DOTTORATO DI RICERCA IN

Computer Science and Engineering

Ciclo XXXVI

**Settore Concorsuale: 09/H1 - Sistemi di Elaborazione delle Informazioni**

**Settore Scientifico Disciplinare: ING-INF/05 - Sistemi di Elaborazione delle Informazioni**

# Embedding AI into Constrained Devices: A Multi-Faceted Approach

**Presentata da:** ***Andrea Agiollo***

**Coordinatore Dottorato**
**Prof. Ilaria Bartolini**

**Supervisore**
**Prof. Andrea Omicini**

**Co-Supervisore**
**Daniele Turrin**

Esame finale anno 2024

# Abstract

Thanks to their data-driven nature, machine and deep learning approaches have recently reached super-human performance, promoting the last artificial intelligence (AI) spring. Accordingly, the application of such techniques to the industrial world has vastly grown in popularity. However, the most common deep learning models, namely neural networks (NNs), are characterised by an intrinsic trade-off between performance and efficiency. Focusing on raw performance, recent efforts produced highly complex NN models made of several millions or even billions of parameters. This complexity hinders the application of AI into industrial devices and appliances characterised by limited computational capabilities and resources. Accordingly, in this thesis, we focus on the *embedding AI into constrained devices* problem, to which we refer to as the open research challenge of applying AI techniques to devices characterised by limited computational capabilities and resources. We tackle the embedding AI task reframing the problem from a NN efficientisation perspective, where the aim is the minimisation of the resource usage of NNs, either during their optimization process or their deployment phase. We propose a pioneer multi-faceted approach in which we consider both *(i)* the available efficientisation approaches – aiming at analysing and overcoming some of their limitations –, and *(ii)* to leverage Neuro-Symbolic integration (NeSy) mechanisms to tackle the efficientisation perspective. As a result of our twofold perspective, we shed new light on the NN efficientisation issue, highlighting the groundbreaking opportunities available leveraging NeSy systems.

**Keywords:** Neural Networks · Embedded Devices · Resource-Constrained Environments · Efficientisation · Neuro-Symbolic Integration

*Failure is a part of the process.*
*You just learn to pick yourself back up.*

# Acknowledgements

Throughout my PhD journey, I had the pleasure of meeting several people who influenced me profoundly. Without them, my PhD wouldn't have been the same, and I wish to express my deepest gratitude to all of them.

Firstly, I'd like to thank from the bottom of my heart my supervisor, Prof. Andrea Omicini. He guided me, sharing his wisdom and his enlightened visions on the research world, as well as his light-hearted spirit and cheerfulness. I'd also like to thank all the amazing people of his research group, with whom I had the opportunity and pleasure of collaborating. In particular, I'd like to thank Dr. Giovanni Ciatto and Dr. Roberta Calegari for being inspirational and for shaping what my idea of "a good researcher" actually is. Finally, I'd wish to thank Matteo Magnini, Andrea Rafanelli, and Federico Sabbatini for being great co-authors and for all the valuable discussions about research and PhD life.

I'd like to thank the awesome people who allowed me to explore new places during this PhD journey. In particular, my deepest gratitude goes to Prof. Catholijn Jonker, Dr. Pradeep Murukannaiah, and Dr. Luciano Cavalcante Siebert for hosting me at TU Delft. Their dedication to scientific and technical knowledge was stimulating and helped me grow significantly. Similarly, I'd like to thank Dr. Rajiv Ashu Khanna for hosting me at Purdue University. His mentorship shined a new light on my perspective of the research world, helping me find new inspiration and motivation to improve myself and thrive.

Along these four years, I had the opportunity of working in many laboratories, where I found a number of friends with whom I've spent some wonderful time. I'd like to thank the colleagues from the LIA laboratory at the University of Bologna for all the wonderful time spent complaining about the glory and misery of a PhD life. Similarly, I'd like to thank the PhD students from the B112 lab at the University La Sapienza of Rome for hosting me whenever I visited and for considering me as an integral part of the group rather than the outlier that I was. A special thank goes to the colleagues at the Purdue University, who made my visiting experience there one of the best memories of this PhD journey.

This PhD journey, and basically my whole life, wouldn't have been the same without the support I received from my wonderful family. I'd wish to thank my parents, Ugo and Cinzia, who supported me in every decision I made, no matter how difficult it was for them. Their selfless spirits encouraged me to pursue my dreams and never give up on them, and pushed me to become the better version of myself. I also thank them for teaching me all the most valuable lessons in life, the value of culture, the importance of learning from your mistakes, and to be kind towards other people. Similarly, a special thank goes to my dearest friend Filippo, whom I consider more like a brother than a long-time friend.

Finally and most importantly, I'd wish to dedicate this thesis to my beloved partner, Enkeleda, who shared with me all the ups and downs I've encountered through this journey. I'd wish to thank her the most, for supporting and enduring me from the get-go. She believed in me whenever I didn't, and she was home when I needed it the most. She pushed me to go the extra mile, and never give up whenever I felt lost, sparking a light to illuminate my world. I don't think I would have ever made it this far without her, and I wish I will never experience what it means to be without her.

# Contents

# Chapter 1

# Introduction

Recently, the proposal of Machine and Deep Learning (ML and DL) approaches gave rise to the new Artificial Intelligence (AI) spring. This increased interest in AI solutions is due to the groundbreaking performance that data-driven ML approaches show, where, humanly-designed models semi-automatically learn task-solving procedures from data via some sort of optimisation mechanism. The variety of tasks which can be tackled in a data-driven way is nearly unlimited, depending on the model and optimisation procedure considered. Accordingly, the application of AI to the industrial world and its appliances has recently grown in popularity, leading to ML and DL techniques becoming the de-facto solution for tackling complex tasks concerning computer vision [Liu et al., 2020], natural language processing [Birjali et al., 2021] and many more [Pawlicki et al., 2022]. However, striving to reach human-like capabilities, most efforts in this field focused solely on increasing the performance of AI technnologies – e.g., Neural Networks (NNs) –, at the expense of their complexity. Only in the last five years, the size of the state-of-the-art NNs have increased by a factor 1000$\times$ for natural language processing tasks and 100$\times$ for computer vision tasks, resulting in applications targeting high-end devices only. However, the application of such AI techniques to industrial devices and appliances characterised by limited computational capabilities and resources remains a fundamental issue for enabling the pervasive deployment of AI solutions. This issue is particularly relevant in the industrial world where appliances are developed focusing on saving costs, reducing resource consumption and relying on computationally—constrained components. Internet of Things platforms represent the most relevant example – but not the only one –, given the ever increasing success of AI-enabled applications in this realm attracting the interest of industries and investors [1].

In this thesis, we refer to the open research challenge of applying complex AI techniques to devices characterised by limited computational capabilities and resources as the *embedding AI into constrained devices* problem. Accordingly, we tackle the embedding task, focusing on the NN efficientisation perspective and proposing a multi-faceted approach in which we consider both *(i)* exploring and overcoming some of the limitations of the available efficientisation approaches, and *(ii)* to leverage a novel family of approaches – relying on the integration of symbolic and sub-symbolic models, namely Neuro-Symbolic integration (NeSy) – to tackle the efficientisation perspective. This multi-faceted perspective represents a pioneer approach in the NN efficientisation context, as it showcases the efficiency improvements achievable through NeSy, previously ignored.

Given the relevance of the embedding task, few recent works proposed different approaches to efficientise complex AI models such as NNs, aiming at enabling their deployment into resource constrained devices. The set of available NN efficientisation approaches can be roughly categorised as:

- *pruning*, representing the class of approaches that identify and remove the non-relevant substructures of a NN [Liang et al., 2021];

- *quantisation*, representing the set of approaches used to reduce the precision – e.g., floating point to integer representation – of NN weights to compress its footprint and consume less memory [Liang et al., 2021];

- *distillation*, representing the set of approaches proposing to couple multiple NNs and force the transfer of knowledge from a large model to a smaller one, without incurring in performance degradation [Gou et al., 2021];

- *architecturing*, representing the class of techniques aiming at identifying efficient architectures of NNs [Ren et al., 2021];

[1] https://straitsresearch.com/report/artificial-intelligence-in-iot-market

- *data coresets*, representing the set of approaches proposing to identify the most relevant samples of a dataset and train the NNs model only on them, thus reducing training time and energy consumption [Feldman, 2020];
- *distributed learning*, representing the class of approaches aiming at distributing the learning process among several devices [Zhang et al., 2021a].

In the remainder of the thesis we refer to these approaches as *classic* approaches, as they represent the popular component of the multi-faceted approach that we followed.

In the context of *classic* efficientisation approaches, we focus specifically on *architecturing*, *data coresets* and *distributed learning*. The motivations for such focus are diverse, but can be quickly summarised as follows: *(i)* architecturing approaches have been proven superior against *pruning*, *quantization* and *distillation* approaches [Blalock et al., 2020], as the design of an ad-hoc model structure allows for higher freedom and achievable efficiency; *(ii)* data coresets approaches target explicitly the data footprint of NN models, thus representing the most effective and popular set of approaches to reduce the resource requirements during the NN training phase; and *(iii)* while it is true that distributed learning approaches reduce local resource consumption, the limit of the achiavable efficientisation is not well explored in these setups, as well as the limitations that arise from the introduction of a distributed framework—such as the need for resource management schemes.

*Architecturing* approaches usually lack mechanisms to limit the architecture complexity – while promoting variability – and learn their inner working principles. Therefore, we aim at overcoming such issues proposing both *(i)* an efficient modular search of basic components of a NN architecture via increasing its depth; and *(ii)* a three-way adversarial learning approach relying on graph neural networks to learn which operations and interconnections between NN components are required for building the best performing architecture, in a single shot setup without exploration processes.

*Data coresets* approaches represent a valid solution for learning with less data. However, they usually rely on complex approaches to identify and select relevant samples to be used for NN optimisation. These complex approaches – usually relying on gradient matching techniques – require large amount of resources, thus hindering the advantages of training on fewer data. In this context, we aim at overcoming such issue proposing to leverage a set of simple and resource efficient properties of NN to identify *clean* samples to be used during training.

*Distributed learning* – of which Federated Learning (FL) represents the de-facto solution – can help reducing the amount of data required for training by leveraging the collective intelligence of distributed devices. By enabling collaboration between parties, FL achieves faster convergence and better generalization performance compared to traditional centralised learning methods. However, the energy and resource utilization optimisation represents an overlooked issue in most FL scenarios. Therefore, we aim at overcoming such issue proposing a novel energy and resource aware client selection approach, showcasing how it is possible to obtain resource usage improvements only leveraging a smart client selection mechanism, achieving minimal impact on the general FL pipeline.

In this thesis we state that, along with popular *classic* efficientisation approaches – e.g., architecturing, pruning, etc. –, it is worth exploring the usage of NeSy systems to study their achievable efficiency improvements. NeSy approaches rely on the hybridisation of symbolic and sub-symbolic realms, two components that have been mostly considered in duality throughout AI history [Besold et al., 2017, d'Ávila Garcez et al., 2019]. Symbolic approaches usually focus on rational intelligence – e.g., deduction, abduction, induction, etc. –, aiming at endowing machines with human-like, automated reasoning capabilities [Brachman and Levesque, 2004]. Conversely, sub-symbolic approaches mostly focus on intuitive intelligence, aiming at enabling information mining and processing from raw data, resulting in powerful pattern matching and recognition capabilities [Rocha et al., 2012]. Amongst the classes of hybridisation techniques, two relevant sets of approaches propose to *inject* and *extract* symbolic knowledge into NNs, and may come in handy when considering the efficientisation perspective. Symbolic Knowledge Injection (SKI) frameworks aim at steering the NN learning process towards a predefined goal—e.g., discrimination-free predictions. Here, the knowledge base injected into the NN carries complex logical information which is otherwise difficult to learn and elaborate from numerical data using bare NNs. Therefore, the simple intuition is that injection can be leveraged as a tool to remove part of the complexity burden from the underlying NN and obtain simpler and more efficient sub-symbolic models. On the other hand, Symbolic Knowledge Extraction (SKE) frameworks aim at producing symbolic knowledge that reflects the behaviour of the NN predictor with high fidelity. Here, the complexity of the extracted symbolic knowledge can be bounded, producing an explainable program more efficient than the original NN model.

While being a theoretically valid assumption that NNs hybridisation results in efficiency benefits, the state-of-the-art lacks a solid empirical proof of such phenomenon. SKI approaches usually measure the quality of their injection mechanism in terms of performance gain over an uneducated ML counterpart [Diligenti et al., 2017b, Xu et al., 2018], while SKE approaches mostly focus on the fidelity of the extracted knowledge [Hailesilassie, 2016, Johansson et al., 2022]. Therefore, the necessity of identifying a set of Quality-of-Service (QoS) metrics for NeSy mechanisms thoroughly analysing their efficiency improvements arises. Exploiting this lack, we highlight efficiency gains that SKI and SKE mechanisms may obtain, focusing on the definition of different efficiency QoS measures such as energy, memory and data savings. The proposed definitions applied to a set of state-of-the-art injection mechanisms highlight how both SKI and SKE bear improvements in data and energy efficiency in various contexts, providing a solid proof of the theoretically assumption originally made.

**Goals of the thesis.** To summarise, the proposed thesis aims at tackling the cumbersome task of embedding complex AI – and especially ML and DL – models in those devices and scenarios characterised by strong resource limitations. Similarly to the state-of-the-art, we focus on efficientisation of NN models as a proxy for the embedding task, analysing the broad set of resources requiring to be efficiently minimised for enabling the embedding procedure. We propose tackling the efficientisation issue in a multi-faceted fashion, focusing on both *(i)* optimising popular approaches dealing with raw efficientisation of ML and DL systems, as well as *(ii)* highlighting the efficientisation opportunities available from the integration of symbolic components into the ML and DL pipelines—namely, NeSy. Along this line, the goals of this thesis can be summarised as follows:

1. analysing the similarity between embedding and ML/DL efficientisation,

characterising the set of resources to be taken into account for successful efficientisation;

2. identifying the most common approaches for dealing with the issue of ML/DL efficientisation in resource-constrained environment;

3. tackling few of the limitations of the current most popular approaches for efficientisation of DL systems;

4. proposing to leverage neuro-symbolic integration techniques to tackle efficientisation, aiming at highlighting efficiency benefits of NeSy systems;

5. proposing a set of rigorous metrics to measure efficiency improvements obtained through NeSy, along with other relevant properties such as robustness and trustworthiness.

**Structure of the thesis.** Following the bi-partite approach followed to tackle the efficientisation issue, the thesis is structured into two main parts, namely *Embedding AI via Classic Efficientisation* and *Efficientisation via Neuro-Symbolic Integration.*

**Embedding AI via Classic Efficientisation.** In Part I we focus on the set of most classical approaches to efficientise ML and DL models, focusing on goals 1 to 3. We start from the definition of embedding and its correlation to ML and DL models efficientisation with respect to a set of resource metrics of interest in Chapter 2 and tackle goal 2 by analysing the set of available efficientisation approaches. We refer to this available techniques as the set of *classic* efficientisation techniques, focusing mostly on raw reduction of resource usage. Then, we accomplish goal 3 in Chapters 3 to 6, by identifying a set of limitations of the available approaches and presenting how we propose to overcome such limitations.

**Efficientisation via Neuro-Symbolic Integration.** In Part II we focus on leveraging NeSy approaches to efficientise ML and DL models, focusing on goals 4 and 5. We start by providing an overview of popular NeSy approaches in Chapter 7, focusing particularly on SKI and SKE, representing the most useful NeSy tools for dealing with NN efficientisation. Thereafter, we focus firstly on SKI in Chapter 8 and SKE in Chapter 9, showcasing their applications in the efficientisation perspective. We conclude Part II, by providing an additional set of metrics to measure relevant NeSy properties related to the efficiency perspective in Chapter 10, and suggesting the possible future of NeSy trustworthiness – for which efficiency is one key component – in Chapter 11.

Finally, Part III concludes the thesis, summarising the conclusions and sketching possible future research directions.

# Part I

# Embedding AI via Classic Efficientisation

One of the first conditions of happiness is that the link between man and nature shall not be broken.

*Lev Nikolàevič Tolstòj*

# Chapter 2

# Embedding as Efficientisation: State of the Art

This chapter contains contributions from [Agiollo and Omicini, 2021, Agiollo et al., 2022].

To kick off our investigation we first consider what are the requirements and goals of the integration of AI models into embedded and resource-constrained devices task. To this end we elicit the set of limitations affecting embedded devices, considering a set of popular resource-constrained boards as our targets and showcasing their shortcomings when it comes to deploying complex ML and DL models. These limitations are tightly linked with the devices hardware limitations, as they are thought as flexible single-purpose devices rather than powerful multipurpose machines. The set of embedded devices limitations translate directly into a set of resource metrics to take into account whenever dealing with such devices. Eliciting such limitations and the corresponding metrics, we translate the integration of AI into resource-constrained devices into the task of efficientisation of NN models—following the state-of-the-art approaches. Accordingly, we define the *efficientisation* task as the process of minimising the resource usage of NN models, either during their optimization process or their deployment phase.

After having defined the *efficientisation* task, we provide an overview on the set of available approaches to tackle this task. Depending on the particular scenario, learning task at hand, and efficiency priority, several different techniques might come in handy. Overall, there exists no silver-bullet solution for tackling the NN efficientisation problem, as each solution presents their advantages and drawbacks.

## 2.1 Embedding as Efficientisation

### 2.1.1 Limitations of Embedded Devices

The design process of low-end commercial appliances is often characterised by a huge focus on costs saving. Indeed, even the introduction of a single hardware component could turn out to be critical to the design of such devices—e.g., adding a costly component to the hardware needed to build the final appliance may substantially increase its consumer price and hinder its chances of commercial success. To account for such issue, embedded devices are usually considered as the hardware platform for running specific complex tasks in these appliances. More in detail, an embedded device is a part of the larger computing system serving a specific purpose, via the execution of a particular task or set of tasks. Also known as a dedicated or single-purpose device, an embedded device is embedded or included within the larger system and is usually chracterised by limited hardware, usually featuring *(i)* 8-bit microcontrollers, *(ii)* application-specific integrated circuits (ASICs), or *(iii)* dedicated digital signal processors (DSPs).

Embedded devices allow to cap the price, while achieving strong flexibility – i.e., most embedded devices are fully programmable – and performance—depending on the target application. However, these devices are characterised by a set of strong limitations when it comes to the integration of AI techniques in commercial appliances. The hardware limitation of embedded devices clashes strongly with the ever-increasing requirements of modern AI approaches, relying on Neural Network (NN) models. Not only embedded devices are characterised by limited computational resources – as they rely on microcontrollers, ASICs or DSPs –, but the amount of memory at their disposal is also limited, as they mostly target scenarios where storage of information is not considered a priority. Finally, depending on the considered scenario, the embedded devices may be battery-powered – e.g, mobile devices, robots, etc. –, thus requiring to take into account the energy consumption during their deployment.

To represent faithfully the limitations of the most commonly used embedded devices we present them along with their computational capabilities and available memory in Table 2.1. In this context, the FLOPS metric represents a measure of the computational capabilities of the selected board and will be inspected more in detail in Section 2.1.2. The limitations of the selected devices vary broadly. The OpenMV H7 Plus microcontroller board represents the most limited device, characterised by an Arm Cortex M7 processor with a single core, low clock speed and small memory to obtain low power consumptions and enable the deployment on edge and battery-powered devices. On the other hand, the Nvidia Jetson-AGX Xavier board is characterised by high clock-speed – similar to most full-fledge personal computers –, high amount of memory and the availability of a powerful GPU. Depending on the scenario at hand, the selected embedded device may vary and consequently its limitations. For example, whenever the Raspberry PI device is used, the memory and processing power limitations represent the most relevant aspects to consider. Meanwhile, the energy consumption aspect represents a more relevant component whenever an Nvidia Jetson-TX1 – or similar – board is used.

Table 2.1: Comparison between available embedded devices.

| **Device Name** | **RAM** | **Cores** | **Work Frequency** | **#FLOPS** | **GPU Availability** |
|---|---|---|---|---|---|
| OpenMV H7 Plus [Agiollo and Omicini, 2021] | 32 MB | 1 | 480 MHz | $0.48 \times 10^9$ | ✗ |
| Raspberry Pi-3B+ [Sajjad et al., 2020] | 1 GB | 4 | 1400 MHz | $5.3 \times 10^9$ | Broadcom VideoCore IV |
| Odroid Xu-4 [Hossain and Lee, 2019] | 2 GB | 8 | 2000 MHz | $6.25 \times 10^9$ | Mali-T628 MP6 |
| Latte Panda [Hossain and Lee, 2019] | 4 GB | 4 | 1440 MHz | $7.01 \times 10^9$ | Intel HD Graphics |
| Raspberry Pi-4 [Süzen et al., 2020] | 8 GB | 4 | 1500 MHz | $13.5 \times 10^9$ | Broadcom VideoCore VI |
| Nvidia Jetson-Nano [Süzen et al., 2020] | 4 GB | 4 | 1400 MHz | $472 \times 10^9$ | 128-Core NVIDIA Maxwell |
| Nvidia Jetson-TX1 [Stamoulis et al., 2018] | 4 GB | 4 | 1700 MHz | $1 \times 10^{12}$ | 256-Core NVIDIA Maxwell |
| Nvidia Jetson-TX2 [Süzen et al., 2020] | 8 GB | 6 | 2000 MHz | $1.3 \times 10^{12}$ | 256-Core NVIDIA Pascal |
| Nvidia Jetson-AGX Xavier [Jiang et al., 2021] | 16 GB | 8 | 1900 MHz | $11 \times 10^{12}$ | 512-Core NVIDIA Volta |

To complete the picture of embedded device limitations, we present the resource requirements of available state-of-the-art NN models in Table 2.2. Most of the available state-of-the-art NNs present footprints greater than the memory availability of the most constrained devices of Table 2.1. Moreover, it is possible to notice that most models are characterised by a large amount of parameters, translating directly to a large amount of computations to be run at inference time. Most limitations for running AI on these devices come from the computational power of the device. Indeed, the inference process of NN models require high amount of computational power to run quickly, thus introducing also latency limitations on embedded devices. To this end, minimising the memory footprint of the NN model – by reducing its number of parameters – result in smaller memory requirements and reduced amount of computations, thus reducing the inference latency as well. However, the proportionality between number of parameters of a model and its inference time is not linerar, as the nature of the NN operations might differ vastly. Similarly, while there exists a correlation between model size and energy consumption, this is not straightforward. For example, deeper models with fewer parameters require large amount of computations and increased latency, as the parallelization over consecutive layers is limited. Therefore, reducing the number of model parameters does not represent the silver bullet solution for solving the integration of AI into constrained devices issue.

Table 2.2: Footprints of available state-of-the-art NNs. The number of parameters of each model is expressed in millions (M). The memory size is expressed in MBs and is computed as four bytes times the number of parameters, since each parameter is a floating point variable. FLOPs are the number of floating points operations required to run a single instance of a given model and are indicated in billions. The upper half refers to models targeting image classification, while the bottom half refers to models targeting object detection.

| **Model** | **#Parameters (M)** | **Footprint (MB)** | **#FLOPs (B)** |
|---|---|---|---|
| 1.0 MobileNet-224 [Howard et al., 2017] | 3.3 | 13.2 | 0.28 |
| EfficientNet-B0 [Tan and Le, 2019] | 5.3 | 21.2 | 0.39 |
| DenseNet-169 [Huang et al., 2017] | 14 | 56 | 3.5 |
| Inception-v3 [Szegedy et al., 2016] | 24 | 96 | 5.7 |
| ResNet-50 [He et al., 2016] | 26 | 104 | 4.1 |
| VGG-16 [Simonyan and Zisserman, 2015] | 138 | 552 | 16 |
| SSD300-MobileNet [Liu et al., 2016b] | 6.8 | 27.2 | 1.2 |
| EfficientDet-D0 [Tan et al., 2020] | 3.9 | 15.6 | 2.5 |
| FasterRCNN-MobileNet [Ren et al., 2015] | 6.1 | 24.4 | 25.2 |
| SSD300-Deeplab [Liu et al., 2016b] | 33.1 | 132.4 | 34.9 |
| FasterRCNN-VGG [Ren et al., 2015] | 138.5 | 554 | 64.3 |
| YOLOv3 [Redmon and Farhadi, 2018] | 40.5 | 122 | 71 |

Inspired by these insights, in this thesis we do not focus solely on the minimization of NN parameters, but rather we keep a general and holistic view on the minimisation of the required resources issue, focusing on all limitations of embedded devices. More in detail, we follow the state-of-the-art approaches, re-framing the integration of AI into constrained/embedded devices problem as a NN efficientisation task. The integration of AI into embedded devices requires the minimisation of the resources required by NN models to be optimised and compute their prediction. Therefore, we define the *NN efficientisation* task as the problem of

> *minimising the resource usage of NN models, either during their optimization process or their deployment phase.*

To this end, we consider tackling the issue by investigating the set of techniques that allow reducing the *(i)* memory and computational footprint of NN models, *(ii)* their latency, *(iii)* their energy consumption, or *(iv)* the amount of data they require to be optimised.

### 2.1.2 Efficientisation Metrics

Having defined the translation from the embedding problem to the NN efficientisation task as the minimisation of resource usage of NN models, we now identify the set of relevant metrics to measure for achieving resource minimisation. Accordingly, we focus on four different metrics that correlate to the limitations of embedded devices and allow to measure effectively if – and to what extent – efficientisation is effective.

**Computational capabilities and requirements.** To measure the computational power of devices – and compare it with the NNs computational requirements – the *FLOPS* and *FLOPs* metrics are commonly used. FLOPS – FLoating point Operations Per Second – refers to the number of floating point operations that a device is capable to complete in one second. FLOPs – FLoating point OPerations –, on the other hand, represents the number of floating points operations required to run a single algorithm—e.g., NN inference. Knowing the number of cores of a device, its clock frequency, and the number of FLOPs per clock cycle that the device is capable to handle, it is possible to compute the theoretical FLOPS performance of a device by means of the following equation:

$$FLOPS = cores \times clock\ frequency \times \frac{FLOPs}{cycle} \tag{2.1}$$

For example, the OpenMV H7 Plus board – presented in Table 2.1 – runs at 480 MHz, and is capable of one FLOP per cycle. Therefore the board has a theoretical performance of $0.48 \times 10^9$ FLOPS. This measure, when compared with the FLOPs requirements of modern NNs – see Table 2.2 –, clearly indicates the strong limitations of that sort of device for AI applications. Indeed, the board is *theoretically* capable of running a single inference of a MobileNet in

$$t = \frac{FLOPs}{FLOPS} = 0.583\, s.$$

This would allow incoming inputs to be processed at a rate of 1.72 Frames Per Second (FPS), which is not enough for most real-time commercial solutions. Moreover, real-world experiments show how the real performance of embedded devices is far from their theoretical limit, thus increasing concerns on the applicability of AI techniques on similar devices [Agiollo and Omicini, 2021].

The *FLOPs* measure represents an intrinsic metric of complexity of the model at hand—being independent from the device on which the model is deployed. Therefore, in this thesis, we will focus almost exclusively on the *FLOPs* measure, while ignoring FLOPS. Similarly, we consider an alternative measure of FLOPs, namely MACs (Multiply-Accumulate Operations). MACs count the number of multiply-accumulate operations, which involve multiplying two numbers and adding the result. This operation is fundamental to many linear algebra operations, such as matrix multiplications and convolutions. Therefore, they are often used as a more specific measure of computational complexity of convolutional neural networks (CNNs).

**Latency.** The amount of time required for a NN model to draw a single prediction is one of the most relevant and impactful efficiency measures. In this thesis, we refer to such time-lapse as *latency*. Although not being totally independent from the device on which the model is deployed, latency is proportional to model complexity. On the same hardware setup, a model achieving a smaller latency – against a slower counterparts – highlights the model ability to compute relevant predictions in useful time, thus suggesting lower model complexity. However, the reduction of the model computational requirements represents only part of the solution to achieve low latency, as the optimisation of the sequence of operations represents a fundamental aspect to take into account. Sparsely-structured operations might slow down the inference process due to their inefficient computation at hardware level. Moreover, input data complexity and quality might alter the latency achieved by the predictor, as the inference over different – yet structurally analogous – samples may take vastly different timings [Shumailov et al., 2021].

As latency represents a device-dependent measure, in the remainder of this thesis, we assume latency to be computed by averaging the time required to draw a number of predictions from a reference test dataset over a specific hardware setup. More formally, we define the latency of a predictor $N$ as the average time required to draw a single prediction from a dataset $T$:

$$\Lambda(N, T, H) = \frac{1}{|T|} \sum_{t \in T} \Theta(N, t, H),$$

where $\Theta(N, t, H)$ represents the time required to draw a prediction from $N$ on the input $t$ over the hardware setup $H$.

**Data footprint.** NN models rely on data-driven training algorithms to learn solving the given task. In this context, the data hungriness of NN models represent a fundamental factor. Indeed, in the NN context there exists a direct proportionality between the amount of data used for optimising the model at hand and the corresponding achievable performance. Moreover, the *quality* of the data – here intended as its representativeness of the task at hand – is crucial for the predictor to learn effectively. Finally, inefficient data management can result in decreased accuracy and increased energy consumption, negatively affecting the system at hand. All such requirements make the data collection process time-costly and possibly affected to subjectivity or uncertainty. Therefore, NN models capable of learning similar concepts from lower amount of data are highly desirable and are to be considered more efficient. Similarly, NN models capable of learning from the same amount of data, but in fewer optimisation iterations are also to be considered more efficient. A smaller amount of data required for the optimisation procedure also translates to fewer energy wastes and quicker convergence.

Inspired by these insights, we here define the *data footprint* of a given predictor as one of the efficientisation metrics to be taken into account. Informally, the data footprint of a predictor $N$ is the amount of data it requires to be trained to reach a certain performance level. Hence, assuming that a predictor $N$ is trained on a dataset $D$ – of samples of potentially different dimensions –, via some training process involving $e$ epochs, and that it reached a performance level $\pi(N, T)$ over a test set $T$ – and according to some test dataset $T$ –, we define its data footprint as follows:

$$\Delta_\pi(e, N, D, T) = \frac{e}{\pi(N, T)} \sum_{d \in D} \beta(d)$$

where $d$ is a single training sample, and $\beta(d)$ is the amount of bytes required for its in-memory representation, and $\pi$ is some performance score of choice. As the reader may notice, the data footprint is *directly* proportional to the number of epochs $e$, to the size of the training set, and to its dimensionality; whereas it is *inversely* proportional to the performance score of the resulting predictor.

**Energy consumption.** The development and optimisation of NN models require significant amounts of energy, especially when working with large datasets or model architectures. Measuring the energy consumption of training and inference of neural networks is crucial for understanding the environmental impact of these processes and identifying opportunities for improvement. Moreover, measuring the energy consumption levels represents an essential task when models are deployed on edge battery-powered devices. By monitoring energy consumption, developers and researchers can identify areas where efficiency improvements can be made, such as optimizing model architecture, reducing its precision, or exploiting hardware acceleration. Therefore, the energy consumption metric represents a fundamental measure to consider whenever dealing with the integration of AI systems in embedded/constrained devices. Finally, measuring energy consumption is essential for ensuring that AI systems are not only accurate but also sustainable.

In the data-driven AI life-cycle it is possible to consider the *model training* and *model deployment* – where the predictor runs multiple times with a frequency depending on the specific application at hand – phases as the most resource hungry. Indeed, training requires a huge – yet predictable – amount of predictor executions and updates, whereas deployment might be very energy demanding depending on the predictor usage frequency and life expectation—which are typically hard to anticipate. Accordingly, as far as energy consumption is concerned, we are interested in measuring the energy consumption of the training and deployment phases, individually. Generally speaking, we can define the average energy consumed by a NN predictor $N$ on a per-single-inference basis:

$$\Upsilon^{\mathsf{i}}(N, T) = \frac{1}{|T|} \sum_{t \in T} \upsilon(N, t),$$

where $\upsilon(N, t)$ identifies a function to measure the energy consumption of a single forward run of a NN predictor $N$ on a single sample $t$. Such a function may measure directly the battery depletion process or estimate the energy consumption level via proxy metrics, such as the heat dissipated by the hardware running the predictor, during the single inference $N(t)$. To account for variability accross samples, the considered measure estimates the inference energy consumption as the *average* over a test dataset $T$.

Relying on the average energy consumption definition, it is possible to define the energy consumed while training $\Upsilon^{\mathsf{t}}$ as:

$$\Upsilon^{\mathsf{t}}(e, N, T) = e \cdot \sum_{t \in T} [\upsilon(N, t) + \beta(N, t)]$$

Our definition assumes the training involves $e$ epochs, and that during each epoch the whole training set $T$ is used to update the predictor $N$. The definition also assumes $\beta(N, t)$ is a function estimating the energy consumed by the optimization phase for each data sample $t$—namely, the backpropagation phase.

## 2.2 Efficientisation State of the Art

In this section, we give an overview on the set of popular techniques available in the literature for tackling the NN efficientisation task. These approaches can be roughly categorised into six classes, namely:

- neural network *pruning* – i.e., the process of systematically removing part of the parameters from an existing NN model
- neural network *quantization* – i.e., the process of storing the weights and activation tensors forming the NN model in a lower bit precision than the 16 or 32-bit precision they are usually trained with
- *neural architecture search* – i.e., the process of automating the construction of effective structures of NNs
- *knowledge distillation* – i.e., the process of transferring the knowledge learnt from a large teacher model to a small student model
- *coreset construction* – i.e., the process of identifying the most informative subset of data samples, so that the model trained on the selected subset achieves similar generalization performance to the model trained on the whole training set
- *distributed learning* – i.e., the process of splitting the learning task and distribute it accross several devices.

Before delving in depth on each class of the available efficientisation approaches, we give a brief background on Neural Network models, defining their main terminologies.

**Brief Background on Neural Networks.** Neural networks are biologically-inspired computational models, made of several elementary units (neurons) interconnected into a *directed-acyclic* graph (DAG) via *weighted* synapses. NN can be *trained* on data via Stochastic Gradient Descent (SGD) and backpropagation [Hecht-Nielsen, 1988] and exploited into both supervised and unsupervised learning tasks such as classification, regression, and anomaly detection. More precisely, during the training phase synapses weights are optimised automatically to minimise a given loss function over the data samples at hand, thus enabling NN learning from data. The SGD method involves arbitrarily-sized subsets of the dataset (a.k.a. batches) to be processed a finite – i.e., controllable – amount of times—i.e., epochs. Hence, the complexity of SGD can be finely controlled and adapted to the computational resources at hand—e.g., by making the learning process incremental, and by avoiding all data to be loaded in memory. Moreover, SGD can be applied to several sorts of predictor families (there including NN and generalised linear models), as it only requires the target loss function to be differentiable w.r.t. the model's parameters. For all these reasons, despite the lack of optimality guarantees, SGD is considered as very effective, scalable, and malleable in practice, hence it is extensively exploited in the modern data science applications.

### 2.2.1 Pruning

One set of popular approaches for reducing the resource requirements of NN during their deployment phase is neural network pruning [Blalock et al., 2020]. Network pruning is defined as the process of systematically removing part of the parameters from an existing network, thus reducing the network complexity while aiming at keeping the performance degradation small. Typically, the starting NN is large and accurate, and the goal is to produce a smaller network with similar level of performance—e.g., accuracy. Mathematically speaking, pruning entails taking as input a model $f(W)$ – where $W$ represents the model parameters – and producing as output a new model $f(M \odot W)$. Here $M \in \{0,1\}^{|W|}$ is a binary mask that fixes certain parameters to 0, and $\odot$ is the elementwise product operator. In practice, rather than using an explicit mask, pruned parameters of $W$ are fixed to zero or removed entirely.

Generally speaking NN pruning algorithms require the network to be firstly trained to convergence. Afterwards, a scoring function is defined to assign a score to each parameter or structural element in the NN. The obtained scores are used to prune the NN accordingly, removing less relevant elements. Pruning reduces the accuracy of the network, thus the NN model is trained further – i.e., fine-tuned – to recover part of the lost performance. The overall process of pruning and fine-tuning is often repeated several times – gradually reducing NN footprint – up until a satisfaction criterion is met, generally based on a trade-off between the number of parameters pruned and the obtained performance loss.

State-of-the-art pruning methods vary primarily in their choices regarding sparsity structure, scoring, and scheduling.

*Unstructured pruning* methods prune individual model's parameters, thus producing a sparse NN. These methods produce smaller – w.r.t. the total number of parameters – networks which, however, may not be arranged in a fashion resulting in computation speedups. Indeed, the intrinsic sequential nature of NN operations does not favour sparse networks for achieving faster inference time or smaller resource consumption. Other methods consider pruning groups of parameters – i.e., *structured pruning* –, removing entire neurons, filters, or channels. While these methods usually output higher performance degradation, they enable the exploitation of hardware and software optimized for dense computation [Li et al., 2017].

NN parameters are usually scored based on their absolute values, trained importance coefficients, or contributions to network activations or gradients. Here, different approaches either assign scores *locally* or *globally*. Local scoring approaches consider pruning the fraction of parameters resulting in the lowest scores within each structural subcomponent of the network [Han et al., 2015]. On the other hand, global scoring approaches compare scores to one another independently of the part of the network in which the parameter resides [Frankle and Carbin, 2019].

Finally, available pruning methods vary depending on the amount of parameters/components they prune for each iteration of the pruning algorithm. The simplest approach would be to prune all desired weights at once as done in [Liu et al., 2017b]. Aiming at achieving higher pruning precision and smaller performance degradation, few other approaches consider to prune a fixed fraction of parameters iteratively over several steps [Han et al., 2015] or vary the rate of pruning according to a specific function [Gale et al., 2019].

### 2.2.2 Quantization

One popular approach for decreasing the computational time and energy consumption of NN models is represented by *quantization*. The underlying idea of neural network quantization it to store the weights and activation tensors forming the NN model in a lower bit precision than the 16 or 32-bit precision they are usually trained with. Quantization represents a powerful approach for reducing the computational requirements of NNs. For example, when moving from 32 to 8 bits, the memory overhead of storing tensors decreases by a factor of 4 while the computational cost for matrix multiplication reduces quadratically by a factor of 16. Moreover, several reserach efforts show how NNs are generally robust to quantization manipulation. In other words, NNs can be quantized to lower bit-widths with a relatively small impact on the obtained performance.

NN quantization approaches can be roughly categorised into two main classes of algorithms: *Post-Training Quantization* (PTQ) and *Quantization-Aware-Training* (QAT) [Nagel et al., 2021]. PTQ requires no re-training or labelled data and is thus a lightweight push-button approach to quantization. PTQ is sufficient for achieving 8-bit quantization with close to floating-point accuracy. However, this often comes at the cost of lower accuracy as compared to QAT in the low-precision regime. For this reason, PTQ approaches often focus on the mitigation of accuracy degradation via different strategies. For example, [Banner et al., 2019, Finkelstein et al., 2019] observe inherent bias in the mean and variance of the weight values following their quantization and propose bias correction methods. Similarly, [Meller et al., 2019, Nagel et al., 2019] show that equalizing the weight ranges (and implicitly activation ranges) between different layers or channels can reduce quantization errors.

On the other hand, QAT requires fine-tuning and access to labeled training data – thus increasing the resource requirements of such approaches –, but enables lower bit quantization with competitive results. In QAT frameworks the usual forward and backward pass are performed on the quantized model in floating point, but the model parameters are quantized after each gradient update. An important subtlety in QAT backpropagation is how the non-differentiable quantization operator is treated. Without any approximation, the gradient of this operator is zero almost everywhere. Most approaches differ in the way they tackle this issue, either ignoring the rounding operation and approximating it with an identity function [Bengio et al., 2013, Stock et al., 2021] or removing the rounding operation in the quantization formula [Bai et al., 2019, Choi et al., 2018].

### 2.2.3 Neural Architecture Search

Throughout the NN training phase only network synapses weights are modified, whereas its overall graph structure (topology henceforth) is not allowed to vary. It is rather assumed to be *manually* engineered by data scientists. Therefore, NN development workflows are deeply influenced by the choices by human experts. Network architectures represent the most relevant aspect requiring human contribution. However, as neither theorems nor methods ensure optimal results, human choices may lead to sub-optimal or inefficient solutions.

To avoid inefficiencies introduced by human errors in NN design, *neural architecture search* (NAS) has been proposed [Miller et al., 1989]. NAS automates network architecture engineering: it aims at learning a network topology that can achieve reasonably-good performances on specific tasks, by letting a search algorithm look for the best network topology among the admissible ones. To keep the computational complexity of NAS acceptable, several approaches have been proposed in the literature. Virtually all of them try to *(i)* reduce the search space size and *(ii)* control the whole search duration by leveraging on a *greedy* or *evolutionary* search strategy.

A common means to restrict the search space is to assume the network topology to be composed by a sequence of units called *cells*. Each cell contains a number of *blocks*, connected over a DAG structure. Blocks, in turn, are groups of neurons having a predefined internal organisation—commonly corresponding to a particular mathematical operator. In Convolutional Neural Networks (CNNs), for instance, blocks are commonly constrained to represent convolutional layers, each one representing different sorts of convolutional filters—e.g. 3×3, 5×5, etc. Directed connections among any two cells $A$ and $B$ are modelled as directed connections among the output block of $A$ and the input block of $B$.

State-of-the-art NAS approaches mostly differ in which and how many blocks and cells are exploited, how these can be connected with each others, or which (meta-)heuristic the search of the optimal topology leverages upon. For instance, a method is proposed in [Liu et al., 2018] where NN is built as a sequence of identical cells—so that only the internal structure of one cell is optimised. On the other hand, some popular approaches consider fixing cells operations and identifying the best cell combination that can compose the NN [Chu et al., 2020, Tan et al., 2019]. Concerning the leveraged search meta-heuristic, several approaches consider relying on evolutionary approaches, exploring different architectures through mutations and selection [Real et al., 2019, Casale et al., 2019]. Reinforcement learning also represents a popular meta-heuristic to search for the best composition, as it allows defining multi-objective oriented exploration algorithms [Chu et al., 2020, Tan et al., 2019].

Although being originally defined targetting raw model performance, NAS approaches can be applied to identify resource efficient NN architectures. Few approaches consider incorporating a model complexity metric into the objective function of the search process, either considering a specific hardware [Tan et al., 2019] or defining a hardware-aware search mechanism [Cai et al., 2019]. These approaches generally rely on directly measuring the latency [Tan et al., 2019, Xu et al., 2020] or the computational requirements [Wu et al., 2019] – either in terms of FLOPs or MACs – of the analysed architecture. However, although NAS can be leveraged to identify reource-efficient architectures, these approaches are still vastly unexplored in the NN efficientisation landscape, as they require relevant amount of resources to explore the search space at hand. Finally, it is relevant to notice that NAS approaches can integrate *pruning* [Ding et al., 2022] and *quantization* [Chen et al., 2018] mechanism in their exploration pipeline, helping tackling the efficientisation task from multiple perspectives.

### 2.2.4 Knowledge Distillation

Knowledge distillation has received rapid increasing attention from the community as a representative type of model compression technique. Knowledge distillation approaches aim at effectively learning a small student NN model from a large teacher NN model, so that the performance obtained by the small NN is similar to the original large model. The underlying idea behind knowledge distillation is that the student model is supervised thorughout its training procedure by the large teacher model, mimicking its predictive behaviour to obtain a competitive performance [Gou et al., 2021]. Knowledge distillation represents a powerful tool for NN model efficientisation, as it allows to define a student model whose complexity can be tuned depending on the scenario at hand. However, there exists a trade-off between the teacher-student complexity compression ratio and the performance degradation achievable using knowledge distillation. Indeed, distilling a complex large NN model usually requires a fairly complex student NN to achieve a satisfactory level of performance.

Generally speaking, knowledge distillation approaches differ depending on how the knowledge is transferred from the teacher model to the small student NN. More in detail, such approaches can be roughly categorized depending on two key components: *(i)* the knowledge tehy consider and *(ii)* the distillation algorithm.

Vanilla knowledge distillation approaches consider the knowledge stored in the logits of the prediction of the large teacher model, defining a novel distillation loss which aims at minimising the distance between teacher logits and student logits [Hinton et al., 2015]. More complex approaches consider relying on the knowledge stored in the neurons or features of the intermediate layers – as well as structural connections between layers [Liu et al., 2019b] – of the teacher model, similarly defining an ad-hoc distillation loss to couple the two models [Huang and Wang, 2017].

Distillation algorithms can be categorised depending on whether the teacher model is updated simultaneously with the student model or not, roughly identifying two main categories, namely: *(i) offline distillation* and *(ii) online distillation* *Offline distillation* approaches consider training first the large teacher model on a set of training samples and subsequently distill the teacher knowledge into the student model, keeping the teacher model fixed [Passalis and Tefas, 2018]. On the other hand, *Online distillation* approaches consider to update simultaneously both the teacher and the student model, thus defining a knowledge distillation framework that is end-to-end trainable [Walawalkar et al., 2020].

### 2.2.5 Coreset Construction

Coreset construction approaches propose to identify the most informative subset of data samples belonging to a dataset, so that the model trained on the selected subset achieves similar generalization performance to the model trained on the whole training set. Identifying a smaller subset of relevant data smaples to be used for training the NN model at hand, coreset construction approaches target directly the minimisation of the model's *data footprint* (see Section 2.1.2). Moreover, the reduction of the number of data samples required for training a model impacts the overall energy consumption of the optimisation process, reducing the convergence time and the resource wastes. Therefore, coreset construction approaches represent a very valuable solution to reduce the environmental impact of the NN and DL models optimisation process.

We now define mathematically the coreset construction problem setup, which is useful for understanding the problem complexity and that will be used extensively in Chapter 5. Consider a learning task in which the given large training set is defined as $\mathcal{T} = \{(\mathbf{x}_i, y_i)\}_{i=1}^{|\mathcal{T}|}$, where $\mathbf{x}_i \in \mathcal{X}$ is the input data sample, $y_i \in \mathcal{Y}$ is the ground-truth label of sample $\mathbf{x}_i$, and $\mathcal{X}$ and $\mathcal{Y}$ denote the input and output spaces, respectively. $\mathcal{V} = \{(\mathbf{x}_i, y_i)\}_{i=1}^{|\mathcal{V}|}$ defines a held-out validation set of samples used to test the performance achieved in the learning task. In this context, given a neural network model $h$ with parameters $\theta$ and a loss function $\mathcal{L}$, we can define the loss on a set $S$ of instances as $\mathcal{L}(\theta, \mathcal{S}) = \sum_{i \in \mathcal{S}} \mathcal{L}(\theta, \mathbf{x}_i, y_i)$. Similarly, $\mathcal{L}_{\mathcal{T}}(\theta, \mathcal{T})$ denotes the training loss over the full training set $\mathcal{T}$, and $\mathcal{L}_{\mathcal{V}}(\theta, \mathcal{V})$ the corresponding validation loss. In this setup, the coreset construction task aims at identifying the most informative subset $\mathcal{S} \subset \mathcal{T}$ such that $|\mathcal{S}| < |\mathcal{T}|$ – preferably $|\mathcal{S}| << |\mathcal{T}|$ –, so that the model $h(\theta_{\mathcal{S}})$ trained on $\mathcal{S}$ achieves similar generalization performance to the model $h(\theta_{\mathcal{T}})$ trained on the whole training set $\mathcal{T}$. Mathematically, we can formulate the coreset construction problem by searching for the coreset $\mathcal{S}$ that allows to minimize the validation loss of the model optimized on the same subset of training samples $\mathcal{S}$:

$$\underset{\mathcal{S} \subset \mathcal{T}}{\operatorname{argmin}} \left\{ \mathcal{L}\left( \underset{\theta}{\operatorname{argmin}} \left\{ \mathcal{L}(\theta, \mathcal{S}) \right\}, \mathcal{V} \right) \right\}. \tag{2.2}$$

We refer to the coreset size by the percentage of training data samples selected to construct the coreset, namely $\mathcal{P} = \frac{|\mathcal{S}|}{|\mathcal{T}|}$.

The most popular approaches in the coreset construction context rely on the importance of a sample as its contribution to the loss or the gradient of the loss function throughout training. For example, both Grad-Match [Killamsetty et al., 2021a] and CRAIG [Mirzasoleiman et al., 2020] select the weighted subsets whose gradients best approximate the loss gradient on the entire training dataset every few epochs during training. CRAIG converts the gradient-distance optimization problem into a submodular function solvable using a greedy approach. On the other hand, GradMatch leverages an orthogonal matching pursuit algorithm to closely match the gradient of the loss function on both training and validation samples. While effective, such approaches require frequently re-selecting the coresets during training, thus limiting resource consumption improvements, as the coreset selection algorithm is expensive. To overcome this issue GraNd [Paul et al., 2021] approximates the L2 norm of the gradient early on in training averaging it over many different training initializations.

Decision boundary-based methods select coresets relying on the assumption that the points closest to the decision boundary are the most informative. However, measuring the sample distance from the decision boundaries remains an open issue addressed by relying on the dissimilarity between the classification likelihood of a sample and its neighbours [Margatina et al., 2021] or by leveraging adversarial examples [Ducoffe and Precioso, 2018]. Few other approaches consider identifying coreset samples, based on how well they cluster together, representing each cluster by its centroid sample. In this context, the available approaches differ depending on the clustering approach or metric used, such as [Chen et al., 2010] which considers distance in the feature space, or [Har-Peled and Mazumdar, 2004] which reframes the problem as a minimax facility location. Finally, some recent approaches consider the minimization of the coreset loss directly as a bilevel optimization problem, either maximizing the log-likelihood on a held-out validation set of samples [Killamsetty et al., 2021b], minimizing directly the loss function [Killamsetty et al., 2021c], or leveraging submodular optimization [Iyer et al., 2021, Iyer and Bilmes, 2013].

### 2.2.6 Distributed Learning

*Distributed learning* approaches propose to reduce the amount of data and resources required for training ML and DL models by leveraging the collective intelligence of distributed devices. In this context, *Federated Learning* (FL) approaches are becoming increasingly popular. Most, if not all, FL scenarios consider a central server interacting with multiple users – also called clients or workers – to train a ML/DL model jointly. In this setup, each worker locally trains its model on its private data, while the central server aggregates local updates upon their reception. The groundbreaking idea behind FL is that the training process of the global model takes into account the data of all clients belonging to the federation while never disclosing their nature. Thanks to its unique design, FL achieves high efficiency improvements over centralised training approaches, thanks to its underlying parallelisation paradigm.

Distributed learning and FL approaches can be leveraged to reduce the resource requirements of the optimization process of large NN models. However, when considering to apply FL on embedded and resource-constrained devices, several issues might arise and should be taken into account. For example, while FL intrinsic parallelisation procedure may help distribute the optimisation workload accross different devices, it may also introduce communication overhead issues and corresponding resource wastes. To tackle this issue few works consider analysing the communication scheme of FL over resource-constrained scenarios, proposing mechanisms to reduce the amount of interactions between the clients and the central server [Zhang et al., 2022], or the size of each message [Shah and Lau, 2023].

Due to the limited energy budget of the resource-constrained clients and the heterogeneity of the clients' hardware setup, computation and communication energy consumption must be considered during the FL process. Indeed, energy wastes may cause clients disconnections as well as convergence issues. In this context, the joint optimisation of the energy consumed by each local client along with the minimisation of the energy consumed by the overall federation process represents an open problem. Therefore, few works consider tackling the energy consumption minimisation issue either proposing to formulate the joint learning and communication problem as an optimization problem [Yang et al., 2021b, Kim and Wu, 2021], or to consider local energy consumption, minimising the probability of clients disconnection from the federation [Hamdi et al., 2022b].

Finally, few approaches focus on the client selection process typical of FL. In this context, few approaches consider selecting clients depending on their model *utility* – defined as its potential improvement over the aggregated model – [Lai et al., 2021, Arouj and Abdelmoniem, 2022]. Few other approaches consider selecting the clients depending not only on their model performance, but also considering the communication burden affecting the considered client, such as [Cho et al., 2020].

## Chapter Synopsis

In this chapter we define the requirements of the integration of AI models into embedded and resource-constrained devices, identifying the major limitations that affect popular embedded devices. We reframe the integration of AI into constrained/embedded devices problem as a NN efficientisation task, requiring the minimisation of the resources consumed by NN models to be optimised and compute their prediction. To this end, we translate the embedded device limitations into a set of efficientisation metrics that we will take into account for the remainder of this thesis. Finally, to complete the NN efficientisation picture, we provide a brief overview on the set of avaialable approaches for tackling this specific task, focusing on their advantages and shortcomings. The key aspect to keep in mind throughout the remainder of this thesis is that none of the available approaches represents a silver-bullet solution, as there still are several open challenges for enabling the integration of complex NN models in resource-constrained scenarios.

# Chapter 3

# Efficientisation via Architecture Search

This chapter contains contributions from [Agiollo et al., 2021].

State-of-the-art neural architectures are characterised by an ever-increasing *structural* complexity – in terms of layers, neurons, and their connections –, which is expected to make neural networks even more precise and accurate. However, the structural complexity of NN brings about a number of drawbacks. For instance, it makes training more eager for computational resources and data. Furthermore, it represents a bottleneck in the engineering process of DL systems—which is commonly performed by data scientists, manually. This is where Neural Architecture Search (NAS) [Elsken et al., 2019, Wistuba et al., 2019] comes into play. The general goal of NAS is to *automate* the identification of the best NN structure for a given task, by defining an architecture search space and a corresponding exploration policy (see Section 2.2.3 for more details).

Along with enabling the identification of the best NN model for a given task, NAS approaches can be leveraged to tackle the NN efficientisation issue. Indeed, few research efforts propose leveraging NAS to automate the identification of NN architectures that minimise the memory footprint of the obtained model or its latency [Tan et al., 2019, Wu et al., 2019]. Generally speaking, these approaches model the NAS as a search problem in the space of all possible NN architectures where the model selection policy takes into account the architecture resource usage. However, no work so far focuses on controlling network structural complexity while encouraging operations variability. Accordingly, in this chapter we propose *Shallow2Deep*, a novel NAS algorithm aimed at keeping NN structural complexity under control, while promoting structural variability over its elements. Our algorithm enforces structural constraints over the searched NN architecture, limiting the NN structural complexity and therefore its resource hungriness. In other words, *Shallow2Deep* fits NN design by providing a means to control the depth of a NN and specialise NN sub-components depending on their location.

Our solution differs from other NAS approaches in a number of ways. First, it promotes local *variability* in NN architectures—meaning that it supports and encourages variability in the different layers composing a NN. Then, it favours local *specialisation* of NN sub-structures—thus letting each layer of the NN specialise on different tasks, depending on their depth. Finally, it promotes progressive complexity, avoiding *overthinking*—a well-acknowledged [Kaya et al., 2019] tendency of deep NN to learn too many concepts, becoming more complex than needed. We present a full operational formalisation of the *Shallow2Deep* algorithm along with a number of experiments showing its practical feasibility and versatility.

## 3.1 Shallow2Deep

In this section we present *Shallow2Deep*, a novel exhaustive NAS algorithm: we first present its architecture design along with the corresponding search space (Section 3.1.1), then we discuss its overall working principle, its fundamental hypotheses, and the details of its composing modules.

### 3.1.1 Architecture Design

Most popular NAS approaches blindly build a NN architecture by repeating the same *elementary* structure (cell) several times—assuming that the internal structure of a cell has been manually optimised. This sounds like a reasonable approach, considering the history of NN development and validation. Handcrafted successful networks (e.g., VGG [Simonyan and Zisserman, 2015], ResNet [He et al., 2016], Mobilenet [Howard et al., 2017], etc.) are composed by repetitions of a certain

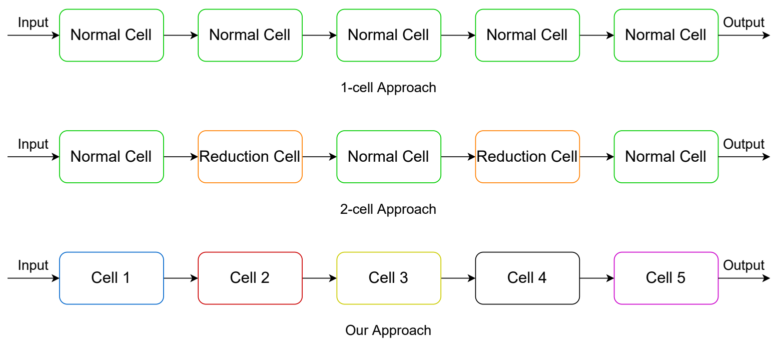


Figure 3.1: *Shallow2Deep* avoids architecture design limitations, common in other NAS algorihtms. Here $C = 5$.

peculiar element (e.g., convolutions, skip connections, inverted mobile bottleneck, etc.). Furthermore, repetitions lead to a reduction of the search space size, when an effective combination of the elementary cells must be automatically computed.

However, although understandable from a computational perspective, such an approach is not reasonable in terms of predictive capability. Relying on the same elementary structure at different depth levels of the network architecture may hinder the predictive performance of the resulting NN as a whole. In fact, assuming a particular elementary structure is good enough to let a network's *shallow*[1] layers perform valuable feature extraction, it is unlikely that the same structure is equally good to provide that network's *deep* layers with more sophisticated pattern matching capabilities. The different layers of a well-trained NN are expected to perform totally different feature-mining tasks. This is why we argue that the best elementary structure for shallow and deep layers of NN are not architecturally equal. *Shallow2Deep* builds on this, providing a means to look for a good cell structure specification for both shallow and deep layers.

More precisely, *Shallow2Deep* constructs NN classifiers with a fixed number $C$ of *different* cells. Cells can differ in terms of the topology they assume and the operations they apply. We define the difference between cells w.r.t. topology and operations as *structure variability*. *Shallow2Deep* promotes local *structure variability* in NN architectures, avoiding design limitations.

[1]By "shallow" (resp. "deep") layers of a NN we mean the *inner* layers close to the input (resp. output) neurons.

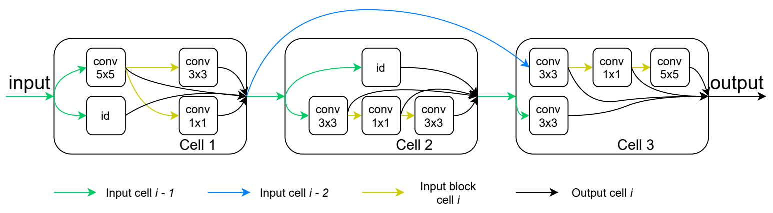


Figure 3.2: Blocks in *Shallow2Deep* cells can get input from previous cell (green), the cell before that (blue) or any other block in the cell (yellow). Cell output is obtained concatenating outputs of all blocks belonging to the cell (black). Here $C = 3$ and $B = 4$.

Figure 3.1 highlights the main difference among state-of-the-art NAS mechanisms and our approach: *Shallow2Deep* lets each cell vary independently from each other. However, similarly to other authors, we assume cells to be ordered from shallow side to the deep side. Accordingly, the $1^{st}$ cell is the closest one to the inputs, while the last one is the closest to the outputs.

In particular, *Shallow2Deep* lets each cell contain $B$ blocks—being $B$ a positive and finite integer. Each block represents a particular sort of NN layer. Following a convention introduced in [Ying et al., 2019], we denote by $\mathbb{O}$ the *finite* set of all possible sorts of blocks—which in turn depend on the particular task the target NN aims to solve. For instance, if the considered NN targets image recognition tasks, we let $\mathbb{O}$ contain simple convolutions, other than the identity block—e.g. some $n \times n$ convolutional layers (for $n = 1, 3, 5, \ldots$), plus the identity layer $f(x) = x$.

Each block of the $i^{th}$ cell can accept as input the output of $i - 1^{th}$ and $i - 2^{th}$ cells and of any other block in the same $i^{th}$ cell. However, loops and cycles among blocks connections are not allowed. In other words, the blocks topology must be a Directed Acyclic Graph (DAG). This is necessary to preserve the feed-forward architecture of the NN. Furthermore, each block can provide output to any amount of other blocks.

The whole output of a cell is attained by concatenating the outputs of all blocks belonging to that cell, as in [Liu et al., 2018]. Figure 3.2 provides an example of an admissible topology that can be created by blocks and cells following the aforementioned rules.

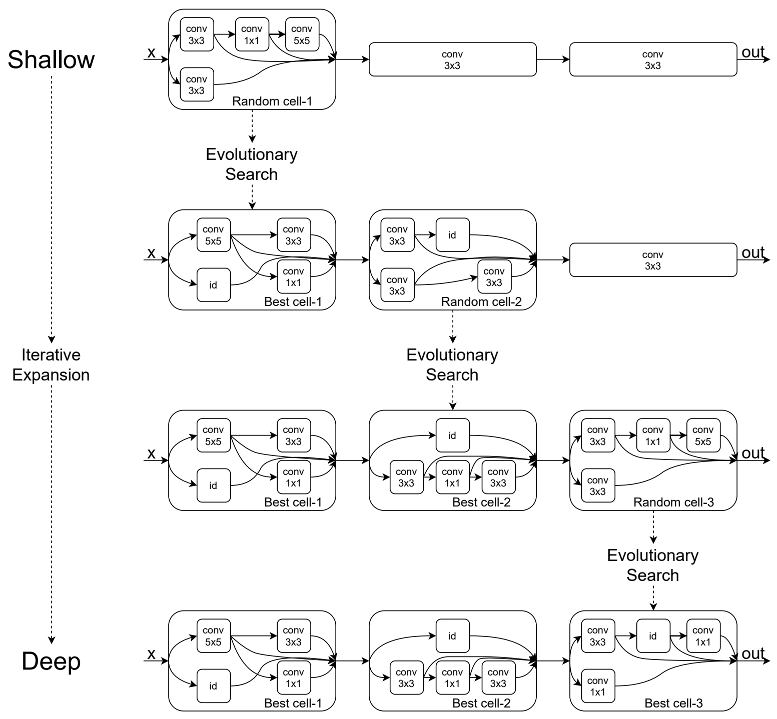


Figure 3.3: *Shallow2Deep* iteratively searches for the best structure of cells going from shallow to deep ones. Shallow bests are searched using simple superstructures to increase the feature expressiveness and reduce training time. Local bests are kept fixed while deeper best are searched, reducing the complexity.

### 3.1.2 Search Algorithm

Virtually all NAS algorithms proposed into the literature so far deal with reduced search spaces attained via strict architectural constraints. Conversely, our approach avoids the excessive simplification of the architectural design by allowing the internal structure of each cell to vary. A greedy search algorithm is then employed to automate the selection of the actual cells structures, in an iterative way. It relies on the successive search of locally-optimal cell structures proceeding from the shallower cells to the deeper ones.

As exemplified in Figure 3.3, *Shallow2Deep* consists of the iterative repetition of a local search algorithm aimed at selecting the (locally) best internal structure of the $i^{th}$ cell. The search algorithm is repeated for all $i = 1, \dots, C$, in such a way that the internal structure of the $i^{th}$ cell is only optimised *after* that $(i-1)^{th}$ one has already been optimised. In particular, we rely on an evolutionary algorithm to tackle local search. During local search, a population of NN is considered based on the structures that need to be analysed. The NN under examination are trained on a subset of the training set in order to find well behaving local structures—i.e. cell. To keep the whole process time-efficient, while optimising the $i^{th}$ cell, all the $j^{th}$ cells ($j \in \{1, \dots, i-1\}$) are left unaffected by the training process. Moreover, to maximize the knowledge extracted at the $i^{th}$ cell during its discovery process, all $k^{th}$ cells ($k \in \{i+1, \dots, C\}$) are built as bare as possible. Following literature, we consider bare cells to be composed of a single block applying a $3\times3$ convolution operation [Golovko et al., 2017, Simonyan and Zisserman, 2015]. In other words, *Shallow2Deep* greedly proceeds from the shallowest cell to the deepest one.

While further details concerning our design choices are provided in Section 3.2, some insights can be provided by the way a well-trained NN operates. The shallow layers of a NN aim to mine low level features. Complex features are extracted by deeper layers, reliably building on top of low level information. Therefore, *Shallow2Deep* searches for structures of deeper cells iteratively, building on the knowledge acquired at previous search steps.

### Cell Search

The *Shallow2Deep* algorithm relies on a local search of the best performing structure for each cell of the NN. The task can be accomplished through a variety of different search algorithms, from reinforcement learning to evolutionary algorithms [Zoph and Le, 2017, Yang et al., 2020]. In *Shallow2Deep* we exploit *evolutionary* (a.k.a. genetic) heuristic algorithms.

Evolutionary algorithms are a family of population-based metaheuristic optimization algorithms inspired by biological evolution. They commonly rely on a set of predefined stochastic mechanisms – namely, generation, mutation, selection, mating, fitness, etc – which let the algorithm randomly explore a vast search space in a smart way. Technically, these algorithms attempt to solve an optimisation problem by generating population of random solutions for the problem at hand, and by simulating evolution for a predefined amount of iterations—a.k.a. *generations*. Solutions are more or less likely, to survive among generations depending on their *fitness*—i.e. a measure of the quality of a particular solution w.r.t. the problem at hand. To prevent the search to step into local optima, evolutionary algorithms may exploit a number of strategies to introduce more randomness in the process, such as mutations—meaning that solutions may randomly mutate while stepping through generations.

We choose to rely on evolutionary algorithms because of their *(i)* flexibility, *(ii)* support to *space pruning* [Luo et al., 2020] – a feature that we plan to support in the future –, other than *(iii)* the many successful works on NAS leveraging on evolutionary approaches as well (cf. [Liu et al., 2019a, Liu et al., 2018, Ying et al., 2019]). In particular, our evolutionary algorithm is inspired to regularised evolution proposed in [Real et al., 2019]. However, we avoid regularisation through aging and introduce a randomised approach to explore untouched areas of the search space.

As any other evolutionary approach, our algorithm mimics biological evolution by letting a *population* of $N$ randomly-generated NN step through a number $\nu$ of generations. More in details, the number of generations (i.e., $\nu$) represents the maximum amount of iterations that the evolutionary algorithm should perform before returning the final solution. While transitioning between generations, NN may probabilistically *mutate*, other than being allowed to survive depending on their *fitness*. Accordingly, while the *mutation* mechanism lets the algorithm *randomly* explore different internal structures for the $i^{th}$ cell, the *fitness measure* lets the algorithm assess how good a particular internal structure of the $i^{th}$ cell actually is. The *Shallow2Deep* algorithm can then go on with its iteration and focus on the $(i+1)^{th}$ cell. Once reached the $\nu^{th}$ generation, the best fitting NN is used to determine the final interal structure to be chosen for the $i^{th}$ cell.

Accordingly, in the remainder of this section, we delve into the details of how mutation and fitness actually work in the particular case of *Shallow2Deep*.

**Algorithm stub.** We denote by $P_n$ the $n^{th}$ generation of the population. Similarly, we denote by $P_0$ the initial population, which is randomly generated. The population size is kept fixed to throughout the local search procedure, as it is commonly done for evolutionary algorithms. In other words, for all $i \in \{1, \ldots, \nu\}$, the population $P_n$ is such that $|P_n| = N$ and all the architectures of all networks in $P_n$ conform to the constraints described in Section 3.1.1.

Then, our evolutionary algorithm refines the population through 3 steps which are repeated at every generation. These steps are:

**train** — where all the NN in $P_n$ are trained on (a subset of) the data set;

**selection** — where the NN which are not among the top-$m$ fittest ones are removed from $P_n$;

**incubation** — where $P_n$ is enriched with new NN – attained via mutation – aimed at replacing the ones cutted off by the selection step.

*Shallow2Deep* assumes the available data to be partitioned into 3 parts, namely the *training*, *validation*, and *test* sets. While the train step only leverages on the training set, the selection step evaluates the fitness measure of each network against the validation set. The test can then be used to assess the performance of the final network architecture output by *Shallow2Deep*.

Concerning the incubation step, it is aimed at helping *Shallow2Deep* both from a performance-maximisation and search-space-exploration-speed perspective. More precisely, it aims at generating new NN following two criteria:

- $c$ networks are attained by mutating as many individuals in $P_n$ through the application of *mutation* transformation;
- $r = N - m - c$ networks are randomly generated from the search space.

Best behaving structures mutation helps performance maximisation, enhancing the focus on those evolutionary paths that have proven to be strong in recent history of the population. Partially randomising incubation helps search space exploration as it allows the evolution to look for points in the space farther apart from previously beaten evolutionary paths.

Once all the three steps have been completed for generation $n$, and a new population has been attained, $P_{n+1}$ and the evolutionary search can proceed with generation $n+1$. The process is repeated $\nu$ times, after which the best performing local structure is considered as found.

**Fitness measure.** Fitness is measured on the *validation* set using the most adequate performance measure for the task at hand. Accordingly, in case the to-be-defined network targets classification tasks, accuracy or F1-score[2] measures may be used. Conversely, in the case of regression tasks, MSE[3], MAE[4], or R2[5] measures may be exploited instead.

In the particular case of image recognition tasks, classification accuracy is an adequate choice. More complex performance metrics may consider also FLOPS [Tan and Le, 2019] and latency [Tan et al., 2019].

**Mutation.** The *mutation* transformation is applied to some NN – referred as the *parent* – in order to attain new architectures—called *children*. It only focuses on the internal structure of the $i^{th}$ cell of the parent network, possibly affecting some of its blocks. In particular, we rely on two possible mutations that can be applied to the blocks of a cell (graphically depicted in Figure 3.4):

**input mutation** — a block $B$ of the $i^{th}$ cell is selected at random, it is detached from its previous input, and the output of either another block $B'$ in the same cell or of the $j^{th}$ cell as whole, with $j \in \{i-1, i-2\}$, is used as the new input of $B$—provided that the new connection does not introduce a loop or a cycle;

**operation mutation** — a block $B$ of type $o \in \mathbb{O}$ is randomly selected from the $i^{th}$, and its type is changed to some other $o' \in \mathbb{O}$ such that $o \neq o'$.

### Greedy Assemble

*Shallow2Deep* requires several NN to be actually trained behind the scenes of its operation. This is true, in particular, for the evolutionary algorithm described above. In fact, while mostly focusing on one cell at a time, the algorithm must still train at least $N \cdot \nu$ networks – only differing for the content of the $i^{th}$ cell –, $C$ times.

[2]https://en.wikipedia.org/wiki/F-score
[3]https://en.wikipedia.org/wiki/Mean_squared_error
[4]https://en.wikipedia.org/wiki/Mean_absolute_error
[5]https://en.wikipedia.org/wiki/Coefficient_of_determination

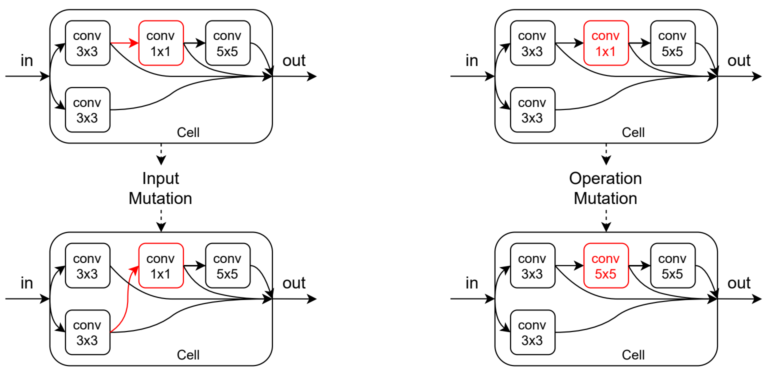


Figure 3.4: Mutation operations available in randomized evolution. When input operation is applied, previous input block is linked with cell output if it has remained pendent, avoiding block removal.

To keep the computational effort feasible, a number of strategies are in place. For istance, while performing the $i^{th}$ evolutionary search, *Shallow2Deep* leaves all cells of index $j$ s.t. $1 \leq j < i$ unaffected, and does not re-train them anymore, as they have already been explored and trained in previous iterations. Dually, the algorithm always assumes all cells of index $j$ s.t. $i < j \leq C$ to only contain a single block. In this way, the whole NN shallowness is preserved. In the particular case of image recognition tasks, that block may for instance consist of a 3×3 convolutional layer. Accordingly, during the $i^{th}$ evolutionary search, only cells whose index is at least equal to $i$ are actually trained over data, and all cells whose index is greather than $i$ have a very minimal structure.

In other words, once the $i^{th}$ local search is completed, the $m$ best performing structures for the $i^{th}$ cell are fixed, and never retrained anymore. As part of the subsequent iterations of *Shallow2Deep*, the network architecture is deepened to produce deeper and more complex NN.

## 3.2 Discussion

Global NN architectures are ideally composed by different local structures whose role depends on their position in the NN. Following this idea, unlike most common NAS frameworks, *Shallow2Deep* does not rely on the replication of the same cell. Rather, *Shallow2Deep* exploits a *progressive* search of the best cells at each possible depth level, from the shallowest to the deepest ones. We here discuss the rationale behind *Shallow2Deep* progressive search.

It is well understood how the complexity of the features extracted by some NN is proportional to the depth of the layer which recognises them [Yosinski et al., 2015, Nguyen et al., 2016]. In fact, while layers that are closer to the input are appointed to extract basic features – such as edges, corners, borders, etc., in image-recognition tasks –, deeper layers aim at recognising more complex features— such as combination of shapes, combination of textures, etc. Accordingly, shallow networks are better suited to tackle simple tasks [Golovko et al., 2017] where only simple features are involved. Conversely, the more complex a to-be-recognised feature is, the deeper a layer capable to recognise it must be. This happens because the recognition of a complex feature in a NN relies on the composition of more basic features extracted by shallow layers. Consequently, the lower is a feature complexity, the shallower can be the NN able to learn it. We call this phenomenon *depth-complexity proportionality* assumption.

There exists a tight link between features complexity and network depth that allows us to propose reasonable shortcuts for exhaustive architecture search methods. *Shallow2Deep* is designed on the assumption that simple features learnt by shallow networks perform reliably for deeper networks as well. Indeed, deeper NN achieve more flexible recognition capabilities than shallower ones [Nam and Han, 2016, Peng et al., 2020]. Moreover, deeper NN may attain higher generalisation capabilities [Rolnick and Tegmark, 2018], being capable of adapting to the features that shallow NN have learnt to recognise.

Accordingly, we argue that NN built from the sequential repetition of the same local structure cannot achieve the astounding results that characterise state-of-the-art NN. Conversely, we believe it is possible to search for reliable shallow architectures and expand them in successive iterations, as done by *Shallow2Deep*. The more simple concepts are reliably learnt by shallow networks, the easier it will be to learn complex notions from their combinations. We call this phenomenon *knowledge greediness* assumption.

The progressive global assemble of *Shallow2Deep* exploits both knowledge greediness and depth-complexity proportionality assumptions to boost the overall time complexity and performance.

In particular, depth-complexity proportionality justifies the deepening of the NN architecture in successive iterations, which in turns supports the trick exploited by *Shallow2Deep* to speed up the local search phase. Indeed, population training in the evolutionary local search is the most expensive and time consuming process. Training shallower networks requires less time to complete, as the parameters to optimize are much less.

Conversely, knowledge greediness justifies *Shallow2Deep*'s strategy of iteratively expanding the depth of the NN architecture under consideration. While this certainly raises NN training time, it also lets deeper architectures rely on previously trained cells. In particular, to boost the overall search, *Shallow2Deep* fixes the parameters of shallower cells, avoiding their re-training. This idea traces back to the well-established idea of re-using pre-trained feature extractors in object detection mechanisms [Ren et al., 2017, Li et al., 2018a]. Indeed, once a network is deepened and its deeper cell structure is determined, the predictive performance of the overall network does not degrade, even if shallow layers are kept fixed.

## 3.3 Experiments

In this section we first present the dataset used in our experiments (see Section 3.3.1). We then introduce the *Shallow2Deep* implementation and the best NN architecture obtained with it in Section 3.3.2. In Section 3.3.3 we then compare obtained architecture with state-of-the-art models that leverage on the same operation set $\mathbb{O}$. We also analyse if *Shallow2Deep* local structures could be reused through repetition in a NN model to obtain better performance/complexity ratio. We make publicly available our implementation of *Shallow2Deep*.[6]

### 3.3.1 Dataset

To demonstrate the validity of our approach we run *Shallow2Deep* on the MNIST fashion dataset [Xiao et al., 2017]. The dataset is a publicly available database containing images of fashion products and is commonly used for training and testing state-of-the-art ML systems. The fashion MNIST serves as a quick replacement

[6]`https://github.com/AndAgio/Shallow2Deep`

of the original MNIST dataset [LeCun et al., 2010], sharing the same image size, data format and structure of training and testing samples. The dataset contains a total of 70,000 grayscale images – each having size of 28x28 pixels – of fashion products from 10 different categories. The dataset was extracted scraping images from the articles of the Zalando website[7]. The classes are perfectly balanced each having 7,000 images. Finally, the training set consists of 60,000 samples, while the test set consists of 10,000 images.

### 3.3.2 *Shallow2Deep* Architecture

We define $\mathbb{O}$ to be the set of available operations that can be selected for each block of a cell. Similar to [Ying et al., 2019], in *Shallow2Deep* $\mathbb{O}$ contains simple convolutions and identity (*1×1 conv*, *3×3 conv*, *5×5 conv*, *identity*). Consider now *Shallow2Deep* search space $\mathbb{S}$—i.e. the space that contains obtainable cells through local search. The search space cardinality – i.e. the number of obtainable cells – is $|\mathbb{S}| = (B+1)! \cdot |\mathbb{O}|^B$. Let now $\mathbb{N}$ be the search space for the overall NN—i.e. the set of obtainable NN architectures. Remembering that *Shallow2Deep* does not rely on cell repetition, the amount of NN architectures available during the overall architecture search is $|\mathbb{N}| = |\mathbb{S}|^C = \left((B+1)! \cdot |\mathbb{O}|^B\right)^C$. For our experiments we set $B = 3$ and $C = 4$, obtaining $|\mathbb{S}| = 1.54 \cdot 10^3$ and $|\mathbb{N}| = 5.57 \cdot 10^{12}$. The amount of possible NN architectures is huge, but it does not reflect the computational complexity. Indeed, thanks to its increasing depth approach, *Shallow2Deep* is capable of searching a space of size $|\mathbb{S}|^C$, while having complexity that is only proportional to $C \cdot |\mathbb{S}|$

For each cell we search for the best structure using the randomised evolution algorithm proposed in Section 3.1.2. We fix the number of generations of the evolutionary algorithm to be $\nu = 5$ for each cell and the population size to be $|P| = 50$. During *incubation* we fixed the number of surviving best models to be $m = 10$, the number of models obtained through mutations to be $c = 20$ and the number of random models added to each generation to be $r = 20$. Each model is trained for 10 epochs using learning rate $\lambda = 0.01$.

To show the effectiveness of *Shallow2Deep* search, we study the behaviour of

[7]https://www.zalando.com

the NN population against the number of generations of the overall algorithm. In *Shallow2Deep*, the user can select the number of cells $C$ that compose the NN and $\nu$, the number of generations that the local search takes. *Shallow2Deep* iteratively searches each of the $C$ cells for $\nu$ generations. Therefore, the overall search of the NN architecture takes $C \cdot \nu$ generations to complete. We study the average performance – i.e. classification accuracy – of the population of NN for each of the $C \cdot \nu$ generations. We also study the accuracy of the best NN in the population for each of *Shallow2Deep* $C \cdot \nu$ generations.

Figure 3.5 shows the behaviour of average and best NN performance against *Shallow2Deep* generations. The classification accuracy increases with the number of generations considered, showing the success of *Shallow2Deep* search. Accuracy increments are limited since even $1^{\text{st}}$ generation NN reach reasonable performances. This is due to the mild complexity of the classification task over the MNIST fashion dataset. Biggest increments in the NN accuracy are found in generations where the cell index $i$ is increased—i.e. local search shifts to the next cell. This behaviour is expected as the increasing complexity – i.e. depth – of the NN extends its reasoning capabilities. It is also interesting to notice that this behaviour is more evident for smaller cell index $i$, while it becomes more attenuated for values of $i$ close to $C$. In our experiments, performance reaches stability for $i = C$—i.e. there exists a negligible difference between accuracy of NN with $i = C - 1$ and $i = C$. Stabilisation of accuracy can be considered a signal that the NN is reaching a complexity limit. Surpassing this limit would increase concepts complexity while not bringing any gain in performance, introducing possible overthinking issues [Kaya et al., 2019]. Therefore, *Shallow2Deep* represents a tool to automatically identify the NN complexity sweetspot over a certain task.

Figure 3.6 shows the architecture of the NN obtained running *Shallow2Deep* on the MNIST fashion dataset.

From its architecture, we point out that *Shallow2Deep* NN identifies sequential operations – i.e. blocks connected in a sequential manner inside a cell – at shallower stages of the NN—i.e. cells 1 and 2. Going deeper in the NN architecture – i.e. cells 3 and 4 –, *Shallow2Deep* building procedure identifies cells composed of parallel branches of convolutional operations. It is possible that sequential operations at shallow sections of NN help the model to learn simple concepts at the basis of their

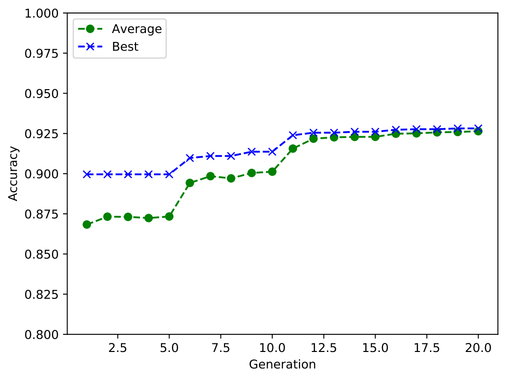


Figure 3.5: Performance – i.e. classification accuracy – of NN architectures considered by *Shallow2Deep* for each generation. We consider both the average performance and the accuracy of the best model in each generation.

reasoning — i.e. edges, corners, simple shapes, etc. Parallel operations approach may, instead, be useful for the NN learning process when complex concepts need to be extracted—i.e. combination of shapes, combination of textures, etc. Therefore, deeper investigation of this result may be interesting.

### 3.3.3 *Shallow2Deep vs.* State of the Art

We now compare the performances obtained by *Shallow2Deep* NN against state-of-the-art models that apply the same basic operations – i.e. convolutions and identity – like VGG [Simonyan and Zisserman, 2015] and ResNet [He et al., 2016]. In order to make the comparison fair, we retrain the *Shallow2Deep* NN, VGG, and ResNet on the MNIST fashion dataset from scratch. Training parameters are the same for every model considered—i.e. 60 epochs and learning rate $\lambda = 0.01$. Moreover, to study the effects of cell structure variability in NN architectures, we consider NN models built from the repetition of single cells found by *Shallow2Deep*

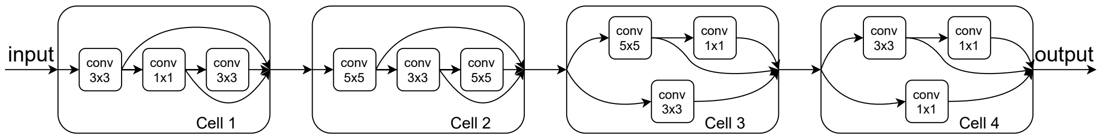


Figure 3.6: NN architecture discovered by *Shallow2Deep* algorithm.

| Model name | Accuracy ± std (%) | Parameters (M) |
|---|---|---|
| *Shallow2Deep* | $93.26 \pm 0.18$ | 0.251 |
| *Shallow2Deep*$_1$ | $92.87 \pm 0.17$ | 0.165 |
| *Shallow2Deep*$_2$ | $93.31 \pm 0.14$ | 0.491 |
| *Shallow2Deep*$_3$ | $92.73 \pm 0.14$ | 0.377 |
| *Shallow2Deep*$_4$ | $92.28 \pm 0.10$ | 0.118 |
| VGG | $93.66 \pm 0.18$ | 0.746 |
| ResNet | $92.77 \pm 0.09$ | 1.626 |

Table 3.1: Comparison between *Shallow2Deep* and state-of-the-art models. We consider also models built through repetition of single *Shallow2Deep* cells—e.g. *Shallow2Deep*$_1$ is the NN built from repetition of *Shallow2Deep* cell 1 in a sequential manner.

local search. In other words, we select *Shallow2Deep* cell $i$ – i.e. the cell discovered during $i^{th}$ local search step – and we build the NN model composed of 4 cells having the same structure of cell $i$. We name these NN architectures *Shallow2Deep*$_i$.

Table 3.1 shows the performance over the test set T– i.e., the average accuracy and its standard deviation over 20 training runs –, the footprint – i.e. number of weights of the NN (expressed in millions, denoted by M) – of *Shallow2Deep* and state-of-the-art NN. *Shallow2Deep* NN with its variants reach state-of-the-art performances over the MNIST fashion classification dataset. NN obtained using *Shallow2Deep* are the most efficient if we consider the accuracy/footprint trade-off—i.e. division between reached accuracy and number of parameters. More in details, *Shallow2Deep* NN reaches accuracy comparable with VGG (only 0.4% less), while requiring a third of the parameters. Performances obtained by the ResNet NN over the dataset under examination are possibly due to overthinking issues. ResNet model complexity – i.e. model footprint – is higher than necessary for the selected task, which brings it to learn too many or too complex concepts, decreasing overall performances.

We also analyse the effects of cell structure variability in NN architectures. Base *Shallow2Deep* NN version intrinsically express high level of cell structure variability, while its variants – e.g. *Shallow2Deep*$_i$ – do not. It is possible to notice that *Shallow2Deep* NN outperforms 3 of its *Shallow2Deep*$_i$ variants out of 4 in terms of absolute performances. Moreover, *Shallow2Deep* NN outperforms all of its *Shallow2Deep*$_i$ variants when the accuracy/footprint trade-off is considered. Therefore, we can safely state that cell structure variability allows NN models to reach higher performances while being complexity-constrained.

## Chapter Synopsis

In this chapter, we focus on NAS as a possible solution to the NN efficientisation task, proposing *Shallow2Deep*, a novel NAS approach that limits NN complexity and promotes local variability in their architectures. *Shallow2Deep* relies on successive searches of local optima and NN expansions – i.e. depth increment – to produce well performing NN models. We show that *Shallow2Deep* can effectively achieve NN complexity reduction, while reaching performances comparable to the state-of-the-art. The experimental analysis demonstrates how variability over local structures that compose NN is a desirable feature to obtain small and well performing models. This idea is in contrast with previously-proposed NN design approaches that neglect local structure variability, opening new possibilities for future research.

# Chapter 4

# Learning vs. Searching

This chapter contains contributions from [Agiollo and Omicini, 2022].

Despite its success, *Neural Architecture Search* (NAS) exhibits some drawbacks, as it relies on *searching* for the best architecture rather than *learning* architectural criteria for building the optimal NN. Moreover, NAS approaches are not flexible with respect to slight changes of the application scenario, require huge amount of resources to run, and are limited by their search space specifications. Inspired by these limitations, in this chapter we present GNN2GNN, a novel tool leveraging graph neural networks to generate NN architectures and learn their optimal design criteria.

GNN2GNN is a meta-learning framework exploiting Graph Neural Networks (GNNs) to learn generating efficient NN structures. GNNs are particular models proposed to tackle *graph*-processing tasks via convolution-equivalent operation over graphs [Wu et al., 2021b]. A NN structure can be seen as a Directed Acyclic Graph (DAG) where nodes represent layers – implementing common operations like convolution, pooling, etc. – and edges represent how the output of one layer is fed to the following one. In this context, we propose a three-way adversarial learning setup to allow GNN to learn the features of an efficient NN structure and generate novel architectures. More in details, a generator GNN is trained to produce plausible architectures, while a discriminator GNN is optimised to distinguish between generated and real architectures. Finally, a valuer GNN aims at optimising the performance of the generated architectures. During training, the

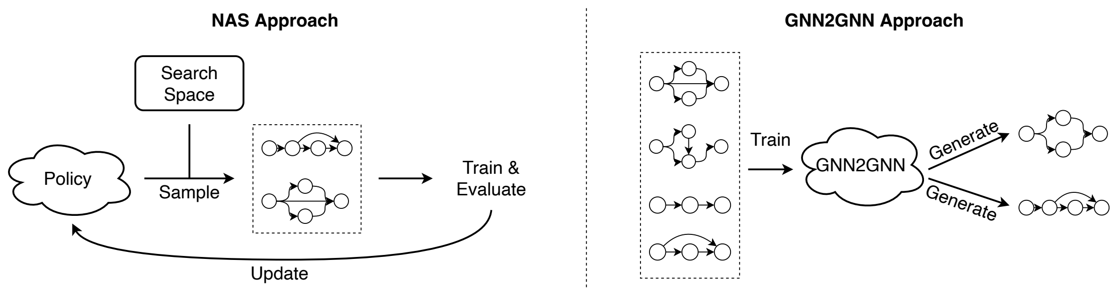


Figure 4.1: **Left**: NAS approaches rely on a recursive sampling, evaluation and optimisation procedure. A NAS policy is used to sample architectures from the search space. The sampled architectures are then trained and evaluated to optimise the NAS policy depending on their performance. Once a convergence criterion is met, NAS identifies the sub-optimal NN architecture. **Right**: The GNN2GNN approach rely on a single training procedure where GNN2GNN learns to propose effective NN architectures. The trained generator is able to produce multiple powerful NN architectures, rather than identifying solely the local sub-optimal NN architecture.

generator loss is defined as a mixture of the discriminator and valuer feedbacks, therefore aiming at enabling the learning of realistic – i.e., discriminator feedback – and powerful – i.e., valuer feedback – architectures.

Differently from NAS techniques, which aim at efficiently searching NN architectures over a set of available ones, GNN2GNN aims at intrinsically learning architectural criteria from a set of available architecture-performance pairs. While NAS consider to recursively propose, evaluate and optimise a set of NN structures (see Figure 4.1 left), we here consider learning to propose architectures from a set of NNs in a single step—i.e., training of the generator GNN (see Figure 4.1 right). Once trained, GNN2GNN is capable of proposing multiple efficient NN architectures at once, rather than focusing solely on the local optimum obtained from the deployed search algorithm. Therefore, GNN2GNN significantly shifts the paradigm of the approach to the problem of NN architecture design, from relying on *searching* architectures to *learning* design criteria.

## 4.1 Preliminaries on Graph Neural Networks

As the proposed framework relies on graph manipulation via GNNs, here we briefly introduce Graph Neural Networks, presenting their fundamental concepts. GNNs have been proposed as an extension of traditional NNs to enable processing of non-rigidly structured data such as graphs. GNNs are mathematical models operating upon directed graphs, whose vertices (respectively, arcs) are labelled with vectors (or matrices, or tensors) of real numbers – $\mathbf{x}_v \in \mathbb{R}^d$ for vertex $v$, and $\mathbf{a}_{v,w} \in \mathbb{R}^c$ for the arc between vertex $v$ and $w$ –, each one carrying further numeric information about the corresponding vertex (resp., arc). GNNs rely on *graph convolution*, which represents the generalisation of a 2-dimensional convolution over graph-structured data. Graph convolution is defined over a single vertex $v$ and its neighbourhood $N(v)$, and relies on three successive phases:

*propagation* — the information $\mathbf{x}_{v'}$ belonging to each vertex $v' \in N(v)$ is weighted by the information $\mathbf{a}_{v,v'}$ belonging to the arc among $v$ and $v'$, then propagated to vertex $v$;

*aggregation* — the information propagated from each vertex $v' \in N(v)$ to $v$ is aggregated via an aggregation function;

*transformation* — the aggregated information corresponding to vertex $v$ is transformed into a new embedding vector and assigned back to vertex $v$, as its new state $\mathbf{x}'_v$ .

The single convolution operation is applied in parallel to each vertex in $G$, updating the whole graph representation.

GNNs have proven to be successful in many tasks involving graph structured data. Most common applications concerns computational chemistry [Fung et al., 2021], social recommendations [Fan et al., 2019], computer vision [Wang et al., 2019b], and many others [Hamilton et al., 2017, Yu et al., 2018]. For a comprehensive review of GNNs and the underlying techniques we refer interested readers to [Wu et al., 2021b, Zhou et al., 2020].

## 4.2 GNN2GNN

In this section we present our framework, namely GNN2GNN. GNN2GNN leverages Graph Neural Networks to Generate Neural Networks. We first present briefly how NN architectures can be mapped into graph structures (Section 4.2.1). We then introduce the general pipeline for generating and processing NN architectures (Section 4.2.2), focusing specifically on its components.

### 4.2.1 Neural Networks as Graphs

NN architectures are uniquely defined by a set of layers $\mathcal{L}$, a set of operations $\mathcal{O}$ applied on layers, and a set $\mathcal{I}$ of interconnections between layers. Each layer $l_v \in \mathcal{L}$, with $v \in [0, |\mathcal{L}|]$, identifies a specific component of the NN architecture and is characterised by a specific operation $o_v \in \mathcal{O}$. Interconnections between layers, on the other hand, define how layers are linked to each other. An interconnection $i_{v,w} \in \mathcal{I}$ identifies that layers $l_v$ and $l_w$ are connected, and more specifically, it identifies that the output of the operation $o_v$ applied at layer $l_v$ is used as an input for the operation $o_w$ applied at layer $l_w$. It is important to notice that, thanks to the feedforwarding nature of standard NNs, there exists total ordering among the layers in $\mathcal{L}$ and interconnections can only exist between successive layers. Mathematically speaking, $\exists i_{v,w} \in \mathcal{I} \iff w > v$.

Following the above notations, NN architectures can be mapped easily into graph structures, specifically to DAGs. Layers in $\mathcal{L}$ are mapped into a set of vertices $\mathcal{V}$ characterised by a set of features $\mathcal{X}$ representing layers operations ($\mathcal{O}$), while interconnections ($\mathcal{I}$) are mapped into a set of directed edges $\mathcal{E}$. Vertices – i.e., layers –, along with their features – i.e., operations –, are defined as vectors $\mathbf{x}_v \in \mathbb{R}^d$, where $v$ enumerates the graph vertices, and $d$ represents operations cardinality. On the other hand, the set of edges – i.e., interconnections –, is denoted by the adjacency matrix $\mathbf{E} \in \mathbb{R}^{|\mathcal{V}| \times |\mathcal{V}|}$, where $\mathbf{e}_{v,w} = 1 \iff \exists i_{v,w}$. Therefore, a NN architecture can be mapped into a DAG defined by $\mathbf{X} \in \mathbb{R}^{|\mathcal{V}| \times d}$, where $row_k(\mathbf{X}) = \mathbf{x}_v$ – i.e. a matrix of vertices characterised by the operations they apply – and $\mathbf{E}$—i.e., the adjacency matrix defining how operations are connected.

### 4.2.2 Adversarially Generate Architectures

The proposed framework relies on a generative adversarial approach (GAN) [Goodfellow et al., 2014], applied over graph structured data leveraging Graph Neural Networks. The proposed framework is presented in Figure 4.2 and relies on three components:

- A *generator* GNN $G$ is in charge of proposing graph structures representing NN architectures.
- A *discriminator* model $D$ is responsible for distinguishing between NN structures proposed by $G$ and real architectures.
- A *valuer* network $V$ is responsible for predicting the architecture performance, therefore optimising the generation towards powerful structures.

GNN2GNN relies on such triplets of GNNs to allow $G$ to intrinsically learn optimal architectural criteria. Indeed, the discriminator and valuer models are used during training to optimise the generator status. More in details, the generator loss is defined as a mixture of the discriminator and valuer feedbacks:

$$\mathcal{L}_G = \lambda \cdot \underbrace{\mathcal{F}_D}_{D \text{ feedback}} + (1 - \lambda) \cdot \underbrace{\mathcal{F}_V}_{V \text{ feedback}} \tag{4.1}$$

Here, $\lambda$ represents the balancing factor between the two feedbacks. Leveraging such mixture loss, $G$ is capable of proposing realistic – i.e., $D$ feedback – and powerful – i.e., $V$ feedback – architectures. Finally, once trained, the GNN2GNN framework exploits solely the generator component to propose significant NN architectures.

#### Generator

Generating graph structures that satisfy specific properties is complex and represents an open research issue [You et al., 2018, Li et al., 2018b]. This task complexity is three-folded:

**Q1** *Generate realistic structures.* For a generated structure to result realistic, the generative framework should learn which nodes should be linked and which not.

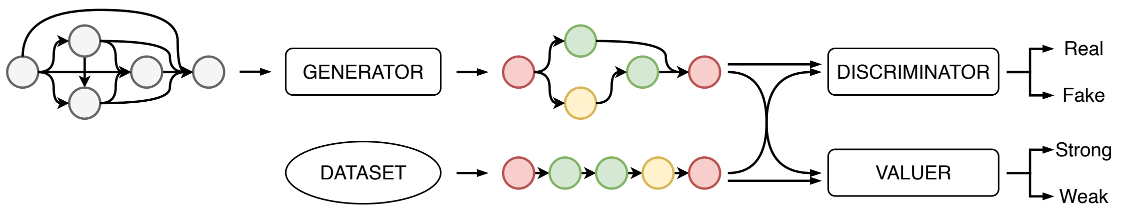


Figure 4.2: The GNN2GNN framework. The generator GNN produces NN architectures, starting from randomly initialised fully connected DAGs. The discriminator GNN aims at distinguishing artificial NN architectures from the real ones. The valuer network aims at predicting architectures performance, distinguishing between strong and weak structures or regressing their accuracy. Different colors of graph nodes represent different operations—embeddings. Red nodes identify input/output nodes, while green and yellow nodes may represent $3 \times 3$ convolution and max-pooling respectively. Gray nodes represent randomly initialised node embeddings.

**Q2** *Generate realistic nodes.* The generated graph should be characterised by nodes having realistic features.

**Q3** *Stopping criteria.* In the generating process, it is important to identify when the generated graph structure has reached its final structure, which is non-trivial.

To tackle the aforementioned problems and generate realistic NN architectures, we here propose a novel generative GNN. Indeed, GNNs are particularly suited for handling interconnections and node features, while they exhibit limitations on stopping criteria identification. However, given the nature of available NAS techniques and datasets, this GNNs limitation is not an issue. Indeed, available NAS techniques restrict their searching space, limiting the number of layers – vertices – that compose the NN architecture. Therefore, publicly available NAS datasets build on top of this rationale, fixing the number of NN layers.

Building on top of the same rationale, we here propose a generator model that receives as input a fully-connected DAG – i.e., where every node is connected with every other node – having $N$ nodes. $N$ represents an hyperparameter of the framework, and can be arbitrarily set depending on the features of the NAS dataset at hand, or, on the complexity of the architecture to generate. Fixing $N$ immediately satisfies property **Q3**, implicitly setting a stopping criteria for the graph generation process. It is also important to notice that value $N$ only provides an upper limit on the number of layers composing the generated architectures. Indeed, architectures having $n \leq N$ can be generated by the proposed approach, thanks to edge removal and node isolation. Finally, node features of the input graph are randomly initialised, mirroring the usual GAN approach.

The proposed generator framework relies on four successive steps, presented in Figure 4.3 along with an example of input graph and generated architecture, and explained in details below.

**Graph convolution.** The generator applies $\mu$ layers of graph convolution to the randomly generated fully-connected graph received in input. Graph convolution layers allow elaboration of the random information received, building the backbone of the generated NN architecture. Depending on the number $\mu$ of convolutional layers selected, we should expect more or less fine-grained embeddings as output of this step. However, given the fully-connected nature of the input graph, a small value of $\mu$ is enough to obtain a meaningful graph embedding.

**Edge scoring & sampling.** Once a proposal of fully-connected architecture is obtained from the graph convolution layers, the generator applies a learnable scoring function to each edge of the graph at hand. This procedure allows different scores to be assigned to each edge of the architecture, depending on their relevance. To score edges we first build edge features vectors, through the concatenation of adjacent vertices features. Mathematically speaking, the feature vector of edge connecting vertex $v$ to vertex $w$ is $\mathbf{e}_{v,w} = \mathbf{x}_w \parallel \mathbf{x}_w$, where $\parallel$ denotes concatenation. Once the edge feature is obtained, the edge relevance is scored using a standard densely-connected layer followed by normalisation in $[0, 1]$, obtaining $\mathbf{e}^{'}_{v,w}$ which represents the score given to the edge between $v$ and $w$. To avoid non-differentiability issues that may arise from scores thresholding, edges are then sampled depending on their scores using a gumbel softmax layer. This procedure ensures the survival of relevant – from the generator perspective – edges only, aiming at satisfying **Q1**. Edge scoring and sampling are here presented as a unique step, given their logical bond. However, it is also possible to conceive these two as separate steps, as done in Figure 4.3 to ease reader understanding of the

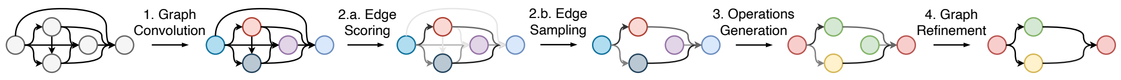


Figure 4.3: The generator receives in input randomly initialised – gray nodes – fully connected DAGs, and process them via graph convolution (1.). The new graph embedding, obtained from (1.) is used to score edges (2.a.). Light gray (dark gray) edges represent links having small (high) score. Edges are sampled (2.b.) and the scores are propagated to the next graph convolution step to obtain operations embedding (3.). Different colors of graph nodes represent different operations—embeddings. Red nodes identify input/output nodes, while green and yellow nodes may represent $3 \times 3$ convolution and max-pooling respectively. Finally, the graph is refined removing unsampled edges and nodes (4.).

framework.

**Layers operations generation.** The aim of this step is to assign one operation to each vertex – i.e., layer – of the graph corresponding to the NN architecture. To do so, the graph embedding obtained from the graph convolution step is combined with the sampled edge scores and used as input for a new layer of graph convolution. A softmax operation is then applied to the output of the convolutional layer to produce the one-hot encoding of the operations corresponding to each node. This specific step, aiming at identifying realistic nodes features – i.e., operations –, is meant to satisfy **Q2**. Here, the layer generation step focuses solely on the layer type – i.e., operation to deploy –, ignoring layers dimensioning issues. Indeed, we consider layers size to be automatically inferrable from the overall NN architecture, as stated in [Ying et al., 2019].

**Graph refinement.** Finally, the generator removes unsampled edges from the graph as well as isolated nodes, obtaining the final NN architecture. Possible cycles and pending nodes are also removed during this step, ensuring therefore to produce a DAG architecture. The graph-refinement operation is left as the last operation to avoid possible non-differentiability issues which may arise from removal of nodes or edges. However, this does not influence the generation of layer operations, since zero-scored edges do not propagate information in the previous convolution step.

### Discriminator

The discriminator model aims at distinguishing between synthetically generated architectures and architectures available in the dataset at hand. In the proposed framework we build the discriminator model stacking $\nu$ layers of graph convolution, followed by a single densely-connected classification layer. Graph convolutional layers extract graph-structured features from the input graph, while the classification layer outputs a binary prediction. The complexity of the discriminator model – i.e., the number of graph convolutions $\nu$ – depends on the complexity of the architectures under inspection. Available NAS datasets consider fairly small architectures, as they deal with identical block structures, therefore in our experiments we set $\nu = 2$.

### Valuer

The valuer model aims at identifying the performance of the architectures given as input. Structurally speaking, we build the valuer model similarly to the discriminator, stacking few layers of graph convolution, followed by a single densely-connected layer. The prediction of the valuer model over the structures generated by $G$ are also used for the generator optimisation, aiming to push $G$ toward the generation of more powerful NN architectures. Indeed, the generator model is trained minimising a combination between the standard GAN loss and the reward loss obtained from the valuer NN:

$$\mathcal{L}_G = \lambda \cdot \underbrace{\log(1 - D(G(z)))}_{\text{standard GAN loss}} + (1 - \lambda) \cdot \underbrace{\mathcal{L}_R(V(G(z)))}_{\text{reward loss}} \tag{4.2}$$

where $z$ represents the randomly initialised graph used as input for $G$, $\mathcal{L}_R$ represents the reward loss and $\lambda$ represents a balancing factor between the two loss terms. The definition of $\mathcal{L}_R$ depends on the role of the final densely-connected layer of $V$, which can be used either to regress the performance of the graph at hand or to binary classify graphs—strong vs. not-strong architecture. In the first approach, $\mathcal{L}_R$ is represented via mean-squared error loss between the predicted performance of generated architecture and the best performing architecture. In the second, the reward loss is represented via cross-entropy loss between predicted classification and strong architecture labels. Our experiments (see Section 4.3.4) show that the second approach is more consistent.

## 4.3 Experiments and Results

In this section we propose a set of experiments to show the effectiveness of GNN2GNN for generating strong NNs. Our source code is available at `https://github.com/AndAgio/GNN-2-GNN`.

### 4.3.1 Datasets

To test our framework performance we rely on the NAS101 [Ying et al., 2019] and NATS [Dong et al., 2021b] benchmark datasets. Both datasets contain a set of NN architectures along with their recorded performance over a specific image classification task. Here, NN architectures are built from the repetition of identical cells, which are the target of our GNN2GNN approach. NAS101 contains 423$k$ NN architectures trained multiple times over CIFAR-10 [Krizhevsky and Hinton, 2009]. On the other hand, NATS contains a set of 15$k$ NN topologies trained over three different datasets: *(i)* CIFAR-10; *(ii)* CIFAR-100; *(iii)* ImageNet-16-120. However, NATS represent operations over graph edges, while GNN2GNN and NAS101 represent operations over graph nodes, as introduced in Section 4.2.1. Therefore, we translate NATS architectures into Section 4.2.1 form and remove possible duplicates, thus obtaining a refined version of NATS consisting of 7$K$ unique architectures.

NAS101 and NATS datasets rely on similar search spaces used for the construction of NNs. Indeed, both consider a small set of operations, containing: *(i)* $3 \times 3$ convolution, *(ii)* $5 \times 5$ convolution, and *(iii)* *pooling*—NAS101 considers max-pooling, while NATS examines average-pooling. NAS101 contains NN cells with at most 7 nodes and 9 edges, while NATS examines cells with at most 8 nodes, without imposing any restriction on the number of edges.

### 4.3.2 Experimental Setup

To test GNN2GNN ability to produce novel architectures and generate strong cells, we remove part of the architectures from the training dataset. We eliminate some randomly picked cells from the dataset, as well as the best 10% of architectures—w.r.t. their classification accuracy. Under these settings, the generator can not extract information from the strongest models during training, rendering the generation task more complex. Therefore, a generator capable of producing the best 10% of architectures is to be considered a strong model. $N = 10$ was selected since in NAS101 there exists quite a significant performance delta between the top-10% architectures and the rest.

Each GNN2GNN instance is trained for 20 epochs over the training set using standard Stochastic Gradient Descent and setting the learning rate to 0.001 and the batch size to 32. Moreover, during the first half of the training procedure we set $\lambda = 1$. This is done to allow $V$ properly learning to distinguish between strong and weak architectures, before leveraging it to optimise $G$ with backpropagation. Indeed, backpropagating information from a partially trained $V$ to the generator $G$ may increase the noise of its training, slowing down or hindering its optimisation. Therefore, in the first 10 epochs the generator model is optimised only through the discriminator $D$. After this setup period $\lambda$ is set back to its desired value, enabling the interaction between $G$ and $V$ as described by Equation (4.2).

### 4.3.3 Evaluation Metrics

Throughout our experiments, we consider only models which always output valid NN architectures, since they output DAGs thanks to some refinement step. Therefore, the metrics that we define refer solely to the quality of the generated architectures. Moreover, since our framework is not directly comparable with NAS approaches, we avoid considering common NAS metrics—e.g., convergence time, etc. *Novelty* measures the percentage of generated architectures not belonging to the training set used. The *Top-N* metrics measure the percentage of generated architectures that belong to the best $N$% of architectures in terms of classification accuracy. $Acc_n$ measures the ratio between the number of generated architectures that reach an accuracy greater than $n$, and the number of generated models that belong to the dataset. Finally, $|Acc|$ measures the average accuracy reached by generated architectures.

### 4.3.4 Ablation Study

To identify the best hyper-parameters setup for GNN2GNN, we propose a thorough ablation study. The ablation study is performed over the NAS101 dataset, given its higher degree of expressiveness w.r.t. NATS.

**Hyper-parameters** We consider the influence of the parametric values that may alter the generation of NN architectures. We take into account the balancing factor $\lambda$ used during training, the temperature $\tau$ of the gumbel softmax layer used to perform edge sampling, and the number of graph convolution layers used by the generator $\mu$. Table 4.1 shows the results of the ablation study on such parameters. It is possible to notice that the model is highly affected by the balancing factor $\lambda$, which injects performance-critical information into the generator. Indeed, leveraging smaller $\lambda$ increases the performance of the proposed architectures, as the generator focuses more on the information received by $V$ through backpropagation. Smaller $\lambda$ values also improve GNN2GNN ability to predict more complex models. Architectures generated using $\lambda = 0.1$ have on average twice the number of parameters of their $\lambda = 1$ counterparts. This phenomenon is encouraging, as it shows that GNN2GNN is capable of mapping the whole space, thanks to $V$. However, smaller $\lambda$ increases the risk of mode collapse issues, as highlighted by the slight drop in novelty obtained with $\lambda = 0.01$. Finally, $\mu$ and $\tau$ do not seem to heavily influence the GNN2GNN performance.

**Valuer mode** The mechanism used by the valuer network $V$ to identify strong and weak architectures may cause variation in the generation performance of GNN2GNN. We distinguish between a classification-based valuer C and a regression-based valuer R. The former identifies strong architectures as the cells belonging to the best half of the dataset. On the other hand, in the regression-based setup, $V$ aims at predicting precisely the classification accuracy of a cell from its structure. We pick the three best models in Table 4.1, retrain them using a regression-based

Table 4.1: Ablation study over hyperparameters of $G$. Bold values highlight the best setup for each metric.

| Parameters | | | Novelty | Top-5 | Top-10 | Top-50 | $Acc_{90}$ | $\lvert Acc \rvert$ |
|---|---|---|---|---|---|---|---|---|
| $\mu$ | $\tau$ | $\lambda$ | | | | | | |
| 1 | 0.01 | 1 | 50.13% | 10.60% | 13.70% | 27.20% | 45.58% | 88.55% |
| | | 0.5 | 71.23% | 34.66% | 40.20% | 52.98% | 75.18% | 90.38% |
| | | 0.1 | 82.32% | 46.30% | 50.50% | 57.00% | 80.50% | 91.53% |
| | | 0.01 | 81.63% | 45.10% | 49.40% | 58.10% | 80.14% | 91.44% |
| | 0.1 | 1 | 51.79% | 12.10% | 15.40% | 25.20% | 40.23% | 88.10% |
| | | 0.5 | 67.81% | 19.01% | 22.62% | 39.20% | 59.23% | 89.48% |
| | | 0.1 | 82.52% | 45.19% | 50.53% | 58.91% | 80.60% | 91.48% |
| | | 0.01 | 82.47% | **46.50%** | 52.14% | **57.30%** | 79.32% | 91.35% |
| 2 | 0.01 | 1 | 51.66% | 8.57% | 11.40% | 26.83% | 41.63% | 88.53% |
| | | 0.5 | 73.84% | 40.41% | 45.28% | 54.61% | 75.01% | 90.89% |
| | | 0.1 | 82.06% | 46.09% | 51.03% | 57.82% | 81.55% | 91.54% |
| | | 0.01 | 82.23% | 45.66% | 50.20% | 57.19% | 79.69% | 91.42% |
| | 0.1 | 1 | 53.57% | 10.08% | 12.63% | 25.21% | 42.15% | 88.49% |
| | | 0.5 | 68.76% | 25.59% | 30.84% | 45.54% | 69.94% | 90.45% |
| | | 0.1 | **82.60%** | 45.91% | **52.21%** | 57.37% | **81.79%** | **92.04%** |
| | | 0.01 | 81.51% | 45.90% | 51.10% | 59.50% | 79.98% | 91.27% |

$V$, and compare them against their classification-based counterparts.

Table 4.2 shows the results of the ablation study, highlighting the superiority of the classification-based setup. Indeed, regressing exactly NN performance from its architecture is complex, mostly since few small architectural modifications may lead to relevant performance changes. Such variability is complex to handle in a regression setup and hinders $V$ ability to predict correctly cells strength.

Table 4.2: Ablation study over evaluation mode adopted by $V$.

| Parameters | | | $V_{mode}$ | Novelty | Top-5 | Top-10 | Top-50 | $Acc_{90}$ | $\lvert Acc \rvert$ |
|---|---|---|---|---|---|---|---|---|---|
| $\mu$ | $\tau$ | $\lambda$ | | | | | | | |
| 2 | 0.1 | 0.1 | C | **82.60%** | **45.91%** | **52.21%** | **57.37%** | **81.79%** | **92.04%** |
| | | | R | 72.10% | 26.59% | 32.92% | 50.10% | 67.11% | 89.93% |
| 1 | 0.1 | 0.01 | C | **82.47%** | **46.50%** | **52.14%** | **57.30%** | **79.32%** | **91.35%** |
| | | | R | 71.06% | 25.90% | 34.13% | 51.07% | 65.79% | 90.07% |
| 2 | 0.1 | 0.01 | C | **81.51%** | **45.90%** | **51.10%** | **59.50%** | **79.98%** | **91.27%** |
| | | | R | 70.33% | 27.04% | 33.54% | 50.97% | 66.43% | 90.01% |

In the remainder of the experiments, we build the GNN2GNN model employing a classification-based $V$ and the best hyperparameters values—i.e. $\mu = 2$, $\tau = 0.1$, $\lambda = 0.1$, as highlighted in Table 4.1. Indeed, these values represent a good starting point for deploying GNN2GNN over multiple scenarios, given NAS101 generality.

### 4.3.5 Performance Comparison

To show the effectiveness of the proposed approach, we compare GNN2GNN against other generative mechanisms. We first consider generating random NN architectures using the Erdös–Rényi model [Erdös et al., 1960]. We then evaluate the strength of our approach against two GAN-based frameworks, relying on different generation strategies:

**MOLGAN-like** The model generates nodes and edges independently and simultaneously, recalling the approach by [De Cao and Kipf, 2018]. Two matrices representing node types and connections between them are generated from a random input vector and sampled using gumbel softmax.

**RNN** The model generates architectures starting from a single input node and appending new vertices – with corresponding edges – until a stopping criteria is met. This approach resembles the one by [Zhang et al., 2019a] and leverages Recurrent NNs to deal with graph construction via recursive node appending.

To make the comparison fair, both the MOLGAN-like and the RNN model are built using the three-way NNs adversarial approach that characterises GNN2GNN. Therefore, the three approaches differ solely on the generation criteria embodied by the generator model $G$.

Table 4.3 shows the performance of the different models. GNN2GNN vastly outperforms the counterparts, as it produces more accurate predictions for strong NN architectures. Moreover, more than 80% of the predictions performed by our model are NNs characterised by an accuracy greater than 90%, while the best counterpart model – i.e., MOLGAN – fails to reach even 60%. This proves GNN2GNN's generation consistency. Indeed, even the simple random generation approach can sporadically generate powerful architectures, as shown also in [Xie et al., 2019a]. However, it suffers in terms of consistency, as it is uncommon to obtain articulated architectures starting from a random empty graph.

Table 4.3: Performance comparison between GNN2GNN and other GAN based approaches to generate NN architectures.

| Dataset | Model | Novelty | Top-5 | Top-10 | Top-50 | $Acc_{90}$ | $\|Acc\|$ |
|---|---|---|---|---|---|---|---|
| NAS101 | GNN2GNN | 82.60% | **45.91%** | **52.21%** | **57.37%** | **81.79%** | **92.04%** |
| | MOLGAN | 65.41% | 22.63% | 27.29% | 45.20% | 59.71% | 89.39% |
| | RNN | **96.34%** | 1.69% | 2.32% | 4.81% | 53.04% | 89.32% |
| | Random | 51.81% | 11.17% | 13.74% | 28.43% | 43.18% | 88.54% |

### 4.3.6 Resistance to Dataset Quality Degradation

To study the flexibility of our approach against poorly-constructed datasets, we analyse GNN2GNN performance when a high number of strong models are removed from the training dataset. More in details, we first remove the best $N\%$ of models from the NAS101 training set, varying $N$ between 10 and 90, then retrain GNN2GNN. Table 4.4 shows these tests results. The performance loss between different setups is minimal, highlighting GNN2GNN strength against dataset quality degradation. Indeed, even when almost all best models are removed from the training set, GNN2GNN produces strong predictions, showing just a 3% loss in the Top-5 metric and a 0.82% decrease of the average accuracy reached by generated models.

Table 4.4: Performance comparison when the top $N\%$ of best architectures is removed from the training dataset.

| Dataset | $N$ | Novelty | Top-5 | Top-10 | Top-50 | $Acc_{90}$ | $\|Acc\|$ |
|---|---|---|---|---|---|---|---|
| NAS101 | 10% | 82.60% | 45.91% | 52.21% | 57.37% | 81.79% | 92.04% |
| | 20% | 83.01% | 45.54% | 52.32% | 58.48% | 82.05% | 91.98% |
| | 30% | 83.67% | 46.10% | 51.07% | 57.04% | 80.14% | 91.68% |
| | 40% | 84.89% | 45.80% | 51.03% | 58.71% | 81.03% | 91.55% |
| | 50% | 85.00% | 44.01% | 48.81% | 56.30% | 79.24% | 91.33% |
| | 60% | 84.58% | 44.40% | 49.12% | 57.72% | 80.30% | 91.30% |
| | 70% | 84.20% | 44.66% | 49.38% | 56.71% | 79.52% | 91.37% |
| | 80% | 84.33% | 43.90% | 48.27% | 55.51% | 78.16% | 91.27% |
| | 90% | 85.71% | 42.77% | 46.70% | 55.72% | 78.24% | 91.22% |

### 4.3.7 Generalisation Between Datasets

We now consider the generalisation ability of our framework. We start by training GNN2GNN over NATS and showing its performance. As Table 4.5 shows, the performance obtained over NATS are poor, probably due to the small size of NATS—i.e., only $7K$ NN architectures. We then apply the generator model trained over NAS101 to NATS, analysing its prediction performance. Table 4.5 shows the results of our generalisation study. While still not being satisfactory, we notice that performance strongly increase when GNN2GNN is transferred from NAS101 to NATS. This is encouraging, especially if we consider the strong difference between NAS101 and NATS. Indeed, only 576 NATS architectures are available also in NAS101, and their performance vary on average by 16.183% between the two datasets.

Table 4.5: Performance of GNN2GNN when generalising between different datasets. Subscript refers to NATS split. $C10$ and $C100$ stand for CIFAR10 and CIFAR100, while $I$ stands for ImageNet.

| Dataset | | Novelty | Top-5 | Top-10 | Top-50 | \|Acc\| |
|---|---|---|---|---|---|---|
| Train | Test | | | | | |
| NATS$_{C10}$ | NATS$_{C10}$ | **76.73%** | 1.65% | 3.39% | 15.61% | 68.91% |
| NAS101 | NATS$_{C10}$ | 73.67% | **2.64%** | **5.08%** | **16.55%** | **70.25%** |
| NATS$_{C100}$ | NATS$_{C100}$ | 71.71% | 1.30% | 3.28% | 13.10% | 33.03% |
| NAS101 | NATS$_{C100}$ | **72.03%** | **2.64%** | **4.82%** | **16.90%** | **35.40%** |
| NATS$_{I}$ | NATS$_{I}$ | 81.03% | 0.91% | 2.17% | 8.42% | 16.75% |
| NAS101 | NATS$_{I}$ | **82.30%** | **1.93%** | **3.64%** | **11.49%** | **18.84%** |

## 4.3.8 Preliminary Comparison Against NAS

GNN2GNN does not represent a traditional NAS technique, as it does not rely on search space exploration and focuses solely on the architecture generation procedure. However, we can compare GNN2GNN against state-of-the-art NAS in terms of the performance obtained by the generated architectures over NAS101. Results shown in Table 4.6 are extracted from [Yu et al., 2020] and consider 1000 GNN2GNN generation samples. The average accuracy of GNN2GNN generation is comparable with other NAS approaches. Meanwhile, results show that GNN2GNN vastly outperforms NAS techniques in terms of best accuracy. Indeed, the best architecture generated by GNN2GNN achieves 94.32%, while NAO tops up at 93.33%. Moreover, the architecture generated by GNN2GNN achieves a lower rank value, meaning that they are closer to the optimal architecture. Indeed, the best achievable accuracy in NAS101 is 95.06%, which represents an increase of only 0.72% compared to what GNN2GNN achieves.

Table 4.6: GNN2GNN performance against state-of-the-art NAS approaches over NAS101. Subscript refers to the percentage of samples removed from NAS101.

| Model | \|Acc\| | Best Acc | Best Rank |
|---|---|---|---|
| DARTS [Liu et al., 2019a] | 92.21% | 93.02% | 57079 |
| NAO [Luo et al., 2018] | **92.59%** | 93.33% | 19552 |
| ENAS [Pham et al., 2018] | 91.83% | 92.54% | 96939 |
| $\text{GNN2GNN}_{90}$ | 92.04% | **94.32%** | **5372** |
| $\text{GNN2GNN}_{60}$ | 91.68% | 94.18% | 7843 |
| $\text{GNN2GNN}_{30}$ | 91.30% | 94.01% | 9371 |
| $\text{GNN2GNN}_{10}$ | 91.22% | 93.69% | 12570 |

## 4.4 Discussion

GNN2GNN relies on an architecture-performance pairs dataset to remove part of the complexity burden of extracting architectures performance. This might represent a possible drawback, as it requires the training of a set of hand-crafted NNs. However, results of Section 4.3.6 show how GNN2GNN learns to generate effective NNs even when most – i.e., 90% – of the arch-performance pairs are not available. Moreover, GNN2GNN does not impose any requirement on the dataset quality, as it can generate powerful architectures even when trained on the worst part of the dataset—i.e., worst 10%. Finally, Section 4.3.7 hints how GNN2GNN can translate the generation process to a new setup, without requiring the extraction of a new dataset.

## Chapter Synopsis

In this chapter, we focus on overcoming the *learning vs. searching* issue characterizing NAS approaches. To this end, we present a novel GNN-based three-way adversarial framework for learning to generate strong NN architectures, without relying on search space exploration. The experiments completed over two state-of-the-art datasets highlight the strength of our approach. We show GNN2GNN ability to predict optimal NN architectures and its superiority against other available generation approaches. Moreover, given its flexibility against dataset quality degradation, the proposed framework represents a step forward towards learning architectural criteria for NNs design. Indeed, the GNN2GNN generator is capable of predicting unseen strong architectures even when dealing with unsound dataset. Finally, some experiments on knowledge transferability suggest the generalisability of our approach. While aiming to overcome NAS limitations – via removal of search algorithms – GNN2GNN can also be integrated into a NAS approach as a proposal technique. Here, the adversarial framework characterising GNN2GNN would require online training, similarly to other NAS approaches.

# Chapter 5

# Coresets as Data Efficient Learning

This chapter contains results of the work done during my visiting at Purdue University.

Data is expensive to gather and annotate, influencing the applicability of ML and DL approaches in many scenarios. Moreover, we currently lack a comprehensive understanding of how NNs learn from complex datasets and what constitutes a *good* or *clean* learning example. In this context, research efforts have focused on *data efficiency* – aiming at learning effective models with fewer data – and *sample importance*—measuring the impact of each data sample on NN optimization. *Coresets construction* approaches propose to identify relevant samples of the dataset – either statically before the training starts or dynamically as the training proceeds – and optimize the NN model using solely those samples [Feldman, 2020]. Alongside, several works analyse the data samples complexity characteristic, relying on different properties of the NN training behaviour, such as forgetting [Toneva et al., 2019] and memorization [Feldman and Zhang, 2020].

Recent empirical findings seem to suggest that there exists a strong relationship between sample complexity – also referred to as typicality [Toneva et al., 2019] or cleanliness [Garg and Roy, 2023] – and their relevance for data efficiency. Inspired by these findings, we propose leveraging memorization scores – as originally defined in [Feldman and Zhang, 2020] – to identify coresets effectively. The experimental results against several state-of-the-art approaches highlight the superiority of our approach for selecting small coresets. However, while memorization scores allow us to overcome state-of-the-art approaches for coreset construction, they require a vast amount of computational resources to be extracted on a given learning task. For example, memorization requires training 4000 NN models to compute the memorization scores of samples on the CIFAR100 dataset [Krizhevsky and Hinton, 2009]. Therefore, approaches relying on memorization scores do not scale over different datasets or learning setups. Armed with this insight, we conduct an empirical study on a set of NN properties that may be useful for approximating memorization scores, while requiring fewer computational resources. Specifically, we consider both well-known properties and novel metrics, relating to various characteristics of the learning task. We refer to such metrics as *proxies* and empirically show their correlation with actual memorization values, along with their efficiency improvements and limitations. Finally, we test if – and to what extent – the correlation between memorization and its proxies propagate to the field of coreset construction, thoroughly testing the performance of our coreset construction approaches – based on memorization proxies – over two datasets and against seven state-of-the-art solutions.

## 5.1 Memorization for Coresets Construction

### 5.1.1 Coreset Construction: Problem Definition

Here, we recall briefly the definition of coreset construction given in Section 2.2.5. Consider a learning task in which the given large training set is defined as $\mathcal{T} = \{(\mathbf{x}_i, y_i)\}_{i=1}^{|\mathcal{T}|}$, where $\mathbf{x}_i \in \mathcal{X}$ is the input data sample, $y_i \in \mathcal{Y}$ is the ground-truth label of sample $\mathbf{x}_i$, and $\mathcal{X}$ and $\mathcal{Y}$ denote the input and output spaces, respectively. $\mathcal{V} = \{(\mathbf{x}_i, y_i)\}_{i=1}^{|\mathcal{V}|}$ defines a held-out validation set of samples used to test the performance achieved in the learning task. In this context, given a neural network model $h$ with parameters $\theta$ and a loss function $\mathcal{L}$, we can define the loss on a set $S$ of instances as $\mathcal{L}(\theta, \mathcal{S}) = \sum_{i \in \mathcal{S}} \mathcal{L}(\theta, \mathbf{x}_i, y_i)$. Similarly, $\mathcal{L}_{\mathcal{T}}(\theta, \mathcal{T})$ denotes the training loss over the full training set $\mathcal{T}$, and $\mathcal{L}_{\mathcal{V}}(\theta, \mathcal{V})$ the corresponding validation loss. In this setup, the coreset construction task aims at identifying the most informative subset $\mathcal{S} \subset \mathcal{T}$ such that $|\mathcal{S}| < |\mathcal{T}|$ – preferably $|\mathcal{S}| << |\mathcal{T}|$ –, so that the model

$h(\theta_{\mathcal{S}})$ trained on $\mathcal{S}$ achieves similar generalization performance to the model $h(\theta_{\mathcal{T}})$ trained on the whole training set $\mathcal{T}$. Mathematically, we can formulate the coreset construction problem by searching for the coreset $\mathcal{S}$ that allows to minimize the validation loss of the model optimized on the same subset of training samples $\mathcal{S}$:

$$\underset{\mathcal{S}\subset\mathcal{T}}{\operatorname{argmin}} \left\{ \mathcal{L}\left( \underset{\theta}{\operatorname{argmin}} \left\{ \mathcal{L}(\theta, \mathcal{S}) \right\}, \mathcal{V} \right) \right\}. \tag{5.1}$$

Throughout this chapter, we refer to the coreset size by the percentage of training data samples selected to construct the coreset, namely $\mathcal{P} = \dfrac{|\mathcal{S}|}{|\mathcal{T}|}$.

### 5.1.2 Memorization Definition

[Feldman and Zhang, 2020] define the memorization metric to capture and quantify how much a learning algorithm – comprised of a model, its training procedure and a training set of data samples – memorizes each sample rather than learning to generalize from its features. The memorization metric is formally defined as:

$$m(\mathbf{x}_i) = P\left[h^{\mathcal{A}}_{\mathcal{T}}(\mathbf{x}_i) = y_i\right] - P\left[h^{\mathcal{A}}_{\mathcal{T}^{\setminus i}}(\mathbf{x}_i) = y_i\right], \tag{5.2}$$

where $\mathcal{A}$ represents a training algorithm operating on the training dataset $\mathcal{T}$ to optimize the parameters $\theta$ of model $h$, while $\mathcal{T}^{\setminus i}$ refers to the dataset $\mathcal{T}$ with $(\mathbf{x}_i, y_i)$ removed. The intuition behind memorization formulation is that an algorithm memorizes the label $y_i$ if its prediction at $\mathbf{x}_i$ based on the rest of the dataset changes significantly once $(\mathbf{x}_i, y_i)$ is added to the dataset.

### 5.1.3 Experiments on Memorization for Coresets

Several works show how relying on clean samples for coreset construction achieves a relevant level of performance over small values of $\mathcal{P}$ [Garg and Roy, 2023, Toneva et al., 2019]. Concurrently, the empirical results in [Feldman and Zhang, 2020] show how memorization scores are useful for identifying and separating clean – i.e., low memorization – and complex – i.e., high memorization – samples. Accordingly, we here propose to leverage memorization scores of training samples for coreset construction procedures. We use memorization estimate $m_i$ as the selection metric,

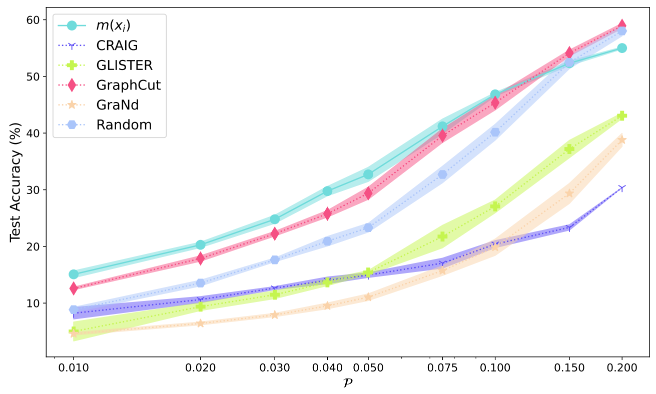


Figure 5.1: Coreset construction based on memorization scores against baselines on the CIFAR100 dataset.

choosing samples with the lowest values to make up our coreset. Hence, we first select the points with the lowest $m_i$ from pretrained networks and then train a new randomly initialized network on these points using the standard cross entropy loss.

We compare memorization coreset construction against 7 baseline approaches: *(i)* random uniform sampling, *(ii)* Glister [Killamsetty et al., 2021b], *(iii)* CRAIG [Mirzasoleiman et al., 2020], *(iv)* GraphCut [Iyer et al., 2021], *(v)* GraNd [Paul et al., 2021], *(vi)* Forgetting [Toneva et al., 2019], *(vii)* Loss curvature [Garg and Roy, 2023]. For methods that require training before coreset selection, we optimize the model for 100 epochs before selecting the coreset. The implementation of the baselines follows the one available in the DeepCore [Guo et al., 2022] library [1]. For each baseline and for the coreset constructed using memorization we let $\mathcal{P}$ vary over the range $[0.01, 0.2]$. We select the CIFAR100 dataset [Krizhevsky and Hinton, 2009] as the training target, since it allows relying on the [Feldman and Zhang, 2020] memorization scores readily available[2]. Figure 5.1 shows the results of our analysis.

Coreset construction relying on memorization scores achieves great performance improvements over all selected baselines through most values of $\mathcal{P}$. More in

[1]https://github.com/PatrickZH/DeepCore
[2]https://pluskid.github.io/influence-memorization/

detail, the coresets obtained via $m_i$ achieves almost a 5% accuracy improvement over the range $\mathcal{P} \in [0.01, 0.1]$. Meanwhile, for $\mathcal{P} = 0.2$ GraphCut represents the best approach, overcoming the sample selection based on memorization. Interestingly, the proposed sampling approach outperforms several popular approaches such as CRAIG, GLISTER and Forgetting over the whole spectrum of $\mathcal{P}$. These results highlight how sampling based on memorization represents the best approach for constructing small coresets, corroborating the *clean samples* hypothesis. Armed with this strong insight, we here investigate if – and to what extent – it is possible to empirically approximate memorization scores of a dataset using some proxy metrics, and how such proxy metrics behave for coreset construction.

## 5.2 Memorization Proxies

Memorization scores represent a relevant property of the learning task at hand, allowing to identify atypical, highly ambiguous or mislabeled data samples – memorized through training to optimize the loss – as well as clean examples—generalized and not memorized. However, memorization – as defined in its original formulation – requires a large amount of resources and time to be extracted. Estimating memorization values with a standard deviation of $\sigma$ requires running the training algorithm on the order of $1/\sigma^2$ times for every data instance. [Feldman and Zhang, 2020] reduce the memorization extraction complexity by leveraging an estimator that looks at the expected memorization of the label of $\mathbf{x}_i$ on a random subset of $\mathcal{T}$. However, the memorization score computation on the CIFAR100 dataset still requires training of 4000 different neural network models. This large amount of required optimization procedures is prohibitive and does not scale whenever it is required to compute memorization scores over a new model, dataset, or learning algorithm. Therefore, it is necessary to identify possible approximations of the memorization scores that are less resource-demanding. To this end, we here consider analysing six different possible proxies for memorization scores. Some of these proxies represent well-known properties of learning algorithms while others are completely novel. To our knowledge, this analysis represents the first attempt to approximate memorization scores.

### 5.2.1 Training History Statistics

We consider two different proxies extractable from the behaviour of the model throughout its training history. These metrics are thought solely from data sample complexity – typicality – perspective and thus represent the simplest possible proxies for memorization.

**Sample Forgettability.** We consider example forgetting events as firstly defined by [Toneva et al., 2019]. Similarly to its original formulation, we consider a forgetting event for a single sample $\mathbf{x}_i$ when it is misclassified during the training process at step $t+1$ after having been correctly classified at step $t$. Mathematically, we denote the predicted label for sample $\mathbf{x}_i$ by model $h$ at epoch $t$ as $\hat{y}_i^t = \underset{c}{\operatorname{argmax}} \{h^t(\mathbf{x}_i)\}$. Similarly, we define a binary variable indicating whether the example is correctly classified at epoch $t$ as $acc_i^t = \mathbb{1}_{\hat{y}_i^t = y_i}$. As such, example $i$ undergoes a forgetting event when $acc_i$ decreases between two consecutive updates: $acc_i^t > acc_i^{t+1}$. The forgetting event definition perfectly resembles the original [Toneva et al., 2019] definition. However, we here extend the forgetting events definition to introduce the concept of sample forgettability, which identifies the frequency of forgetting events for the sample at hand over the span of the whole training. As such, we count the forgetting events that occurred while training for $T$ epochs and normalize the obtained value by $T$. Mathematically, the sample forgettability is defined as:

$$F(\mathbf{x}_i) = \frac{\sum_{t=1}^{T} \mathbb{1}_{acc_i^t > acc_i^{t-1}}}{T}. \tag{5.3}$$

As highlighted by [Toneva et al., 2019], lower forgettability values correspond to *clean* samples, while high forgettability corresponds to complex or mislabelled samples.

**E2L: Epochs to Learn.** Inspired by [Toneva et al., 2019] which focused mostly on forgetting events, we here define a novel metric to be used as a proxy for identifying the complexity of samples. More in detail, we here consider the number of epochs required by the model to learn a sample $\mathbf{x}_i$ and never forget it anymore.

Studying only the number of forgetting events required by one model to learn the sample at hand is not sufficiently expressive. Indeed, while it is reasonable for *clean* samples to require few forgetting events to be correctly classified, throughout the training process complex samples might either *(i)* be wrongly classified for the majority of training after which the model learns the sample and never forgets (resulting in low forgettability) it, or *(ii)* be correctly and wrongly classified, switching from a correct state to a bad one (resulting in high forgettability). To account for such limitations of the forgettability metric, we here consider the *epochs to learn* metric defined as:

$$E2L(\mathbf{x}_i) = \begin{cases} \dfrac{\underset{t}{\text{argmax}} \left\{acc_i^t > acc_i^{t-1}\right\}}{T} & \text{if } acc_i^T = 1 \\ 1 & \text{otherwise.} \end{cases} \tag{5.4}$$

Similarly to memorization and sample forgettability, the proposed metric ranges between 0 – when a sample $\mathbf{x}_i$ is learnt at the very first epoch and never forgotten – and 1—when a sample is never learnt.

**Complexity** Samples forgettability and *E2L* require the training of a single model $h$ over the algorithm $\mathcal{A}$ and the dataset $\mathcal{T}$ to compute the value of $F(\mathbf{x}_i)$ for each sample in $\mathcal{T}$. In practice, as the randomness of the training algorithm influences the outcome of $F(\mathbf{x}_i)$, we run the training process $n$ times and average the obtained proxy values over the $n$ different runs.

### 5.2.2 Loss Flatness

**Loss curvature.** [Garg and Roy, 2023] studied the second order of the loss function of trained and partially trained models over dataset samples, highlighting an empirical correlation between the loss curvature and the samples cleanliness. Inspired by their work, we here follow their approach and leverage the loss curvature of trained models as a possible proxy for the memorization scores. More in detail, given a trained models $h$ and a loss function $\mathcal{L}$, [Garg and Roy, 2023] defines the loss curvature metric around a sample $\mathbf{x}_i$ as

$$\gamma(\mathbf{x}_i) = \left\| \nabla_{\mathbf{x}_i} \mathcal{L}\left(\mathbf{x}_i + h \cdot z\right) - \nabla_{\mathbf{x}_i} \mathcal{L}(\mathbf{x}_i) \right\|, \tag{5.5}$$

where $z = \frac{sign\left(\nabla_{\mathbf{x}_i} \mathcal{L}(\mathbf{x}_i)\right)}{\left\|sign\left(\nabla_{\mathbf{x}_i} \mathcal{L}(\mathbf{x}_i)\right)\right\|}$ and $h$ represents a hyperparameter empirically set to $h = 3$. While relevant for identifying simple samples, the authors of this metric lack comparing the obtained curvature scores with memorization, enabling our analysis. Similarly to the *sample forgettability* metric, the loss curvature represents both a possible proxy and a readily available baseline for building coresets, and will be considered dually in our experimental evaluation of Section 5.3.

**SAM's $\epsilon$.** Inspired by the empirical findings of [Garg and Roy, 2023] on the correlation between sample cleanliness and loss curvature, we here consider an alternative proxy for memorization scores that relies on the loss flatness metric as defined in [Foret et al., 2021]. More in detail, [Foret et al., 2021] defines a Sharpness-Aware Minimization (SAM) approach as an alternative to the popular Stochastic Gradient Descent (SGD)[3] optimization algorithm. SAM relies on simultaneously minimizing loss value and loss sharpness to achieve generalization benefits. To minimize sharpness, SAM's optimization procedure relies on bounding the model's generalization ability in terms of neighborhood-wise training loss, computing the maximum difference between the loss from the current weights $w$ and the losses near the current weights $w + \epsilon$. To this end, SAM computes the optimal $\epsilon$ by analysing the adversarial direction of the gradient to achieve the maximum loss difference. We consider leveraging SAM's definition of $\epsilon$ as a possible proxy for the memorization scores. The underlying assumption is that very flat minima result in small $\epsilon$ values, while sharp minima correspond to high $\epsilon$. Therefore, $\epsilon$ can be theoretically leveraged to measure the flatness of the loss around a given sample $\mathbf{x}_i$. To this end, we consider computing SAM's $\epsilon$ per-sample rather than per-batch and only at the end of model training. Therefore, we can define

[3] https://en.wikipedia.org/w/index.php?title=Stochastic_gradient_descent

the $\epsilon$ proxy for memorization scores as:

$$\epsilon(\mathbf{x}_i) = \rho \cdot sign\left(\nabla_{\mathbf{x}_i}\mathcal{L}(\mathbf{x}_i)\right) \cdot \frac{|\nabla_{\mathbf{x}_i}\mathcal{L}(\mathbf{x}_i)|}{\sqrt{\|\nabla_{\mathbf{x}_i}\mathcal{L}(\mathbf{x}_i)\|}}, \tag{5.6}$$

where $\rho$ represents the neighborhood size and is set to $\rho = 0.5$ following the experimental findings of [Foret et al., 2021].

**Complexity.** Similarly to [Garg and Roy, 2023], we compute the loss flatness metrics $\gamma$ and $\epsilon$ on sample $\mathbf{x}_i$ after the model $h$ has been optimized on $\mathcal{T}$ with $\mathbf{x}_i \in \mathcal{T}$. Therefore, similarly to the history statistics proxies, $\gamma$ and $\epsilon$ require training one single model $h$ to compute the proxies over all training data samples. In practice, we run the training process $n$ times and average the obtained proxy values over the $n$ different runs. Finally, during our experimentation, we empirically found that there exists no relevant difference between computing loss flatness using the model trained on the sample – i.e., $h^{\mathcal{A}}_{\mathcal{T}}$ – or without the sample – i.e., $h^{\mathcal{A}}_{\mathcal{T}\setminus i}$ –, when the trained model does not suffer overfitting issues.

### 5.2.3 SAMIS: SAM - SGD

Inspired by [Kim et al., 2024], we propose a novel approach that combines both Sharpness-Aware Minimization (SAM) and Stochastic Gradient Descent (SGD) to approximate memorization scores in neural networks. They introduced an entropy-based metric that leverages influence scores to assess the reliance of a model's prediction for a given test point on a limited subset of training data. Their findings suggest that predictions for test points that are highly dependent on specific training examples exhibit a more pronounced discrepancy in test accuracy between models trained with SAM and those trained with SGD.

We hypothesize that this phenomenon can be extrapolated to training data points with high memorization scores. Such points, akin to outliers, would similarly exhibit a heavy dependency on a few training examples were they part of a test set. This is because a data point with a high memorization score is, by definition, atypical. In contrast, if the data point was typical, surrounded by numerous similar examples, it would likely have a lower memorization score if it was in the training set and low dependency if it was in the test set since its prediction would be influenced by a broader array of training data. Building on this understanding, we propose using the discrepancy in predictions between SAM and SGD models as a proxy for memorization scores. In particular, we start from the definition of the memorization score as the difference between model output probabilities. The memorization definition considers the model trained on the whole dataset $\mathcal{S}$ and on the dataset $\mathcal{S}^{\setminus i}$ lacking sample $\mathbf{x}_i$ for which it is necessary to compute the memorization score. We assume that SAM can attain higher generalization capabilities than SGD and consider replacing $h_{\mathcal{S}}^{SGD}$ with $h_{\mathcal{S}^{\setminus i}}^{SAM}$. Although this is not a valid assumption in general, this allows replacing the expensive term of memorization computation $P\left[h_{\mathcal{S}}^{SGD}(\mathbf{x}_i) = y_i\right]$ with a less computationally demanding term $P\left[h_{\mathcal{S}^{\setminus i}}^{SAM}(\mathbf{x}_i) = y_i\right]$.

We define the SAM − SGD (SAMIS) formulation as the average absolute difference in output distribution between the models obtained using SAM and SGD.

$$S_P(\mathbf{x}_i) = \frac{\sum_{c=1}^{C} |P\left(h^{\alpha}(\mathbf{x}_i) = c\right) - P\left(h^{\sigma}(\mathbf{x}_i) = c\right)|}{C}, \tag{5.7}$$

where $h^{\alpha}$ represents the model optimised using SAM ($h_{\mathcal{S}^{\setminus i}}^{SAM}$), $h^{\sigma}$ its SGD counterpart ($h_{\mathcal{S}^{\setminus i}}^{SGD}$) and $C$ the number of classes in the dataset at hand.

Inspired by previous works analysing loss properties, we here consider defining a dual variant of the SAMIS metric focusing on the difference between the SAM based training and the SGD model in terms of loss obtained on the data sample at hand. Mathematically:

$$S_L(\mathbf{x}_i) = |\mathcal{L}\left(h^{\alpha}(\mathbf{x}_i), y_i\right) - \mathcal{L}\left(h^{\sigma}(\mathbf{x}_i), y_i\right)|, \tag{5.8}$$

Taking into account a standard cross entropy loss function, defined as:

$$\mathcal{L} = -\sum_{c} y_{i,c} \cdot log(P\left[h(\mathbf{x}_i) = c\right]), \tag{5.9}$$

the proposed $S_L$ formulation can be rewritten as a

$$\begin{aligned} S_L(\mathbf{x}_i) = &\left| -log\left[P(h^{SAM}_{\mathcal{S}^{\setminus i}}(\mathbf{x}_i) = y_i)\right] \right. \\ &\left. + log\left[P(h^{SGD}_{\mathcal{S}^{\setminus i}}(\mathbf{x}_i) = y_i)\right] \right|, \end{aligned} \tag{5.10}$$

which resembles the memorization score definition of Equation (5.2) logarithmically scaled.

**Complexity.** Replacing $h^{SGD}_{\mathcal{S}}$ with $h^{SAM}_{\mathcal{S}^{\setminus i}}$ allows to train fewer models to approximate correctly the metric, as it requires only $\mathbf{x}_i$ to not be a part of the training samples. Therefore, it is possible to approximate $h^{SAM}_{\mathcal{S}^{\setminus i}}$ with $h^{SAM}_{\mathcal{S}^{\setminus \{i,j,k,\ldots\}}}$ and reuse it for computing the proxy values for $(x_j)$, $(x_k)$ and all samples removed from the training set. This assumption is valid when training is performed on a big enough subset of the original training set $\mathcal{S}$ such that $h^{SAM}_{\mathcal{S}^{\setminus i}} \simeq h^{SAM}_{\mathcal{S}^{\setminus \{i,j,k,\ldots\}}}$. To this end, we consider splitting the training dataset for which it is required to compute the SAMIS proxy into $m$ splits – namely $\mathcal{S}_j\ \forall j \in [1, m]$ – such that $\mathcal{S} = \bigcup_{j=1}^{m} \mathcal{S}_j$. We then consider keeping each split as the validation set and train two models – one using SAM, and one using SGD – on the remaining samples belonging to $\mathcal{S}^{\setminus k}\ \forall k \in \mathcal{S}_j$. Thus, leveraging $h^{SAM}_{\mathcal{S}^{\setminus k}\ \forall k \in \mathcal{S}_j}$ and $h^{SGD}_{\mathcal{S}^{\setminus k}\ \forall k \in \mathcal{S}_j}$ it is possible to compute the SAMIS proxy for all samples $k$ in the split $\mathcal{S}_j$. The desired approach requires training $2 \cdot m$ models to compute the proxies over all training data samples. In practice, we run the training process $n$ times and average the obtained proxy values over the $n$ different runs. Therefore, overall the samis metrics require the optimization of $2 \cdot m \cdot n$ models to compute the memorization proxy scores with precision. Throughout our evaluation we set $m = 10$ and $n = 10$, thus requiring the optimization of 200 models to approximate the memorization scores.

### 5.2.4 Proxies Quality Analysis

To analyse the quality of the six different proxies considered, we compare them against the memorization scores available from [Feldman and Zhang, 2020] for the CIFAR100 dataset [Krizhevsky and Hinton, 2009]. More in detail, we compute each proxy measure for each sample $\mathbf{x}_i$ belonging to the training set of the CIFAR100 dataset over 10 different runs and consider the final proxy metrics as the average scores of the metrics over the 10 runs. For each run, we vary the initial conditions by seeding differently the training procedure of the models required to compute the proxy, but we keep the learning hyperparameters untouched. To be as consistent as possible with the memorization setup we use the same model architecture and same training hyperparameters of [Feldman and Zhang, 2020]. In particular, we use a ResNet50 architecture, and SGD with momentum 0.9, a batch size of 512 and a base learning rate of 0.4. We train each model for 160 training epochs, in which the learning rate is scheduled to grow linearly from 0 to the base learning rate in the first 15% of epochs, and then decay linearly back to 0 in the remaining epochs. Finally, when the proxy values are not bounded, as it is for the loss curvature, we normalize the score using a min-max normalization approach to fit them in $[0, 1]$.

**Rank correlation** We first consider comparing the rank correlation between each proxy and memorization by using the Spearman rank correlation $\rho$. The correlation between memorization and the obtained proxies averaged over a different number of runs is shown in Figure 5.2. Interestingly, the results show how there exists a positive non-negligible correlation between memorization scores and all proxies. These results highlight the promising nature of all the proxies considered, showing how it is possible to reproduce the sample ranking obtained through memorization scores. More in detail, $S_L(\mathbf{x}_i)$ achieves the highest correlation reaching up to $\rho = 0.92$ for $k = 10$, highlighting that replacing $h_{\mathcal{S}}^{SGD}$ with $h_{\mathcal{S}^{\setminus i}}^{SAM}$ represents a reasonable assumption given the generalization properties of SAM. Interestingly, very simple proxies such as *sample forgettability* (Equation (5.3)) and *E2L* (Equation (5.4)) achieve high correlation with memorization, even if defined in a completely agnostic manner.

**Correlation nature** Analysing only the rank correlation coefficient is not sufficiently effective for studying the similarity nature between memorization scores and the defined proxy metrics. Therefore, we consider analysing for which values of the memorization spectrum each proxy is capable of approximating the baseline scores. To this end, we first analyse the number of common samples between memorization and each proxy over different splits of the CIFAR100 dataset. We

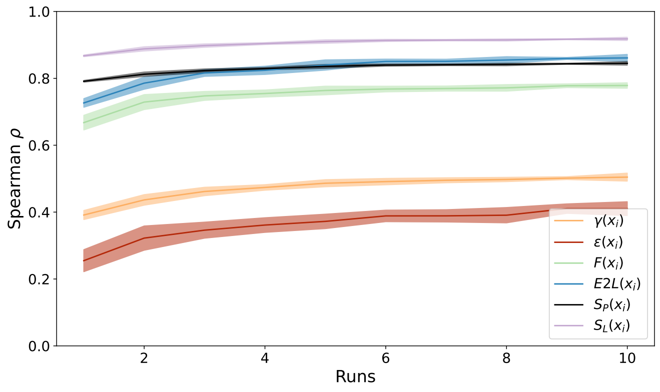


Figure 5.2: Correlation between memorization and proxies averaged over the number of runs.

sort the memorization scores and the proxy counterparts and split the CIFAR100 dataset into 10 splits each of which containing 5000 samples gathered from the sorted memorization and proxies indices. For each split we compute the percentage of common samples between the memorization scores and the proxies, aiming to study how the correlation varies through the memorization scores spectrum. Figure 5.3 shows the results of such analysis. Most proxies can approximate quite well the search for clean samples since low memorization scores have been empirically linked with sample cleanliness. On the other hand, most proxies fail to effectively approximate the high-end of the memorization spectrum, as they focus on different aspects of complex samples. Indeed, the only relevant proxy at the highest end of the memorization scores is represented by $S_L(\mathbf{x}_i)$. A similar visualization of such phenomenon is also made available in Figure 5.5.

**Scores distribution** To corroborate the findings on the correlation distribution we analyse the values distribution for each proxy. Figure 5.3 shows how the distribution of scores differs between memorization and the considered proxies. The obtained results support the findings of Figure 5.3, highlighting how most proxies can approximate well – to some extent – the memorization scores distribution for lower values of the spectrum. On the other hand, none of the proxies is capable of approximating effectively the high end of the memorization spectrum—in which memorization scores are characterised by a small but significant rise in memoriza-

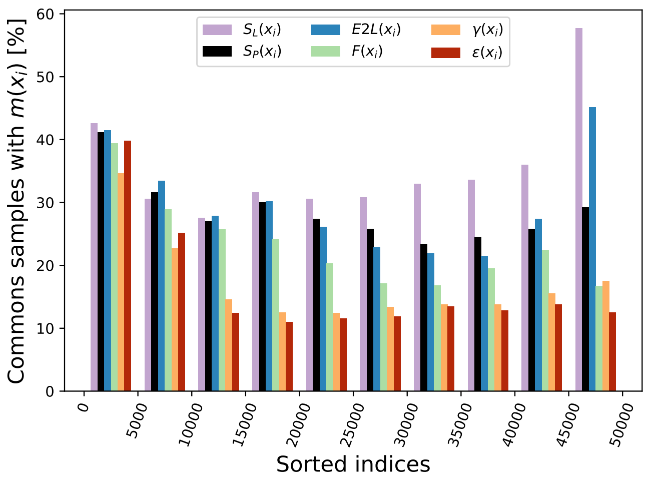


Figure 5.3: Common samples between the memorization scores and the proxies over low to high memorization scores.

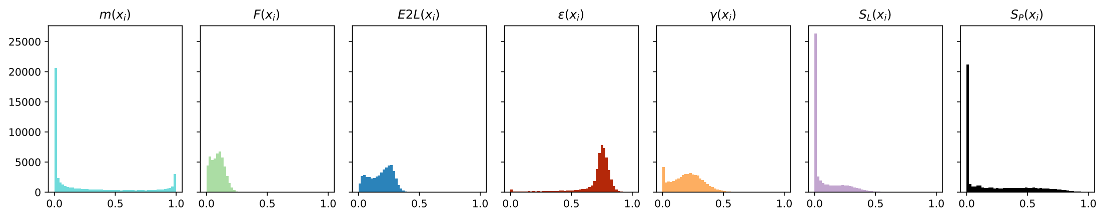
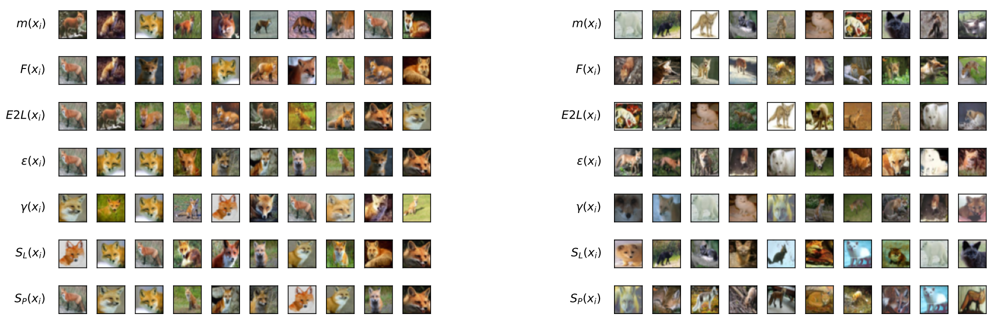


Figure 5.4: Histogram of memorization and proxies scores distribution.

tion values. The best proxies are the SAMIS variants which approximate very well the memorization behaviour on its lower end of the spectrum and only miss the small memorization rise close to 1. Meanwhile, the flatness-based proxies fail to replicate memorization scores over its whole spectrum, as it is less likely for the loss flatness or curvature to reach values very close to zero.

**Sample visualization** To investigate the discrepancy between memorization and proxies at the high end of the memorization spectrum, we plot the 5 samples achieving the lowest and highest scores for a specific class of the CIFAR100 dataset. Figure 5.5 shows the obtained samples for the class *fox*. The visual results confirm the findings on proxies correlation. Low values result in clean images of orange foxes on a natural background, containing either a close-up of the fox's



(a) Samples with lowest memorization/proxies scores

(b) Samples with highest memorization/proxies scores

Figure 5.5: Sample visualization for the class *fox* of the CIFAR100 dataset over low (a) and high (b) memorization/proxies scores.

head or the full body of the fox. Several images are identical between memorization and the proxies – especially the SAMIS variants –, showing almost perfect correlation. These results highlight how all of the considered proxies can be used to identify clean, simple samples independently of their capability to approximate the memorization values. On the other hand, the samples extracted from the metrics' highest values differ from one another, while still seemingly representing complex – unconventional – samples. More in detail, each metric seems to focus specifically on some of the image properties. For example, memorization seems to highlight the colour of the fox inside the image, as the extracted samples contain only black and white foxes. The *E2L* metric seems to focus on shape-based properties of the image while disregarding the colour criterion since its images contain cropped or further away foxes as well as hand-drawn foxes. Finally, the SAMIS images seem to represent a mixture of colour and shape properties (images contain black foxes as well as cropped images). It is important to notice that this analysis on colour and shape is valid only for this specific class, as for other classes the difference between memorization and proxies images seem to rely on other characteristics of the class at hand.

**Correlation over classes** Finally, we investigate the correlation between memorization and proxy scores over each of the classes composing CIFAR100 to better understand the impact of class complexity on the proxy. SAMIS and *E2L* proxies

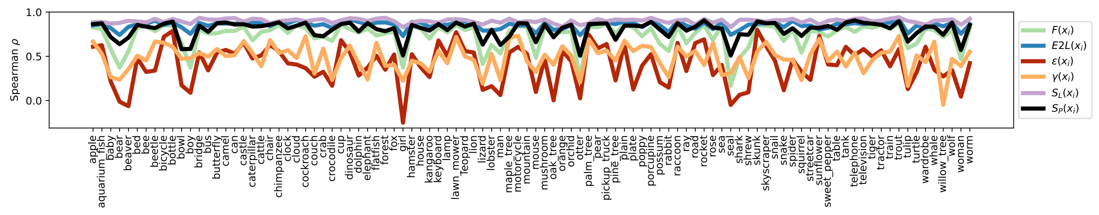


Figure 5.6: Pearson correlation between memorization and proxies over each CIFAR100 class.

represent the most effective metrics, as shown in Figure 5.6. Interestingly, the correlation behaviour of most proxies is rather turbulent. Indeed, only the metrics reaching an overall Pearson correlation greater than 0.8 showcase a sufficiently uniform correlation value over all classes. The *girl*, *bear* and *seal* classes represent the most complex instances to be well approximated by proxies. For these instances, even the best-performing proxy suffers a decrease in correlation up to 0.2, showcasing the complexity of memorization approximation. Conversely, the *bottle*, *lawn mower* and *palm tree* classes represent the easiest instances to be well approximated.

## 5.3 Memorization Proxies for Coreset Construction: Experiments

In this section, we propose leveraging the proxy metrics for memorization scores for coreset construction procedures. Similarly to the memorization coreset construction approach, we here use proxy estimate $\pi_i$ as the selection metric, choosing samples with the lowest values to make up our coreset.

### 5.3.1 Experiments Setup

We run experiments on CIFAR10 and CIFAR100 [Krizhevsky and Hinton, 2009] datasets. The CIFAR10 and CIFAR100 datasets contain 50000 training images split into 10 and 100 classes respectively, where the number of samples per class is perfectly balanced. Each data sample is a 32-by-32 RGB image.

We compare each of the performance of the coresets constructed leveraging each of the proxies – as well as memorization itself – against all of the baselines considered in Section 5.1.3. Here it is relevant to notice that *sample forgettability* and *loss curvature* proxies were already used by [Toneva et al., 2019] and [Garg and Roy, 2023] for coreset construction, and as such we consider them both as part of the baselines and part of the proxies. As various levels of complexity characterise the different datasets, the proxy metrics are computed using a ResNet architecture of slightly different sizes depending on the dataset. More in detail, we leverage ResNet50 to compute proxies on CIFAR100 and ResNet18 for proxies extraction on all other datasets. For each dataset the set of coreset sizes $\mathcal{P}$ and the hyperparameters used for coreset training are available in Table 5.1.

| Dataset | $\mathcal{P}$ | Epochs | $\lambda$ | $\beta$ | $\lambda_s$ | $\lambda_f$ |
|---|---|---|---|---|---|---|
| CIFAR100 | $[0.01, 0.2]$ | 160 | 0.1 | 32 | 50 | 0.1 |
| CIFAR10 | $[0.01, 0.1]$ | 160 | 0.1 | 32 | 50 | 0.1 |
| SVHN | $[0.01, 0.2]$ | 30 | 0.1 | 32 | 10 | 0.1 |
| Fashion-MNIST | $[0.01, 0.2]$ | 10 | 0.1 | 32 | 3 | 0.1 |

Table 5.1: Coreset size and hyperparameter of coreset training used for each dataset. $\lambda$ = starting learning rate, $\beta$ = batch size, $\lambda_s$ = number of epochs after which the learning rate decays, $\lambda_f$ = learning rate decay factor.

### 5.3.2 Proxies vs. Baselines

Figures 5.7 and 5.8 show the results obtained from the comparison of the considered proxies against the selected coreset construction baselines.

The obtained results show that selection based on memorization scores still represent the best approach, outperforming all other coreset methods over all datasets. $S_L(i)$ represents the best performing proxy, following closely the behaviour of coreset construction based on memorization. Interestingly, other highly correlated proxies such as *E2L* and *sample forgettability* do not perform as closely to memorization as SAMIS. This behaviour is caused by the decrease in correlation for the lowest values of memorization scores that these metrics achieve (see Figure 5.3). Overall, these results represent an encouraging finding, as they prove

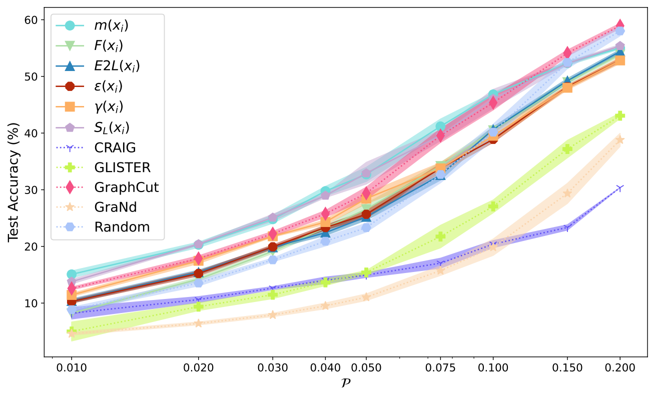


Figure 5.7: Coreset construction performance using memorization proxies against state-of-the-art solutions on CIFAR100.

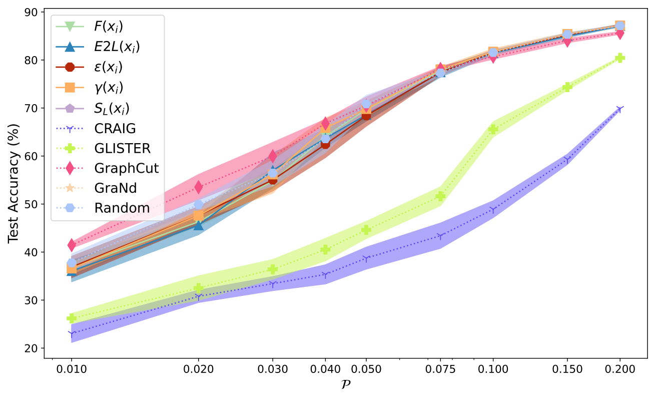


Figure 5.8: Coreset construction performance using memorization proxies against state-of-the-art solutions on CIFAR10.

how it is possible to approximate the expensive memorization scores $m_i$ – using $S_L(i)$ – while retaining the same behaviour on coreset construction.

Once again, the strongest competitor is the submodular function-based method GraphCut. We also note that for higher coresets sizes the random sampling baseline forms a surprisingly strong approach (also noted in [Guo et al., 2022]). Finally, it is possible to notice a slight decrease in the delta of performance between memorization coreset construction approach – along with its proxies – and the considered baselines over larger coreset sizes—e.g., $\mathcal{P} = 0.15$ and $\mathcal{P} = 0.2$ for

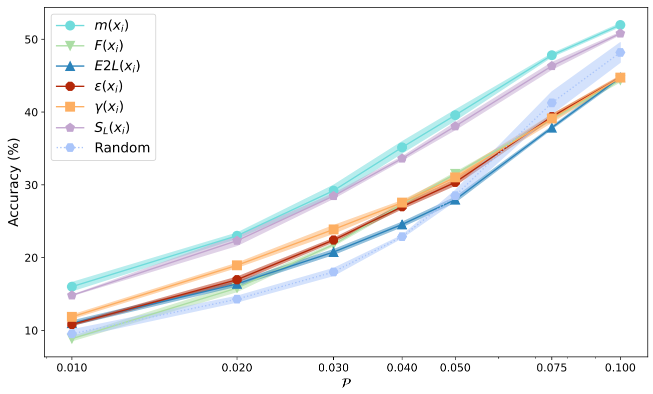


Figure 5.9: Cross architecture performance of proxies computed using ResNet18 and transfered to WideResNet.

CIFAR100. This behaviour hints that the amount of clean samples in the considered datasets is overabundant, as for $\mathcal{P} = 0.2$ it is more effective to also consider complex samples rather than considering only the set of top-20% clean samples.

### 5.3.3 Cross-Architectures Coresets

The proxy metrics considered in this chapter require less resources than memorization to be computed. However, they still require training a relevant amount of models—at least 10 in our experiments. Therefore, we here consider if – and to what extent – memorization and proxy scores can transfer through different architectures during the coreset construction procedure. To this end, we consider training a model architecture – e.g., MobileNet – on the clean samples obtained from the proxy scores extracted from a different model architecture—e.g., ResNet50. In our experiments we select ResNet18 [He et al., 2016], ResNet50 [He et al., 2016], VGG-19 [Simonyan and Zisserman, 2015], MobileNetV3Small [Howard et al., 2019], MobileNetV3Large [Howard et al., 2019] and InceptionNetV4 [Szegedy et al., 2017] and train them using the coresets constructed with proxy measures extracted from a ResNet18 architecture.

Figure 5.9 shows the results obtained for each proxy metric when trained over the WideResNet architecture. We plot also the random sampling baseline – which forms a strong baseline as noted in Section 5.3.2 – as a reference for the expected architecture performance. The obtained results highlight how memorization and proxies can transfer well across different architectures, given the cleanliness of the extracted samples. These findings suggest that the data properties identified via proxy metrics over an architecture can generalize well to other models, thus rendering the memorization proxies almost completely independent of the considered architecture. Similar results can be obtained for all analysed architectures and are not shown to avoid redundancy.

### 5.3.4 Balance Memorization and Proxies

Results in Figure 5.7 highlight a slight decrease in coreset construction performance of memorization scores – and its proxies – at larger coreset sizes. Inspired by such an insight, we here consider defining a weighted sampling approach for constructing coresets based on memorization and its proxies. Rather than constructing coresets selecting the lowest memorization samples, we here consider assigning a sampling probability inversely proportional to the memorization – or proxy – score of the sample. The underlying assumption is that assigning a small, but non-negligible sampling probability also to mildly complex data samples allows to diversify the training set information in scenario where the selected coreset size is not prohibitively small. Indeed, while training only on the lowest memorization/proxy data samples is effective for very small coresets, it is also true that clean samples are usually overabundant in common datasets, while the information extractable from rough samples should still be taken into account during training. To avoid sharp differences between the assigned sampling probability, we first consider smoothing the sample scores using a sigmoid function and we apply softmax to the corresponding output obtaining a probability distribution over all samples. Mathematically, we denote with $p_i$ the sampling probability of sample $\mathbf{x}_i$ and define it as:

$$p_i = \frac{e^{\sigma_i}}{\sum_{j \in S} e^{\sigma_i}} \quad \text{with} \quad \sigma_i = \frac{1}{1 + e^{\tau \cdot (s_i - \kappa)}}, \tag{5.11}$$

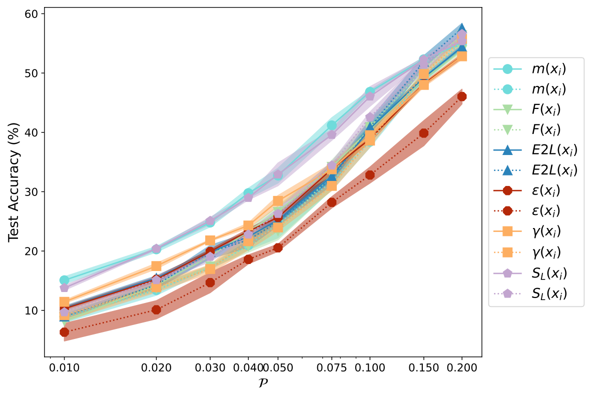


Figure 5.10: Coreset construction performance when balanced sampling based on memorization/proxies scores is considered against sampling directly the samples resulting in the minimum memorization/proxy scores.

where $s_i$ represents the memorization/proxy score of sample $\mathbf{x}_i$, and $\tau$ and $\kappa$ represent two hyperparameters of the sigmoid function that we set empirically to $\tau = 8$ and $\kappa = 0.5$.

The results obtained for coresets contruction over the CIFAR100 dataset using the novel balanced sampling approach are shown in Figure 5.10. As expected the balanced sampling approach performs poorly for very small coreset sizes, as it enables the sampling of mildly complex samples that hinder the training process if not sorrounded by enough clean samples. On the other hand, as the coreset size increases, the balanced sampling approach seem to catch up with the minimum sampling procedure. For most of the proxies the balanced sampling approach outperforms slightly its counterpart when the percentage of dataset samples used to construct the coreset overcomes 0.15. These results confirm the original assumption that as the training set size increases it becomes more helpful to add complex or mildly complex samples to the learning procedure, since the amount of information that the model can extract from only clean samples is limited. The obtained results also seem to corroborate the overabundance of clean samples in the CIFAR100 dataset, as for $\mathcal{P} = 0.2$ it is more effective to consider complex samples rather than considering only the set of top-20% clean samples.

## Chapter Synopsis

In this chapter, we investigate the effectiveness of leveraging data sample memorization scores and their proxies for data efficient learning task. We first show that a simple sampling of data attaining lowest memorization scores results in more effective coresets when a small number of training examples is considered—i.e., few-shot learning. To account for the resource hungriness of memorization computation we analyze six different proxy metrics, showcasing their high level of correlation and the nature of their similarity with memorization. Thereafter, we replicate the memorization-based coreset construction approach leveraging the considered proxies and compare them with state-of-the-art approaches, highlighting their effectiveness. Approximating the memorization effect facilitates the investigation of *clean* vs. *atypical* or *mislabelled* data instances over large datasets where the optimization of more than 1000 models would be unfeasible for most. Overall, improving the efficiency of coreset construction approaches, the proposed approach represents a step towards enabling the training of complex NNs on resource-constrained devices.

# Chapter 6

# Resource Management in Federated Learning

This chapter contains contributions from [Agiollo et al., 2024b].

Federated Learning (FL) – the de-facto solution for distributed learning scenarios – reduces the amount of data and resources required for training ML and DL models by leveraging the collective intelligence of distributed devices. However, the energy and resource utilization optimisation represents an overlooked issue in most FL scenarios as most recent efforts focus on model performance and privacy [Zhang et al., 2021a, Lyu et al., 2020, Agiollo et al., 2024a]. This phenomenon – along with the intrinsic complexity of the *performance vs. resource consumption* trade-off – hinders the adoption of FL over constrained scenarios. Inspired by the above issue, in this chapter we introduce EneA-FL as a pioneering serverless FL framework, uniquely equipped with a smart energy management module tailored for resource-constrained clients. Conceived to address the energy management challenges inherent in the real-world deployment of FL solutions, this innovative middleware is crafted to facilitate the training and deployment of FL models across a spectrum of heterogeneous Internet of Things (IoT) devices. In particular, the core is an orchestrator that dynamically manages the FL process by shipping to participant nodes a containerised environment capable of monitoring the current status of the hosting machine in terms of: *(i)* computing and networking capabilities, *(ii)* energy budget, and *(iii)* current accuracy over local samples.

EneA-FL dynamically assesses the effort required by each IoT client to fulfil the specified resource constraints. Subsequently, it autonomously applies the most suitable policy, by minimising any intervention by FL users and FL-based application developers. Our automated energy management scheme has been tested over a wide variety of clients and energy requirements while showing its benefits over standard FL approaches. In particular, we tested our solution over CPU- and GPU-enabled microcontrollers with limited computing capabilities and reduced energy budgets [Agiollo and Omicini, 2021]. The adaptable container-based approach proposed here has shown to be able to achieve between 30% and 60% lower energy consumption against popular client selection approaches available in the literature.

## 6.1 Related Work

In this section we provide an overview of recent research efforts in the research areas intersecting our proposal—i.e., Serverless containerisation for ML, and resource management for FL and fog computing scenarios.

**Serverless containerisation for Machine Learning:** As real-world ML applications continue to expand, there is a growing demand for significantly increased computational power to execute training processes [Sevilla et al., 2022]. In this context, cloud resources prove invaluable, enabling the execution of computationally intensive operations that would be impractical for individuals and mid-sized companies. In addition, the combination of serverless paradigm with containerisation approaches enables remarkable scalability at minimal overhead cost [Rudyy et al., 2019]. While ML-based container orchestration [Zhong et al., 2022, Rovnyagin et al., 2020] has demonstrated the benefits of ML in Serverless computing, the exploration of the applicability of containerised serverless scheme for ML is still pending validation. In this context, several works demonstrate the minimal impact of containerisation on deep learning application performance. [Xu et al., 2017] results pinpoint how containers have a 0.2%-0.5% overhead compared with host execution time, proving the deep learning containerisation feasibility.

In the distributed ML realm, several works focused on serverless paradigms for

FL, such as [Grafberger et al., 2021] where the authors illustrate the benefits of a middleware solution that eases the interaction between multiple Software-as-a-Service (SaaS) cloud providers. Similarly, [Singh et al., 2022] present a distributed real-time privacy-preserving data analytic solution for smart grid systems based on a Serverless cloud computing FL approach to predict the energy needs of Home Area Networks (HANs). Despite the remarkable relevance of the above-mentioned solutions, none of them have already addressed the feasibility of Serverless computing for FL in a fog environment.

**Resource Management in Fog Computing:** Extensive analysis of resource management in fog computing environments [Yi et al., 2015, Puliafito et al., 2019] is meticulously explored by [Ghobaei-Arani et al., 2019], delving deeply into a comprehensive examination of resource management within fog computing settings. The investigation results pinpoint task offloading [Hamdi et al., 2022a, Hussein and Mousa, 2020] as the predominant resource management approach in the literature. In this context, the need for ensuring transparency [Shan et al., 2019, Weiner et al., 2022] in offloading decisions for application developers is crucial. Additionally, the heterogeneity of mobile nodes [Zhang et al., 2017] and their energy performance seems still underexplored. This forward-looking perspective aims to promote efficient and developer-friendly offloading strategies in the evolving landscape of mobile computing. Relevantly, the majority of academic efforts tackle offloading strategies by showcasing innovative solutions leveraging simulation tools [Gupta et al., 2017, Calheiros et al., 2010, Puliafito et al., 2020] and ignoring QoS factors—i.e., only 18% of the reviewed articles [Zhao et al., 2016, Meng et al., 2017] take energy considerations into account. These factors represent relevant limitations for enabling real-world deployment of such solutions.

**Resource Management in Federated Learning:** FL represents the most popular technology for enabling multi-party joint training of ML models. In this context, multiple entities collaborate to locally optimise a shared model by sending their local updates either relying on a central controller or in a fully decentralised fashion. Relying on the local optimisation procedure of the shared model, FL ensures data privacy, while enforcing heavy computational constraints on the federation entities. Therefore, the application of FL to resource-constrained devices – e.g., IoT devices, battery-powered devices, etc. – represents an open research issue [Nguyen et al., 2021, Imteaj et al., 2022]. In this context, many of the research efforts focus on the identification of effective learning strategies to optimise local model training. [Wu et al., 2018b] propose relying on lower bit-length integers to reduce the computational costs of training and inference models. Similarly, tensor rematerialisation [Jain et al., 2020], recomputation [Chen et al., 2016], and efficient architecture strategies [Cai et al., 2020] have been proposed to reduce memory requirements of NN training. Although these proposals do not target directly the FL realm, these works focus on resource optimisation during model training and thus can be applied to FL to achieve a more efficient federation scheme. Few works have specifically dealt with the optimisation of resources in the field of FL when the federation is composed of inherently constrained and heterogeneous devices—in terms of either available energy or computational power [Trindade et al., 2021]. The problem of energy-efficient model transmission for FL over wireless communication networks is analysed in [Yang et al., 2021b], where both local computation energy and transmission energy are taken into account, formulating the FL convergence problem as a system energy minimisation problem. Meanwhile, [Zaw and Hong, 2021] formulate an energy-conscious resource management problem for FL where the federation clients aim to minimise time over a set of energy and communication constraints. Here, the problem is formulated as a Nash equilibrium problem [Nash Jr, 1950], solved in a decentralised fashion. In [Xu et al., 2022], the optimisation of the local update frequency and the compression ratio of the model to effectively decrease the time required for optimisation is proposed, thus reducing the consumed resources. [Cui et al., 2022] aims at reducing FL resource usage by optimising their allocation. To this extent, the authors analyse resource block allocation introducing a mixed-integer linear programming strategy to better allocate resource blocks over federation clients.

In this resource efficientisation context, few approaches focus on the client selection process typical of FL. In this context, OORT [Lai et al., 2021] represents the most popular node selection framework where clients are selected depending on their model ”utility”, defined as the potential improvement over the aggregated model. [Arouj and Abdelmoniem, 2022] propose an improved version of OORT – to which we will refer as OORTv2 for the remainder of the chapter – prioritizing clients having higher battery levels to maximize the overall system efficiency.

Diversely from these approaches which focus solely on the computation efficiency perspective, [Cho et al., 2020] analyses the computation and communication efficiency perspective, proposing a *power-of-choice*-based solution. Total time to convergence represents another relevant factor in FL setups, attracting several research efforts such as [Kim and Wu, 2021, Wang et al., 2019a]. More in detail, [Kim and Wu, 2021] propose to jointly optimise time to convergence and energy consumption in data heterogeneity scenarios, proposing a reinforcement learning client selection algorithm. Similarly, the results in [Wang et al., 2019a] highlight reduced time execution when communication data collection and experiments upscaling over simulated environments are considered.

Although relevant, these approaches do not sufficiently address the academic gap in optimizing power consumption within real and heterogeneous environments. While some of the proposed solutions neglect to consider power consumption information entirely, others consider energy optimization as a direct consequence of training time minimization, which is not the case when heterogeneous nodes participate in the training process. Ultimately, none of the existing solutions prioritizes the concerns of application developers, leading to a lack of seamless integration in a serverless environment. The absence of consideration for developers hinders the smooth and effortless incorporation of these solutions into real-world applications. Therefore, up to our knowledge, there exists no study on the effectiveness of resource management solutions in FL when a set of resource bounds is considered, taking into account heterogeneous resource requirements and availability over multiple clients. Table 6.1 summarizes the limitations of the existing solution and highlights the academic gap that EneA-FL is meant to fill.

## 6.2 FL Energy Consumption Modelling in Fog Deployment Environments

One of the objectives of fog computing in several vertical domains of application is to process data in near real-time. Here, a set of edge nodes receives data from IoT devices and performs stream processing with millisecond-grade response time. Given the opportunistic and heterogeneous nature of those scenarios, the

Table 6.1: Comparison of related work based on their features. Legend: **EA** = Energy awareness, **RE** = Real energy data collection, **NH** = Node Heterogeneity, **C** = Comparison with other solutions, **SA** = Serverless architecture.

| **Work** | **EA** | **RE** | **NH** | **C** | **SA** |
|---|---|---|---|---|---|
| Grafberger et al. [Grafberger et al., 2021] | - | X | - | - | X |
| He et al. [He et al., 2021] | - | - | - | - | X |
| Singh et al. [Singh et al., 2022] | X | - | - | - | X |
| Yang et al. [Yang et al., 2021b] | X | X | - | - | - |
| Zaw et al. [Zaw and Hong, 2021] | X | - | - | - | - |
| Kim et al. [Kim and Wu, 2021] | X | X | X | - | - |
| Xu et al. [Xu et al., 2022] | - | X | X | - | - |
| Cui et al. [Cui et al., 2022] | - | X | X | X | - |
| Cho et al. [Cho et al., 2020] | X | - | - | - | - |
| Wang et al. [Wang et al., 2019a] | - | X | - | - | - |
| Lai et al. [Lai et al., 2021] | - | - | X | X | - |
| Arouj et al. [Arouj and Abdelmoniem, 2022] | X | - | X | X | - |
| EneA-FL | X | X | X | X | X |

unavailability of a node reaching the end of its energy budget is an eventuality to be avoided at any cost. Generally speaking, if we define $E(i)$ as the energy consumption of a fog node $i$, and consider the canonical FL scenario made up of an arbitrary set of workers of size $n$ and one aggregator node – where ideally, all working nodes presents the same networking and computational capabilities – we can compute the energy consumption of the entire system $E_s$ simply as the sum of the individual contribution of each worker and the aggregator node. Therefore, we can then model $E_s$ as:

$$E_s = \Sigma_i E(i) = n * E_w + E_a,$$

where $E_w$ refers to the energy consumption of a single worker and $E_a$ to the energy consumed for the aggregation process.

By further investigating the contribution of each node in a FL scenario, for a generic worker $w$ the energy consumption $E_w$ is directly proportional to the local model complexity – referred to as $M_c(w)$ – and to the size of the local dataset—referred to as $S_d(w)$. Given that in a FL scenario all the participating nodes share the same model architecture, we have that:

$$M_c(w_1) = M_c(w_2) = \ldots = M_c(w_{n-1}) = M_c(w_n) = M_c$$

Meanwhile, as far as the aggregator node $a$ is concerned, the computation $E_a$ is usually an average-like operation with a limited corresponding computational cost. However, this operation can become expensive with a high number of tensor to be averaged. This consideration leads us to the conclusion that $E_a$ is directly proportional to the number of workers $n$.

The energy modelling scheme described above would be incomplete if the relationship between energy, latency, and accuracy would not be made explicit. In fact, the main priority of a FL pipeline is to provide the best accuracy as possible while minimising the overall latency. While the better accuracy $Acc$ usually corresponds to a higher $M_c$, the minimisation of the overall latency $\Delta Lat$ is beneficial to energy consumption, too. The optimisation of these constrains would be easy to solve in an ideal scenario with a low level of heterogeneity among participating working nodes. Theoretically speaking:

$$\begin{cases} \min(\Delta Lat) \widehat{=} \min(M_c) \\ \max(Acc) \widehat{=} \max(M_c) \\ \min(E_s) \widehat{=} \min(M_c) \end{cases} \tag{6.1}$$

Then, the minimisation of $M_c$ would be beneficial for $E_w$, $E_a$ and $\Delta Lat$. In other words, the one between $Acc$ and $M_c$ would be the only trade-off to solve to minimise energy consumption while satisfying QoS requirements.

Unfortunately, the erratic nature of fog environment brings into play a higher-dimensional space of possible solutions. Given that each worker node may be involved in more than one task and that it may have a dynamic percentage of bandwidth at its disposal over time, the selection of a specific node during the aggregation phase may be tricky and negatively impactful not only in terms of $Acc$ and $\Delta Lat$, but also for $E_s$. For this reason, the dynamic selection of workers over the time is a primary task to address in order to apply FL in real-world fog scenarios.

## 6.3 The EneA-FL Serverless Middleware

In this section, we present the architecture of our novel serverless middleware for FL in fog environments, namely EneA-FL. Inspired by the emerging Cloud Continuum paradigm [Moreschini et al., 2022], EneA-FL presents the first middleware capable of bringing together the best characteristics of serverless and fog computing, showcasing its applicability to FL settings with highly dynamic and heterogeneous devices.

### 6.3.1 Serverless Computing and Energy Awareness in Constrained Scenarios

The harmonic combination of microservice architecture and energy awareness emerges as a natural consequence of applying the serverless paradigm in fog computing scenarios. The microservice architecture, with its modular and decentralised approach, enables the development of flexible and scalable applications by breaking down complex functionalities into smaller independent services, by easing software portability over heterogeneous IoT devices at the same time. Meanwhile, energy awareness focuses on optimising resource consumption and power utilisation to achieve energy efficiency in constrained computing environments. When integrated into fog computing, which extends cloud services to the edge of the network, the serverless paradigm brings its on-demand execution and resource management capabilities. This allows applications to leverage microservices while efficiently utilising resources and minimising energy consumption in the edge and fog nodes. By combining these elements, fog scenarios can harness the benefits of microservice-based application development while maintaining energy-conscious operations. This fusion facilitates the creation of responsive, adaptable, and energy-efficient systems, making it an advantageous approach for deploying applications in dynamic and resource-constrained edge environments.

The synergy between microservice-oriented architecture and energy awareness for serverless fog computing represents a significant step towards building sustainable and high-performance applications at the edge of the network. In this context, EneA-FL is an original and relevant contribution, by providing FL developers with the first framework supporting the hybrid composition of serverless and fog computing by taking into account the resources consumed by participating nodes. By bridging the gap between serverless computing and fog scenarios, EneA-FL capitalises on the benefits of on-demand execution and resource optimisation, while extending these advantages to the edge of the network.

### 6.3.2 EneA-FL Architecture

EneA-FL consists of three main modules:

**Energon** Prometheus[1] is a well-established open-source systems monitoring and alerting toolkit originally built at SoundCloud [Rabenstein and Volz, 2015]. Inside EneA-FL, Energon is the Prometheus-compliant exporter for IoT and edge devices that keeps track of each participating device status in a completely transparent way. It is shipped to participating nodes as a container that passively collects the energy metrics of the host, independently of the local operating system and offers to the aggregator an endpoint for polling energy and network metrics. It scrapes a wide set of Linux-based microcontrollers by reading specific registries at the operating system level. Energon is published as a Pypi package[2] to help the community track down the system metrics of edge device transparently.

**Furcifer** A novel container orchestrator that handles the communication between participants nodes by offering an overlay network [Lua et al., 2005] to each node inside the cluster. This enables peer-to-peer communication between participating nodes and corresponding containers over a virtual network managed through a Docker DNS interface. In addition, it provides each participating node with a kernel-compliant container-based application.

**Magister** A policy manager that takes care of the aggregation process by choosing the aggregation policy and selecting the participant clients for each aggregation round. It is also in charge of deciding when the learning process is

[1]`https://github.com/prometheus/prometheus`
[2]`https://pypi.org/project/energon-prometheus-exporter/`

completed depending on the grade of satisfaction of the specified QoS requirements for the FL application.

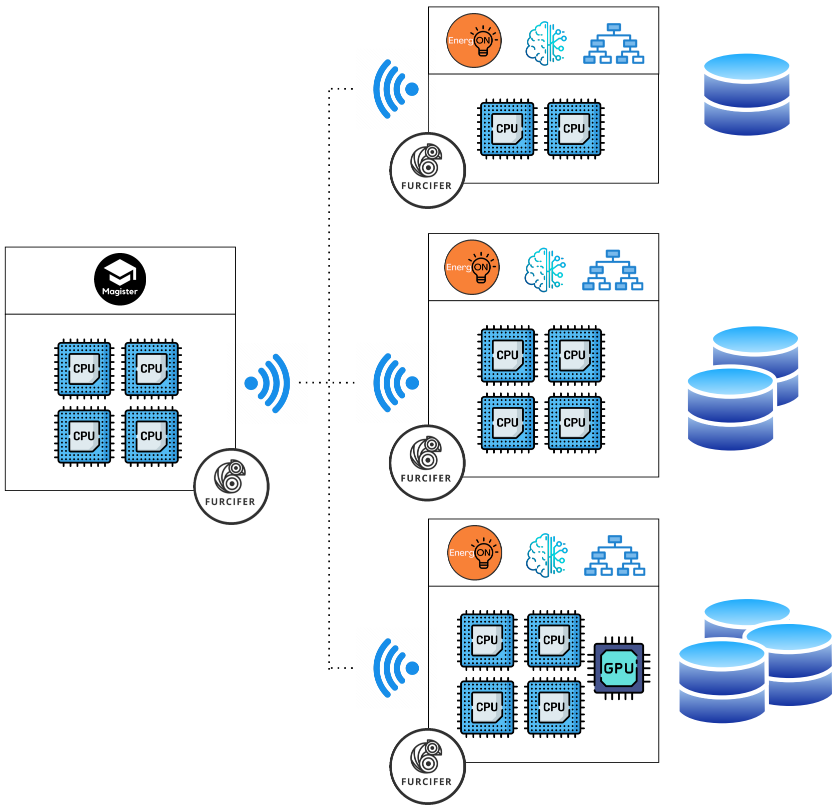


Figure 6.1: Serverless middleware architecture.

### Energon for Transparent Energy Awareness

Energon is a modular monitoring tool for IoT and edge devices. It keeps track of an extensible set of system metrics about energy consumption, network channel quality, and resource utilisation, among others. Collected metrics are compliant with the Prometheus exporting standard, which was recognised as graduated project maturity level in 2016 by Cloud Native Computing Foundation, the open-source vendor-neutral hub of cloud-native computing. Diagnostic information can be obtained by HTTP requests to the */metrics* endpoint on the IoT device. This allows both scanning with application-dependent business logic and a smooth interaction with the Prometheus ecosystem.

The ubiquitous nature of Energon allows the user to monitor critical metrics and to make real-time decisions at the application level, without generating any additional overhead for the constrained devices. When it comes to system monitoring, one of the primary objectives is to minimise the additional operations necessary to collect the desired metrics while the target applications are running. Energon has been meticulously designed to ensure complete isolation from the rest of the system. It operates independently and does not require any interaction with the running applications, thereby eliminating any potential interference or performance overhead caused by monitoring processes. Running as a separate process allows Energon to efficiently collect the desired metrics and perform monitoring operations without being tightly coupled to the application's execution and its business logic. By adopting this approach, Energon efficiently and seamlessly gathers essential metrics without impacting the performance and behaviour of the monitored applications, making it an effective and non-intrusive solution for system monitoring tasks. This helps developers focus solely on the development of their applications without the need to worry about logging the device's state for later historical analysis or real-time decision-making.

In addition to raw data monitoring and exposition, Energon can be customised to send an event when a specific condition is met. Inside our EneA-FL middleware, Energon plays the central role of keeping the orchestrator updated about the current state of the monitored nodes, thus allowing the policy manager (i.e., Magister) to select the best workers available in terms of residual energy, instantaneous power consumption, and peer-to-peer communication quality. About the querying interface, Energon wraps PromQL, the functional expression language defined by Prometheus, with easier high-level REST APIs. Those JSON-based endpoints can be further customised depending on QoS requirements. Scraped metrics can be stored locally on the orchestrator side, as done inside EneA-FL, or they can be saved on a separate database for later use—e.g., time series analysis for designing new better policies. All metrics are stored as time series data identified by a metric name and a set of key-value pairs. Sharding and federation are also possible with minimal additional settings.

### Furcifer: Container Orchestrator for IoT Devices

Furcifer is a centralised microservice manager, specialised for constrained devices and enabling communication between participant nodes. It is available in multiple versions through a set of containers for different operating systems and architectures. Furcifer is meant to fill the gap between potential policy-oriented adaptation and the heterogeneous nature of fog computing, by exploiting container orientation – and not virtual machines –, largely accepted as more suitable for these deployment environments. The introduction of an additional abstraction layer offered by containers is justified by the minimal impact on system resources and the need for higher flexibility for context adaptation purposes. In particular, when focusing on FL deployment over edge devices, we have to consider model drifting as a critical circumstance where model performance drops in an unpredicted manner. When model drifting is detected, it is very likely that a new model architecture has to be deployed on all edge devices. In a traditional setup, where all participating nodes are executing their local training at the OS level without any kind of containerisation, a manual deployment of the updated model will have to take place, with additional effort and substantial impracticality in real-world scenarios—e.g., required intervention for each participant IoT device. On the other hand, if a new model architecture has to be deployed in a containerised environment the only operation required by workers is to pull a new image from the container registry offered by Furcifer. This wraps and extends Harbor[3], a well-known open-source container registry supporting Kubernetes-based applications, for fog scenarios. In addition, since the majority of the dependencies are likely to be unchanged, only the last layer of the image will be downloaded, minimising at the same time maintenance operations and bandwidth utilisation. When a container image is built, the platform builder attempts to reuse layers from earlier builds and if a layer of an image is unchanged, then the builder picks it up from the build cache without any additional download. As a consequence, even if the container occupies more memory compared with bare metal solutions, only the device initialisation phase is affected; while real-time adaptations minimally impact the context switch latency.

Furcifer use of container images significantly simplifies the deployment and

[3]https://goharbor.io/

scaling of applications. Once an updated container image is available, it can be easily distributed to all participating edge devices through the container registry, ensuring that the latest version of the application or model architecture is seamlessly deployed across the network. This streamlined process minimises manual intervention and reduces operational overhead, making it feasible to manage a large fleet of IoT devices efficiently. Furthermore, Furcifer overcomes the lack of hardware acceleration support on constrained devices by integrating *NVIDIA Engine Runtime* for GPU-enabled IoT devices in a transparent way. The orchestrator checks automatically the availability of hardware acceleration and reserves the GPU for the FL training process before it takes place. This feature has been successfully tested inside EneA-FL on NVIDIA Jetson family boards.

In terms of storage utilisation, containers occupy a relatively larger amount of additional space compared to OS-level applications. However, it is crucial to note that the business-logic application and ML model utilise less than 5% of the total space, while the majority of storage is dedicated to user libraries. This efficient distribution means that when deploying a new model or communication strategy, only the last layer of the image needs to be shipped to the worker node, equivalent to the size of an OS-level application. Consequently, the additional storage introduced by containerisation significantly impacts initialisation time, but once the application is running, it does not exacerbate overall latency. This highlights the advantage of containers in efficiently managing application dependencies and minimising the impact on run-time performance once the containerised application is operational. In addition to providing support for real-time adaptation scenarios and dynamic resource allocation, Furcifer offers an essential advantage in terms of security and isolation. Thanks to containerisation, each service and application running on the constrained devices is encapsulated within its own container, creating a boundary that restricts its access to system resources. Isolation ensures that if one container is compromised, the security of other containers and the host system remains intact. This level of security is particularly crucial in edge computing and FL environments, known to be susceptible to attacks [Lyu et al., 2020, Agiollo et al., 2024a]. In addition, defining the communication interface at the container level provides an additional layer of security in the system. With this setup, any potential malicious node attempting to communicate with the central aggregator would face significant hurdles. Such a node would need to compromise and flawlessly replicate an existing image used in the container to establish communication.

### Magister for Fog-Oriented Context Switch Decision Making

The worker selection problem in FL has been approached from various perspectives, including faster convergence [Wu and Wang, 2022, Cho et al., 2020], higher trust level [Rjoub et al., 2022], and energy-awareness [Aloqaily et al., 2022]. However, selecting the most suitable workers in terms of both energy consumption and training speed becomes challenging in scenarios where multiple heterogeneous devices interact and multiple tasks must be executed simultaneously. Complexity arises from the diverse capabilities and resource constraints of the devices involved, making it crucial to develop innovative and efficient approaches to address the worker selection dilemma in such dynamic and diverse environments. While the identification of the most efficient device can be done in a static and self-evident way by considering the amount of FLOPS per Watt on each device, a more dynamic adaptation is required when additional constraints take place. For example, the most efficient device may not be available due to the limited battery duration or to the parallel execution of a different task with higher priority. In addition, network channel quality can also play a central role when dealing with mobile devices. For these reasons, even though in a controlled environment a static energy-preserving policy may be suitable, this is not applicable when moving to real-world scenarios. A higher-dimensional space of constraints has to be taken into account to save as much energy as possible while meeting QoS requirements. Inside EneA-FL, Magister is the module of the container orchestrator in charge of optimising the clients selection policy depending on the state of each worker in terms of system metrics and QoS satisfaction. In particular, Magister takes into account the consumed energy of participating workers, the time required to perform the local training procedure and the accuracy improvement compared with the previous training epochs. Communication is not taken into account at this level.

At the beginning of each federation round, Magister collects the clients' resource usage from Energon – using Furcifer – and selects clients accordingly. The local training procedure is not impacted by Magister, which upon the reception of local updates from the selected clients decides whether to keep the distributed training process alive or to stop it—depending on the achieved QoS metrics. Therefore, the global model aggregation process is also not affected by our solution. As a result, Magister represents a flexible client selection component which can be integrated with any custom training and aggregation mechanism for FL. To effectively identify the Magister smart selection process, here we analyse how the worker selection process affects the consumed energy, execution time and accuracy improvements. First, we define the workers available to the federation process a $\mathcal{W} = \{w_1, w_2, \dots w_N\}$, where $N$ represents the total number of workers that compose the federation. Out of these $N$ workers only a subset $\mathcal{W}_S \in \mathcal{W}$ is selected by the aggregating entity for each round of the optimisation process. The selection function used to identify $\mathcal{W}_S$ is defined as $\mathcal{S}\{\mathcal{W}\}$, and is usually considered to be a simple random selection process in most FL setups. Magister's objective is to identify some novel selection procedure $\mathcal{S}^*$ minimising the amount of energy and latency required by the federation process while maintaining the performance of the aggregated model untouched. To this extent, we first consider the selection function $\mathcal{S}$ to be dependent on the history of the federation process, accounting for smart selection of highly impactful nodes and disregarding unreliable workers.

To identify the optimal selection function, we need to take into account the dependency between the achieved performance of the aggregated global model and the amount of energy and time spent by the FL system to reach this global optimum. To this end, we here define the neural network model trained by worker $w_i$ at the $t^{th}$ optimisation step of the federation process as $\mathcal{N}_i^{(t)}$. Consequently, the amount of energy used to obtained $\mathcal{N}_i^{(t)}$ via local computation is defined as $E_c(\mathcal{N}_i^{(t)})$. The time it takes for the optimisation process in worker $w_i$ to compute $\mathcal{N}_i^{(t)}$ is written as $\tau(\mathcal{N}_i^{(t)})$. Meanwhile, at the aggregator node, the neural network model computed at the end of the $t^{th}$ federation step is defined as $\mathcal{N}_a^{(t)} = \mathcal{A}\{\mathcal{N}_i^{(t)} \; \forall i \in \mathcal{W}_s^{(t)}\}$, where $\mathcal{A}$ represents a custom aggregation function used to compute the global model from the local updates. FedAvg [McMahan et al., 2017], where $\mathcal{A}$ corresponds to the average weights of the local model, represents the most popular aggregation solution thanks to its simplicity of competitor solutions [Pillutla et al., 2022, Ma et al., 2022]. If we consider relying on a test set of examples to compute the goodness of the obtained aggregated global model, we can define its performance – e.g., accuracy, f1-score, etc. – as $\mathcal{P}(\mathcal{N}_a^{(t)})$. The overall aim of any FL process is to increase the performance of the aggregated model $\mathcal{P}(\mathcal{N}_a^{(t)})$ at each federation step $t$, so that $\mathcal{P}(\mathcal{N}_a^{(t+1)}) \geq \mathcal{P}(\mathcal{N}_a^{(t)})$ and the global optimum is achieved at the end of the federated optimisation process.

To account for smart selection of the federation workers, we here consider relying on a selection function that aims at maximising the performance of the aggregated model, while minimising the energy and time spent. Therefore, we would be ideally able to define a new selection function $\mathcal{S}$ that selects the next set of workers $\mathcal{W}_S^{(t+1)}$, such that:

$$\mathcal{W}_S^{(t+1)} \; s.t. \begin{cases} \max\left(\mathcal{P}\left(\mathcal{N}_a^{(t+1)}\right)\right) \\ \min\left(\sum_{\mathcal{W}_S^{(t+1)}} E\left(\mathcal{N}_i^{(t+1)}\right)\right) \\ \min\left(\sum_{\mathcal{W}_S^{(t+1)}} \tau\left(\mathcal{N}_i^{(t+1)}\right)\right) \end{cases} \tag{6.2}$$

However, while desirable, such a setup cannot be solved directly, as the nodes that mostly affect the most the computation of the aggregated model $\mathcal{N}_a^{(t+1)}$ cannot be known *a priori*. Therefore, we here rely on the assumption that the setup can be computed *a posteriori*. Thus, we aim at identifying the set of nodes that affect mostly positively the performance of the aggregated model for previous steps $t, t-1, \dots, 1$, while consuming less resources – i.e., energy and time –, and rely on these nodes also for aggregation step $t+1$. Therefore, the selection process becomes a combinatorial search problem, where we aim at identifying the subset of best workers $\hat{\mathcal{W}}_S^{(t)}$, such that their combination achieves high performance while keeping both energy and time requirements under control.

As it represents a combinatorial optimisation problem, finding the optimal set of workers $\hat{\mathcal{W}}_S^{(t)}$ to maximise performance and minimise resources is not scalable when the number of workers increases. Therefore, relying on such an optimal procedure would lead to huge computational waste from the aggregator node end. While we aim at minimising the amount of resources spent at workers level, shifting the computational burden to the aggregator node would not represent a feasible solution. Thus, we here consider relying on a simplified approach that does not require the solution of the combinatorial optimisation problem. To this end, we here define an effectiveness measure aiming at identifying how much the local model update from a worker $w_i$ impacts the performance and resource consumption of the whole federation setup.

Accordingly, we define the effectiveness metric as:

$$\mathcal{E} = \alpha \cdot \frac{E\left(\mathcal{N}_i^{(t)}\right)}{\max\{E^{(t)}\}} + (1-\alpha) \cdot \frac{\tau\left(\mathcal{N}_i^{(t)}\right)}{max\{\tau^{(t)}\}} - \beta \cdot \Delta\left(\mathcal{P}\left(\mathcal{N}_a^{(t)}\right), \mathcal{P}\left(\mathcal{N}_{a\ominus i}^{(t)}\right)\right), \quad (6.3)$$

where $\max\{E^{(t)}\}$ represents the maximum energy spent by any worker at step $t$, namely $\max\{E^{(t)}\} = \max\{E\left(\mathcal{N}_j^{(t)}\right) \forall j\}$. Similarly, $\max\{\tau^{(t)}\}$ represents the maximum time taken by any worker at step $t$, namely $\max\{\tau^{(t)}\} = \max\{\tau\left(\mathcal{N}_j^{(t)}\right) \forall j\}$. Meanwhile, $\Delta$ represents the difference in performance between the aggregated model $\mathcal{N}_a^{(t)}$ and the model $\mathcal{N}_{a\ominus i}^{(t)}$ obtained aggregating all workers updates except $w_i$. Finally, $\alpha$ and $\beta$ represent the hyperparameters that identify the trade-off between the relevance of energy consumption, time requirements, and the performance improvement achieved. Intuitively, a local worker $w_i$ whose model has a high impact and is obtained by spending little resources ensures a small score. On the other hand, workers whose models have little-to-no impact on the model performance and/or are obtained consuming a vast amount of resources result in high scores. To account for devices that are not capable of sending an update to the aggregation entity, we set a handicap value $h$ to the effectiveness metric of those devices that do not guarantee an update when selected, namely:

$$\mathcal{E}\left(w_i^{(t)}\right) = \begin{cases} \mathcal{E} & \text{if received update} \\ h & \text{otherwise.} \end{cases} \quad (6.4)$$

Finally, to take into account the reputation history of workers, we consider measuring the reputation score of each worker as its average effectiveness score, namely:

$$\mathcal{R}\left(w_i^{(t)}\right) = \frac{\sum_{l=1}^{t} \mathcal{E}\left(w_i^{(l)}\right)}{t}. \quad (6.5)$$

Considering the reputation score of Equation (6.5), we can now redefine the problem of identifying optimal workers as the process of identifying

$$\hat{\mathcal{W}}_S^{(t)} \; s.t. \; \min\left(\mathcal{R}\left(w_i^{(t)}\right)\right). \tag{6.6}$$

Therefore, in our novel workers selection process, the workers selected to run the optimisation procedure for the next aggregation step $t+1$ are

$$\mathcal{W}_S^{(t+1)} = \left\{\hat{\mathcal{W}}_S^{(t)} \; s.t. \; \min\left(\mathcal{R}\left(w_i^{(t)}\right)\right) \oplus \mathcal{S}_r\left(\mathcal{W} - \hat{\mathcal{W}}_S^{(t)}\right)\right\}, \tag{6.7}$$

where $\hat{\mathcal{W}}_S^{(t)}$ represents the set workers with optimal performance from step $t$ – to account for incentivisation of highly performing nodes – and $\mathcal{S}_r\left(\mathcal{W} - \hat{\mathcal{W}}_S^{(t)}\right)$ represents a random sampling – i.e., $\mathcal{S}_r$ – of the remaining non-optimal nodes—i.e., $\mathcal{W} - \hat{\mathcal{W}}_S^{(t)}$. The selection of a set of random non-optimal nodes is necessary for the federation process to take into account nodes that were not present in previous aggregation procedures, which may still impact positively the global model. The cardinality of the set of optimal nodes and randomly selected ones accounts for the trade-off between static behaviour over time and workers coverage. Thus, we add a third hyperparameter value $k$ that balances the cardinality of optimal nodes and randomly selected ones. In particular, if we identify with $O$ the cardinality of $\hat{\mathcal{W}}_S^{(t)}$ – i.e., the number of optimal nodes selected from one iteration to the next – and with $R$ the cardinality of $\mathcal{S}_r\left(\mathcal{W} - \hat{\mathcal{W}}_S^{(t)}\right)$—i.e., the randomly sampled non-optimal nodes, and with $n_r$ the number of workers to be selected at each round, we obtain $O = \lfloor k \cdot n_r \rfloor$ and $R = \lceil (1-k) \cdot n_r \rceil$. Taking into account energy consumption, time, and reached performance level of each client in the federation – Equation (6.3) –, EneA-FL handles device heterogeneity in terms of computational capabilities, computational efficiency, and data availability.

## 6.4 Experiments

As already stated, our primary objective is to address the pressing need for more energy-efficient and sustainable machine learning models, especially in the era of ubiquitous data and resource constraints. By leveraging the collaborative power of FL, we aim at demonstrating significant energy savings without compromising model performance.

### 6.4.1 FL Training Process and Model Complexity

The experiments are conducted on a real networking testbed consisting of heterogeneous edge devices, which mimics a real-world deployment environment with diverse computational capabilities. We use representative datasets from LEAF[4], which includes different use case data coming from Computer Vision and Natural Language Processing areas. From LEAF we select two datasets: MNIST and Sent140 to test the feasibility of our energy-aware Federated Learning selection on multiple tasks. While MNIST is the distributed version of the well-known MNIST dataset for image classification, Sent140 consists of a corpus of 1,600,000 tweets extracted using the Twitter API for sentiment analysis. Finally, we also employ the N-BaIoT dataset [Meidan et al., 2018] to include a dataset containing realistic traffic data gathered from 9 commercial IoT devices authentically infected by Mirai and BASHLITE. Here, the malicious traffic is divided into 10 different attack classes (e.g., network scanning and firmware) plus 1 benign class. On all datasets, we implement a small neural network composed either of a few blocks of convolution operations or linear ones. The corresponding model complexity is evaluated in terms of both MAC (Multiply-Accumulate Operations) and the total number of parameters, reported in Table 6.2. The overall model complexity of the model for image classification on MNIST is more than three times the one for sentiment analysis over Sent140 and 1000 times the one for attack identification in N-BaIoT.

| | MAC | n params |
|---|---|---|
| Sent140 | 16.106 M | 2.006 M |
| MNIST | 5.962 M | 2.278 M |
| N-BaIoT | 4.384 K | 4.491 K |

Table 6.2: Complexity of leveraged NN models.

[4] `https://leaf.cmu.edu`

### 6.4.2 Container vs Operating System FL Training Execution

Before delving into finer details, we perform a comparison between the containerised FL training application and the execution at operating system level. We execute EneA-FL with and without containerisation to estimate the additional overhead brought by a higher level of abstraction. The training process consists of five local epochs for each dataset where both os-level and container training execution follow the same node selection policy. For this reason, the device is selected or ignored depending on the current state of the federation. This unveils noteworthy similarities in their energy consumption, with a minor difference of about 5% in terms of training time. As can be seen in Figure 6.2, both executions demonstrate comparable energy usage, indicating that the containerisation process does not significantly impact overall power consumption during training tasks. However, the containerised environment exhibits a slight delay, with training times being approximately 1-2% slower compared to the operating system level environment. As it can be seen, the most powerful device, the Jetson AGX Orin, exhibits significant variability and dispersion in its values; while the two most efficient devices, the Jetson Nano and the Jetson Xavier tend to manifest a consistent and steady behaviour, in particular when equipped with hardware acceleration. Generally speaking, over all the devices considered, the usage of GPU seemingly leads to lower data sparsity and a more predictable trend. The overall marginal disparity in terms of startup required time proves the minimal overhead versus the benefits of GPU-enabled containerisation. The improved portability, isolation, and ease of deployment remain highly advantageous, making it a viable and efficient choice for FL training tasks in fog environments.

### 6.4.3 Container Startup Latency Assessment

In the dynamic landscape of IoT and serverless architecture, a crucial aspect of consideration revolves around the comparison of container startup latencies when utilising IoT devices with and without GPU hardware acceleration. Serverless computing, which enables on-demand execution of functions without the need for

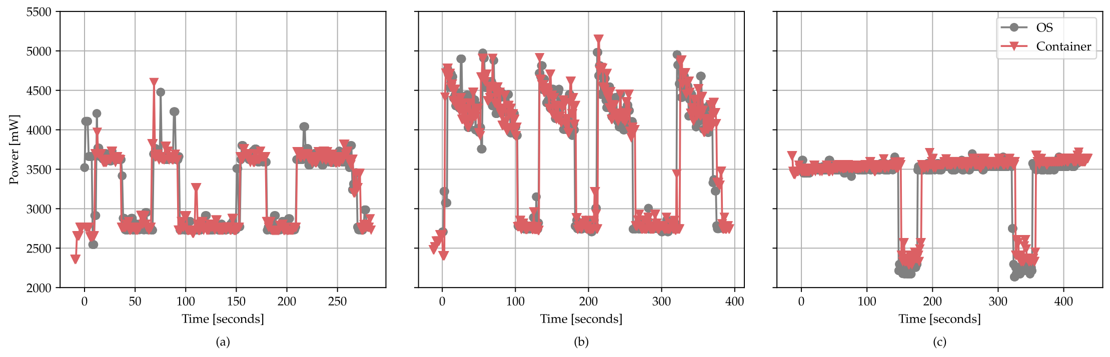


Figure 6.2: Time execution and energy consumption comparison on GPU enabled Jetson Nano over 5 training epochs.
(a) On Sent140 dataset. (b) On MNIST dataset. (c) On N-BaIoT dataset.

server management, is particularly attractive in IoT environments where data is generated at the network's edge. Containers, acting as self-contained units of application code and dependencies, can significantly impact the responsiveness of IoT applications during initialisation and deployment. By assessing and contrasting the startup latencies with and without GPU hardware acceleration, we gain valuable insights into the potential performance gains and resource optimisation for serverless architectures in the context of IoT applications. This section analyses the minimal additional overhead given by containerisation during training execution. In particular, we evaluated the startup latency for all the devices taking into account over 100 startup cycles.

The data presented in Figure 6.3 demonstrates the startup latencies of various configured devices, revealing that the presence of hardware acceleration does not result in significantly higher delays. On average, the additional latency incurred with hardware acceleration is merely 5-10%. Moreover, across all devices, the observed latencies range between 200 ms and 300 ms, underscoring the minimal overhead introduced by containerisation techniques on the overall training execution. These findings highlight the efficiency and effectiveness of utilising hardware-accelerated containers, as they offer near-instantaneous startup times while providing substantial benefits in terms of flexibility and modularity during training processes.

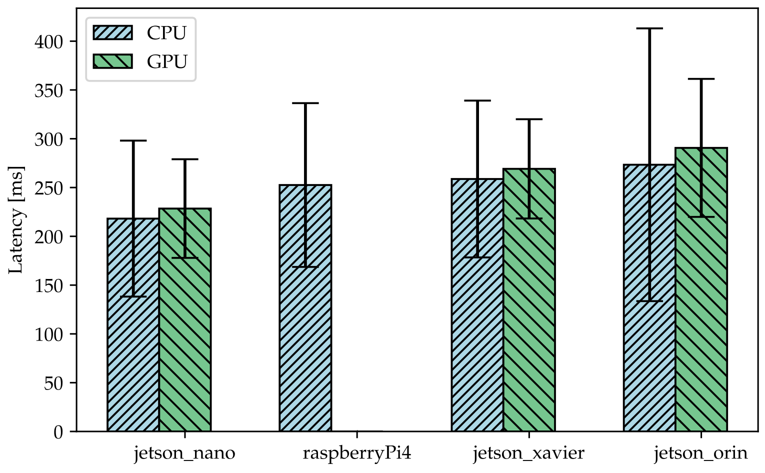


Figure 6.3: Time startup comparison of CPU containers vs. GPU-enabled containers.

### 6.4.4 Energy Consumption and Training Execution Time on IoT Devices

To evaluate the performance and the effectiveness of energy optimisation, we used the following metrics: model accuracy, communication rounds and clock time required for convergence, and total energy consumption. We first evaluate the energy consumption of a wide range of IoT devices commonly used in fog and edge scenarios. In particular, we measure the energy consumption of our models on a Raspberry Pi model 4, a Jetson Nano developer kit, a Jetson Xavier NX board, and a Jetson AGX Orin developer kit. In addition, each of the Jetson boards has been tested with and without GPU support to check the energy cost of hardware acceleration in IoT training processes. Figure 6.4 illustrates the experiments performed to measure the energy consumption of all devices tested in five training epochs. The Jetson AGX Orin developer kit is the fastest, but it is also the one with the higher instantaneous power consumption. On the contrary, the slowest device is the Jetson Nano without GPU, while the hardware-accelerated version of the device appears the be the best compromise in terms of energy consumption and training time.

Surprisingly, the use of low-resource devices, like the well-known Raspberry Pi model 4, proves to be unfavourable in terms of both overall energy consumption

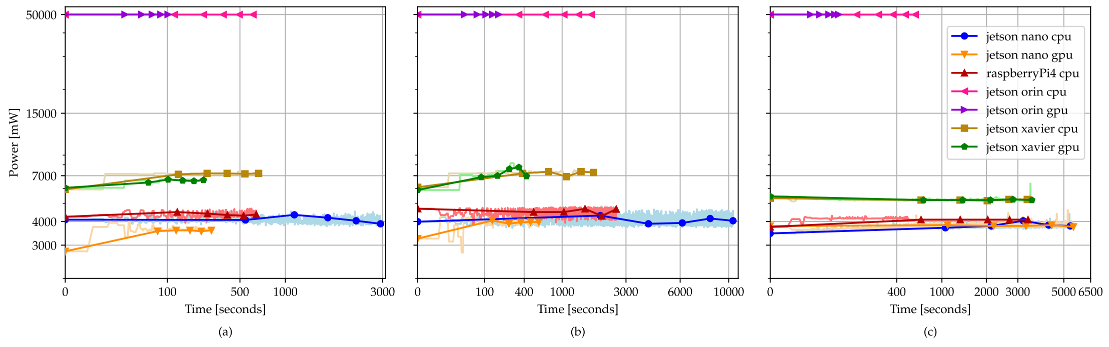


Figure 6.4: Time execution and energy consumption on considered devices over 5 training epochs.
(a) On Sent140 dataset. (b) On MNIST dataset. (c) On N-BaIoT dataset.

and training execution time. When considering the total energy cost over the whole training, employing a higher-powered device may lead to lower energy consumption. Additionally, the experiments underscore the remarkable reduction in training execution time achieved with higher-tier devices, exemplified by the Jetson AGX Orin, albeit at the expense of consuming approximately 10 times more energy compared to other devices. On the other hand, the Raspberry Pi 4 requires over five times more time for training execution compared to the majority of other constrained devices. Notably, in the first two datasets, the difference in training time between GPU-enabled devices and their CPU-limited versions is significant. However, on the N-BaIoT dataset, this difference is minimal. As expected, in this case, the lower complexity of the model does not benefit significantly from parallel computation.

Figure 6.5 showcases the energy efficiency of the examined devices, determined by computing the amount of Joules using Simpson's rule for numerical approximation integration [Tallarida and Murray, 1987]. This method allows us to precisely evaluate the energy consumption of each device and compare their efficiency in performing the given task. By employing Simpson's rule, we gain valuable insights into the energy performance of the devices, providing a comprehensive understanding of their capabilities in optimising power utilisation during the integration process. According to the results, the Jetson AGX Orin emerges as the least efficient device in terms of energy consumption, indicating higher energy usage during the inte-

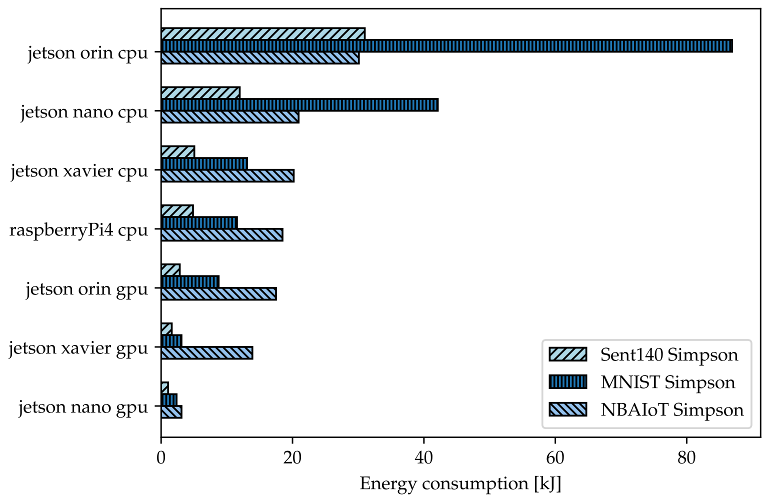


Figure 6.5: Ranking of overall energy consumption over considered devices.

gration process. On the other hand, the Jetson Nano and Jetson Xavier stand out as more energy-efficient options, as they exhibit lower power consumption relative to their execution times.

### 6.4.5 EneA-FL Performance Results and Discussion

In this section, we present the results obtained during the testing phase of EneA-FL framework. More in detail, we first describe the setup used for the analysis of our framework in Section 6.4.5. In Section 6.4.6 we perform an ablation study to identify EneA-FL best hyperparameters. We then study the performance of the EneA-FL framework in terms of reached accuracy when a set of resources budgets are considered (see Section 6.4.7). Sections 6.4.8 and 6.4.9 analyse respectively the impact of the number of selected devices per round and the inactive devices on the performance of EneA-FL. Finally, in Section 6.4.10 we study the flexibility of EneA-FL over variations of the distribution of device types in the federation.

**Experiments setup** To test the proposed EneA-FL framework we implement a federated learning simulation tool that takes into account the testbed results obtained in Section 6.4 to emulate the deployment of containerised workers on each of the seven selected devices. More in detail, we implement the proposed framework using PyTorch[5] for the definition of models and their learning process and rely on FedAvg [McMahan et al., 2017] as the aggregation algorithm during the FL training process. Through all our experiments, we consider a federation network composed of 100 different workers and for each worker we identify the probability of it being one of the seven device types considered in Section 6.4. We refer to the distribution of these probabilities as the distribution of device types and, apart from the experiments in Section 6.4.10, we build the federation using a uniform distribution of device types. In particular, since we consider 7 different type of devices – see Section 6.4.4 – we deploy a federation composed by 14.29% of each device type. Thus we obtain a setup where each device type covers one-seventh of the whole federation hardware.

The energy consumption and execution time of the local training process of each device is emulated by the worker belonging to the federation to keep track of the total energy consumption and total execution time. More in detail, for each training step of the worker local training process, we perform a sampling from the normal distribution of the energy consumption and time requirements that characterise the device type at hand. The characteristic normal distribution is extracted from the testbed measurements obtained in Section 6.4, thus representing faithfully the real-world energy consumption and latency of the selected devices.

The implemented framework also allows for a flexible definition of several additional options, such as device type distribution, device lifetime, data distribution, QoS target definition and many more. The device lifetime is modelled as an exponential random variable that models the number of available federation rounds and is used in Section 6.4.9 to assess the impact of device discharge on EneA-FL. Meanwhile, QoS targets definition supports various types of resource budgets – investigated in Section 6.4.7 – and target performance. Finally, if not diversely specified, throughout our experiments we consider uniformly distributed data samples over all workers belonging to the federation. For maximum results reproducibility, the developed code and documentation for all the experiments presented in this article are published on a dedicated EneA-FL GitHub repository[6].

[5] `https://pytorch.org`
[6] `https://github.com/AndAgio/EneA-FL`

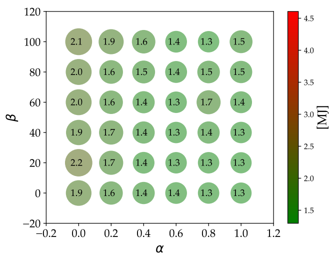
### 6.4.6 Ablation study

EneA-FL relies on its Magister component for optimisation of the device selection process throughout federation learning. In this context, the energy-aware policy defined in Section 6.3.2 takes into account a mixture of energy consumption, latency and achieved performance to promote energy-effective workers. To balance these three components EneA-FL relies on three hyperparameters to select the devices of each federation round, namely: *(i)* $\alpha$, balancing energy consumed and execution time; *(ii)* $\beta$, to account for local models' accuracy; and *(iii)* $k$, adding a randomness component to the selection process. Given the relevance of these hyperparameters in EneA-FL we here propose an ablation study aiming to identify the best Magister setup.

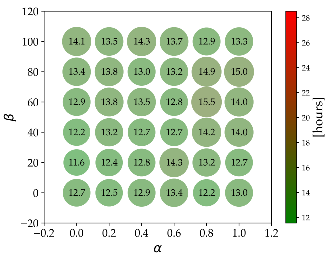
#### $\alpha$ and $\beta$ values

The energy effective metric proposed in Equation (6.3) relies on $\alpha$ and $\beta$ hyperparameters to weigh the relevance of minimal power consumption, training execution time and achieved accuracy. By leveraging the values of $\alpha$ and $\beta$, EneA-FL can balance the trade-off between energy consumption over training execution time. Therefore, it represents a significant aspect of the identification of the best $\alpha$ and $\beta$ setup for running EneA-FL. Indeed, setting these hyperparameters is not a straightforward process, as $\alpha$ is the component setting a trade-off between energy and time, while $\beta$ represents the parameter tuning the relevance of the achieved accuracy. More in detail, higher values of $\alpha$ define an optimisation process where the energy is considered more valuable than the execution time of the learning step, thus giving higher priority to very efficient – possibly slow – devices. Meanwhile, smaller $\alpha$ prioritises raw optimisation speed over energy efficiency, valuing faster devices that may consume more energy. On the other hand, $\beta$ represents ideally the relevance given to the reached accuracy of local models. Therefore, higher $\beta$ allows Magister to promote devices whose model reaches higher accuracy improvements, without regarding their energy consumption and latency. Meanwhile, smaller $\beta$ allows Magister to focus solely on optimisation energy and latency, disregarding the reached accuracy.

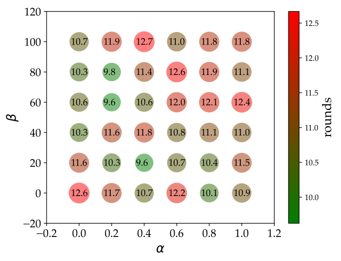
To analyse the impact of $\alpha$ and $\beta$ values on the federation optimisation we



(a)



(b)



(c)

Figure 6.6: Impact of ($\alpha$, $\beta$) values on energy consumption (a), time execution (b) and rounds required to converge (c) on MNIST dataset.

let their values vary between 0 and 1, and 0 and 100 respectively. We perform 10 federation optimisation experiments for each $\alpha$ and $\beta$ combination, setting the number of federation rounds to 30. Training is performed over the MNIST dataset. For each experiment, we keep track of the *(i)* overall energy consumption; *(ii)* total execution time; and *(iii)* number of rounds required to converge. Here, convergence is considered to be achieved when the global model achieves 97% of accuracy on the test set.

Figure 6.6 shows the average values for energy, time and convergence achieved over the 10 experiments of each $\alpha$ and $\beta$ setup. More in detail, Figure 6.6a presents the average energy consumption. Here, it is possible to notice that smaller $\alpha$ values lead to higher energy consumption, while higher $\alpha$ leads to smaller energy values, confirming the goodness of the proposed policy. Interestingly, selecting $\alpha = 0$ can lead to almost doubled energy consumption over $\alpha = 1$. Meanwhile, concerning $\beta$, it is possible to notice a slight increment of consumed energy for higher $\beta$ values. This behaviour is expected, as higher $\beta$ values force Magister to promote devices depending mostly on their models' reached accuracy, discarding their energy efficiency.

Figure 6.6b presents the average time execution. Here, it is possible to notice that smaller $\alpha$ values lead to smaller latency, while higher $\alpha$ leads to time increments. Meanwhile, $\beta$'s impact is less evident, probably due to the fact that selecting devices with high accuracy improvements usually leads to improved convergence time. Differently from the energy analysis, the difference in execution time is confined as it is possible to save at most only a couple of hours when

| Approach | Energy | Time | Rounds |
|---|---|---|---|
| Standard FL | 4.6 MJ | 28.5 h | 9.7 |
| OORT | 1.8 MJ | 13.2 h | 11.2 |
| OORTv2 | 2.2 MJ | 13.8 h | 13.8 |
| EneA-FL worst | 2.1 MJ | 14.1 h | 10.7 |
| EneA-FL best | 1.3 MJ | 12.7 h | 10.8 |

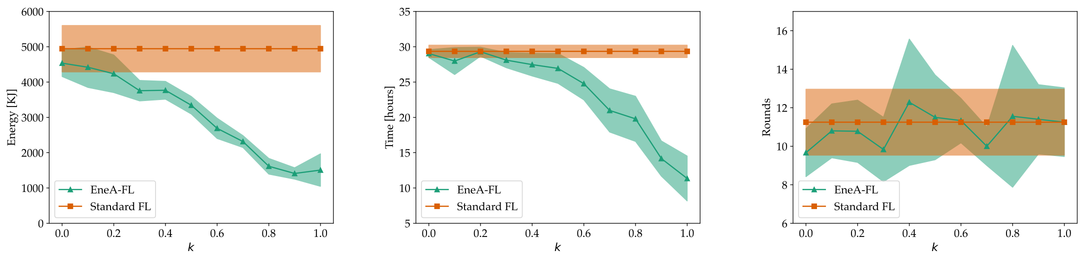
Table 6.3: Comparison of energy consumption, time execution and rounds required to converge on MNIST dataset between different worker selection approaches and the best and worst ($\alpha$, $\beta$) EneA-FL setups. The EneA-FL best setup is obtained for $\alpha = 0.6$, $\beta = 40$, while its worst setup is obtained for $\alpha = 0$ and $\beta = 100$. For each metric, we highlight in green the best approach.

selecting properly $\alpha$ and $\beta$.

Figure 6.6c presents the average number of rounds required to converge to 97% of accuracy. Here, it is possible to notice that $\alpha$ and $\beta$ values do not impact clearly the convergence time. Indeed, convergence time is mostly influenced by the quality of the data that each device holds rather than its efficiency.

Given the results of Figure 6.6, it is possible to select the best $\alpha$ and $\beta$ for EneA-FL. Throughout our experiments, we select $\alpha = 0.6$ and $\beta = 40$ to be the best EneA-FL setup, as it allows to consume only 1.3 MJ of energy, while requiring 12.7 hours to complete 30 federation rounds and converge in 10.8 steps. Finally, these results show that Magister can define hybrid systems that adapt to the scenario at hand depending on the chosen $\alpha$ and $\beta$ values. Indeed, in a federation setup where energy represents the most relevant component we suggest selecting high $\alpha$ and small $\beta$ values. Meanwhile, when the latency is the most valuable aspect we suggest selecting small $\alpha$. Finally, when both energy and latency are relevant, Magister can reach improved efficiency by selecting a middle ground between the previous two options.

Having obtained the best $\alpha$ and $\beta$ setup, we compare our proposed solution with the standard FL setup – which relies on random node selection – and the OORT [Lai et al., 2021] and its extended version (OORTv2) [Arouj and Abdelmoniem, 2022] in Table 6.3. For a fair comparison, we compare both the best and worst Magister setups. The best setup is achieved for $\alpha = 0.6$ and $\beta = 40$,



(a) (b) (c)

Figure 6.7: Impact of $k$ values on energy consumption (a), time execution (b) and rounds required to converge (c) on MNIST dataset.

while the worst counterpart is obtained by selecting $\alpha = 0$ and $\beta = 100$. The results highlight that Magister offers a favourable solution for minimising energy consumption and execution time against all baselines. Meanwhile, Magister does not impact the convergence time—i.e., the number of rounds required to reach 97% of accuracy. The best Magister setup consumes 3.5 times less energy than its standard FL counterpart and 30% less energy than OORT which represents the best baseline. These findings highlight the significant advantages of the Magister approach, offering substantial energy savings and improved training efficiency when compared to standard FL, making it a highly promising and effective solution for FL scenarios.

### $k$ values

As expressed in Equation (6.7), Magister relies on a hybrid worker selection strategy, where a portion of workers is selected from the best workers of previous rounds and the remaining portion is picked randomly from the devices available. The random component is used to allow Magister to explore the effectiveness of federation devices throughout the federation history and avoid getting stuck in selecting always the same devices. The balance between the number of workers selected via the effectiveness metric and the number of randomly selected workers is given by the parameter $k$ in Equation (6.7). To assess the relevance of $k$ we compare EneA-FL against a standard FL setup, while varying $k$, aiming at identifying the best $k$ for deploying EneA-FL.

Figure 6.7 shows the results for the ablation study on the values of $k$. We let $k$ vary between 0 and 1. Here, $k = 0$ represents the setup where EneA-FL and random node selection are equivalent, as all devices are selected randomly also in EneA-FL. Meanwhile, $k = 1$ represents the EneA-FL setup where workers are only selected based on their energy effectiveness metric and no worker is selected randomly. Figures 6.7a to 6.7c highlight that higher values of $k$ allow the federation to reach smaller energy consumption and execution time. Meanwhile, a similar convergence time is achieved for all $k$ setups. Interestingly, the results show a high level of energy and time savings even for small values of $k$—e.g., $k = 0.5$. As a result, Magister shows that it is possible to save a high amount of energy and time even when a large part of the workers are selected randomly. From the obtained results, it is possible to identify the best $k$ value, that we select to be equal to 0.8. We avoid considering $k = 1$ as we want to allow Magister to explore all federation devices over its history. In its best setup, Magister reaches 3 times less energy consumption, while requiring almost 2 times less time to complete the 30 federation rounds.

### 6.4.7 Resources Budget

In real-world scenarios, one desideratum of distributed optimisation scenarios such as FL is to identify efficient processes over limited resources budgets. In particular, we here consider scenarios where the federation network has a pre-defined budget of energy or time that can be invested into the FL optimisation. Indeed, when considering FL setups in the real world, it is common to impose an upper bound on the time that the FL can take for optimising the model at hand. Similar requirements can be expressed for the total amount of energy that the FL process should take to reduce costs and have a restricted environmental impact. Therefore, in order to assess the performance of EneA-FL over limited resource budgets, we compare its performance – i.e., accuracy – against the available baselines when we set energy and time limits (budgets) over the MNIST, Sent140 and N-BaIoT datasets.

Table 6.4 shows the average performance – over 10 iterations – achieved by the federation when training using the given energy budget. The EneA-FL pol-

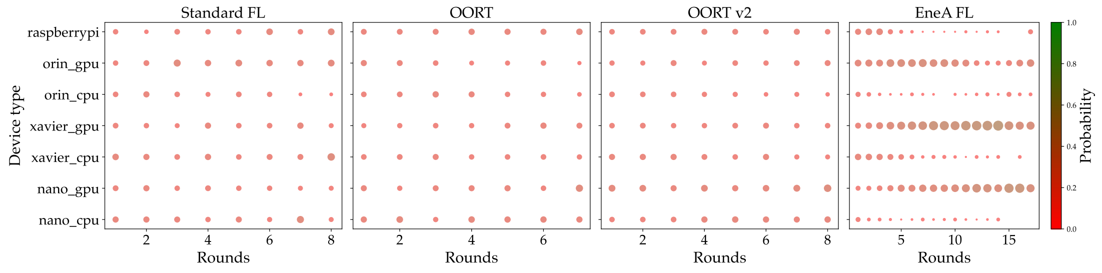


Figure 6.8: Selected devices distribution over rounds for various client selection policies when federation is optimised given an energy budget of 1 MJ on the MNIST dataset.

| Dataset | Budget | Standard FL | OORT | OORTv2 | EneA-FL |
|---|---|---|---|---|---|
| MNIST | 1 MJ | 95.3% | 95.4% | 95.7% | 97.4% |
| Sent140 | 500 KJ | 72.1% | 72.3% | 72.3% | 72.5% |
| NBIoT | 50 KJ | 87.8% | 87.5% | 86.7% | 89.3% |

Table 6.4: Average accuracy on each dataset for EneA-FL against selected baselines when the federation is optimised given an energy budget. For each dataset, we highlight in green the best approach.

icy vastly outperforms the selected baselines, achieving higher accuracy over all datasets and showing a statistically significant improvement over the experiment iterations. This is due to the higher longevity achieved by the federation when implementing the energy management policy—i.e., Magister. Indeed, the number of optimisation rounds that the federation can survive with the given energy budget is on average more than doubled with respect to the standard FL policy. This behaviour can be seen in Figure 6.8, where we plot the average selected devices distribution over federation rounds on the MNIST training. On the $x$ axis it is possible to notice that the baselines survive at most 8 federation rounds, while

| Dataset | Budget | Standard FL | OORT | OORTv2 | EneA-FL |
|---|---|---|---|---|---|
| MNIST | 8h | 96.4% | 96.2% | 96.1% | 97.0% |
| Sent140 | 4h | 72.3% | 72.5% | 72.4% | 72.9% |
| NBIoT | 10m | 87.9% | 88.0% | 87.8% | 89.9% |

Table 6.5: Average accuracy on each dataset for EneA-FL against selected baselines when the federation is optimised given a time budget. For each dataset, we highlight in green the best approach.

EneA-FL reaches up to 17 optimisation rounds.

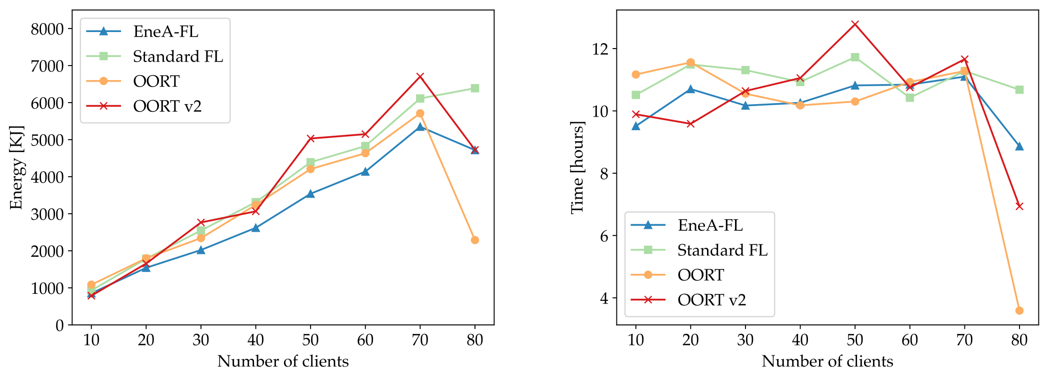
To assess the effectiveness of our policy in selecting the most efficient devices, we study the effect of the Magister policy on the device selection distribution. We plot the average selected devices distribution over the optimisation rounds in Figure 6.8—one box for each client selection policy. The size and the colour of each blob represent the probability of each device type being selected for a specific round of the federation. Interestingly, all baselines behave similarly to the random context, where every device has the same uniformly distributed probability of being selected. Meanwhile, for EneA-FL the probability of selecting the Jetson Nano with GPU and the Jetson Xavier with GPU increases over the number of rounds, reaching the point where these devices represent almost the totality of devices selected for the last time stages of the federation. Similarly, the probability of selecting very slow and energy-hungry devices such as the Raspberry Pi v4 decreases over time, enabling efficiency improvements of FL. This behaviour highlights the quality of our policy for selecting efficient devices over their counterparts.

Table 6.5 shows the performance improvements obtained by EneA-FL against the available worker selection approaches when time budgets are considered.

### 6.4.8 Number of Clients

Given the Magister focus on client selection, we here analyse the impact of the number of selected devices per round on Magister effectiveness. The number of workers selected at each federation step represents a fundamental parameter of FL settings, as it defines the broadness of data gathered at each optimisation step. Selecting a higher amount of workers for each round leads to a higher number of updates for each round, but also to increased energy and latency. Therefore, identifying smart workers selection policies such as Magister represents a key aspect to efficiently optimise the federation while avoiding selecting a high number of clients for each round.

To study the impact of the number of clients per round on the federation, we consider selecting $n$ workers per round and let $n$ vary between 10 and 80. Overall the federation consists of 100 devices, therefore $n$ ranges from selecting only a small portion of the federation workers to almost selecting them all. We perform



(a) (b)

Figure 6.9: Impact of the number of selected clients per round on energy consumption (a), and time execution (b) on MNIST dataset. Here, we consider training the federation to reach a target accuracy of 97%.

10 federation optimisation experiments for each $n$, letting the federation converge to 97% accuracy when the optimisation process is stopped. Training is performed over the MNIST dataset and for each experiment we keep track of the *(i)* overall energy consumption; and *(ii)* total execution time.

Figure 6.9 shows the results of our experiments. More in detail, Figure 6.9a presents the results concerning energy consumption. Here, it is relevant to notice how Magister outperforms all selection baselines for almost every value of $n$. For small $n$ values – e.g. $n = 30$, $n = 40$ – EneA-FL requires almost 30% less amount of energy to converge to the same accuracy level. Meanwhile, for higher $n$s the difference is less evident. As the number of workers selected increases, Magister ends up selecting both efficient and inefficient devices – since the number of efficient devices is limited –, therefore decreasing the advantage of smart device selection. Concerning the total execution time (Figure 6.9b), EneA-FL seems to perform similarly to the available client selection baselines, while outperforming the standard FL setup.

### 6.4.9 Device Discharge

One of the issues to face when dealing with FL scenarios in edge and IoT domains is linked with the lifetime span of edge devices. In particular, in real-world setups, it is common for the edge devices to be battery-powered. Therefore, the devices belonging to the federation could suffer discharging issues, leaving the optimisation process unattended. To study if – and to what extent – the proposed energy management policy can help in these scenarios, we here study the effect of device discharging on EneA-FL and compare it to the standard FL scenario.

We model the discharging process of devices as an exponentially distributed event over the federation rounds that a device can complete and set its average value to be equal to a random value between 1 and 5. Therefore, in this experimental setup, each device belonging to the federation is capable of completing a variable number of federation rounds, after which it discharges completely and stops sending updates if selected. To account for the discharge process, the proposed energy management mechanism relies on a handicap $h$ value that is assigned to the energy effectiveness score of each worker whenever it does not provide an update to the aggregator (recall from Section 6.3.2). In our experiments, we set $h = 5$ and measure the average number of discharged selected workers for each round. The measurements are obtained over ten iterations of the experiments to account for process variability.

Figure 6.10 shows the percentage of dead selected devices over the federation round, comparing EneA-FL with the selected baselines. As expected, at first the number of discharged devices is zero, as it is considered impossible for devices to join the federation when already dead. The number of selected devices that can not produce updates increases over the first few steps of the federation process for both approaches, as it is at this point that a few devices start to drain their batteries. However, after a few rounds, the behaviours of EneA-FL and the baselines diverge. Indeed, all baselines keep selecting devices unable to produce updates, reaching peaks of up to 80% of selected workers with drained batteries. Meanwhile, the energy management approach kicks in for EneA-FL, showing a stable percentage of dead devices selection. EneA-FL allows for this percentage to stay always below 40%. This is thanks to the handicap given to devices that do not produce any update. Moreover, it is also possible to notice that the energy management mechanism allows the federation to converge more rapidly, as the updates are more consistently produced by workers. Finally, it is relevant to notice that although EneA-FL assigns handicap to inactive devices, it is still impossible to avoid completely selecting drained devices. Indeed, part of the EneA-FL selection process is random. Moreover, it is also favourable to avoid disregarding completely the drained devices, as they could become available once again if charged during the federation optimisation process.

### 6.4.10 Impact of Device Type Heterogeneity

When considering real-world scenarios where FL frameworks can be deployed, one commonly changing aspect is represented by the heterogeneity of device types belonging to the federation, along with their distribution. In this context, we refer to device type as to their hardware component, thus indicating the seven different devices selected for our experimentation. The distribution of these devices may vary radically depending on the scenario at hand and this variability can impact greatly the performance of the federation process. Therefore, it is required to anal-

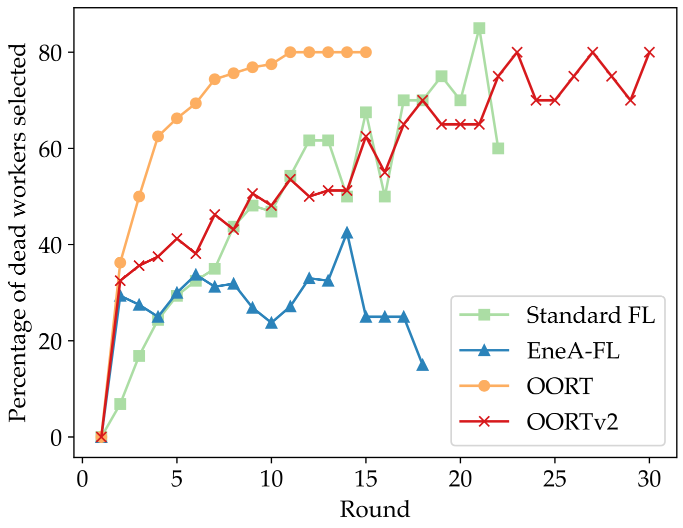


Figure 6.10: Percentage of selected dead devices for each round of optimisation.

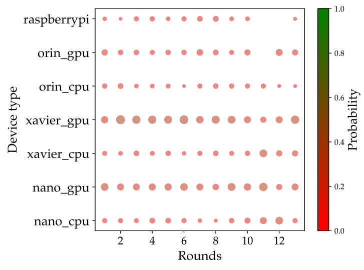
yse the flexibility of a given FL framework towards the different setups of device distributions that can be encountered. Therefore, we here consider investigating how the device distribution can impact the effectiveness of the proposed EneA-FL framework.

To study the impact of devices distribution on EneA-FL we consider running different experiments where the distribution of device types differ vastly. More in detail, we focus on the devices that are found to be more efficient (see Figure 6.5) and define three experimental setups where the probability of the federation containing those devices varies. In particular, we focus on the Jetson Nano with GPU and Jetson Xavier with GPU devices. We then define three experimental setups, namely:

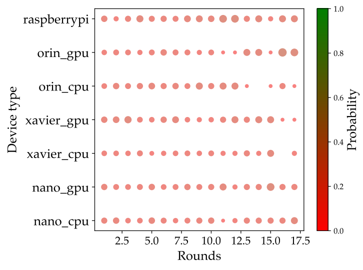
- *Likely setup*, where Jetson Nano with GPU and Jetson Xavier with GPU are more likely to appear in the federation. Here, the probability of a random device being one of these two types is equal to 0.25.
- *Uniform setup*, where Jetson Nano with GPU and Jetson Xavier with GPU are equally likely to appear in the federation. Here, the probability of a random device being one of these two types is 0.14, similarly to the uniform device type distribution scenario considered so far.
- *Rare setup*, where Jetson Nano with GPU and Jetson Xavier with GPU are less likely to appear in the federation. Here, the probability of a random device being one of these two types is equal to 0.05.

Finally, we study whether EneA-FL and the standard FL selection policy can select efficient devices. We focus solely on the standard FL baseline, as previous experiments highlight the similarity between available client selection policies and standard FL in terms of the distribution of selected devices—see Figure 6.8.

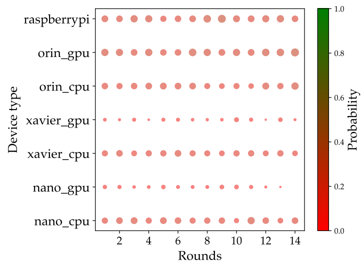
Figure 6.11 shows the distribution of the selected device types over the federation rounds for the standard FL scenario. Similarly to Figure 6.8, the size and the colour of each blob represent the probability of each device type being selected for a specific round of the federation. As expected, for the random selection policy, the distribution of the selected devices follows the distribution of their deployment. Here, the probability of an efficient device to be selected is equal throughout the



(a)

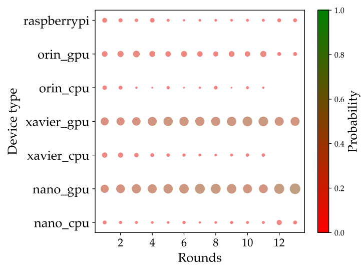


(b)

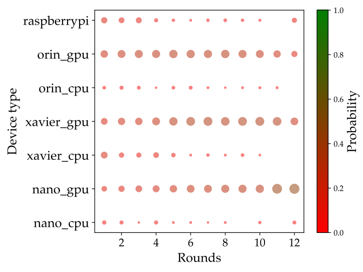


(c)

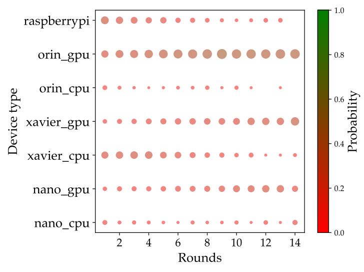
Figure 6.11: Distribution of selected devices for the standard FL policy over rounds when more efficient devices (Jetson Nano with GPU and Jetson Xavier with GPU) are more likely (a), equally likely (b) or least likely to be selected.



(a)



(b)



(c)

Figure 6.12: Distribution of selected devices for the EneA-FL policy over rounds when more efficient devices (Jetson Nano with GPU and Jetson Xavier with GPU) are more likely (a), equally likely (b) or least likely (c) to be selected.

optimisation, being higher for the *likely setup* (Figure 6.11a) and smaller for the *rare setup* (Figure 6.11c).

Figure 6.12 shows the same distribution study for the EneA-FL setup. Here, it is possible to notice that the probability of efficient devices being selected increases over time, independently of their likelihood of being in the federation or not. Indeed, in the *likely setup* (Figure 6.12a) the Jetson Nano with GPU and Jetson Xavier with GPU end up being almost the only devices selected from the federation mechanism. Even in the *rare setup* (Figure 6.12c) the Jetson Nano with GPU and Jetson Xavier with GPU end up being frequently selected devices, although there are only a handful of those devices in the federation. Moreover, in the *rare setup* it is relevant to notice that the other type of device popularly selected ends up being the Jetson Orin with GPU which represents the third most efficient solution according to our measurements. Finally, in all setups, it is possible to see that the least efficient devices – e.g., Jetson Orin without GPU and Jetson Nano without GPU – are also the least popular selections, independently of their proportion.

The results show the superiority of EneA-FL over the standard FL setups in terms of flexibility against device type distribution. Indeed, EneA-FL is capable of selecting efficient devices even when such devices represent a very small component of the federation devices.

## Chapter Synopsis

In this chapter, we tackle the frequently overlooked issue of resource management in FL over constrained devices. We introduce EneA-FL, a novel serverless computing framework for FL in fog and constrained environments. In these environments, the resource management of devices participating in the learning process represents a fundamental component to take into account, as possible discharging issues and high-update latency can hinder the overall optimisation process. To tackle this issue, EneA-FL presents a promising solution to enhance worker selection in FL applications focusing on energy awareness, latency reduction and performance improvements. To achieve its goal EneA-FL relies on three novel components, namely *(i) Energon* – a novel energy monitoring tool –; *(ii) Furcifer* – an ad-hoc orchestrator –; and *(iii) Magister*—an hybrid energy management process. EneA-FL's unique hybrid composition and energy-conscious approach relies on a reputation system to balance energy, latency, and performance of each device, by selecting the most appropriate for each learning step. Extensive experiments show that EneA-FL yields remarkable results, demonstrating a 30% to 60% reduction in energy consumption, and faster convergence of models to the target accuracy when compared to available client selection policies.

# Part II

# Efficientisation via Neuro-Symbolic Integration

The optimist sees the donut, the pessimist sees the hole.

*Oscar Wilde*

# Chapter 7

# Background on Symbolic and Sub-Symbolic Integration

This chapter contains contributions from [Ciatto et al., 2024].

In the ML and DL context, *learning* is automated via data-driven algorithms, often implying *numeric* processing of data—which in turn enables the detection of fuzzy patterns or statistically-relevant regularities in the data, that algorithms can learn to recognise. This promotes a data-driven approach to the engineering of intelligent computational systems where hard-to-code tasks are (semi-)automatically learned from data rather than manually programmed by human developers. The data-driven nature of ML and DL approaches is fundamental to support the automatic acquisition of otherwise hard-to-formalise behaviours for computational systems. However, flexibility comes at the cost of poorly-*interpretable* solutions, as state-of-the-art *sub-symbolic* predictors – such as neural networks – are often exploited behind the scenes. Recent research efforts have focused on novel AI paradigms aiming at blending the subsymbolic perspective of ML and DL agents with symbolic AI solutions focusing on high-level symbolic (human-readable) representations of problems, logic, and search: this is where *neuro-symbolic integration systems* (NeSy) stand today. NeSy integrate neural (subsymbolic) and symbolic AI solutions aiming at suitably complementing their strengths and weaknesses, introducing reasoning and cognitive capabilities (the symbolic way) while preserving fast-learning capabilities (the subsymbolic way).

While representing a popular target for attaining more interpretable AI systems, NeSy can be leveraged as a possible helping tool to tackle the NN efficientization task. NeSy systems allow to consider an a-priori knowledge that should not be extracted and learnt from the data used to optimize the sub-symbolic predictor, thus reducing the learning burden. Accordingly, in this chapter, we focus on the set of NeSy approaches which can be leveraged as helping tools for tackling the efficientization of NNs task. We do so by promoting two complementary activities, namely *symbolic knowledge extraction* (SKE) and *injection* (SKI) from and into sub-symbolic predictors. In both cases, "symbolic" refers to the way knowledge is represented. In particular, we consider as symbolic any language that is intelligible and interpretable for both human beings and computers. This includes a number of logic formalisms, and excludes the fixed-sized tensors of numbers commonly exploited in sub-symbolic ML.

Intuitively, SKE is the process of distilling the knowledge a sub-symbolic predictor has grasped from data into symbolic form. Generally speaking, SKE enables the *inspection* of the inner operation of an opaque predictor constructing an *interpretable replacement* (a.k.a. surrogate model) for the original predictor. In this context, the extracted surrogate model complexity can be limited to obtain a more efficient and resource-friendly version of the original sub-symbolic predictor. Moreover, upon further inspection, the symbolic surrogate model can be pruned removing its non-relevant portions, enabling efficiency fine-tuning otherwise unachievable leveraging only the original sub-symbolic predictor.

Conversely, SKI is the process of letting a sub-symbolic predictor follow the symbolic knowledge possibly encoded by its human designers. It enables a higher degree of *control* over a sub-symbolic predictor and its behaviour, by constraining it with human-like common-sense—suitably encoded into symbolic form. In this context, the injection process can remove part of the learning burden from the data-driven nature of NN models, as the available symbolic knowledge contains the information readily available. Therefore, SKI can reduce the learning time by providing straight away the very knowledge that predictors would otherwise struggle to learn by processing huge amounts of data. Moreover, SKI mitigates the issues arising from the lack of sufficient amounts of training data – as under-represented situations can be suitably represented in symbols –, thus enabling to learn from fewer data.

Apart from insights, notions such as SKE and SKI have rarely been described in general terms into the scientific literature—despite the multitude of methods falling under their umbrellas. Hence, the aim of this chapter is to provide general definitions and descriptions of these topics, other than providing durable taxonomies for categorising present and future SKE/SKI methods. Arguably, these contributions should take into account the widest possible portion of scientific literature, so as to avoid subjectivity. Accordingly, we propose a systematic literature review following the three-folded purpose of *(i)* collecting and categorising existing methods for SKE and SKI into clear taxonomies, *(ii)* providing a wide overview of the state of the art and technology, and *(iii)* detecting open research challenges and opportunities. In particular, we analyse 132 methods for SKE and 117 methods for SKI, classifying them according to their purpose, operation, expected input/output data and predictor types. For each method, we also probe the existence/lack of software implementations. We elicit a meta-model for SKE (resp. SKI) according to how existing and future extraction (resp. injection) methods can be categorised and described.

## 7.1 Background on Symbolic Knowledge and Computational Logic

Symbolic KR has always been regarded as a key issue since the early days of AI, as no intelligence can exist without knowledge, and no computation can occur in lack of representation. When compared to arrays of numbers, symbolic KR is far more flexible and expressive, and, in particular, more intelligible—both machine- and human-interpretable. Historically, most KR formalisms and technologies have been designed on top of *computational logic* [Lloyd, 1990], that is, the exploitation of formal logic in computer science. Consider, for instance, *deductive databases* [Green and Raphael, 1968], *description logics* [Baader, 2003], *ontologies* [Cimiano, 2006], *Horn* logic [McNulty, 1977], *higher-order* logic [Van Benthem and Doets, 2001], just to name a few.

## Formal Logics

Many kinds of logic-based KR systems have been proposed over the years, mostly relying on *first-order logic* (FOL) – either by restricting or extending it –, e.g., on description logics and modal logics, which have been used to represent, for instance, terminological knowledge and time-dependent or subjective knowledge. Here, we briefly recall the state of the art of FOL and its most relevant subsets.

**First-order logic** FOL is a general-purpose logic which can be used to represent knowledge symbolically, in a very flexible way. More precisely, it allows both human and computational agents to express (i.e., write) the properties of – and the relations among – a set of entities constituting the *domain of the discourse*, via one or more *formulæ*—and, possibly, to reason over such formulæ by drawing inferences. There, the domain of the discourse $\mathbb{D}$ is the set of all relevant entities which should be represented in FOL to be amenable of formal treatment, in a particular scenario.

Informally, the syntax for the general FOL formula is defined over the assumption that there exist: *(i)* a set of *constant* or *function* symbols, *(ii)* a set of *predicate symbols*, and *(iii)* a set of *variables*. Under such assumption, a FOL formula is any expression composed of a list of quantified variables, followed by a number of *literals*, i.e., *predicates* that may or may not be prefixed by the negation operator ($\neg$). Literals are commonly combined into expressions via *logic connectives*, such as conjunction ($\wedge$), disjunction ($\vee$), implication ($\rightarrow$), or equivalence ($\leftrightarrow$).

Each predicate consists of a predicate symbol, possibly applied to one or more *terms*. Terms may be of three sorts, namely *constants*, *functions*, or *variables*. Constants represent entities from the domain of the discourse. In particular, each constant references a different entity. Functions are combinations of one or more entities via a *function symbol*. Similarly to predicates, functions may carry one or more terms. Being containers of terms, functions enable the creation of arbitrarily complex data structures combining several elementary terms into composite ones. Such kind of composability by recursion is what makes the aforementioned definition of "symbolic" valid for FOL. Finally, variables are placeholders for unknown terms—i.e., for either individual or groups of entities.

Predicates and terms are very flexible tools to represent knowledge. While terms can be used to represent or reference either entities or groups of entities from the domain of the discourse, predicates can be used to represent relations among entities, or the properties of each single entity.

**Intensional vs. extensional** In logic, one may define concepts – i.e., describe data – either *extensionally* or *intensionally*. Extensional definitions are *direct* representations of data. In the particular case of FOL, this implies defining a relation or set by explicitly mentioning the entities it involves. Conversely, intensional definitions are *indirect* representations of data. In the particular case of FOL, this implies defining a relation or set by describing its elements via other relations or sets. Recursive intensional predicates are very expressive and powerful, as they enable the description of infinite sets via a finite (and commonly small) amount of formulæ—and this is one of the key benefits of FOL as a means for KR.

### Expressiveness vs. Tractability: Notable Subsets of FOL

Tractability deals with the theoretical questions: "can a logic reasoner compute whether a logic formula is true (or not) in *reasonable* time?". Such aspects are deeply entangled with the particular reasoner of choice. Depending on which and how many features a logic includes, it may be more or less *expressive*. The higher the expressiveness, the more the complexity of the problems which may be represented via logic and processed via inference increases. This opens to the possibility, for the solver, to meet queries which cannot be answered in practical time, or by relying upon a limited amount of memory—or just cannot get an answer at all. Roughly speaking, more expressive logic languages make it easier for human beings to describe a particular domain – usually, requiring them to write less and more concise clauses –, at the expense of a higher difficulty for software agents to draw inferences autonomously—because of computational tractability. This is a well-understood phenomenon in both computer science and computational logic [Levesque and Brachman, 1987, Brachman and Levesque, 2004], often referred to as the *expressiveness/tractability* trade-off.

FOL, in particular, is considered very expressive. Indeed, it comes with many undecidable, semi-decidable, or simply intractable properties. Hence, several relevant subsets of FOL have been identified into the literature, often sacrificing expressiveness for tractability. Major notions concerning these logics are recalled below.

**Horn logic** Horn logic is a notable subset of FOL, characterised by a good trade-off among theoretical expressiveness and practical tractability [Makowsky, 1987].

Horn logic is designed around the notion of *Horn clause* [Horn, 1951]. Horn clauses are FOL formulæ having no quantifiers, and consisting of a disjunction of predicates, where only at most one literal is non-negated—or, equivalently, an implication having a single predicate as post-condition and a conjunction of predicates as pre-condition: $h \leftarrow b_1,\ \ldots,\ b_n$. There, $\leftarrow$ denotes logic implication from right to left, commas denote logic conjunction, and all $b_i$, as well as $h$, are predicates of arbitrary arity, possibly carrying FOL terms of any sort—i.e., variables, constants, or functions. Horn clauses are thus if-then rules written in reverse order, and only supporting conjunctions of predicates as pre-conditions.

Essentially, Horn logic is a very restricted subset of FOL where: *(i)* formulæ are reduced to clauses, as they can only contain predicates, conjunctions, and a single implication operator, therefore *(ii)* operators such as $\vee$, $\leftrightarrow$, or $\neg$ cannot be used, *(iii)* variables are implicitly quantified, and *(iv)* terms work as in FOL.

**Datalog** Datalog is a restricted subset of FOL [Ajtai and Gurevich, 1994], representing knowledge via function-free Horn clauses—defined in the previous paragraph. So, essentially, Datalog is a subset of Horn logic where structured terms (i.e., recursive data structures) are forbidden. This is a direct consequence of the lack of function symbols.

Similarly to Horn logic, Datalogs's knowledge bases consist of sets of function-free Horn clauses.

**Description logics (DL)** Description logics are a family of subsets of FOL, generally involving some or no quantifiers, no structured terms, and no $n$-ary predicates such that $n \geq 3$. In other words, description logics represent knowledge by only leveraging on constants and variables, other than atomic, unary, and binary predicates.

Differences among specific variants of DL lay in which and how many logic connectives are supported, other than, of course, whether negation is supported or not. The wide variety of DL is due to the well known expressiveness/tractability trade-off. However, depending on the particular situation at hand, one may either prefer a more expressive ($\approx$ feature rich) DL variant at the price of a reduced tractability (or even decidability) of the algorithms aimed at manipulating knowledge represented through that DL, or *vice versa*.

Regardless of the particular DL variant of choice, it is common practice in the scope of DL to call *(i)* constant terms, as "individuals" – as each constant references a single entity from a given domain –, *(ii)* unary predicates, e.g., as either "classes" or "concepts" – as each predicate *groups* a set of individuals, i.e., all those individuals for which the predicate is true –, *(iii)* binary predicates, e.g., as either "properties" or "roles"—as each predicate *relates* two sets of individuals. Following such a nomenclature, any piece of knowledge can be represented in DL by tagging each relevant entity with some constant (e.g., an URL), and by defining concepts and properties accordingly.

Notably, binary predicates are of particular interest as they support connecting couples of entities altogether. This is commonly achieved via subject-predicate-object *triplets*, i.e., ground binary predicates of the form $\langle \mathtt{a}\ f\ \mathtt{b} \rangle$ – or, alternatively, $f(\mathtt{a}, \mathtt{b})$ –, where $\mathtt{a}$ is the subject, $f$ is the predicate, and $\mathtt{b}$ is the object. Such triplets allow users to *extensionally* describe knowledge in a readable, machine-interpretable, and tractable way.

Collections of triplets constitute the so-called *knowledge graphs* (KG), i.e., directed graphs where vertices represent individuals, while arcs represent the binary properties connecting these individuals. These may explicitly or implicitly instantiate a particular *ontology*, i.e., a formal description of classes characterising a given domain, and of their relations (inclusion, exclusion, intersection, equivalence, etc.), as well as the properties they must (or must not) include.

**Propositional logic** Propositional logic is a very restricted subset of FOL, where quantifiers, terms, and non-atomic predicates are missing. Hence, propositional formulæ simply consist of expressions involving one or many 0-ary predicates

– i.e., *propositions* –, possibly interconnected by ordinary logic connectives. There, each proposition may be interpreted as a Boolean variable – which can either be true or false –, and the truth of formulæ can be computed as in the Boolean algebra. So, for instance, a notable example of propositional formula could be as follows: $p \wedge \neg q \rightarrow r$ where $p$ may be the proposition "it is raining", $q$ may be the proposition "there is a roof", whereas $r$ may be the proposition "the floor is wet".

The expressiveness of propositional logic is far lower than the one of FOL. For instance, because of the lack of quantifiers, each relevant aspect/event should be explicitly modelled as a proposition. Furthermore, because of the lack of terms, entities from a given domain cannot be explicitly referenced. Such lack of expressiveness, however, implies computing the *satisfiability* of a propositional formula is a *decidable* problem—which may be a desirable property in some application scenarios.

Despite propositional logic may appear too trivial to handle common decision tasks where non-binary data is involved, it turns out a number of apparently complex situations can indeed be reduced to a propositional setting. This is the case for instance of any expression involving numeric variables or constants, arithmetical comparison operators, logic connectives, and nothing more than that. In fact, formulæ containing comparisons among variables or constants (or among each others) can be reduced to propositional logic by mapping each comparison into a proposition.

## 7.2 Definitions & Categorization Methodology

The goal of this chapter is to categorise the many SKE and SKI algorithms proposed into the literature so far, hence shaping a clear picture of what SKE and SKI mean today.

Following this purpose, we *(i)* start from broad and intuitive definitions of both SKE and SKI (provided in Section 7.2.1); we then *(ii)* define a number of research questions aimed at delving into the details of actual SKE and SKI methods; along this line, we *(iii)* explore the literature looking for contributions matching the broad definitions from step *(i)* (following a strategy described in Section 7.2.2). Finally, by analysing such contributions, we *(iv)* provide answers for the research questions from step *(ii)* (in Section 7.3), and, in doing so, we *(v)* synthesise general, bottom-up taxonomies for both SKE and SKI (in Sections 7.3.1 and 7.3.2).

### 7.2.1 Definitions for Symbolic Knowledge Extraction and Injection

Here we provide broad definitions for both symbolic knowledge extraction and injection, following the purpose of drawing a line among what methods, algorithms, and technologies from the literature should be considered related to either SKE or SKI, and what should not. Notably, we tune our definitions so as to comprehend and generalise the many methods and algorithms surveyed in this chapter and considered in this thesis. Indeed, looking for a wider degree of generality, our definitions commit to no particular form of symbolic knowledge, nor sub-symbolic predictor—despite many surveyed techniques come with commitments of that sort. Hence, in what follows we write "symbolic knowledge" meaning "any chunk of intelligible information expressed in *any* possibly sort of logic", as well as any sort of information which can be rewritten in logic form. Similarly, we write "sub-symbolic predictor" meaning "any sort of *supervised* ML model which can be fitted over *numeric* data to eagerly solve classification or regression tasks".

#### Extraction

Generally speaking, SKE serves the purpose of generating intelligible representations for the sub-symbolic knowledge an ML predictor has grasped from data during learning.

**Definition** We define SKE as

> *any* algorithmic *procedure accepting* trained *sub-symbolic predictors as input and producing* symbolic *knowledge as output, so that the extracted knowledge reflects the behaviour of the predictor with high* fidelity*.*

Notably, this definition emphasises a number of key aspects of SKE which are worth to be described in further detail.

First, SKE is modelled as a class of *algorithms* – hence finite-step recipes – characterised by what they accept as input and what they produce as output.

As far as the inputs of SKE procedures are concerned, the only explicit requirement is on *trained* ML predictors. There is no constraint w.r.t. the nature of the predictor itself, hence SKE procedures may be designed for any possible predictor family, in principle. Yet, this requirement implies that the predictor's training has already occurred, and it has reached some satisfying performance w.r.t. the task it has been trained for. Hence, in an ML workflow, SKE should occur *after* training and validation were concluded.

As far as the outputs of SKE procedures are concerned, the only explicit requirement is about the production of *symbolic* knowledge. "Symbolic" is here intended, in a broader sense, as a synonym of "intelligible" (for the human being), hence admissible outcomes are logic formulæ as well as decision trees, or bare human-readable text.

In any case, for an algorithm to be considered a valid SKE procedure, the output knowledge should mirror the behaviour of the original predictor w.r.t. the domain it was trained for, as much as possible. This involves some fidelity score aimed at measuring how well the extracted knowledge mimics the predictor it was extracted by, w.r.t. the domain and the task that predictor was trained for. This, in turn, implies that the extracted knowledge should, in principle, act as a predictor as well, thus being queryable as the original predictor would. Thus, for instance, if the original predictor is an image classifier, the extracted knowledge should let an intelligent agent classify images of the same sort, expecting the same result. The agent may then be either computational (i.e., a software program) or human, depending on whether the extracted knowledge is machine- or human-interpretable. Notably, the exploitation of *logic* knowledge as the target of SKE is of particular interest as it would enable both options.

### Injection

Generally speaking, SKI serves a dual purpose w.r.t. to SKE. In particular, SKI aims at letting an ML predictor keep some symbolic knowledge into account when drawing predictions.

**Definition** We define SKI as

> *any* algorithmic *procedure affecting how sub-symbolic predictors draw their inferences in such a way that predictions are either* computed *as a function of, or made* consistent *with, some* given *symbolic knowledge.*

This definition emphasises a number of key aspects of SKI which are worth to be described in further detail. Similarly to SKE, it is modelled as a class of *algorithms*. Yet, dually w.r.t. extraction, SKI algorithms are procedures accepting symbolic knowledge as input and producing ML predictors as output.

About the inputs of SKI procedures, the only explicit requirement is that knowledge should be symbolic and user-provided—hence *human*-interpretable. However, since any input knowledge should be algorithmically manipulated by the SKI procedure, we elicit an implicit requirement here, constraining the input knowledge to be *machine*-interpretable as well. This implies that some formal language – e.g., some formal logic, or some decision tree – should be employed for knowledge representation, while free text or natural language should be avoided.

Along this line, another implicit requirement is that the input knowledge should be *functionally analogous* w.r.t. the predictors undergoing injection. In other words, if a predictor aims at classifying customer profiles as either worthy or unworthy for credit, then the symbolic knowledge should encode decision procedures to serve the exact same purpose, and observe the exact same information.

About the outcomes of SKI procedures, our definition identifies two relevant situations—which are not necessarily mutually-exclusive. On the one side, SKI procedures may enable sub-symbolic predictors to accept symbolic knowledge as input. SKI procedures of this sort essentially consist of pre-processing algorithm aimed at encoding symbolic knowledge in sub-symbolic form, hence enabling sub-symbolic predictors to accept them as input. In this sense, SKI procedures of this sort enable sub-symbolic predictors to (learn how to) *compute* predictions as functions of the symbolic knowledge they were fed with—assuming it has been conveniently converted into sub-symbolic form. On the other side, SKI procedures may alter sub-symbolic predictors so that they draw predictions which are *consistent* with the symbolic knowledge—according to some notion of *consistency*. SKI procedures of this sort essentially affect either the structure or the training process of the sub-symbolic predictors they are applied to, in such a way that the predictor must then keep the symbolic knowledge into account when drawing predictions. In this sense, SKI procedures of this sort force sub-symbolic predictors to learn not only from data but from symbolic knowledge as well.

In any case, regardless of their outcomes, SKI procedures fit the ML workflow in its early phases, as they may affect both pre-processing and training.

Notably, consistency plays a pivotal role in SKI, dually w.r.t. what fidelity does for SKE. Along this line, our definition involves some consistency score aimed at measuring how well the predictor undergoing injection can take advantage from the injected knowledge, w.r.t. the domain and the task that predictor was trained for. So, for instance, if a knowledge base states that loans should be guaranteed to people from a given minority – as long as annual income overcomes a given threshold –, then any predictor undergoing injection of that knowledge base should output predictions respecting that statement—or at least minimise violations w.r.t. it.

### 7.2.2 Review Methodology

The overall review workflow is inspired by the goal question metric approach by [Caldiera and Rombach, 1994]. In short, the workflow requires some clear research *goal(s)* to be fixed, and then decomposed into a number of research question the survey will then provide answers to. To produce such answers, the workflow requires of course scientific papers to be selected, and analysed. To serve this purpose, the workflow requires a pool of *queries* to be identified. Such queries must be performed on most relevant bibliographic search engines (e.g., Google Scholar, Scopus). Finally, the workflow requires the query results to be selected (or excluded) for further analyses following a reproducible criterion. Any subsequent analysis is then devoted to answer the aforementioned research questions, hence drawing useful classifications and general conclusions.

For the sake of reproducibility, in the remainder of this subsection we delve into the details of how our review on symbolic knowledge extraction and injection is conducted.

We start by defining three different research goals (**G**):

**G1** – *"understanding which are the features of SKE algorithms"*,

**G2** – *"understanding which are the features of SKI algorithms"*.

**G3** – *"probing the current level of technological readiness of SKE/SKI technologies"*.

Then, we break them down in the following research questions (**RQ**):

**RQ1** (from **G1**) – *"which sort of ML predictors can SKE be applied to?"*

**RQ2** (from **G1**) – *"is there any requirement on the input data for SKE?"*

**RQ3** (from **G1**) – *"which kind of SK can be extracted from ML predictors?"*

**RQ4** (from **G1**) – *"for which kind of AI tasks can SKE be exploited?"*

**RQ5** (from **G1**) – *"how does SKE work?"*

**RQ6** (from **G2**) – *"which sorts of ML predictors can SKI be applied to?"*

**RQ7** (from **G2**) – *"which kind of SK can be injected into ML predictors?"*

**RQ8** (from **G2**) – *"for which kind of AI tasks can SKI be exploited?"*

**RQ9** (from **G2**) – *"how does SKI work?"*

**RQ10** (from **G3**) – *"which and how many SKE/SKI algorithms come with runnable software implementations?"*

Notice that research questions about SKE are analogous to those about SKI. In both cases, research questions are devoted to clarify which kind of information can SKE (resp. SKI) methods accept as input (resp. produce as output), how do they work, which AI tasks they can be used for (e.g., regression, classification), and which ML predictors they can be applied to (e.g., NN, SVM, etc.).

In order to answer the research questions above, we identify a number of queries to be performed on widely-available bibliographic search engines. In detail, queries involve the following keywords:

- (‘rule extraction’ $\lor$ ‘knowledge extraction’) $\land$ (‘neural networks’ $\lor$ ‘support vector machines’)
- (‘pedagogical’ $\lor$ ‘decompositional’ $\lor$ ‘eclectic’) $\land$ (‘rule extraction’ $\lor$ ‘knowledge extraction’)
- ‘symbolic knowledge’ $\land$ (‘deep learning’ $\lor$ ‘machine learning’)
- ‘embedding’ $\land$ (‘knowledge graphs’ $\lor$ ‘logic rules’ $\lor$ ‘symbolic knowledge’)
- ‘neural’ $\land$ ‘inductive logic programming’

As far as bibliographic search engines are concerned, we exploit Google Scholar[1], Scopus[2], Springer Link[3], ACM Digital Library[4], and DBLP[5].

For each search engine and query pair, we consider the first two pages of results. For each result, we inspect the title, abstract, and – in case of ambiguity –, the introduction, while trying and classifying it according to three disjoint circumstances: *(i)* the paper is a *primary work* describing some SKE or SKI method matching the broad definitions from 7.2.1, *(ii)* the paper is a *secondary work* surveying some portion of literature overlapping SKE or SKI (or both), *(iii)* the paper is *unrelated* w.r.t. to both SKE and SKI, hence it is not relevant for this chapter. Notably, secondary works selected in step *(ii)* are valuable sources of primary works, hence we recursively explored their bibliographies to further select other primary works. In particular, in this phase we leverage upon relevant secondary works such as [Andrews et al., 1995, Besold et al., 2017, Calegari et al., 2020, Wang et al., 2017, d’Avila Garcez et al., 2001, Guidotti et al., 2018, Hailesilassie, 2016, Huysmans et al., 2006b, von Rueden et al., 2021, Xie et al., 2019b, Zilke et al., 2016]—which we acknowledge as noteworthy (even though less extensive) surveys in the field of SKE or SKI.

We select 249 primary works, of which 132 works concern SKE, and 117 concern SKI. We then analyse each primary work individually, in order to provide answers

[1]https://scholar.google.com
[2]https://www.scopus.com
[3]https://link.springer.com
[4]https://dl.acm.org
[5]https://dblp.uni-trier.de

to the aforementioned research questions. While doing so, we construct bottom-up taxonomies for both SKE and SKI.

Finally, we inspect each primary work for assessing its technological status. In particular, we look for runnable software implementations corresponding to the method described in the primary work. In case no software tool is clearly mentioned in the primary work, or if the software is not technically accessible (e.g., Web site or repository is private or non-reachable) at the time of writing this chapter, then we consider the method as lacking software implementations. Otherwise, we further distinguish among methods coming with reusable software libraries, and methods coming with experimental code. In the first case, the software is ready for re-use, either because it is published on public software repositories such as PyPi, or because it is structured in such a way to let users exploit it for custom purposes. Vice versa, if the software is tailored on the experiments mentioned in the primary work, then we consider it experimental.

## 7.3 Categorization Results

This section summarises the results of our literature analysis. In particular, answers for the research questions outlined in Section 7.2.2 are provided here.

Accordingly, we group research questions according to their main focus (SKE or SKI), and we answer to each question individually—grouping answers when convenient, for the sake of conciseness. Answers consist of brief statistical reports showing the distribution of the surveyed SKE/SKI methods w.r.t. some dimension of interest for either SKE or SKI. Interesting dimensions are presented on the fly, as part of our answers. This is deliberate, since we select as 'interesting dimension' any relevant way of clustering the surveyed methods. In other words, we let taxonomies emerge from the literature rather than super-imposing any particular view of ours.

### 7.3.1 Symbolic Knowledge Extraction

By building upon secondary works, such as the work by [Calegari et al., 2020] and the survey of [Andrews et al., 1995], we identify three relevant dimensions by which

SKE methods can be categorised, namely: *(i)* the learning task(s) they support; *(ii)* the method's translucency; *(iii)* the shape of the extracted knowledge. By analysing the surveyed SKE methods, we find these categories adequate. However, we identify new dimensions, namely: *(iv)* the sort of input data the predictor undergoing extraction is trained upon, and *(v)* the expressiveness of the extracted knowledge. In what follows we answer research questions **RQ1**–**RQ5** and **RQ10** by focusing on such dimensions, individually. Conversely, in the supplementary materials, we provide an overview of the 132 methods selected for SKE.

**RQ1: Which sort of ML predictors can SKE be applied to? RQ5: How does SKE work?**

Answers for questions **RQ1** and **RQ5** are deeply entangled, as they are both related to SKE methods' *translucency*. Translucency deals with the need of SKE methods to inspect the internal structure of the underlying black-box model, while producing the extracted rules.

SKE methods provide for translucency in two ways [Andrews et al., 1995], and can be labelled accordingly as

**decompositional** if the method needs to inspect (even partially) the internal parameters of the underlying black-box predictor, e.g., neuron biases or connection weights for NN, or support vectors for SVM;

**pedagogical** if the algorithm *does not* need to take into account any internal parameter, but it can extract symbolic knowledge by only relying on the predictor's outputs.

Along this line, we observe that surveyed SKE methods can be grouped into as many big clusters, depending on how they treat the predictor undergoing extraction.

W.r.t. **RQ1**, it is worth highlighting that pedagogical methods can be applied to any sort of supervised ML predictor, in principle—despite the literature may only report particular cases of application to specific predictors. Conversely, each decompositional method focuses on a specific sort of supervised ML predictor. Hence, decompositional SKE methods can be further categorised w.r.t. which sort

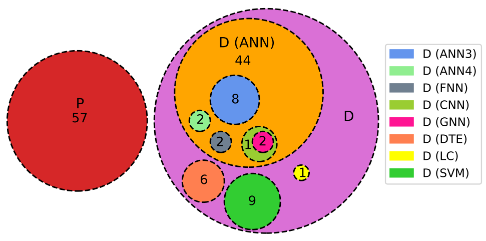


Figure 7.1: Venn diagram categorising SKE methods w.r.t. *translucency*: pedagogical (P) or decompositional (D). For decompositional methods, we report the target predictor type: ANN$\langle n \rangle$ = artificial NN (possibly, having exactly $\langle n \rangle$ layers), CNN = convolutional NN, GNN = graph NN, FNN = fuzzy NN, SVM = support vector machines, DTE = decision tree ensembles, LC = linear classifiers.

of supervised ML predictors they are tailored upon. As detailed by Figure 7.1, the translucency is far from uniform for SKE methods. Indeed, nearly a half of the surveyed methods are pedagogical, while the rest are tailored on feed-forward NN (possibly, with fixed amounts of layers), SVM, linear classifiers, or decision tree ensembles.

W.r.t. **RQ5**, it is worth highlighting that pedagogical methods treat the underlying predictor as an *oracle*, to be queried for predictions the symbolic knowledge shall emulate. Conversely, decompositional methods must look into the internal structure of predictors, hoping to detect meaningful patterns. For instance, SKE methods focusing on NN may try to interpret inner neurons as meaningful expressions combining their ingoing synapses.

## RQ2: Is there any requirement on the input data for SKE?

This question can be answered by looking at the accepted input data type of the surveyed SKE methods. In most cases data is structured, i.e., it consists of tables of numbers, where features are of three different sorts:

**binary** if the feature can assume only two values, generally encoded with 0 and 1 (or -1 and 1, or `true` and `false`);

**discrete** if the feature can assume values drawn from a *finite* set of admissible values; notably, when this is the case, data science identifies two relevant sub-sorts of features: **ordinal** if the set of admissible values is *ordered* (hence,

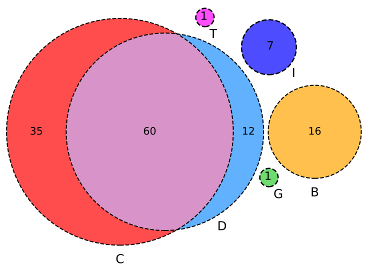


Figure 7.2: Venn diagram categorising SKE methods w.r.t. the *input data type* required by the underlying predictor: binary (B), discrete (D), continuous (C), images (I), text (T), graphs (G).

enabling the representation of the feature via some range of integer numbers), or **categorical** if that set is *un*ordered (hence, enabling the representation of the feature via one-hot encoding);

**continuous** if the feature can assume any real numeric value.

Alternatively, data may consist of

**images** i.e., matrices of pixels, possibly with multiple channels;

**text** i.e., sequences of characters of arbitrary length;

**graphs** i.e., data structures of variable size, consisting of nodes/vertices interconnected by edges/arcs.

In Figure 7.2, we report absolute occurrence of the sorts of input features accepted by the surveyed SKE methods, as described by their authors. As the reader may notice, the vast majority of surveyed methods are tailored on structured data with *continuous* and/or *discrete* features.

**RQ3: Which kind of SK can be extracted from ML predictors?**

Broadly speaking, any extracted SK should mirror (i.e., mimic) the operation of the ML predictor it has been extracted from. For *supervised* ML, this means the extracted knowledge should express a *function*, mapping input features into output features (e.g. classes, for classification tasks). Functions can be represented in symbols in several ways. Indeed, the SK extracted by the surveyed methods comes in various form.

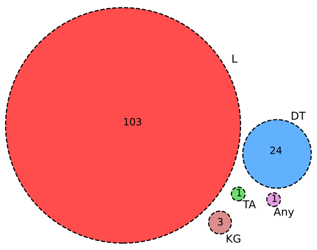


Figure 7.3: Venn diagram categorising SKE methods w.r.t. the output knowledge's *shape*: rule lists (L), decision trees (DT) or tables (TA), knowledge graphs (KG).

Notably, such forms can be categorised under both a *syntactic* or *semantic* perspective. There, syntax refers to the *shape* of the extracted SK, whereas semantic refers to what kind of logic formalism the extracted knowledge may leverage upon—which is a matter of *expressiveness*.

**Shape of the extracted knowledge** As far as syntax is concerned, decision *rules* [Freitas, 2014, Huysmans et al., 2011, Murphy and Pazzani, 1991] and *trees* [Breiman et al., 1984, Quinlan, 1993] are the most widespread human-comprehensible formats for the output knowledge, thus the vast majority of surveyed methods adopt one of these. However, other solutions have been exploited as well—e.g., decision *tables*. In all cases, however, a common trait is that functions of real numbers are expressed by using *symbols* to denote the same input and output features the underlying ML predictor was trained upon.

W.r.t. surveyed SKE methods, we identify four major admissible shapes:

**lists** of rules, i.e. sequences of logic rules to be read in some predefined order;

**decision trees** , i.e. hierarchical decision rules to be followed to compute a prection;

**decision tables** i.e., concise visual rule representations specifying one or more conclusions for each set of different conditions. They can be *exhaustive* – if all the possible combinations are listed –, or *incomplete*—otherwise. Generally speaking, decision tables are structured as follows: there is a column (row) for each input and output variable and a row (column) for each rule. Each cell $c_{ij}$ ($c_{ji}$) contains the value of the $j$-th variable for the $i$-th rule.

**knowledge graphs** see Section 7.1.

Figure 7.3 sums up the occurrence of the different shapes of output rules required for SKE algorithms. As the reader may notice, the majority of the surveyed methods target rule *lists*. Arguably, this trend may be motivated by the great simplicity of rule lists, in terms of readability, and their algorithmic tractability.

**Expressiveness of the extracted knowledge** Despite the extracted knowledge may contain statements of different shapes (e.g., rules, trees, tables), the readability, conciseness, and tractability of the extracted rules heavily depend on what can those statements contain—which, in turn, dictate what can (or cannot) be expressed. In the general case, statements may contain *predicates* or *relations* among the symbols representing input or output features. These may (or may not) contain logic connectives as well as arithmetic or logic comparators. SKE methods can be categorised w.r.t. which and how many ways of combining symbols are admissible within statements.

Along this line, we identify five major formats for statements in the surveyed SKE methods:

**propositional rules** are the simplest format, where statements consist of *propositions* – i.e. symbols denoting *boolean* input/output features –, possibly interconnected via logic connectives (negation, conjunction, disjunction, etc.). Notice that statements containing relations (e.g., arithmetic comparisons) among *single*, *continuous* features and *constant* values are indeed propositional as well.

**fuzzy rules** are propositional rules where the truth value of conditions and conclusions are not limited to 0 and 1, but can assume any value $\in [0,\ 1]$;

**oblique rules** have conditions expressed as inequalities involving linear combinations of the input variables. This is different from the propositional case, as features may be compared to other features (rather than constants alone).

**$m$-of-$n$ rules** are particular sorts of rules where *boolean* statements are grouped by $n$ and each rule is *true* only if at least $m$ literals (out of $n$) are *true*, with $m \leq n$. Notice that $m$-of-$(X_1, \ldots, X_n)$ is just a concise way of writing the disjunction among the conjunction of all possible $m$-sized combinations of

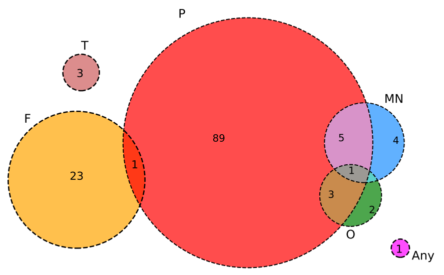


Figure 7.4: Venn diagram categorising SKE methods w.r.t. the output knowledge's *expressiveness*: propositional (P), $M$-of-$N$ (MN), fuzzy (F), or oblique (O) rules; or triplets (T).

$n$ boolean literals $X_1, \ldots, X_n$. Hence, $m$-of-$n$ rules are just a concise way of writing rules of other sorts: if $X_1, \ldots, X_n$ are all predicative statements, then the expression $m$-of-$(X_1, \ldots, X_n)$ is predicative as well—and the same is true if $X_1, \ldots, X_n$ are oblique statements.

**triplets** see Section 7.1.

Figure 7.4 summarises the occurrence of the different SK formats produced by the surveyed SKE algorithms. As the reader may notice, the vast majority of surveyed SKE methods produce predicative rules, i.e. rules composed of several boolean statements about individual input features, possibly interconnected via logic connectives. Arguably, this trend may be motivated by the great tractability of propositional rules, and by their simplicity. In fact, to construct propositional rules, SKE algorithms may follow a *divide-et-impera* approach by focusing on each single input feature at a time—hence enabling the simplification of the extraction process itself.

## RQ4: For which kind of AI tasks can SKE be exploited?

ML methods are commonly exploited in AI to serve specific purposes, e.g. classification, regression, clustering, etc. Regardless of the particular means by which SKE is attained, extraction aids the human users willing to inspect *how* those methods work. However, the particular AI tasks ML predictors have been designed for play a pivotal role in determining what outputs users may expect from those predictors. A similar argument holds for extraction procedures, as the extracted knowledge should reflect the inner behaviour of the original predictor. Along this line, it is interesting to categorise SKE methods w.r.t. the AI task they assume for the ML predictors they are applied to.

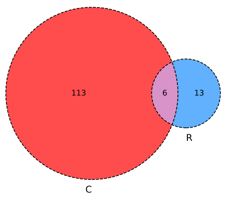


Figure 7.5: Venn diagram categorising SKE methods w.r.t. the *targeted AI task*: classification (C) or regression (R).

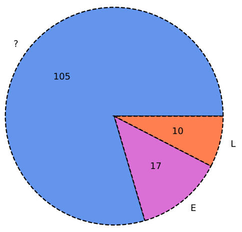


Figure 7.6: Pie chart categorising SKE methods presence/lack of software implementations. There, 'L' denotes the presence of a reusable library, 'E' denotes experiment code, and '?' denotes lack of known technologies.

Figure 7.5 summarises the occurrence of tasks among the surveyed SKE methods. Notably, most of them can be applied uniquely to *classifiers*, whereas a small portion of them is explicitly designed for *regressors*. Only few methods can handle both categories.

In general, we observe how the surveyed methods are tailored on either classification or regression tasks—when not both. In either cases, surveyed methods focus on supervised ML tasks. To the best of our knowledge, currently, there are no SKE procedures tailored on unsupervised or reinforcement learning tasks.

**RQ10: which and how many SKE algorithms come with runnable software implementations?**

Among the 132 surveyed methods for SKE, we found runnable software implementations for 27 (20.5%). Of these, 10 consist of reusable software libraries, while the others are just experimental code. Figure 7.6 summarises this situation. In the supplementary materials, we provide details about these implementations—there including the algorithm they implement and the link to the repository hosting the source code.

### Summary about SKE

Table 7.1 summarises our analysis regarding the 132 surveyed SKE methods. Notably, the table enumerates SKE methods in chronological order (w.r.t. publication year), grouping them by five-year periods. Furthermore, coherently w.r.t. the sections above, the table reports the translucency, the required task and input type, and the output format and shape of each surveyed method.

Table 7.1: Summary of the knowledge-extraction algorithms. Values from the columns “Translucency”, “Task”, “Input”, “Expressiveness”, “Shape”, and “Technology” refer the corresponding figures from Section 7.3.1.

| # | Method | Trans. | Task | Input | Express. | Shape | Tech. |
|---|---|---|---|---|---|---|---|
| 1 | [Breiman et al., 1984] | P | C+R | C+D | P | DT | L[6],[7] |
| 2 | [Quinlan, 1986] | P | C | D | P | DT | E[6] |
| 3 | [Saito and Nakano, 1988] | P | C | D | P | L | ? |
| 4 | [Clark and Niblett, 1989] | P | C | C+D | P | L | E[8] |
| 5 | [Masuoka et al., 1990] | D (ANN3) | C | C | F | L | ? |
| 6 | [Hayashi, 1990] | D (ANN) | C | B | F | L | ? |
| 7 | [Towell and Shavlik, 1991] | D (ANN) | C | D | MN | L | ? |
| 8 | [Berenji, 1991] | D (ANN) | C | C | F | L | ? |
| 9 | [Brunk and Pazzani, 1991] | P | C | C+D | P | L | ? |
| 10 | [Murphy and Pazzani, 1991] | P | C | D | MN | DT | ? |
| 11 | [Horikawa et al., 1992] | D (ANN) | C | C | F | L | ? |
| 12 | [Tresp et al., 1992] | D (ANN) | R | C | P | L | ? |
| 13 | [Towell and Shavlik, 1993] | D (ANN) | C | D | P | L | ? |
| 14 | [Thrun, 1993] | D (ANN) | C | C | P+MN | L | ? |
| 15 | [Cohen, 1993] | P | C | C+D | P | L | ? |
| 16 | [Quinlan, 1993] | P | C | C+D | P | DT | L[9] |
| 17 | [Fu, 1994] | D (ANN) | C | D | P | L | ? |
| 18 | [Halgamuge and Glesner, 1994] | D (ANN) | C | C | F | L | ? |
| 19 | [Mitra, 1994] | D (ANN) | C | C+D | F | L | ? |
| 20 | [Craven and Shavlik, 1994] | P | C | B | P+MN | L | E[7] |

[6] https://scikit-learn.org/stable/modules/tree.html
[7] https://github.com/psykei/psyke-python
[8] https://github.com/alessiamondolo/cn2-rule-based-classifier
[9] https://en.wikipedia.org/wiki/C4.5_algorithm#Implementations

Table 7.1: Summary of the knowledge-extraction algorithms (Continued).

| # | Method | Trans. | Task | Input | Express. | Shape | Tech. |
|---|---|---|---|---|---|---|---|
| 21 | [Fürnkranz and Widmer, 1994] | P | C | D | P | L | E[10] |
| 22 | [Sestito and Dillon, 1994] | P | C | C+D | P | L | ? |
| 23 | [Pop et al., 1994] | P | C | B | P | L | ? |
| 24 | [Andrews and Geva, 1995] | D (ANN) | C | C+D | P | L | ? |
| 25 | [Matthews and Jagielska, 1995] | D (ANN) | C | B | F | L | ? |
| 26 | [Matthews and Jagielska, 1995] | D (ANN) | C | B | F | L | ? |
| 27 | [Cohen, 1995] | P | C | C+D | P | L | L[11] |
| 28 | [Setiono and Liu, 1996] | D (ANN3) | C | B | P | L | ? |
| 29 | [Tickle et al., 1996] | P | C | B | P | L | ? |
| 30 | [Yuan and Zhuang, 1996] | P | C | D | F | L | ? |
| 31 | [Craven and Shavlik, 1996] | P | C | B | P+MN | DT | E[12][7] |
| 32 | [Hong and Lee, 1996] | P | C | C | F | L | ? |
| 33 | [Setiono and Liu, 1997] | D (ANN3) | C | C+D | O | L | ? |
| 34 | [Setiono, 1997] | D (ANN3) | C | D | P | L | ? |
| 35 | [Nauck and Kruse, 1997] | D (ANN) | C | D | F | L | ? |
| 36 | [Saito and Nakano, 1997] | D (ANN) | R | C | P | L | ? |
| 37 | [Benítez et al., 1997] | D (ANN) | C | C | F | L | ? |
| 38 | [Ishibuchi et al., 1997] | P | C | C | F | L | ? |
| 39 | [Taha and Ghosh, 1999] | D (ANN) | C | C | P | L | ? |
| 40 | [Taha and Ghosh, 1999] | D (ANN) | C | C+D | P | L | ? |
| 41 | [Krishnan et al., 1999b] | D (ANN) | C | B | P | L | ? |
| 42 | [Nauck and Kruse, 1999] | D (ANN) | R | D | F | L | E[13] |
| 43 | [Taha and Ghosh, 1999] | P | C | B | P | L | ? |
| 44 | [Krishnan et al., 1999a] | P | C | C | P | DT | ? |
| 45 | [Schmitz et al., 1999] | P | C+R | C+D | P | DT | L[14] |
| 46 | [Hong and Chen, 1999] | P | C | C | F | L | ? |
| 47 | [Setiono, 2000] | D (ANN3) | C | B | MN | L | ? |

[10] https://github.com/buoto/irep-rule-induction
[11] https://github.com/imoscovitz/wittgenstein
[12] https://github.com/abarthakur/trepan_python
[13] http://fuzzy.cs.ovgu.de/nefprox/
[14] https://github.com/fantamat/ruleex

Table 7.1: Summary of the knowledge-extraction algorithms (Continued).

| # | Method | Trans. | Task | Input | Express. | Shape | Tech. |
|---|---|---|---|---|---|---|---|
| 48 | [Tsukimoto, 2000] | D (ANN) | C | C+D | P | L | ? |
| 49 | [Kim and Lee, 2000] | D (ANN4) | C | C+D | P | DT | ? |
| 50 | [Setiono and Leow, 2000] | D (ANN) | R | C+D | P+MN+O | DT | ? |
| 51 | [Zhou et al., 2000] | P | C | C+D | P | L | ? |
| 52 | [Hong and Chen, 2000] | P | C | C | F | L | ? |
| 53 | [Sato and Tsukimoto, 2001] | D (ANN3) | C | C+D | P | DT | E[15] |
| 54 | [Parpinelli et al., 2001] | P | C | C+D | P | L | L[16] |
| 55 | [Castillo et al., 2001] | P | C+R | C+D | F | L | ? |
| 56 | [Saito and Nakano, 2002] | D (ANN) | R | C+D | P | L | ? |
| 57 | [Setiono et al., 2002] | D (ANN3) | R | C+D | P | L | ? |
| 58 | [Liu et al., 2002] | P | C | C+D | P | L | ? |
| 59 | [Boz, 2002] | P | C | C+D | P | DT | ? |
| 60 | [Markowska-Kaczmar and Trelak, 2003] | P | C | C+D | F | L | ? |
| 61 | [Zhou et al., 2003] | P | C | C+D | P | L | ? |
| 62 | [Setiono and Thong, 2004] | D (ANN3) | R | C+D | P | L | ? |
| 63 | [Fu et al., 2004] | D (SVM) | C | C+D | P | L | ? |
| 64 | [Markowska-Kaczmar and Chumieja, 2004] | P | C | C+D | P | L | ? |
| 65 | [Rabuñal et al., 2004] | P | C | C+D | P | L | ? |
| 66 | [Chen, 2004] | P | C | C | P | L | ? |
| 67 | [Liu et al., 2004] | P | C | C+D | P | L | E[17] |
| 68 | [Browne et al., 2004] | P | C | C+D | P+MN | DT | ? |
| 69 | [Zhang et al., 2005] | D (SVM) | C | C | P | L | ? |
| 70 | [Barakat and Diederich, 2005] | D (SVM) | C | C | P | DT | ? |
| 71 | [Fung et al., 2005] | D (LC) | C | C | P | L | ? |
| 72 | [Chaves et al., 2005] | D (SVM) | C | C | F | L | ? |
| 73 | [Torres and Rocco, 2005] | P | C | C+D | P+MN | DT | ? |
| 74 | [Etchells and Lisboa, 2006] | P | C | C+D | P | L | ? |
| 75 | [He et al., 2006] | P | C | C+D | P | DT | ? |
| 76 | [Huysmans et al., 2006a] | P | R | C | P | L | E[7] |
| 77 | [Bader et al., 2007] | D (ANN) | C | B | P | L | ? |
| 78 | [Schetinin et al., 2007] | D (DTE) | C | C+D | P | DT | ? |
| 79 | [Chen et al., 2007] | D (SVM) | C | C | P | L | ? |

[15] https://github.com/zju-vipa/awesome-neural-trees
[16] https://github.com/febo/myra
[17] https://rdrr.io/github/adriansidor/antminer/src/R/antminer3.R

Table 7.1: Summary of the knowledge-extraction algorithms (Continued).

| # | Method | Trans. | Task | Input | Express. | Shape | Tech. |
|---|---|---|---|---|---|---|---|
| 80 | [Barakat and Bradley, 2007] | D (SVM) | C | C+D | P | L | ? |
| 81 | [Saad and Wunsch II, 2007] | P | C | C+D | O | L | E[14] |
| 82 | [Martens et al., 2007] | P | C | C+D | P | L | ? |
| 83 | [Núñez et al., 2008] | D (SVM) | C | C | P+O | L | ? |
| 84 | [Setiono et al., 2008] | P | C | C+D | P+O | L | ? |
| 85 | [Odajima et al., 2008] | P | C | D | P | L | ? |
| 86 | [König et al., 2008] | P | C+R | C+D | P+F | DT | ? |
| 87 | [Bader, 2009] | D (ANN) | C | B | P | L | ? |
| 88 | [Martens et al., 2009] | D (SVM) | C | C+D | any | any | ? |
| 89 | [Lehmann et al., 2010] | P | C | B | P | L | ? |
| 90 | [Augasta and Kathirvalavakumar, 2012] | P | C | C+D | P | L | ? |
| 91 | [Sethi et al., 2012] | P | C | C+D | P | TA | ? |
| 92 | [Zilke et al., 2016] | D (ANN) | C | C+D | P | DT | E[14] |
| 93 | [Chan and Chan, 2017] | D (ANN) | R | C | P | L | ? |
| 94 | [Biswas et al., 2017] | P | C | C+D | P | L | ? |
| 95 | [Yedjour and Benyettou, 2018] | P | C | B | P | L | ? |
| 96 | [Chakraborty et al., 2018] | P | C | C+D | P+O | L | ? |
| 97 | [Obregon et al., 2019] | D (DTE) | C | C+D | P | L | ? |
| 98 | [Chan and Chan, 2020] | D (ANN) | R | C | P | L | ? |
| 99 | [Wang et al., 2020] | D (DTE) | C | C | P | L | ? |
| 100 | [Chen et al., 2020] | D (ANN) | C | I | P | L | E[18] |
| 101 | [Mahdavifar and Ghorbani, 2020] | D (ANN) | C | B | P | L | ? |
| 102 | [Vasilev et al., 2020] | D (ANN) | C | C+D | P | DT | ? |
| 103 | [Odense and d'Avila Garcez, 2020] | D (ANN) | C | I | MN | L | ? |
| 104 | [Jia et al., 2020] | D (ANN) | C | I | P | DT | ? |
| 105 | [Li et al., 2020] | D (ANN) | C | C | F | L | ? |
| 106 | [Hayashi and Takano, 2020] | D (ANN) | C | C+D | P | L | ? |
| 107 | [Chakraborty et al., 2020] | D (ANN) | C | C+D | P | L | ? |
| 108 | [Sabbatini et al., 2021] | P | R | C | P | L | E[7] |
| 109 | [Yu and Liu, 2021] | D (ANN) | C | C | P | L | ? |
| 110 | [Yan et al., 2021] | D (ANN) | C | C | P | DT | ? |
| 111 | [Dattachaudhuri et al., 2021] | P | C | C+D | P | L | ? |
| 112 | [Dong et al., 2021a] | D (DTE) | C | C+D | P | L | ? |

[18] `https://github.com/SeekingDream/FSE20_DENAS`

Table 7.1: Summary of the knowledge-extraction algorithms (Continued).

| # | Method | Trans. | Task | Input | Express. | Shape | Tech. |
|---|---|---|---|---|---|---|---|
| 113 | [Shams et al., 2021] | D (ANN) | C | C | P | L | L[19] |
| 114 | [Yedjour, 2021] | P | C | C+D | P | L | ? |
| 115 | [Marshakov, 2021] | D (ANN) | C | C | F | L | ? |
| 116 | [Yang et al., 2021a] | D (GNN) | C | G | KG | T | E[20] |
| 117 | [Bastos et al., 2021] | D (GNN) | C | T | KG | T | E[21] |
| 118 | [Horta et al., 2021] | D (ANN) | C | I | KG | T | ? |
| 119 | [Bologna, 2021] | D (DTE) | C | C | P | L | ? |
| 120 | [Espinosa Zarlenga et al., 2021] | D (ANN) | C | C | P | L | E[19] |
| 121 | [Sabbatini and Calegari, 2022] | P | R | C | P | L | E[7] |
| 122 | [Johansson et al., 2022] | P | R | C | P | DT | ? |
| 123 | [Barbiero et al., 2022] | D (ANN) | C | I | P | L | L[22] |
| 124 | [Ferreira et al., 2022] | D (ANN) | C | I | P | L | ? |
| 125 | [Diao et al., 2022] | D (ANN4) | C | C+D | P | L | ? |
| 126 | [Barbado et al., 2022] | D (SVM) | C | C+D | P | L | L[23] |
| 127 | [de Campos Souza and Lughofer, 2022] | D (FNN) | C | C+D | F | L | ? |
| 128 | [Salimi-Badr and Ebadzadeh, 2022] | D (FNN) | R | C | F | L | ? |
| 129 | [Irfan et al., 2022] | D (CNN) | C | I | P | L | ? |
| 130 | [Sabbatini and Calegari, 2023] | P | C+R | C | P | DT | E[7] |
| 131 | [Obregon and Jung, 2023] | D (DTE) | C | C+D | P | L | E[24] |
| 132 | [Ciravegna et al., 2023] | P | C | C+D | P | L | L[25] |

### 7.3.2 Symbolic Knowledge Injection

As far as SKI is concerned, we take into account no prior taxonomy. Indeed, despite the methods surveyed in this subsection come from well-studied (yet disjoint) research communities – such as neuro-symbolic computation [Besold et al., 2017] and knowledge graph embedding [Wang et al., 2017] –, we are not aware of any prior work attempting to unify these research areas under the SKI umbrella.

[19] https://github.com/mateoespinosa/remix
[20] https://github.com/BUPT-GAMMA/CPF
[21] https://github.com/ansonb/RECON
[22] https://github.com/pietrobarbiero/pytorch_explain
[23] https://github.com/AlbertoBarbado/rule_extraction_xai
[24] https://github.com/jobregon1212/rulecosi
[25] https://github.com/pietrobarbiero/logic_explained_networks

Along this line, we cluster the surveyed SKI methods according to four orthogonal dimensions, namely: *(i)* the type of SK they can inject, *(ii)* the strategy they follow to attain injection, *(iii)* the kind of predictors they can be applied to, *(iv)* the aim they pursue while performing injection. In what follows, we answer research questions **RQ6**–**RQ10** by focusing on such dimensions, individually. Conversely, in the supplementary materials, we overview the 117 methods selected for SKI.

## RQ7: Which kind of SK can be injected into ML predictors?

Generally speaking, SKI methods support the injection of knowledge expressed by various formalisms—despite each surveyed method focuses on some particular formalism. Along this line, a key discriminating factor is whether the chosen formalism is machine-interpretable or not—other than human-interpretable.

W.r.t. the formalism the input knowledge should adopt to support SKI, we may cluster the surveyed methods into two major groups, namely:

**logic formulæ** or **knowledge bases (KB)** (i.e., sets of formulæ) adhering to either FOL or some of its subsets, which are therefore both machine- and human-interpretable. Here, admissible sub-categories reflect the kinds of logics described in Section 7.1. Ordered by decreasing expressiveness, these are:

- **full first-order logic** formulæ including recursive terms, possibly containing variables, predicates of any arity, and logic connectives of any sorts, possibly expressing definitions;
- **Horn logic** (a.k.a. **Prolog**-like) where knowledge bases consist of head–body rules, involving predicates and terms of any sorts;
- **Datalog** i.e., Horn clauses without recursive terms (only constant or variable terms allowed);
- **modal logics** i.e., extensions of some logic above with modal operators (e.g., $\Box$ and $\Diamond$), denoting the *modality* in which statements are true (e.g., *when*, in temporal logic);

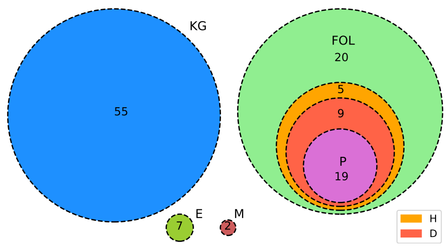


Figure 7.7: Venn diagram categorising SKI methods w.r.t. the *input knowledge* type: knowledge graphs (KG), propositional logic (P), first-order logic (FOL), expert knowledge (E), Datalog (D), Horn logic (H), or modal logic (M).

- **knowledge graphs** i.e., a particular application of description logics aimed at representing entity–relation graphs;
- **propositional logic** where expressions are simply expressions involving boolean variables and logic connectives.

**expert knowledge** i.e., any piece of human- (but not necessarily machine-) interpretable knowledge by which data generation can be attained. This might be the case of physics formulae, syntactical knowledge, or any form of knowledge that is usually held by a set of human experts, and, as such, is only accessible to human beings. For this reason, expert knowledge injection requires some data to be generated to reify its information in tensorial form. Of course, expert knowledge may be cumbersome to extract and requires human engineers to take care of data generation before any injection can occur.

In Figure 7.7 we categorise the surveyed SKI methods w.r.t. their formalism of choice. There, KG are the most prominent cluster (including almost a half of the surveyed methods), whereas model logic is the smallest one. Methods tailored on FOL or its subsets (apart from KG) form another relevant cluster. Among the FOL subsets, propositional logic plays a pivotal role, as it involves the relative majority of methods.

Notably, as long as the logic formalism is concerned, we consider and report the *actual* logic used in the papers. Indeed, this is rarely explicitly stated by the authors into their papers. So, we deduce the actual logic used by each SKI method from the constraints its logic is subject to, according to its authors.

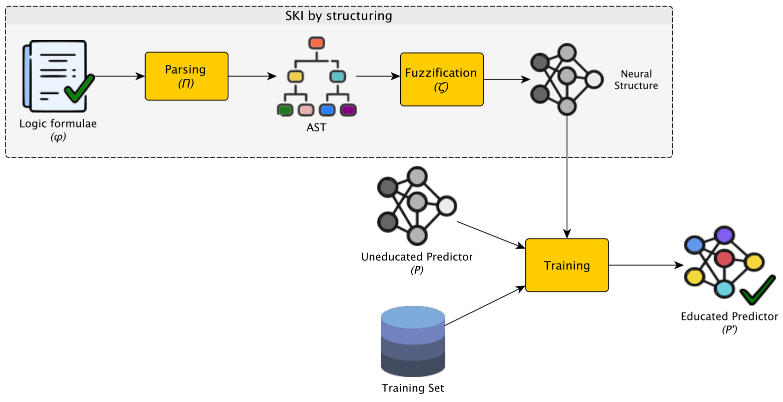


Figure 7.8: Structuring strategy: a (portion of a) neural network is constructed, mirroring the symbolic knowledge.

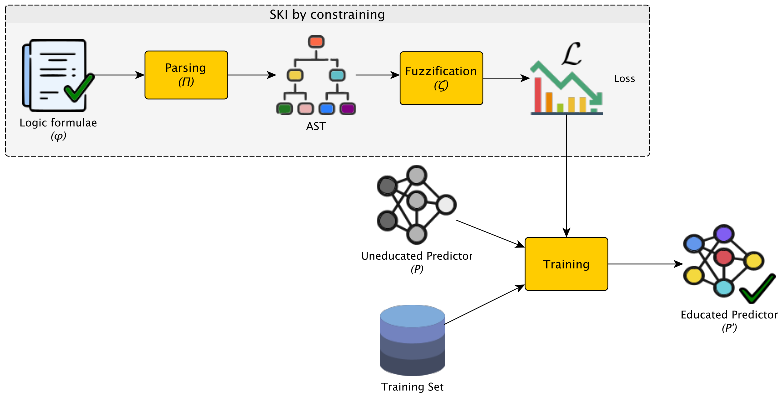


Figure 7.9: Guided learning strategy.

**RQ9: How does SKI work?**

By analysing the surveyed SKI methods, we acknowledge great variety in the actual way injection is performed. Arguably, however, such variety can be tackled by focusing on tree major *strategies*, depicted in Figures 7.8 to 7.10 and summarised below:

**predictor structuring** where (a part of) a sub-symbolic predictor (commonly, NN) is created to mirror the symbolic knowledge via its own internal structure. In other words, a predictor is created or extended to mimic the behaviour of the SK to be injected. For instance, when it comes to NN, their internal structure is crafted to represent logic predicates via neurons, and

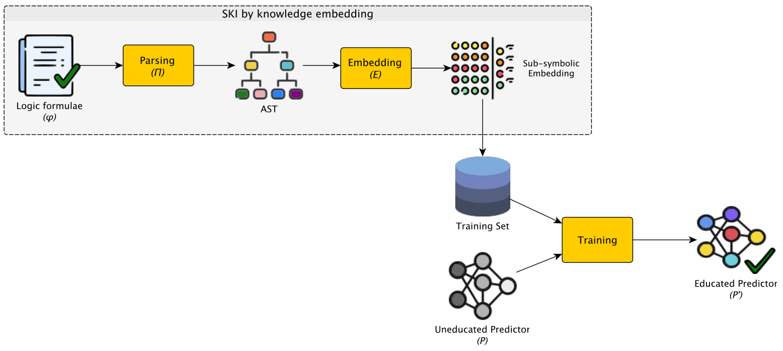


Figure 7.10: Embedding strategy: the symbolic knowledge is converted in tensorial form and ML predictors are fed “as usual”.

logic connectives via synapses;

**knowledge embedding** where SK is converted into numeric-array form – e.g., vectors, matrices, tensors, etc. – to be provided as “ordinary” input for the sub-symbolic predictor undergoing injection. In other words, numeric data is generated out of symbolic knowledge. Any numeric representation of this sort is called *embedding* [of the original symbolic knowledge]. For example, this is the common strategy exploited by the knowledge graph embedding community [Wang et al., 2017], as well as by graph NN [Lamb et al., 2020];

**guided learning** (a.k.a., **constraining**) where SK is used to steer the learning process of ML predictors, by either penalising inconsistent behaviours or by incentivising consistent behaviours w.r.t. the SK. When the predictor undergoing injection is trained via some optimisation process involving loss functions being minimised (e.g., NN), guided learning is achieved by altering those loss functions in such a way that violations w.r.t. the SK increase the loss. A dual statement holds for predictors requiring training to step through maximization processes. The recent book by [Gori, 2018] nicely overviews methods of these kinds.

Figure 7.11 summarises the frequency of these strategies among the surveyed SKI algorithms. Notably, the distribution of surveyed SKI methods among the three categories above is quite balanced.

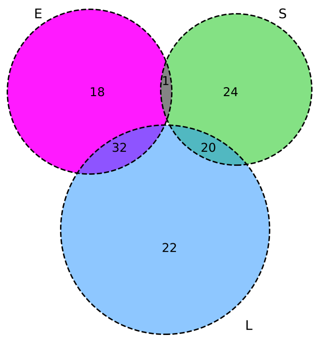


Figure 7.11: Venn diagram categorising SKI methods w.r.t. *strategy*: structuring (S), embedding (E), or guided learning (L).

**RQ6: Which sorts of ML predictors can SKI be applied to?**

Virtually *all* surveyed SKI methods are designed to inject knowledge into NN. However, as our analysis spans over 2 decades, the sorts of NN supported by SKI methods are manifold—despite each method is tailored on specific sorts of NN.

Accordingly, surveyed SKI methods can be classified w.r.t. the particular sort of NN they support. As detailed by Figure 7.12, admissible choices along this line fit many sorts of NN, namely:

**feed-forward NN** multi-layered NN where neurons from layer $i$ are only connected with layer $i + 1$, and multiple ($\geq 2$) layers may exist;

**convolutional NN** particular cases of feed-forward NN, involving convolutional layers as well;

**graph NN** particular cases of convolutional NN tailored on graph-like data;

**recurrent NN** particular cases of NN admitting loops among layers;

**Boltzmann machine** a particular neural architecture where connections are undirected—i.e., every node is connected to every other node;

**transformer** particular case of NN that leverage a self-attention mechanism—i.e., differentially weighting parts of the input data depending on their significance;

**auto-encoders** particular case of feed-forward NN, characterised by a bottleneck architecture used to learn reduced data encodings through learning to regenerate the input from the encoding;

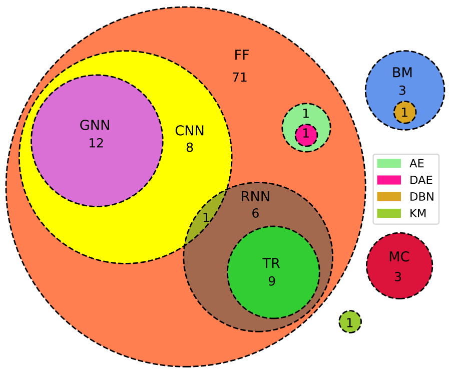


Figure 7.12: Venn diagram categorising SKI methods w.r.t. the *targeted predictor* type: feed-forward (FF), convolutional (CNN), graph (GNN) or recurrent (RNN) neural networks, Boltzmann machines (BM), Markov chains (MC), transformers (TR), auto-encoders (AE), deep belief networks (DBN), denoising auto-encoders (DAE), kernel machines (KM).

**deep belief networks** a composition of multiple Boltzmann machines, stacked altogether, in a feed-forward fashion;

**denoising auto-encoder** particular case of auto-encoders working over corrupted input.

Notable exceptions are:

**kernel machines** ML models relying on kernels—i.e., similarity measures between observed patterns;

**Markov chains** state machines with probabilities on state transitions, modelling stochastic phenomena.

The reason why the vast majority of methods rely on (some sort of) NN is straightforward: methods tailored upon GNN (resp. CNN) assume the networks to accept specific kinds of data as input, e.g. graphs (resp. images), while ordinary feed-forward NN accept raw vectors of real numbers.

## RQ8: For which kind of AI tasks can SKI be exploited?

Unlike SKE methods – which uniquely serve the purpose of inspecting black-box predictors by mimicking the way they address supervised learning tasks –, SKI methods from the literature may serve multiple purposes. As outlined by Figure 7.13, we identify two major purposes SKI methods may pursue, by targeting

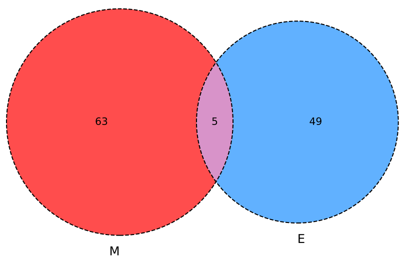


Figure 7.13: Venn diagram categorising SKI methods w.r.t. *aim*: knowledge manipulation (M) or enrich (E).

either symbolic or sub-symbolic AI tasks. More precisely, SKI methods may pursue:

**symbolic knowledge manipulation** where SKI enables the *sub*-symbolic manipulation of *symbolic* knowledge, by letting sub-symbolic predictors treat SK similarly to what done by symbolic engines. In doing so, SKI supports symbolic-AI tasks such as

**logic inference** in its many forms (e.g. deductive, inductive, probabilistic, etc.), i.e. drawing conclusions out of symbolic KB;

**information retrieval** looking for information in symbolic KB;

**KB completion** finding (and adding) missing information in symbolic KB;

**KB fusion** merging several KB into a single one, taking care of (possibly, syntactically different) overlaps;

The key point here is supporting tasks where both inputs and outputs are symbolic in nature, but leveraging upon sub-symbolic methods to gain speed, fuzziness, and robustness against noise.

**learning support** (a.k.a., **enrich**) where SKI lets *sub*-symbolic methods consume *symbolic* knowledge to either improve or enrich learning capabilities. In doing so, SKI supports ordinary ML tasks – such as classification –, by allowing ML predictors to process (or take advantage by) structured symbolic knowledge. The underlying idea of such approaches is that there exist some concepts that are cumbersome or troublesome to learn from examples—e.g., syntactical concepts, semantics, etc. Therefore, SK expressing these high-level concepts may be injected directly into the model to be trained.

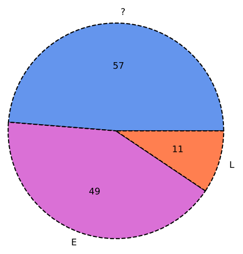


Figure 7.14: Pie chart categorising SKI methods presence/lack of software implementations. There, 'L' denotes the presence of a reusable library, 'E' denotes experiment code, and '?' denotes lack of known technologies.

As the reader may note from the picture, surveyed SKI methods are quite balanced w.r.t. the categories above, with a slight preference for SK manipulation.

**RQ10: which and how many SKI algorithms come with runnable software implementations?**

Among the 117 surveyed methods for SKI, we found runnable software implementations for 60 (51.3%). Of these, 11 consist of reusable software libraries, while the others are just experimental code. Figure 7.14 summarises this situation. In the supplementary materials, we provide details about these implementations—there including the algorithm they implement and the link to the repository hosting the source code.

### Summary about SKI

Table 7.2 summarises our analysis regarding the 117 surveyed SKI methods. Notably, the table enumerates SKI methods in chronological order (w.r.t. publication year), grouping them by five-year periods. Furthermore, coherently w.r.t. the sections above, the table reports the strategy followed by each SKI method, as well as type of knowledge it can inject, the type of neural network it supports, and overall purpose it supports injection for.

Table 7.2: Summary of knowledge-injection algorithms. Values from the columns "Strategy", "Input", "Predictor", "Purpose", and "Technology" refer the corresponding figures from Section 7.3.1.

| # | Method | Strategy | Input | Predictor | Purpose | Tech. |
|---|---|---|---|---|---|---|
| 1 | Ballard, 1986 | S | FOL | BM | M | ? |
| 2 | Towell et al., 1990 | S | P | FF | E | L[26] |
| 3 | Pinkus, 1991 | S | FOL | BM | M | ? |
| 4 | Tresp et al., 1992 | L+S | P | FF | E+M | ? |
| 5 | Giles and Omlin, 1993 | S | E | RNN | E+M | ? |
| 6 | Tan, 1997 | S | P | FF | E | ? |
| 7 | d'Avila Garcez and Zaverucha, 1999 | S | P | FF | M | L[27] |
| 8 | Basilio et al., 2001 | L+S | FOL | FF | M | ? |
| 9 | d'Avila Garcez and Gabbay, 2004 | S | FOL | FF | M | ? |
| 10 | Bader et al., 2005 | S | FOL | FF | M | ? |
| 11 | Chang et al., 2007 | E | E | MN | E | ? |
| 12 | Bader et al., 2008 | L | E | FF | E | ? |
| 13 | Mintz et al., 2009 | E | KG | FF | M | ? |
| 14 | Nickel et al., 2011 | E | KG | FF | M | L[28] |
| 15 | Bordes et al., 2011 | E | KG | FF | M | E[29] |
| 16 | Kimmig et al., 2012 | S | FOL | MN | M | L[30] |
| 17 | Bordes et al., 2012 | E | KG | FF | M | ? |
| 18 | Pinkas et al., 2013 | S | FOL | BM | M | ? |
| 19 | Bordes et al., 2013 | E+L | KG | FF | M | E[31] |
| 20 | Socher et al., 2013 | E+S | KG | FF | M | E[32] |
| 21 | França et al., 2014 | S | P | RNN | M | E[33] |
| 22 | Wang et al., 2014 | E+L | KG | FF | M | E[34] |
| 23 | García-Durán et al., 2014 | E+L | KG | FF | M | ? |
| 24 | Bian et al., 2014 | E+L | E | FF | E | ? |
| 25 | Chang et al., 2014 | E | KG | FF | M | ? |
| 26 | Bordes et al., 2014 | E | KG | FF | M | E[35] |
| 27 | Dong et al., 2014 | E | KG | FF | M | ? |
| 28 | Fan et al., 2014 | E+L | KG | FF | M | ? |
| 29 | Wang et al., 2015 | E | KG | FF | M | ? |

[26] https://github.com/psykei/psyki-python
[27] https://sourceforge.net/projects/cil2p/
[28] https://github.com/mnick/rescal.py
[29] https://github.com/glorotxa/SME
[30] https://github.com/linqs/psl
[31] https://github.com/Lapis-Hong/TransE-Knowledge-Graph-Embedding
[32] https://github.com/dddoss/tensorflow-socher-ntn
[33] https://github.com/vakker/CILP
[34] https://github.com/thunlp/KB2E
[35] https://github.com/usherwang02/SemanticMatchingEnergy-Theano

Table 7.2: Summary of knowledge-injection algorithms (continued).

| # | Method | Strategy | Input | Predictor | Purpose | Tech. |
|---|---|---|---|---|---|---|
| 30 | Wei et al., 2015 | E | KG | MN | M | E[36] |
| 31 | Rocktäschel et al., 2015 | E+L | KG | FF | M | E[37] |
| 32 | Lin et al., 2015 | E+L | KG | FF | M | E[34] |
| 33 | Yang et al., 2015 | E+L | KG | FF | M | ? |
| 34 | Che et al., 2015 | L | KG | FF | E | ? |
| 35 | Ji et al., 2015 | E+L | KG | FF | M | ? |
| 36 | Feng et al., 2016 | E+L | KG | FF | M | ? |
| 37 | Xiao et al., 2015 | E+L | KG | FF | M | ? |
| 38 | He et al., 2015 | E+L | KG | FF | M | ? |
| 39 | Tran and Garcez, 2016 | S | P | DBN | E | ? |
| 40 | Hu et al., 2016a | S | P | CNN | E | ? |
| 41 | Guo et al., 2016 | E+L | KG | FF | M | ? |
| 42 | Nickel et al., 2016 | E+L | KG | FF | M | E[38] |
| 43 | Trouillon et al., 2016 | E+L | KG | FF | M | E[39] |
| 44 | Demeester et al., 2016 | L | KG | FF | M | ? |
| 45 | Hu et al., 2016b | S | P | FF | E | ? |
| 46 | Mrksic et al., 2016 | L | KG | FF | E | E[40] |
| 47 | Liu et al., 2016a | E | KG | FF | M | ? |
| 48 | Ji et al., 2016 | E+L | KG | FF | M | ? |
| 49 | Xiao et al., 2016b | E+L | KG | FF | M | ? |
| 50 | Xiao et al., 2016a | E+L | KG | FF | M | ? |
| 51 | Kipf and Welling, 2017 | E+L | KG | GNN | M | L[41] |
| 52 | Rocktäschel and Riedel, 2017 | L+S | D | FF | M | E[42] |
| 53 | Liu et al., 2017a | E+L | KG | FF | M | E[43] |
| 54 | Stewart and Ermon, 2017 | L | E | CNN | E | ? |
| 55 | Allamanis et al., 2017 | L | P | RNN | E | E[44] |
| 56 | Diligenti et al., 2017a | L | FOL | KM | M | E[45] |
| 57 | Diligenti et al., 2017b | L | P | CNN | M | ? |
| 58 | Marino et al., 2017 | E | KG | GNN | E | ? |
| 59 | Chang et al., 2017 | E | E | FF | E | E[46] |
| 60 | Choi et al., 2017 | E | KG | FF | E | E[47] |
| 61 | Fang et al., 2017 | L | KG | CNN | E | ? |

[36] `https://github.com/ZhuoyuWei/fpMLN`
[37] `https://github.com/uclnlp/low-rank-logic`
[38] `https://github.com/mnick/holographic-embeddings`
[39] `https://github.com/thunlp/openke`
[40] `https://github.com/nmrksic/counter-fitting`
[41] `https://github.com/tkipf/pygcn`
[42] `https://github.com/uclnlp/ntp`
[43] `https://github.com/quark0/ANALOGY`
[44] `https://github.com/mast-group/eqnet`
[45] `https://sites.google.com/site/semanticbasedregularization/home/software`
[46] `https://github.com/mbchang/dynamics`
[47] `https://github.com/mp2893/gram`

Table 7.2: Summary of knowledge-injection algorithms (continued).

| # | **Method** | **Strategy** | **Input** | **Predictor** | **Purpose** | **Tech.** |
|---|---|---|---|---|---|---|
| 62 | [Xu et al., 2018] | L | P | CNN | E | E[48] |
| 63 | [Evans and Grefenstette, 2018] | L+S | D | FF | M | E[49] |
| 64 | [Sourek et al., 2018] | S | D | FF | M | E[50] |
| 65 | [Velickovic et al., 2018] | E+L | KG | GNN | M | E[51] |
| 66 | [Ma and Zhang, 2018] | L | KG | AE | E | ? |
| 67 | [Zhou et al., 2018] | E | KG | GNN | E | E[52] |
| 68 | [Liang et al., 2018] | S | KG | FF | E | E[53] |
| 69 | [Glavas and Vulic, 2018] | E+L | KG | FF | E | E[54] |
| 70 | [Marra et al., 2019] | L | P | FF | E | E[55] |
| 71 | [Goodwin and Demner-Fushman, 2019] | L | KG | FF | E | ? |
| 72 | [Sun et al., 2019] | E+L | KG | FF | M | ? |
| 73 | [Zhang et al., 2019b] | L | KG | TR | E | ? |
| 74 | [Peters et al., 2019] | E+L | KG | TR | E | L[56] |
| 75 | [Daniele and Serafini, 2019] | S | FOL | DFF | E | L[57] |
| 76 | [Fischer et al., 2019] | L | D | DFF | E | E[58] |
| 77 | [Dong et al., 2019] | S+L | H | FF | M | E[59] |
| 78 | [Badreddine et al., 2022] | S | FOL | FF | E+M | L[60] |
| 79 | [Zhang et al., 2020] | E+L | KG | FF | M | E[61] |
| 80 | [Jiang et al., 2020] | S+L | FOL | RNN | E | ? |
| 81 | [Ren and Leskovec, 2020] | S+L | KG | DFF | M | E[62] |
| 82 | [Guo et al., 2020] | L+E | KG | FF | M | E[63] |
| 83 | [Riegel et al., 2020] | S+L | FOL | FF | M | L[64] |
| 84 | [Yu and Liu, 2021] | S | P | DAE | E | ? |
| 85 | [Manhaeve et al., 2021] | S | H | FF | E+M | L[65] |
| 86 | [Dash et al., 2021] | E | P | GNN | E | E[66] |

[48] https://github.com/UCLA-StarAI/Semantic-Loss/
[49] https://github.com/crunchiness/lernd
[50] https://github.com/GustikS/GNNwLRNNs
[51] https://github.com/PetarV-/GAT
[52] https://github.com/tuxchow/ccm
[53] https://github.com/julianschoep/SGRLayer
[54] https://github.com/codogogo/explirefit
[55] https://github.com/GiuseppeMarra/lyrics
[56] https://github.com/allenai/kb
[57] https://github.com/DanieleAlessandro/KENN
[58] https://github.com/eth-sri/dl2
[59] https://github.com/google/neural-logic-machines
[60] https://github.com/logictensornetworks/logictensornetworks
[61] https://github.com/MIRALab-USTC/KGE-HAKE
[62] https://github.com/snap-stanford/KGReasoning
[63] https://github.com/StudyGroup-lab/SLRE
[64] https://github.com/IBM/LNN
[65] https://github.com/ML-KULeuven/deepproblog
[66] https://github.com/tirtharajdash/VEGNN

Table 7.2: Summary of knowledge-injection algorithms (continued).

| # | Method | Strategy | Input | Predictor | Purpose | Tech. |
|---|---|---|---|---|---|---|
| 87 | Giunchiglia and Lukasiewicz, 2021 | S+L | P | CNN | E | E[67] |
| 88 | Bosselut et al., 2021 | S | KG | TR | M | ? |
| 89 | Peng et al., 2021 | E | E | TR | E | ? |
| 90 | West et al., 2022 | L E | KG | TR | E | ? |
| 91 | Marino et al., 2021 | S+L | KG | TR | E | L[68] |
| 92 | Xie et al., 2021 | S+L | M | RNN | E | ? |
| 93 | Cheng et al., 2021 | L+E | H | FF | M | ? |
| 94 | Li et al., 2023a | S+L | D | GNN | M | ? |
| 95 | d'Amato et al., 2021 | L+E | KG | FF | M | E[69] |
| 96 | Dash et al., 2022 | S | P | GNN | E | E[70] |
| 97 | Rodríguez et al., 2022 | L | KG | CNN | E | E[71] |
| 98 | Yu et al., 2022 | S+L | FOL | CNN | E | ? |
| 99 | Wei et al., 2022 | S+L | P | GNN | M | E[72] |
| 100 | Smirnova et al., 2022 | L | P | FF | E | E[73] |
| 101 | Magnini et al., 2022a | S | D | FF | E | E[26] |
| 102 | Spillo et al., 2022 | E | FOL | DFF | E | E[74] |
| 103 | Tang et al., 2022 | L+E | FOL | RNN | M | E[75] |
| 104 | Zhu et al., 2022 | L | FOL | GNN | M | E[76] |
| 105 | Li et al., 2022a | L+E | KG | GNN | M | ? |
| 106 | Sen et al., 2022 | S+L | D | FF | M | ? |
| 107 | Magnini et al., 2022c | S+L | D | DFF | E | E[26] |
| 108 | Werner et al., 2023 | S | FOL | GNN | E | E[77] |
| 109 | Giannini et al., 2023 | L | FOL | FF | E | ? |
| 110 | Cunnington et al., 2023 | S+L | D | FF | E | E[78] |
| 111 | Pourvali et al., 2023 | S+L | FOL | TR | E | ? |
| 112 | Ahmed et al., 2023 | L | P | FF | E | E[79] |
| 113 | Marconato et al., 2023 | L | H | DFF | M+E | E[80] |
| 114 | Li et al., 2023b | S+L | KG | TR | E | E[81] |
| 115 | Lin et al., 2023 | S+L | H | TR | E | ? |

[67] https://github.com/EGiunchiglia/C-HMCNN/
[68] https://github.com/facebookresearch/mmf
[69] https://github.com/Keehl-Mihael/TransROWL-HRS
[70] https://github.com/tirtharajdash/BotGNN
[71] https://github.com/JulesSanchez/X-NeSyL
[72] http://github.com/jinnanli/CogKG
[73] https://github.com/eXascaleInfolab/Nessy_RE
[74] https://github.com/giuspillo/RepoNeSyRecSys2022
[75] https://github.com/XiaojuanTang/RulE
[76] https://github.com/DeepGraphLearning/GNN-QE
[77] https://gitlab.inria.fr/tyrex-public/kegnn
[78] https://github.com/DanCunnington/FFNSL
[79] https://github.com/UCLA-StarAI/Semantic-Strengthening
[80] https://github.com/ema-marconato/NeSy-CL
[81] https://github.com/senticnet/SKIER

Table 7.2: Summary of knowledge-injection algorithms (continued).

| # | Method | Strategy | Input | Predictor | Purpose | Tech. |
|---|---|---|---|---|---|---|
| 116 | [Bai et al., 2023] | L | M | GNN | M | ? |
| 117 | [Nguyen et al., 2023] | L+E | FOL | FF | M | E[82] |

## 7.3.3 Taxonomy Summary

Figure 7.15 summarises the taxonomies for SKE and SKI we induced from the surveyed literature. Generally speaking, such taxonomies are useful tools to categorise present (and, hopefully, future) SKE/SKI methods, and to highlight the relevant features of each particular method. In this way, the interested readers may figure out what to expect from any given SKE/SKI method, as well as draw general analyses concerning the state of the art. Accordingly, in this section we analyse our taxonomies, elaborating on the current challenges and future perspectives.

It is worth mentioning that our taxonomies involve both "stable" and "contingent" categories by which SKE/SKI methods can be described. These are represented as either white or grey boxes in Figure 7.15. Stable categories are time-independent and they are not susceptible to change in the near future, while contingent categories are subject to trends and may evolve. Consider for instance SKE methods (see Figure 7.15), categorised w.r.t. their output knowledge. While expressiveness is a stable sub-category, its actual sub-sub-categories are contingent, meaning that new ones may be added in the future.

### SKE Taxonomy

As shown in Figure 7.15, SKE methods can be classified by *(i)* translucency, *(ii)* targeted AI task, *(iii)* nature of the input data, and *(iv)* form of the output knowledge. W.r.t. item *(i)*, SKE methods can either be categorised as pedagogical or decompositional. In the particular case of decompositional methods, the actually targeted predictor is relevant too—and possibilities *currently* include NN, DT, SVM, and linear classifiers. W.r.t. item *(ii)*, SKE methods may target classification or regression tasks, or both. In any case, they *currently* target supervised ML

[82] `https://github.com/nlp-tlp/cyle`

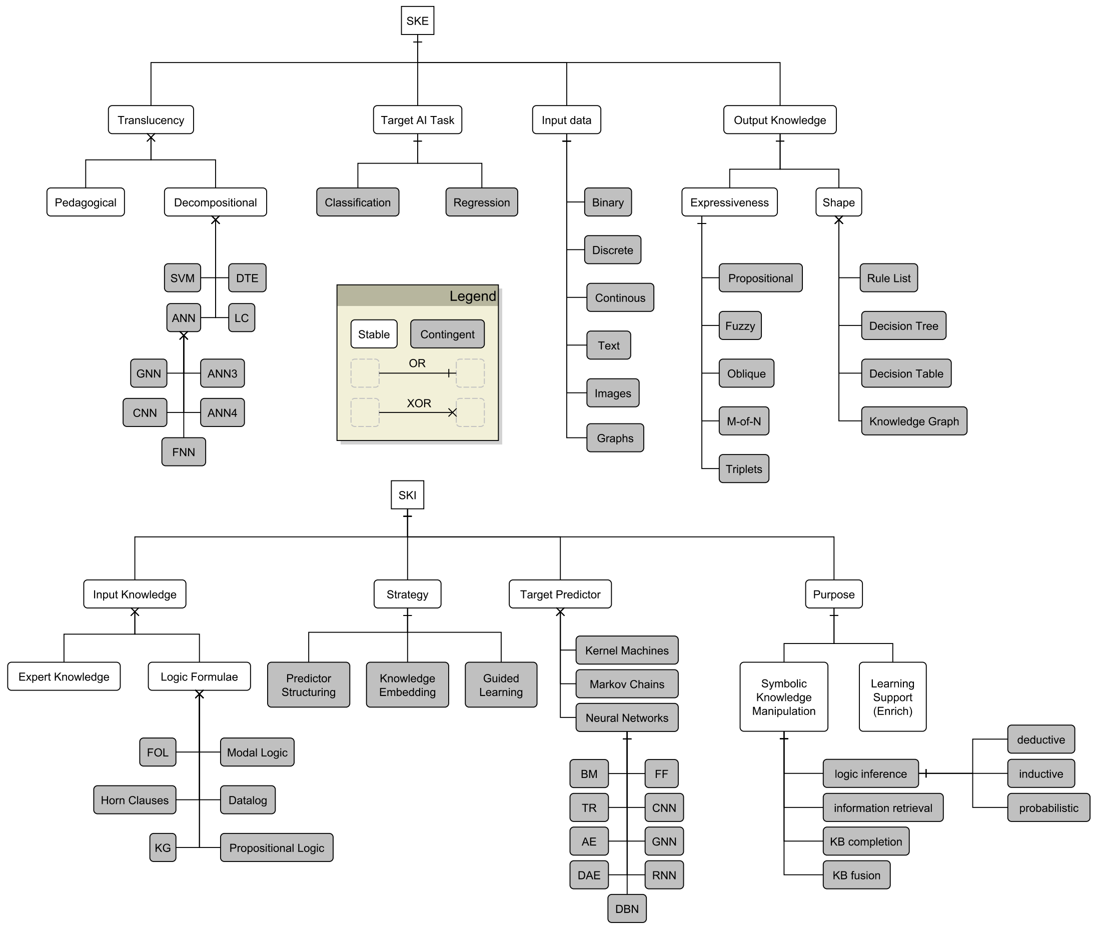


Figure 7.15: Summary of SKE and SKI taxonomies derived from the literature.

tasks alone. W.r.t. item *(iii)*, SKE methods accept predictors trained upon binary, discrete, or continuous data, as well as images, graphs and text. Finally, w.r.t. item *(iv)*, SKE methods may produce symbolic knowledge of different shapes, and with different expressiveness. Shapes may *currently* involve rule lists, as well as graphs, decision trees or tables. Conversely, as long as expressiveness is involved, symbolic knowledge may be propositional or fuzzy – possibly including $M$-of-$N$-like statements – , or be expressed as triplets or oblique rules.

**About translucency** It is worth stressing the relevance of pedagogical methods from the engineering perspective. Indeed, if properly implemented, pedagogical methods may be exploited in combination with predictors of any sorts. Of course, they are expected to reach lower performances w.r.t. decompositional ones, as they access less information. On the other side, decompositional methods may be more precise at the expense of generality.

**About input data** We recall that binary features are particular cases of discrete features, while discrete features are, in turn, particular cases of continuous features. Hence, it is worthwhile noticing that extractors requiring only binary features can be applied to categorical datasets by pre-processing discrete attributes via one-hot encoding (OHE). Analogously, extractors requiring discrete features can work with continuous attributes if those continuous features are *discretised*. Finally, continuous features can be converted into binary ones by performing discretisation and OHE, in this exact order.

While these transformations can always be applied in the general case, some authors have included them in their SKE methods at the design level. Hence, some papers explicitly account discretisation or OHE as part of the SKE methods they propose. This is the case, for instance, of the methods enclosed in the intersection between the "C" and "D" sets in Figure 7.2 (and labelled as "C+D" in the supplementary materials). Other methods may instead rely upon other discretisation strategies, such as the ones surveyed by [Yang et al., 2010].

**About output knowledge** It is worth stressing that differences among rule lists, decision trees, and tables are mostly syntactic, as conversions among these forms are possible in the general case (cf. the supplementary materials for examples). As far as expressiveness is concerned, we remark that all logic formalisms currently in use for SKE are essentially particular cases of propositional logic—possibly, under a fuzzy interpretation. This implies that the full power of FOL is far from being fully exploited in practice.

Finally, we point out some correlations among the expressiveness of output rules and the nature of the predictor they are extracted from, as well as the input data it is trained upon. For instance, SKE methods working with *continuous* input data are more likely to adopt oblique rules—or, at least, propositional rules with arithmetic comparisons. In fact, decisions are there drawn by comparing numeric variables with constants or among each other. Another example: some *decompositional* SKE methods focusing upon NN adopt $M$-of-$N$ statements. Arguably, the reason is that $M$-of-$N$ expressions aggregate several elementary statements into a single formula, similarly to how neurons aggregate synapses from previous layers in NN—hence such methods approximate neurons via $M$-of-$N$ expressions.

### SKI Taxonomy

As shown in Figure 7.15, SKI methods can be classified by *(i)* form of the input knowledge, *(ii)* followed strategy, *(iii)* targeted predictor type, and *(iv)* purpose. W.r.t. item *(i)*, SKI methods can either accept logic formulæ or expert knowledge as input. In the former case, *current* possibilities include FOL and its subsets, and in particular knowledge graphs. W.r.t. item *(ii)*, SKI methods may *currently* follow one of three strategies, namely: predictor structuring, knowledge embedding, or guided learning. W.r.t. item *(iii)*, SKI methods *currently* mostly target NN-based predictors, other than Markov chains and kernel machines. Finally, w.r.t. item *(iv)*, SKI methods may pursue two kinds of purposes, non-exclusively: manipulating symbolic knowledge or supporting/enriching learning. In the former case, *current* possibilities involve symbolic AI related tasks such as logic inference (and its many forms), information retrieval, and KB completion/fusion.

**About input knowledge and injection strategies** Logic formulæ are the most common approach to define prior concepts to be injected. This is true, in particular, for SKI approaches following the model structuring or guided learning strategies. Indeed, via logic formulæ, they express criteria that sub-symbolic models should satisfy or emulate. However, methods of these sorts often require formulæ to be grounded. Grounding introduces computational burden and hinders capability of representing recursive or infinite data structures—hence limiting what can actually be injected.

Conversely, knowledge graphs are the most common knowledge representation approach when it comes to perform SKI following the knowledge embedding strategy. This is unsurprising, given that "knowledge graph embedding" is a research line *per se*.

**About target predictors** Neural networks play a pivotal role in SKI. Arguably, the reason lies in the great *malleability* of NN w.r.t. their structure and training, as well as their *flexibility* w.r.t. feature learning. In fact, NN come in different shapes as different architectures may be constructed by connecting neurons in various ways. This is fundamental to support SKI via *predictor structuring*. Furthermore, as long as their architectures are DAG, NN can be trained via gradient descent, i.e. by minimising a loss function of arbitrarily defined. This is, in turn, fundamental to support SKI via *guided learning*. Finally, feature learning is a characterising capability of NN, making them capable to automatically elicit the relevant aspects they should focus upon, w.r.t. input data. This is the reason why NN are well suited for the knowledge embedding strategy as well. Accordingly, to the best of our knowledge, there exists no other sort of predictor having similar flexibility and malleability.

## Chapter Synopsis

In this chapter we survey the state of the art of symbolic knowledge extraction and injection. Stemming from two original definitions, we *systematically* explore the literature of both SKE and SKI, spanning a period of four decades. Our goal is to elicit the major characteristics of SKE/SKI algorithms form the literature (**G1**–**G2**), hence deriving general taxonomies which we hope other researchers may exploit. Along this line, we design ten research questions (**RQ1**–**RQ10**), and we engineer *ad hoc* queries to be performed on most relevant search engines for scientific literature. We select 249 primary works, almost evenly distributed among SKE and SKI, other than 11 secondary works. By analysing these papers, we define and discuss two general taxonomies for both SKE and SKI, which are general enough to categorise present (and possibly future) methods. Roughly, surveyed methods are categorised w.r.t. what they accept as input and produce as output (in terms of symbolic knowledge or predictors), other than how they operate and why. Finally, we also collect data about which and how many SKE/SKI methods come with runnable software implementations (namely, 87, i.e., 34.9%).

This chapter will provide a useful guide for the remainder of Part II of this thesis, where we will delve more in detail on how SKI and SKE approaches can help tackling the NN efficientization task.

# Chapter 8

# Can Symbolic Knowledge Injection Help Efficientisation?

This chapter contains contributions from [Agiollo et al., 2022, Agiollo et al., 2023a].

SKI represents a possible solution to several issues of *data-driven* AI approaches – i.e., ML and DL –, such as *(i) data greediness* – meaning that the learning requires huge amounts of examples concerning the phenomena to learn –; *(ii) resource greediness* – meaning that the learning process requires specialized hardware and long time to run –; and *(iii) reduced understandability*—as most ML and DL models are black-boxes.

The benefits of SKI to the training of ML predictors [Calegari et al., 2020] include: *(i)* mitigating the issues arising from the lack of sufficient amounts of training data – as under-represented situations can be suitably represented in symbols –; *(ii)* reducing the learning time by providing straight away the very knowledge that predictors would otherwise struggle to learn by processing huge amounts of data; *(iii)* improving predictors' predictive performance in corner cases – as in the case of unbalanced and overlapping classes –; *(iv)* preventing predictors from working as full black boxes during their training—hence overriding the need for explanations; and *(v)* harmonising the symbolic and sub-symbolic components of intelligent systems. Hence, ML researchers may take advantage of SKI to endow intelligent systems with common sense – encoded in some suitable symbolic formalism –, and finely govern their sub-symbolic components—e.g., by tuning them according to the given symbolic beliefs.

It is a common practice to assess SKI mechanisms in terms of the performance gain they introduce w.r.t. some injection-free counterpart [Diligenti et al., 2017b, Xu et al., 2018]. However, performance gain is not the only relevant metric that an AI designer may intend to optimise. For instance, when dealing with resource-constrained environments it may be required to minimise the energy used to train ML predictors, as well as the computational resources required for their execution. Analogously, whenever a human being needs to interact with the intelligent system under development, it is fundamental to maximise the intelligibility of its decision-making processes. Overall, there are several aspects of sub-symbolic predictors that designers could optimise via SKI. Along this line, we here sketch a set of quality-of-service (QoS) metrics for SKI covering several aspects—ranging from energy-related to computational-cost-related ones. The proposed set of QoS metrics focuses on measuring the achievable efficiency gains attainable when leveraging SKI against an injection-free counterpart and represents – to the best of our knowledge – the first attempt to introduce QoS metrics for SKI.

To measure the reliability of our metrics – along with the achievable efficiency improvements –, we extend the PSyKI[1] library, by proposing a full modelling of the sketched QoS metrics. PSyKI represents an open source library that supports the exploitation of SKI methods in arbitrary ML workflows [Magnini et al., 2022b]. Integrating our novel QoS metrics into the PSyKI library we enable measuring the efficiency gains achievable by few different SKI approaches over several datasets. The code to reproduce our experimental evaluation is made publicly available[2]. As expected, the experimental results demonstrate that it is indeed possible to leverage SKI approaches to achieve efficiency gains w.r.t. injection-free approaches, while also proving the soundness of our QoS metrics.

## 8.1 Preliminary Definitions

Formally, given an injection procedure $\mathcal{I}$, some symbolic knowledge $K$, and a sub-symbolic predictor $N$ aimed at solving some supervised learning task, we define

[1]`https://github.com/psykei/psyki-python`
[2]see `https://github.com/pikalab-unibo/ski-qos-jaamas-experiments-2022`

the "knowledge-aware" predictor $\hat{N}$ as the result of the application of $\mathcal{I}$ to $K$ and $N$:

$$\hat{N} = \mathcal{I}(K, N)$$

There, we call $N$ the *uneducated* predictor – as it has not yet undergone injection –, and $\hat{N}$ the *educated* one.

Focussing on the inputs of SKI – namely, the symbolic knowledge $K$ and the sub-symbolic predictor $N$ –, nearly all SKI methods and techniques available in the literature assume that: *(i)* $K$ is a *logic* knowledge base of logic formulæ, encoded via some subset of first-order logic [Smullyan, 1968], while *(ii)* $N$ is a neural network. To support this statement, we refer the reader to Table 7.2.

### 8.1.1 Injection Assessment

It is common for works in the SKI realm to measure the strength of their mechanism as the gain in performance achieved by the SKI predictor against its *uneducated* counterpart. In that case, the effectiveness of the injection mechanism $\mathcal{I}$ when applied to a neural network $N$ to inject the knowledge $K$ is measured via some performance score $\pi$ (accuracy, F1-score, MSE, etc.), aimed at assessing the performance of $N$ with respect some test dataset $T$. More formally:

$$\epsilon_{K,N,\pi,T}(\mathcal{I}) = \pi(\mathcal{I}(K, N), T) - \pi(N, T) \tag{8.1}$$

In other words, the effectiveness of some injection mechanism $\mathcal{I}$ may be assessed differently depending on which knowledge base, neural network, and dataset it is applied to. While being indicative of the quality of the SKI approach w.r.t. predictive performance, that metric does not capture every aspect of the knowledge injection, as there exist multiple properties that one may be willing to optimise through SKI—see Section 8.2.1.

## 8.2 SKI Quality-of-Service Metrics Definition

In this section we propose and analyse the novel set of metrics for identifying the quality of SKI systems. An overview of our proposals, along with a brief classification, is provided in Section 8.2.1. Roughly speaking, we introduce metrics for measuring SKI method's *efficiency*—under multiple goodness criteria.

### 8.2.1 Overview

The current practice of SKI assessment relies exclusively on measuring improvements in the predictive performance of some educated predictor over an equivalent uneducated counterpart. However, predictive performance is not the only relevant benefit of SKI one may be willing to measure.

There exist multiple aspects of neural predictors which may be affected by SKI—and for which metrics should be defined. In the context of this thesis, we focus on the *efficiency* properties of predictors and – recalling Section 2.1.2 – consider the model's:

**memory footprint,** i.e., the size of the predictor under examination;

**latency,** i.e., the time required to run a predictor for inference;

**data efficiency,** i.e., the amount of data required to train the predictor;

**energy consumption,** i.e., the amount of energy required to train/run the predictor;

other than, of course:

**predictive performance,** e.g. accuracy, F1-score, mean squared error, etc.

For the sake of brevity, we also denote as *efficiency metrics* any function aimed at measuring some efficiency property.

Efficiency metrics provide a score measuring how much some efficiency property $P$ of a given uneducated predictor $N$ improves in its educated counterpart $\hat{N}$. Of course, the resulting score may be largely influenced by a number of different aspects, such as:

**(A1)** *Knowledge quality and coverage.* The educated predictor $\hat{N}$ is attained by injecting some input knowledge $K$. Furthermore, both $N$ and $\hat{N}$ are aimed at addressing the same learning task – say, classification or regression –, and they are both trained upon some training dataset $D$, which describes the task. Questions that may arise are, for instance: *(i)* are $K$ and $D$ coherent?, *(ii)* is $K$ covering situation which the data in $D$ exemplifies?, *(iii)* is $K$ consistent, coherent, and correct? *(iv)* can we say the same for $D$? Regardless of the particular efficiency property $P$ being measured, the resulting score may greatly vary depending on the answers to these questions. So, in other words, efficiency measures depend on the particular input knowledge ($K$) and data ($D$) being used during SKI.

**(A2)** *Baseline quality.* As both the educated ($\hat{N}$) and uneducated ($N$) predictors are targetting the same learning task, one may wonder if $N$ is adequate enough to address that learning task. In this setting, questions that may arise are: *(i)* is $N$ biased [Piedmont, 2014] in statistical sense? *(ii)* in case it is, can we expect $\hat{N}$ to carry any observable improvement on some efficiency measure $P$? *(iii)* can we expect $\hat{N}$ to carry any observable improvement on some efficiency measure $P$? *(iv)* event in case where $N$ is not biased, is the selected injection mechanism $\mathcal{I}$ adequate for $N$? From these questions we understand that efficiency measures may also depend on the nature of the input predictor ($N$), and, of course, on the injection mechanism of choice ($\mathcal{I}$).

**(A3)** *Task at hand.* The learning task targeted by both $N$ and $\hat{N}$ determines the training dataset as well as the test dataset $T$. The choice of $T$ impacts the assessment of both $N$ and $\hat{N}$. Therefore, it may impact the score of any efficiency measure as well. So, efficiency measures may finally also depend on the target learning task, and, consequently on the test data ($T$).

Summarising, efficiency measures assess some injection method $\mathcal{I}$ in a very specific setting that depends on *(i)* the particular knowledge to be injected, *(ii)* the sort of predictor undergoing injection, *(iii)* the training and *(iv)* test data adopted for training. In other terms, any efficiency measure should be parametric w.r.t. $K$, $N$, $D$, and $T$.

Accordingly, in the following we propose the implementation and formalisation of metrics to assess the efficiency of SKI. In particular, as discussed at the beginning of this section, *memory footprint*, *latency*, *energy consumption*, and *data efficiency* are introduced as key performance indicators of SKI. Our objective is to assess the efficiency of SKI in terms of computational resource usage, and to provide insight into how these metrics could be used to inform the design and optimisation of SKI-based systems. Therefore, in the following, we consider retracing the definitions of the efficiency metrics previously proposed in Section 2.1.2 and extend them to the SKI scenario.

### 8.2.2 Memory Footprint

Several metrics for measuring the memory footprint of sub-symbolic predictors have been recently proposed in the literature [Kang et al., 2018, Wu et al., 2018a, Liberis et al., 2021]. For instance, it is possible to measure the neural networks' memory footprint by counting the amount of parameters they are composed by—i.e., essentially, the amount of synapses composing each neural network [Wu et al., 2018a]. Alternatively, some authors leverage metrics such as Floating Point OPerations (FLOPs) [Huang et al., 2018] or Multiplication Addition Computations (MACs) [Cheng et al., 2019], which measure the amount of total operations or multiplications and additions required to perform a single inference respectively. MACs consider solely multiplications and summations as they represent the most common computations in NNs. The available measures are indicative of the amount of memory required either to fit the whole sub-symbolic predictor – total number of parameters – or to run it—FLOPs and MACs. Given the proven effectiveness of these metrics, we here consider leveraging them to analyse the efficiency gain of SKI approaches. In other terms, we consider the ability of SKI mechanisms to produce *lightweight* sub-symbolic predictors—in terms of memory occupation.

The key insight here is that knowledge injection may lift part of the learning burden from the predictor at hand, by relieving the network from the need to learn complex or data-uncovered notions via trial-and-error. Indeed, the a-priori concepts carried by the input knowledge might now be injected instead of being learnt in a data-driven way. As a result, the amount of notions that sub-symbolic predictors must learn in a data-driven way might be significantly reduced. Fewer concepts to be learned are typically associated with fewer parameters, FLOPs, and MACs—or, in other words, a smaller memory footprint [Wu, 2019]. In the context of SKI, we define the memory footprint *improvement* score $\mu_{\Psi,K,N}(\mathcal{I})$ as the amount of memory saved by the educated network $\hat{N}$ w.r.t. its uneducated counterpart $N$. The higher the score, the more memory efficient the educated predictor is w.r.t. the uneducated one. However, as one may measure the memory footprint of a sub-symbolic predictor in different ways – e.g., by counting the number of parameters, FLOPs or MACs –, our scoring function is parametric in $\Psi$—which represents the memory footprint metric of choice. More formally:

$$\mu_{\Psi,K,N}(\mathcal{I}) = \Psi(N) - \Psi(\mathcal{I}(K, N)) \tag{8.2}$$

where $\hat{N} = \mathcal{I}(K, N)$ represents the educated predictor attained by injecting $K$ into $N$.

It is worth noticing how the proposed memory footprint score may be influenced by quality and coverage of the input knowledge **(A1)**, as well as by the memory footprint of the input predictor $N$ **(A2)**. About **(A1)**, the reason is simple: the better the input knowledge, the lower the expected memory requirements of the educated predictor. Similarly, as far as **(A2)** is concerned, the better the input predictor, the lower the expected memory footprint improvements of the educated predictor. However, one may also notice from Equation (8.2) that our memory footprint score is not parametric when the current task is taken into account **(A3)**. The reason is simple: the memory footprint of a neural network is *not* task-dependent, as it is a structural property of the neural network itself.

Finally, we stress that memory footprint of the educated predictor is expected to be lower than the one of the uneducated predictor. Indeed, our score is measuring the memory footprint *improvement*. A negative score $\mu_{\Psi,K,N}(\mathcal{I})$ means that the educated predictor is more memory hungry than the uneducated one—i.e., that the SKI approach is not effective in reducing the memory footprint of the input predictor.

### 8.2.3 Energy Consumption

To function effectively in resource-constrained environments, it is required for AI systems to consume a reduced amount of energy. One approach could be to use energy-efficient hardware, such as low-power processors, or to use distributed and federated learning techniques that can distribute computation complexity across multiple devices [Savazzi et al., 2021]. Another approach could exploit more efficient algorithms and data structures so as to reduce the amount of computation required by the AI system to process data. Along that line, the integration of sub-symbolic predictors, which require less memory and computational resources than conventional symbolic AI approaches, could meaningfully reduce energy consumption. However, improvements can still be made from an energy point of view by making the sub-symbolic systems more efficient. For instance, several techniques rely on ad-hoc strategies to compress or optimise sub-symbolic predictors.

In this context, we see SKI approaches as providing human designers with a huge opportunity. In fact, the introduction of injection mechanisms in the data-driven pipeline of sub-symbolic training mechanisms may reduce the amount of computations required to train and run sub-symbolic predictors. Indeed, knowledge injection reduces the complexity of the learning process, providing another source of knowledge other than the training data itself. Thus, one may be interested in assessing whether and to what extent SKI mechanisms contribute to reducing the amount of computations required by a sub-symbolic predictor along its life-cycle.

We propose a new score aimed at measuring the energy consumption of SKI approaches. This is tightly related with memory footprint score from Section 8.2.2. Indeed, it is usually the case for smaller predictors to require fewer amounts of energy to train and run. However, there may exist memory efficient predictors requiring a higher amount of energy to train and run, such as sparse ones. Indeed, sparsity induces a lower amount of operations, but is not usually effectively implemented at hardware level, increasing power consumption [Huang et al., 2018]. Therefore, energy consumption is a property which is worth to be measured by itself.

In order to analyse energy consumption as well as the possible improvements that SKI could bring about, we first need to define the life cycle of AI predictors, analysing hungriness of each component resource. In order to build and deploy a data-driven AI solution, a number of stages need to be completed, namely:

**Model definition,** where data scientists analyse the task at hand and select the most adequate sorts of sub-symbolic predictors, and the most promising hyperparameters assignments for those predictors.

**Model training,** where the parameters of the sub-symbolic predictor of choice are automatically tuned on the training data via some sort of training algorithm. There, the amount of training samples (as well as their dimensionality) may impact energy consumption. Indeed, training algorithms commonly require running the predictor on the data and updating it several times.

**Model testing,** where the predictor is tested against a – limited – set of testing samples to check if the performance are satisfactory. As for the training case, energy consumption here may be affected by the amount (and dimensionality) of testing samples.

**Model deployment,** where the predictor runs multiple times, which a frequency which really depends on the specific application at hand

From the definition of the data-driven AI life-cycle, it is possible to highlight that the *training* and *deployment* phases are the most resource hungry. Indeed, training requires a huge – yet predictable – amount of predictor executions and updates, whereas deployment might be very energy demanding depending on the predictor usage frequency and life expectation—which are typically hard to anticipate. Accordingly, as far as energy consumption is concerned, we are interested in measuring the energy consumption of the training and deployment phases, individually. More precisely, for the training phase, we are interested in measuring the energy consumption of the training algorithm itself, hence excluding the cost of the inferences drawn during the training process—as their cost is expected to be analogous to the one of the deployment phase. Notably, this distinction allows us to evaluate the impact of SKI during both the training and deployment phases—which may, in general, be significantly different. In fact, we expect SKI to decrease the energy consumption of the deployed predictors, at the price of an increased energy consumption of the training phase.

Delving into the details of the energy consumption measurements, we start by defining the $\Upsilon^{\mathrm{i}}$ score, aimed at measuring the average energy consumed by a sub-symbolic predictor $N$ on a per-single-inference basis:

$$\Upsilon^{\mathsf{i}}_{\upsilon}(N, T) = \frac{1}{|T|} \sum_{t \in T} \upsilon(N, t) \tag{8.3}$$

Our definition assumes a function $\upsilon(N, t)$ is available to measure the energy consumption of a single forward run of a sub-symbolic predictor $N$ on a single sample $t$. Such a function may for instance estimate the heat dissipated by the hardware running the predictor, during the single inference $N(t)$. Under that assumption, Equation (8.3) measures the *average* energy consumption of a sub-symbolic predictor $N$ on a test dataset $T$ composed by several samples.

We now define the $\Upsilon^{\mathsf{t}}$ score, aimed at measuring the average energy consumed while training a sub-symbolic predictor $N$ on a training dataset $T$:

$$\Upsilon^{\mathsf{t}}_{\upsilon,\gamma}(e, N, T) = \frac{\gamma(e, N, T)}{e \cdot |T|} - \Upsilon^{\mathsf{i}}_{\upsilon}(N, T) \tag{8.4}$$

Our definition assumes the training involves $e$ epochs, and that during each epoch the whole training set $T$ is used to update the predictor $N$. The definition also assumes $\gamma(e, N, T)$ is a function estimating the overall energy consumed by the training phase as whole—including the energy consumed by the inferences drawn during the training process. Similarly to $\upsilon$, function $\gamma$ may for instance estimate the heat dissipated by the hardware running the predictor, during the whole training process. Under such assumptions, Equation (8.4) measures the *average* energy consumption required by the predictor $N$ for a single update, during its training on the dataset $T$.

We can now define the energy consumption *improvement* of a SKI mechanism as the amount of energy saved by the educated predictor, compared to its uneducated counterpart. Again, we distinguish between energy consumption during training and energy consumption during inference. Along this line, we introduce two scores, namely $\varepsilon^{\mathsf{i}}_{\upsilon,K,N,T}(\mathcal{I})$ (resp. $\varepsilon^{\mathsf{t}}_{\upsilon,\gamma,K,N,T}(\mathcal{I})$), aimed at measuring the energy consumption improvement of a SKI mechanism $\mathcal{I}$, during inference (resp.

training). More formally:

$$\begin{aligned} \varepsilon^{\mathsf{i}}_{\upsilon,K,N,T}(\mathcal{I}) &= \Upsilon^{\mathsf{i}}_{\upsilon}(N,T) - \Upsilon^{\mathsf{i}}_{\upsilon}(\mathcal{I}(K,N),T) \\ \varepsilon^{\mathsf{t}}_{\upsilon,\gamma,e,K,N,T}(\mathcal{I}) &= \Upsilon^{\mathsf{t}}_{\upsilon,\gamma}(e,N,T) - \Upsilon^{\mathsf{t}}_{\upsilon,\gamma}(e,\mathcal{I}(K,N),T) \end{aligned} \tag{8.5}$$

where $\hat{N} = \mathcal{I}(K, N)$ represents the educated predictor attained by injecting $K$ into $N$, and $T$ is a reference dataset of choice—most commonly, the training set in the case of $\varepsilon^{\mathsf{t}}$, and the test set in the case of $\varepsilon^{\mathsf{i}}$.

It is worth noticing how the proposed energy consumption scores may be influenced by all aspects **(A1)**–**(A3)**. About input knowledge **(A1)** the reason is simple: the more complex the input knowledge, the higher (resp. lower) the expected energy consumption of the educated predictor during training (resp. inference). Similarly, as far as the input predictor is concerned **(A2)**, the more energy-hungry it is, the higher we expect the educated predictor's energy consumption improvements to be. Lastly, the task at hand **(A3)** has a clear effect on our scores, as they are both parametric in the dataset—energy consumption improvements are typically task-specific.

### 8.2.4 Latency

The amount of time required to draw a single prediction is one of the most relevant and impactful efficiency measures for sub-symbolic predictors. In what follows, we refer to such time-lapse as *latency*. A small latency indicates that a sub-symbolic predictor is able to compute relevant predictions in useful time—which is an important property in real-world applications. For instance, low latency is essential in those scenarios where human lives depend on the timely response of some AI system, such as intelligent transportation [Grigorescu et al., 2020] and e-health [Esteva et al., 2021]. This is why recent research efforts in the AI field are focussing on time-sensitive predictors.

One possible solution available to address this problem is the use of SKI approaches. By incorporating symbolic representations, SKI approaches can reduce the amount of computations required to process data, leading to reduced latency. Furthermore, the use of symbolic representations could help to simplify the complexity of the system, making it easier to predict the behavior of the system and identify the root causes of an increased latency. As a result, we believe it is crucial to measure latency in order to assess the computational efficiency of present AI systems.

Formally, we define the latency of a predictor $N$ as the average time required to draw a single prediction from a dataset $T$:

$$\Lambda(N, T) = \frac{1}{|T|} \sum_{t \in T} \Theta(N, t) \tag{8.6}$$

where $\Theta(N, t)$ represents the time required to draw a prediction from $N$ on the input $t$.

As far as SKI is concerned, we are interested in assessing the *latency gain* brought by a given SKI mechanism $\mathcal{I}$ w.r.t. some uneducated predictor. Along this line, one may be interested in figuring out whether injection increases or decreases the latency of a given predictor. Hence, we define the latency gain $\lambda_{K,N,T}(\mathcal{I})$ introduced by some SKI method $\mathcal{I}$ as the average difference between the inference time of the educated predictor $\hat{N}$ and its uneducated counterpart $N$, over a reference test dataset $T$. More formally:

$$\lambda_{K,N,T}(\mathcal{I}) = \frac{1}{|T|} \sum_{t \in T} \left(\Theta(N, t) - \Theta(\hat{N}, t)\right) = \Lambda(N, T) - \Lambda(\hat{N}, T) \tag{8.7}$$

where $\hat{N} = \mathcal{I}(K, N)$ represents the educated predictor attained by injecting $K$ into $N$.

Similarly to the energy measurement, the latency metric is tightly related to the complexity of the educated sub-symbolic predictor and therefore with memory footprint. However, like energy consumption, latency is not always directly proportional to the amount of operations that construct the predictor at hand. Sparsely-structured operations might slow down the inference process due to their inefficient computation at hardware level. Moreover, input data complexity and quality might alter the latency achieved by the predictor. Indeed, inference over different – yet structurally analogous – samples may take vastly different timings, as shown in the attack proposed in [Shumailov et al., 2021].

It is worth noticing how the proposed latency score may be influenced all aspects **(A1)**–**(A3)**. About input knowledge **(A1)**, we argue it may have both a positive and a negative effect on the latency gain. In fact, on the one hand, some SKI mechanisms might introduce additional computations—such as the ones required to process the input knowledge $K$ in structuring methods—see Section 7.3.2. We expect this effect to be magnified in the case of large knowledge bases, as the number of operations required to process them is expected to be higher. On the other hand, SKI systems might also reduce the inference time of the given predictor, by reducing the number of computations required to draw a prediction—likely, at the expense of higher training times. As far as the input predictor in concerned **(A2)**, the more time-consuming it is, the higher we expect the educated predictor's latency gain to be. Lastly, the task at hand **(A3)** has a clear effect on our score, as latencies are computed over task-specific test sets.

### 8.2.5 Data Efficiency

Sub-symbolic predictors which rely on data-driven training algorithms, can provide groundbreaking performance and flexibility, but the data-driven procedure comes with several shortcomings. These predictors require collecting significant amounts of data samples for each task to be tackled, leading to increased data storage and processing requirements. Furthermore, not only the quantity but also the *quality* of the data – here intended as its representativeness of the task at hand – is crucial for the predictor to learn effectively. All such requirements make the data collection process time-costly – and depending on the application – possibly affected to subjectivity or uncertainty—e.g., emotion recognition [Deng and Ren, 2021].

For all these reasons, recent research efforts have focused on proposing data-frugal predictors [Sanchez-Iborra and Skarmeta, 2020]. Among them, knowledge injection mechanisms play a significant role [Xu et al., 2018]. Indeed, leveraging a-priori knowledge, SKI relieves the learning process from part of its computational burdens. Concepts that an uneducated predictor would need to learn from data might now be injected into the educated predictor, instead. Hopefully, this would let the educated predictor's learning process require lower amounts of data to attain acceptable performance levels. In this sense, SKI might be considered as a data-efficiency mechanism.

We are here interested in computing the *data-efficiency gain* brought by a given SKI mechanism w.r.t. some uneducated predictor. To do so, we firstly need to define the *data footprint* of a given predictor. Informally, the data footprint of a predictor $N$ is the amount of data it requires to be trained to reach a certain performance level. Hence, assuming that a predictor $N$ is trained on a dataset $D$ – of samples of potentially different dimensions –, via some training process involving $e$ epochs, and that it reached a performance level $\pi(N, T)$ over a test set $T$ – and according to some test dataset $T$ –, we define its data footprint as follows:

$$\Delta_\pi(e, N, D, T) = \frac{e}{\pi(N, T)} \sum_{d \in D} \beta(d) \tag{8.8}$$

where $d$ is a single training sample, and $\beta(d)$ is the amount of bytes required for its in-memory representation, and $\pi$ is some performance score of choice. As the reader may notice, the data footprint is *directly* proportional to the number of epochs $e$, to the size of the training set, and to its dimensionality; whereas it is *inversely* proportional to the performance score of the resulting predictor.

We define the data-efficiency *gain* $\delta_{e,K,N,D,D',T}(\mathcal{I})$ of a given SKI mechanism $\mathcal{I}$ as the difference between the data footprint of the uneducated predictor $N$ – trained upon some dataset $D$ – and that of the educated predictor $\mathcal{I}(K, N)$—trained upon some other dataset $D'$. The score assumes that the two predictors have been trained for the same number of epochs $e$, and that their performance is assessed using the same performance score $\pi$, on the same test set $T$—in order to keep the comparison fair. More formally:

$$\delta_{e,K,N,D,D',T}(\mathcal{I}) = \Delta_\pi(e, N, D, T) - \Delta_\pi(e, \mathcal{I}(K, N), D', T) \tag{8.9}$$

The simplest approach to improve data efficiency in SKI mechanisms is to reduce the amount of samples that compose the training dataset—i.e. $|N|$ in Equation (8.8). However, one may also consider the option of decreasing the size of samples either by reducing their dimensionality or by compressing their representations—in a nutshell, by reducing $\beta(d)$ for all $d \in D$.

To increase the data-efficiency gain, one may also consider engineering SKI and, consequently, the educated predictor. Along this line, the best strategy consist in reducing the size of the training set $D'$ for the educated predictor by letting the input knowledge $K$ compensate for such lack of data. Notably, this is possible because our score is sensitive to both aspects **(A1)** and **(A3)**. In other words, both the input knowledge and the task at hand have a measurable effect on the data-efficiency gain. Finally, as far as the baseline predictor is concerned **(A2)**, we argue that the more data hungry it is, the more the data-efficiency gain will be.

## 8.3 Integration of SKI QoS Metrics into PSyKI

In this section we thoroughly discuss the PSyKI system by first providing the reader with a comprehensive overview of the system, then delving into the specifics of how QoS metrics are integrated into the PSyKI library.

PSyKI – acronym for "Platform for Symbolic Knowledge Injection" – is a Python library that provides support for the injection of prior symbolic knowledge into sub-symbolic predictors by letting the users choose the most adequate method with respect to the ML task to accomplish [Magnini et al., 2022b]. PSyKI is a tool for intelligent systems engineers who need to either experiment with already-existing SKI algorithms or invent new ones.[3]

Currently, PSyKI can be used with predictors created by Tensorflow[4] and supports the following SKI algorithms:

- **KINS**: Knowledge Injection via Network Structuring [Magnini et al., 2022a] is a structuring-based injection mechanism where an ordinary multi-layer neural network is extended with ad-hoc neural modules aimed at mimicking the provided symbolic knowledge. The weights of the neural modules are trained together with the weights of the original neural network.

- **KILL**: Knowledge Injection via Lambda Layer [Magnini et al., 2022c] is a guided-learning-based approach that constrains the training phase as follows.

[3]PSyKI is public and currently available at `https://github.com/psykei/psyki-python`.
[4]`https://www.tensorflow.org`

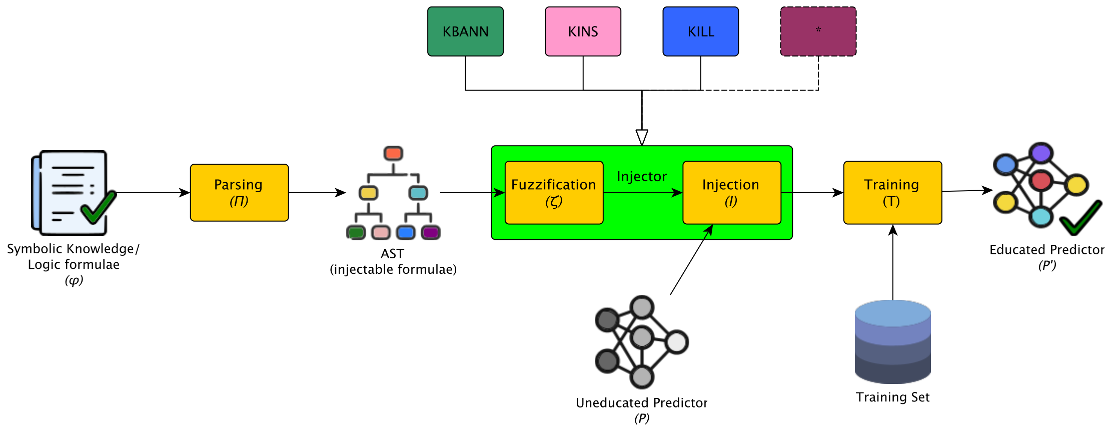


Figure 8.1: PSyKI design. Each SKI algorithm follows the workflow represented in the figure. The four yellow boxes represent the four main steps of the workflow. The first step is the parsing ($\Pi$) of the symbolic knowledge. The second step is the fuzzification ($\zeta$) of the parsed knowledge. The third step is the injection ($\mathcal{I}$) of the fuzzified knowledge into the uneducated target predictor ($P$). The fourth step is the training ($\mathcal{T}$) of the new predictor, making it educated ($P'$).

Before each back-propagation step, a penalty value that represents the degree of violation w.r.t. the input knowledge is added to the loss function. Therefore, the training phase is forced to minimize the loss function and the penalty value at the same time—hence maximising the compliance w.r.t. the knowledge to be injected.

- **KBANN**: Knowledge-Based Artificial Neural Network [Towell et al., 1990] is one of the first structuring-based SKI algorithms proposed in the literature. It creates a NN from a set of propositional rules: each piece of each formula is converted into partial neural structures, to be then composed in a single network. The main difference w.r.t. to KINS is that the network created by KBANN is *entirely* constructed from the knowledge, while KINS extends an existing network with ad-hoc modules.

Essentially, PSyKI is designed around the notion of *injector*, whose block diagram is shown in Figure 8.1. An injector is any algorithm accepting as input a ML predictor and prior symbolic knowledge (typically logic formulæ) and producing a new predictor as output. In order to properly perform injection, injectors may require additional information, such as algorithm specific hyperparameters.

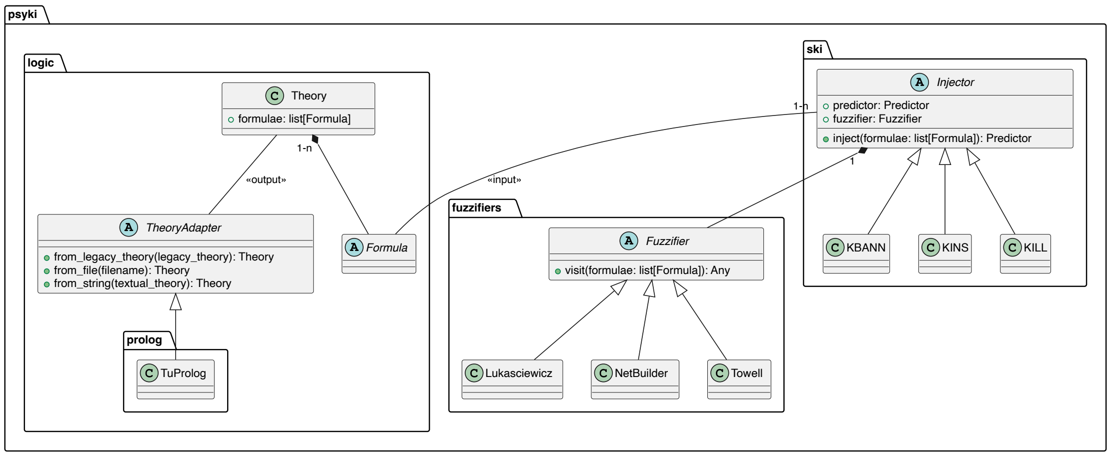


Figure 8.2: Class diagram of PSyKI. Main entities are `Injector`, `Formula`, and `Fuzzifier`.

PSyKI supports the processing of symbolic knowledge represented via logic formulæ. Based on the sort of logic adopted, user can build an abstract syntax tree (AST) for each formula. The AST can be inspected through a *fuzzifier* via pattern visitor [Gamma et al., 1993] to encode the symbolic knowledge into a sub-symbolic form (e.g. fuzzy logic functions, ad-hoc layers). The resulting sub-symbolic object can finally be used by an injector to create a new predictor. This process – denoted with $\zeta$ in Figures 7.8, 7.9, 7.10 – is injector-specific; instead, the same parser $\Pi$ can be used independently of the injector for logic formulæ of the same type.

The software is organised into well-separated packages and interfaces, so as to ensure extensibility towards new sorts of logics and fuzzifiers—see Figure 8.2. A formula AST is represented in the software via instances of the `Formula` abstract class and its manifold subtypes (not shown in the figure)—aimed at covering the many logic-specific aspects supported by PSyKI. Ad-hoc implementations of `Formula` are included in PSyKI, one for each the logic formalism supported by the framework – currently, Prolog, Datalog, and their sub-sets –, and more may be introduced in the future by interested researchers. The same holds for fuzzifiers (resp. injectors), i.e., sub-types of the `Fuzzifier` (resp. `Injector`) abstract class.

However, in its original state PSyKI does not include any particular facility to assess SKI. This is why in the remainder of this chapter we propose a PSyKI extension aimed at supporting engineers in need of practically assessing the efficientisation benefits of their SKI workflows.

### 8.3.1 QoS Metrics Implementation in PSyKI

QoS metrics are implemented as a set of classes that extend the `Metric` abstract class. Each class corresponds to a specific metric and is responsible for computing the corresponding score. Therefore, the `Metric` class provides a common interface for all metrics.

In particular, it provides two methods to compute the metric value between two predictors. The first method is `compute_during_training` and it is used to compute the metric during the training phase of the predictors. The second method is `compute_during_inference` and it is used to compute the metric when predictors are already trained. Both methods, accept the predictors to compare as input parameters. Additional parameters can be passed to the methods to customise the computation of the metric to meet the specific needs of the user (e.g., training set, batch size, etc.).

Implemented metrics are:

1. `Memory`: memory consumption efficiency of the predictors—Equation (8.2);
2. `Energy`: energy consumption efficiency of the predictors—Equation (8.5);
3. `Latency`: latency efficiency of the predictors—Equation (8.7);
4. `DataEfficiency`: data efficiency of the predictors—Equation (8.9).

Metrics are included into the `psyki.qos` package. It is worth noting that all the metrics can be computed using any kind of predictors: no need to have one uneducated and one educated predictor. Instead, one can also compare, say, two educated predictors, or two uneducated predictors of any sort.

## 8.4 Experiments

In this section we present several experiments aimed at assessing the effectiveness of the proposed QoS metrics, as implemented in PSyKI. We first describe the

| **Symbol** | **Adenine** | **Cytosine** | **Guanine** | **Thymine** |
|---|---|---|---|---|
| `d` | ● | | ● | ● |
| `m` | ● | ● | | |
| `r` | ● | | ● | |
| `s` | | ● | ● | |
| `y` | | ● | | ● |

Table 8.1: Mapping of aggregative symbols and the four nucleotides. Each symbol can be substituted with one base on the right that has a dot.

experimental setup, the datasets we adopt, and the rationale behind their choice. Then, we present the results of our experiments, and we discuss them.

The design of our experiments is as follows:

1. we select three relevant classification tasks from the literature, covering different application domains, and coming with datasets of increasing cardinality;
2. for each task and its corresponding dataset $D$, we *(i)* train some uneducated neural predictor $N$ over the data in $D$ – of course performing train/test-set splitting –, and we *(ii)* select some symbolic knowledge base $K$ to be injected in $N$;
3. for each uneducated predictor $N$ we then apply SKI multiple times, one per each injection technique currently supported by PSyKI, namely KBANN, KINS, and KILL—hence attaining as many educated predictors;
4. finally, for each educated predictor $\hat{N}$, we compute our QoS metrics, hence comparing that $\hat{N}$ and $N$ w.r.t. *(i)* data efficiency, energy consumption, memory footprint, latency, and accuracy variation.

The rationale behind this setup is to demonstrate the effectiveness of our QoS metrics in assessing the efficiency SKI techniques of different sorts.

### 8.4.1 Datasets

We select three different datasets from the UCI repository[5]: BCW, PSJGS, and CI.

[5]`https://archive.ics.uci.edu/ml/index.php`

**Breast cancer Wisconsin dataset (BCW)** [Wolberg, 1992] The BCW dataset contains 699 instances of breast cancer biopsy results, each with 9 features – summarising biological characteristics – and one class label. Values are integers in the range $[1, 10]$. The feature *BareNuclei* has 16 missing values, which are replaced with the value zero. The dataset's target variable is a binary indicator of whether a biopsy was benign (`B`) or malignant (`M`), class repartition is 458 and 241 respectively. The purpose of the dataset is to develop predictors that can accurately diagnose breast cancer based on biopsies using the information contained in the features.

**Primate splice junction gene sequences (PSJGS)** [Towell and Shavlik, 1994] The PSJGS dataset includes information regarding gene splicing. The dataset includes 3,190 instances, each representing a sequence of 60 DNA nucleotides. Each nucleotide is represented by one of the four letters `A` (adenine), `T` (thymine), `C` (cytosine), and `G` (guanine). Each sequence begins at position -30 and ends at 30, position zero is excluded.

One DNA sequence can be classified as an exon–intron (`EI`) boundary, an intron–exon (`IE`) boundary, or none (`N`) of them. Class frequencies are 50% for `N`, 25% for both `EI` and `IE`.

In addition to the four nucleotides, the dataset also includes other letters that indicate that for one specific position different nucleotides are allowed. For our experiments, we preprocess the dataset by binarising the nucleotides. In other words, each nucleotide is represented by a vector of 4 elements, where each element is 0 except for the one corresponding to the nucleotide itself, which is 1. Table 8.1 reports the complete binarisation of the nucleotides.

**Census income (CI)** [Kohavi and Becker, 1996] The CI dataset contains individual information from the 1994 United States Census. The dataset contains 48,842 instances, each corresponding to one census participant. Each data row includes information such as age, education, and occupation, as well as income data about a single person. The purpose of the dataset is to predict whether an individual's annual income is greater than or less/equal than/to 50,000 USD based on their demographic information. Hence, the target variable is binary—37,155 earn less/equal than/to 50,000 USD and 11,687 earn more than that amount per year.

For our experiments, we convert the target *Income* to a binary output (1 if the income exceeds 50,000 USD and 0 otherwise). We also drop three features – namely *Fnlwgt*, *Education*, and *Race* – as they are irrelevant for our experiment (*Fnlwgt* is a similarity metric computed over the other features, the information provided by *Education* is already present thanks to the feature *EducationNumeric*) or possibly introduce cultural bias (*Race*). The remaining features are discretised. In particular, *CapitalGain* and *CapitalLoss* are binarised, while the remaining nominal categorical features are transformed into one-hot-encoded data.

We choose these datasets because of their increasing cardinality, which ranges from $10^2$ to $10^4$. In this way, we are able to observe the scalability and robustness of our predictors and metrics in handling datasets of different volume or dimensionality. This is important to get a broader overview about the performance of the different predictors both in terms of their accuracy and in terms of the various efficiency metrics proposed in this work.

We divide each dataset into train and test sets, with a ratio of 2/3 and 1/3 respectively.

Finally, we attain the knowledge bases to be injected in a task-specific way. As far as the PSJGS dataset is concerned, we rely on the knowledge base described into the corresponding paper [Towell et al., 1990], which we suitably convert in Prolog form. Conversely, as far as the BCW and CI datasets are concerned, we leverage upon symbolic knowledge *extraction* [Sabbatini et al., 2022] to automatically generate knowledge bases in Prolog form out of trained predictors. Table 8.2 lists the logic rules that constitute the symbolic knowledge used in the SKI algorithms for the breast cancer dataset. A similarly-shaped knowledge has been used also in the SKI algorithms for the census income dataset.

## 8.4.2 Methodology

We define and train several neural predictors, for each dataset—in particular, one uneducated network and multiple educated counterparts. We attain educated

Table 8.2: Knowledge used for the breast cancer dataset.

| **Symbolic knowledge (Prolog formalism)** |
|---|
| *diagnosis*(*BareNuclei*, *BlandChromatin*, *ClumpThickness*, *MarginalAdhesion*, *Mitoses*, *NormalNucleoli*, *SingleEpithelialCellSize*, *UniformityCellShape*, *UniformityCellSize*, `benign`) :− *UniformityCellSize* < 3, *NormalNucleoli* > 2, *BareNuclei* < 1. |
| *diagnosis*(*BareNuclei*, *BlandChromatin*, *ClumpThickness*, *MarginalAdhesion*, *Mitoses*, *NormalNucleoli*, *SingleEpithelialCellSize*, *UniformityCellShape*, *UniformityCellSize*, `malignant`) :− *UniformityCellSize* < 3, *NormalNucleoli* > 2, *BareNuclei* > 1. |
| *diagnosis*(*BareNuclei*, *BlandChromatin*, *ClumpThickness*, *MarginalAdhesion*, *Mitoses*, *NormalNucleoli*, *SingleEpithelialCellSize*, *UniformityCellShape*, *UniformityCellSize*, `benign`) :− *UniformityCellSize* > 3, *BareNuclei* < 0. |
| *diagnosis*(*BareNuclei*, *BlandChromatin*, *ClumpThickness*, *MarginalAdhesion*, *Mitoses*, *NormalNucleoli*, *SingleEpithelialCellSize*, *UniformityCellShape*, *UniformityCellSize*, `malignant`) :− *UniformityCellSize* > 3, *BareNuclei* > 0, *BlandChromatin* > 4. |
| *diagnosis*(*BareNuclei*, *BlandChromatin*, *ClumpThickness*, *MarginalAdhesion*, *Mitoses*, *NormalNucleoli*, *SingleEpithelialCellSize*, *UniformityCellShape*, *UniformityCellSize*, `malignant`) :− *UniformityCellSize* > 3, *BareNuclei* > 0, *BlandChromatin* < 4, *UniformityCellSize* < 4, *NormalNucleoli* < 1. |
| *diagnosis*(*BareNuclei*, *BlandChromatin*, *ClumpThickness*, *MarginalAdhesion*, *Mitoses*, *NormalNucleoli*, *SingleEpithelialCellSize*, *UniformityCellShape*, *UniformityCellSize*, `benign`) :− *UniformityCellSize* > 3, *BareNuclei* > 0, *BlandChromatin* < 4, *UniformityCellSize* < 4, *NormalNucleoli* > 1. |
| *diagnosis*(*BareNuclei*, *BlandChromatin*, *ClumpThickness*, *MarginalAdhesion*, *Mitoses*, *NormalNucleoli*, *SingleEpithelialCellSize*, *UniformityCellShape*, *UniformityCellSize*, `malignant`) :− *UniformityCellSize* > 3, *BareNuclei* > 0, *BlandChromatin* < 4, *UniformityCellSize* > 4, *MarginalAdhesion* > 1. |
| *diagnosis*(*BareNuclei*, *BlandChromatin*, *ClumpThickness*, *MarginalAdhesion*, *Mitoses*, *NormalNucleoli*, *SingleEpithelialCellSize*, *UniformityCellShape*, *UniformityCellSize*, `malignant`) :− *UniformityCellSize* > 3, *BareNuclei* > 0, *BlandChromatin* < 4, *UniformityCellSize* > 4, *MarginalAdhesion* < 1, *NormalNucleoli* < 3. |
| *diagnosis*(*BareNuclei*, *BlandChromatin*, *ClumpThickness*, *MarginalAdhesion*, *Mitoses*, *NormalNucleoli*, *SingleEpithelialCellSize*, *UniformityCellShape*, *UniformityCellSize*, `benign`) :− *UniformityCellSize* > 3, *BareNuclei* > 0, *BlandChromatin* < 4, *UniformityCellSize* > 4, *MarginalAdhesion* < 1, *NormalNucleoli* > 3. |
| *diagnosis*(*BareNuclei*, *BlandChromatin*, *ClumpThickness*, *MarginalAdhesion*, *Mitoses*, *NormalNucleoli*, *SingleEpithelialCellSize*, *UniformityCellShape*, *UniformityCellSize*, `benign`) :− *NormalNucleoli* < 2, *BareNuclei* < 4. |
| *diagnosis*(*BareNuclei*, *BlandChromatin*, *ClumpThickness*, *MarginalAdhesion*, *Mitoses*, *NormalNucleoli*, *SingleEpithelialCellSize*, *UniformityCellShape*, *UniformityCellSize*, `malignant`) :− *BareNuclei* > 4, *SingleEpithelialCellSize* < 1. |
| *diagnosis*(*BareNuclei*, *BlandChromatin*, *ClumpThickness*, *MarginalAdhesion*, *Mitoses*, *NormalNucleoli*, *SingleEpithelialCellSize*, *UniformityCellShape*, *UniformityCellSize*, `benign`) :− *SingleEpithelialCellSize* > 1. |

networks by applying SKI via the *KINS*, *KILL*, and *KBANN* algorithms—each one exploiting some different approach to perform knowledge injection—see Section 8.3. By constructing all such predictors, we are able to compare and evaluate their performance and their metrics on each dataset.

For each uneducated predictor, we tune the structural hyperparameters (i.e. amount of layers and neurons per layer) by using a grid search with cross-validation. Networks attained via *KBANN* are a notable exception here, as in those cases the entire architecture of the network is dictated by KBANN, as a function of the input knowledge. In particular, we chose to vary the number of layers (from 1 to 3) and the number of neurons per layer (10, 50, and 100). The same process of grid search with cross cross-validation is repeated for the "educated" predictors. In this way, we can ensure good hyperparameters selection – in terms of predictive

Table 8.3: Results of a comprehensive grid search on various datasets. The models evaluated in this study are the uneducated one, and the three different educated models with KBANN, KILL, and KINS. Note that the model used by KBANN is identical to the model obtained via grid search for the uneducated model. The table provides a summary of the number of layers and neurons used in each model for each dataset.

| Dataset | Model | Layers | Neurons |
|---|---|---|---|
| BCW | Uneducated | 3 | [100, 10, 50] |
| | KBANN | 3 | [100, 10, 50] |
| | KILL | 3 | [100, 10, 50] |
| | KINS | 3 | [100, 10, 10] |
| PSJGS | Uneducated | 3 | [100, 10, 10] |
| | KBANN | 3 | [100, 10, 10] |
| | KILL | 3 | [100, 10, 10] |
| | KINS | 3 | [50, 10, 10] |
| CI | Uneducated | 3 | [10, 50, 50] |
| | KBANN | 3 | [10, 50, 50] |
| | KILL | 3 | [10, 50, 10] |
| | KINS | 3 | [10, 50, 10] |

performance –, while still keeping the computation time reasonable. Table 8.3 shows the selected hyperparameters for each dataset and predictor.

In order to tune the (hyper-)parameters of each predictor in a statistically significant way, we repeat the training 30 times, each time with different initial conditions and/or random seeds, grasping statistics about the average accuracy along the way. This lets us reduce the variability of the results and obtain a more accurate estimate of a predictor's actual performance. The outcome of this procedure is shown in Table 8.4.

After calculating the average accuracy, we proceed in computing predictors' efficiency metrics, for each dataset. In particular, we compute data efficiency, energy, memory, and latency metrics—see Section 8.2. The corresponding scores are presented in Table 8.4, and discussed in the following section.

Table 8.4: Comparison of the performance of different models (KBANN, KILL, and KINS) respect to the uneducated one on three datasets (Breast Cancer, Splice Junction, and Census Income) in terms data efficiency, energy consumption, memory usage, latency, and accuracy. Note that the train accuracy is the mean of 30 runs.

| Dataset | Model | Set | Data efficiency (KB) | Energy (mWh) | Memory (FLOPs) | Latency (ms) | Accuracy (% ) |
|---|---|---|---|---|---|---|---|
| BCW | uneducated | | – | – | – | – | 94.53 |
| | KBANN | train | 35.89 | -1.47 | 3933 | -1.70 | 95.45 |
| | | test | | -0.10 | | | |
| | KILL | train | 4.09 | -0.99 | 0 | 0.35 | 94.63 |
| | | test | | 0 | | | |
| | KINS | train | -9.97 | -1.22 | -559 | -1.41 | 94.29 |
| | | test | | -0.09 | | | |
| PSJGS | uneducated | | – | – | – | – | 93.91 |
| | KBANN | train | -4946.81 | -4.67 | -66944 | -2.56 | 92.84 |
| | | test | | -0.22 | | | |
| | KILL | train | 553.89 | -3 | 0 | 0.04 | 94.02 |
| | | test | | 0 | | | |
| | KINS | train | -954.80 | -6.53 | -161779 | -4.75 | 93.70 |
| | | test | | -0.51 | | | |
| CI | uneducated | | – | – | – | – | 84.63 |
| | KBANN | train | 1653.79 | -1.41 | -2468 | -0.43 | 84.78 |
| | | test | | -0.02 | | | |
| | KILL | train | 4016.90 | -0.70 | 4200 | 0 | 84.81 |
| | | test | | 0 | | | |
| | KINS | train | 4263.50 | -1.41 | -6220 | -0.44 | 84.77 |
| | | test | | -0.02 | | | |

### 8.4.3 Discussion

In the following we thoroughly analyse and interpret the results of our experiments. Accordingly, we examine the columns of Table 8.4 from left to right.

It is worth noticing how data-efficiency scores vary hugely across predictors and datasets. We recall that a positive data-efficiency score indicates that the educated predictor is more efficient than its uneducated counterpart, whereas a negative score indicates the opposite. In general, as stated in Section 8.2, it is important to consider how data-efficiency scores can be affected by all three aspects **(A1)**–**(A3)**. Thus, for instance, the high variation of this score points out the importance of selecting the most appropriate predictor for a given task **(A3)**. For instance, the KINS-based solution has a lower data-efficiency score than the other predictors tailored on the BCW dataset. This may indicate that KINS is not the best solution for this task. In contrast, we note that the CI dataset shows positive data-efficiency scores for all three predictors, indicating that, in terms of data efficiency, an improvement is obtained by using all three SKI algorithms proposed in this work.

The second column of Table 8.4 shows the energy metrics for both train and test. With regard to each predictor and dataset, we mostly see negative values for this metric. Again, it is important to note that energy consumption scores can be affected by a number of factors, including input knowledge **(A1)**, input predictor **(A2)**, and task to be performed **(A3)**. In most cases, the table indicates that the KBANN-based solution consumes more energy than the other predictors. In contrast, the KILL-based solution consumes significantly less energy than the other predictors. Additionally, it is important to shift the emphasis towards input knowledge **(A1)**. As stated in Section 8.2, it is expected that the more complex the input knowledge, the more energy the educated predictor will consume during training. Hence, in terms of data efficiency, we argue that more complex knowledge may produce a gain for the educated predictor—possibly at the price of higher expenses in terms of energy consumption.

The third column of Table 8.4 shows the results of the memory metric. We recall that a positive value here indicates that the educated predictor consumes less memory than the uneducated one. Conversely, a negative value indicates that the educated predictor consumes more memory. For example, in the case of the BCW dataset and the KBANN-based solution, the educated predictor shows a positive difference in memory consumption—which means it uses less memory than the uneducated one. In the PSJGS dataset, both KBANN- and KINS-based solutions show negative memory metrics. This suggests that, in this case, those educated predictors are more memory intensive than the uneducated one. Regarding the KILL-based solution, it often shows a memory metric of 0, indicating that there is no difference in memory between the educated and uneducated predictors.

The fourth column of Table 8.4 shows the latency results. Comparing the latency of educated predictors with the uneducated ones, we observe that, as far as KILL is concerned, the results between the two solutions are very similar—i.e., the metric is close to 0 in both cases. KBANN and KINS, on the other hand, have a slightly-worse latency, on all three datasets. As discussed in Section 8.2, we argue this is due to the complexity of the injected input knowledge, which can lead to negative effects in terms of latency—especially in structuring-type SKI methods, such as KBANN.

Finally, by looking at the accuracy scores – see the last column of the Table 8.4 –, we observe that the educated and uneducated predictors show very similar results. Furthermore, the results indicate that the accuracy of all predictors on all three datasets is very similar and consistent. In general, results suggest that all predictors perform well and can accurately predict the results of the datasets.

To conclude, in terms of data efficiency, the educated predictor generally requires less data to achieve similar accuracy than the uneducated one. This is a positive result, as it suggests that the trained predictor is able to make accurate predictions using less data, which can be a nice-to-have feature in resource-constrained settings.

As far as energy is concerned, our results show a gain in energy during the training phase for the uneducated predictor, but during the inference phase this difference is close to 0. We argue that this is due to the knowledge injection process, which in these experiments required more energy expense for the educated predictor than the uneducated one. About memory, the results are somehow mixed: the educated predictor sometimes requires more memory and sometimes less memory than the uneducated one. Finally, as far as latency is concerned, results indicate that the uneducated predictor tends to have a slightly lower latency than the educated one.

## Chapter Synopsis

In this chapter we propose a set of quality-of-service (QoS) metrics for SKI mechanisms, focusing on the efficiency gains achievable through SKI. Along this line, we formally define four metrics, namely: *(i) memory footprint efficiency*—i.e., gain in terms of predictor's complexity; *(ii) energy efficiency*—i.e., gain in terms of total energy required to train and deploy a sub-symbolic predictor; *(iii) latency efficiency*—i.e., improvements in terms of time required for inference; and *(iv) data efficiency*—i.e., improvement in terms of amount of data required to optimise a sub-symbolic predictor.

Enabled by PSyKI, we perform a number of experiments aimed at demonstrating the effectiveness of our metrics. Overall, our experiments show that the proposed metrics can be exploited to grasp insights about whether a given SKI mechanism is actually able to improve the efficiency of a given predictor or not. As a by-product of our experiments we also show that SKI approaches sometimes help reduce the amount of computations required to process data, leading to *improved data efficiency* and *reduced memory footprint*.

# Chapter 9

# Can Symbolic Knowledge Extraction Help Efficientisation?

This chapter contains results of the work done during my visiting at TU Delft and published in [Agiollo et al., 2023b].

SKI techniques have been proven useful for achieving efficiency gains by reducing the learning burden via the straight away integration of the very knowledge that predictors would otherwise struggle to learn from huge amounts of data. On the other hand, theoretically speaking, SKE approaches can be used to extract a surrogate model whose complexity can be limited, thus obtaining a more efficient and resource-friendly version of the original sub-symbolic predictor. However, this assumption concerning SKE approaches has not been tested so far in the literature, as most SKE works focus on the *fidelity* of the surrogate model. Indeed, SKE approaches are affected by a *fidelity vs. complexity* trade-off. To extract a high fidelity surrogate model requires increasing its complexity, allowing it to take into account all the intricacies of the sub-symbolic model to be replicated. Therefore, it represents an open challenge to meaure if – and to what extent – SKE techniques can be leveraged to achieve efficiency improvements over NNs.

Motivated by this insight we here propose to measure directly SKE efficiency. We focus on the Natural Language Processing (NLP) domain, as it represents the scenario where it is common for very large NN models to be used to tackle quite simple classification tasks. In this context, we propose building *global post-hoc explainers* from the output of a Local Post-hoc Explainer (LPE). We leverage a neuro-symbolic process [Kautz, 2022, Agiollo et al., 2023a, Agiollo and Omicini, 2023, Agiollo et al., 2022], aiming at extracting the Large Language Model (LLM) knowledge under the form of a logic program equivalent to the sub-symbolic model at hand, similarly to [Calegari and Federico, 2022, Sabbatini et al., 2022]. More in detail, the presented knowledge extraction framework – namely, Global Explanations from Local Post-hoc Explainers (GELPE) – relies on the LPE's outputs to identify the set of most relevant components in sentences, and optimise a transparent-by-design – such as Classification And Regression Tree (CART) – surrogate model to mimic the LLM predictions. Once the transparent model is optimised, an equivalent logic program is extracted from the model, allowing for the inspection of the global reasoning process of the LLM. Identifying small and efficient surrogate programs over several tasks, the proposed framework enables the deployment of intelligent techniques over resource-constrained environments where LLMs represent a limited solution [Sarkar et al., 2023, Agiollo and Omicini, 2021].

We test the proposed framework over a large set of text classification domains, ranging from simple scenarios – e.g., spam text classification [Almeida et al., 2011, Alberto et al., 2015] – to challenging tasks such as the Moral Foundation Twitter Corpus (MFTC) [Hoover et al., 2020]. As the performance of the proposed approach largely depends on the LPE technique used, we first consider analysing how different LPEs correlate with each other. Surprisingly, our experiments show how the explanations of different LPEs are far from being correlated, highlighting how explanation quality is highly dependent on the chosen eXplainable AI (XAI) approach and the respective scenario at hand. There are huge discrepancies in the results of different state-of-the-art local explainers, each of which identifies a set of relevant concepts that largely differs from the others—at least in terms of relative impact scores. These results highlight the fragility of XAI approaches for NLP, caused mainly by the complexity of large NN models, their inclination to extreme fitting of data and the lack of sound techniques for comparing XAI mechanisms.

Under the efficientisation perspective, the proposed experiments also highlight how GELPE enables the extraction of reliable surrogate logic programs from LLMs with high fidelity over a broad set of datasets. The extracted knowledge is not only faithful to the original model, but also quite simple, as the complexity of the logic program is kept bounded depending on the number of relevant lemmas selected. Throughout our experimental evaluation, we analyse the computation requirements of the proposed extraction process and the efficiency of the extracted logic program. Numerical results highlight the efficiency of the extracted surrogate model, improving over the original LLM in terms of required processing time and consumed energy. The results show how the proposed framework enables the deployment of intelligent solutions over resource-constrained environments via identifying transparent surrogate models. Also, we highlight that leveraging on LLMs to tackle a learning task in NLP does not always represent the best option, as alternative equivalent solutions that are simple, small and transparent are actually available.

## 9.1 Background: Explanations in NLP

The set of explanations extraction mechanisms available in the XAI community are often categorised along two main axis [Guidotti et al., 2018, Adadi and Berrada, 2018]: *(i) local* against *global* explanations, and *(ii) self-explaining* against *post-hoc* approaches. In the former context, *local* identifies the set of explainability approaches that given a single input produce an explanation of the reasoning process followed by the NN model to output its prediction for the given input [Luo et al., 2021]. In contrast, *global* explanations aim at expressing the reasoning process of the NN model as a whole [Hailesilassie, 2016, Ibrahim et al., 2019]. Given the complexity of the NN models leveraged for tackling most NLP tasks, it is worth noticing how there is a significant lack of *global* explainability systems, whereas a variety of *local* XAI approaches are available [Lundberg and Lee, 2017, Ribeiro et al., 2016].

About the latter aspect, we define *post-hoc* as those set of explainability approaches which apply to an already optimised black-box model for which it is required to obtain some sort of insight [Madsen et al., 2022]. Therefore, a *post-hoc* approach requires additional operations to be performed after that the model outputs its predictions [Danilevsky et al., 2020]. Conversely, inherently explainable – *self-explaining* – mechanisms aim at building a predictor having a transparent reasoning process by design—e.g., CART [Loh, 2014]. Therefore, a self-explaining approach can be seen as generating the explanation along with its prediction, using the information emitted by the model as a result of the prediction process [Danilevsky et al., 2020].

In the context of *local post-hoc* explanation approaches, a popular solution in NLP is to explain the reasoning process by highlighting how different portions of the input sample impact differently the produced output, by assigning a relevance score to each input component. The relevance score is then highlighted by using some saliency map to ease the visualisation of the obtained explanation. Therefore, it is also common for local post-hoc explanations to be referred to as *saliency* approaches, as they aim at highlighting salient components.

## 9.2 Methodology

In this section, we present our methodology for comparing LPE mechanisms and building global explanations from LPE's outputs. We first overview the proposed approach in Section 9.2.1. Subsequently, the set of LPE mechanisms adopted in our experiments are presented in Section 9.2.2, and the aggregation approaches leveraged to obtain global impact scores from LPE outputs are described in Section 9.2.3. In Section 9.2.4 we present the metrics used to identify the correlation between LPEs. Finally, in Section 9.2.5 we propose GELPE as a novel methodology to build global explanations of LLMs on top of LPE's outputs.

### 9.2.1 Overview

Measuring different LPE approaches over single local explanations is a complex task. This is why we first consider measuring how much LPEs correlate with each other over a set of fixed samples. The underlying assumption of our framework is that various LPE techniques aim at explaining the same NN model used for prediction. Therefore, while explanations may differ over local samples, one could reasonably assume that reliable LPEs when applied over a vast set of samples – sentences or set of sentences – should converge to similar (correlated) results. Indeed, the underlying LLM considers being relevant for its inference always the same set of concepts—lemmas. A lack of correlation between different LPE mechanisms would hint to the existence of a conflict among the set of concepts that each explanation mechanisms consider as relevant for the LLM—thus making at least one, if not all, of the explanations unreliable.

We first analyse the correlation between a set of LPEs over the same pool of samples, and define $\epsilon_{NN}$ as a LPE technique applied to a NN model at hand. Being local, $\epsilon_{NN}$ is applied to the single input sample $\mathbf{x}_i$, producing as output one impact score for each component (token) of the input sample $l_k$. Throughout the remainder of the chapter, we consider $l_k$ to be the lemmas corresponding to the input components. Mathematically, we define the output impact score for a single token or its corresponding lemma as $j\left(l_k, \epsilon_{NN}(\mathbf{x}_i)\right)$. Depending on the given $\epsilon_{NN}$, the corresponding impact score $j$ may be associated with a single label, making $j$ a scalar value, or with a set of labels, making $j$ a vector—one scalar value for each label. To enable the comparison between different LPE, we define the aggregated impact scores of a LPE mechanism over a NN model and a set of samples $\mathcal{S}$ as $\epsilon_{NN}(\mathcal{S})$. In our framework we obtain $\epsilon_{NN}(\mathcal{S})$ aggregating $\epsilon_{NN}(\mathbf{x}_i)$ for each $\mathbf{x}_i \in \mathcal{S}$ using an aggregation operation $\mathcal{A}$—mathematically:

$$\epsilon_{NN}\left(\mathcal{S}\right) = \mathcal{A}\left(\{\epsilon_{NN}(\mathbf{x}_i) \text{ } \textit{for each } \mathbf{x}_i \in \mathcal{S}\}\right). \tag{9.1}$$

By defining a correlation metric $\mathcal{C}$, we obtain from Equation (9.1) the following for describing the correlation between two LPE techniques:

$$\begin{aligned}\mathcal{C}\left(\epsilon_{NN}\left(\mathcal{S}\right), \epsilon'_{NN}\left(\mathcal{S}\right)\right) = \mathcal{C}\big(&\mathcal{A}\left(\{\epsilon_{NN}(\mathbf{x}_i) \text{ } \textit{for each } \mathbf{x}_i \in \mathcal{S}\}\right), \\ &\mathcal{A}\left(\{\epsilon'_{NN}(\mathbf{x}_i) \text{ } \textit{for each } \mathbf{x}_i \in \mathcal{S}\}\right)\big)\end{aligned} \tag{9.2}$$

where $\epsilon_{NN}$ and $\epsilon'_{NN}$ are two LPE techniques applied to the same NN model.

The aggregated explanations $\epsilon_{NN}\left(\mathcal{S}\right)$ obtained from LPE's outputs can also be leveraged as a starting point for building transparent surrogate models of the original LLM, as they highlight the impact of each lemma or token in the LLM decision process. Constructing a transparent surrogate model allows for extracting explanations of the *global* reasoning process of the black-box LLM, enabling knowledge extraction, model debugging, and interaction with a human user. To this extent, we here propose GELPE as a novel framework for constructing a logic program

– represented as a set of Prolog clauses – that mimics the LLM behaviour starting from a set of locally relevant lemmas $\epsilon_{NN}(\mathcal{S})$. More in detail, GELPE relies on transparent-by-design models such as CART optimised over the LLM outputs, rather than the dataset considered. The input sentences are converted in a binary format, expressing the presence or absence of relevant lemmas and their combinations. The binarised input is used to optimise the underlying CART model, from which it is possible to extract the equivalent logic program $\mathcal{P}$. Mathematically, we represent the knowledge extraction procedure as:

$$\mathcal{P} = \mathcal{H}\left\{(bin_{\epsilon_{NN}(\mathcal{S})}(\mathbf{x}_i),\ NN(\mathbf{x}_i))\ \forall\ \mathbf{x}_i \in \mathcal{S}\right\}, \tag{9.3}$$

where $\mathcal{H}$ identifies the transparent-by-design models used to extract the explanations logic program $\mathcal{P}$, $bin_{\epsilon_{NN}(\mathcal{S})}$ represents the binarization process used to convert the sentence $\mathbf{x}_i$ into a corresponding binary vector of lemmas occurences and $NN(\mathbf{x}_i)$ identifies the output of the LLM when fed with input sentence $\mathbf{x}_i$.

### 9.2.2 Local Post-hoc Explanations

In our framework, we consider seven different LPE approaches for extracting local explanations $j\,(l_k, \epsilon_{NN}(\mathbf{x}_i))$ from an input sentence $\mathbf{x}_i$ and the trained LLM—identified as $NN$. The seven LPEs are selected in order to represent as faithfully as possible the state-of-the-art of XAI approaches in NLP. Subsequently, we briefly describe each of the seven selected LPEs. However, a detailed analysis of these LPEs is out of the scope of this thesis and we refer interested readers to [Samek et al., 2021, Danilevsky et al., 2020, Luo et al., 2021].

#### Gradient Sensitivity Analysis (GS)

The Gradient Sensitivity Analysis (GS) probably represents the simplest approach for assigning relevance scores to input components. GS relies on computing gradients over inputs components as $\frac{\delta f_c(\mathbf{x}_i)}{\delta \mathbf{x}_{i,k}}$, which represents the derivative of the output with respect to the the $k^{th}$ component of $\mathbf{x}_i$. Following this approach local impact scores of an input component can be thus defined as:

$$j\left(l_k, \epsilon_{NN}(\mathbf{x}_i)\right) = \frac{\delta f_{\tau_m}(\mathbf{x}_i)}{\delta \mathbf{x}_{i,k}}, \tag{9.4}$$

where $f_{\tau_m}(\mathbf{x}_i)$ represents the predicted probability distribution of an input sequence $\mathbf{x}_i$ over a target class $\tau_m$. While simple, GS has been shown to be an effective approach for understanding approximate input components relevance. However, this approach suffers from a variety of drawbacks, mainly linked with its inability to define negative contributions of input components for a specific prediction—i.e., negative impact scores.

### Gradient × Input (GI)

Aiming at addressing few of the limitations affecting GS, the Gradient × Input (GI) approach defines the relevance scores assignment as GS multiplied – element-wise – with $\mathbf{x}_{i,k}$ [Kindermans et al., 2019]. Therefore, mathematically speaking, GI impact scores are defined as:

$$j\left(l_k, \epsilon_{NN}(\mathbf{x}_i)\right) = \mathbf{x}_{i,k} \cdot \frac{\delta f_{\tau_m}(\mathbf{x}_i)}{\delta \mathbf{x}_{i,k}}, \tag{9.5}$$

where notation follows the one of Equation (9.4). Being very similar to GS, GI also inherits most of its limitations.

### Layer-wise Relevance Propagation (LRP)

Building on top of gradient-based relevance scores mechanisms – such as GS and GI –, Layer-wise Relevance Propagation (LRP) proposes a novel mechanism relying on conservation of relevance scores accross the layers of the NN at hand. Indeed, LRP relies on the following assumptions: *(i)* NN can be decomposed into several layers of computation; *(ii)* there exists a relevance score $R_d^{(l)}$ for each dimension $\mathbf{z}_d^{(l)}$ of the vector $\mathbf{z}^{(l)}$ obtained as the output of the $l^{th}$ layer of the NN; and *(iii)* the total relevance scores across dimensions should propagate through all layers of the

NN model, mathematically:

$$f(\mathbf{x}) = \sum_{d \in L} R_d^{(L)} = \sum_{d \in L-1} R_d^{(L-1)} = \cdots = \sum_{d \in 1} R_d^{(1)}, \tag{9.6}$$

where, $f(\mathbf{x})$ represents the predicted probability distribution of an input sequence $\mathbf{x}$, and $L$ the number of layers of the NN at hand. Moreover, LRP defines a propagation rule for obtaining $R_d^{(l)}$ from $R^{(l+1)}$. However, the derivation of the propagation rule is out of the scope of this thesis, thus we refer interested readers to [Ali et al., 2022, Bach et al., 2015]. In our experiments we consider as impact scores the relevance scores of the input layer, namely $j\left(l_k, \epsilon_{NN}(\mathbf{x}_i)\right) = R_d^{(1)}$.

## Layer-wise Attention Tracing (LAT)

Since LLMs rely heavily on self-attention mechanisms [Tay et al., 2021], recent efforts propose to identify input components relevance scores analysing solely the relevance scores of attentions heads of LLM models, introducing Layer-wise Attention Tracing (LAT) [Abnar and Zuidema, 2020, Wu et al., 2020]. Building on top of LRP, LAT propose to redistribute the inner relevance scores $R^{(l)}$ across dimensions using solely self-attention weights. Therefore, LAT defines a custom redistribution rule as:

$$R_i^{(l)} = \sum_{k \ s.t. \ i \ is \ input \ for \ neuron \ k} \ \sum_{h} \mathbf{a}^{(h)} R_{k,h}^{(l+1)}, \tag{9.7}$$

where, $h$ corresponds to the attention head index, while $\mathbf{a}^{(h)}$ are the corresponding learnt weights of the attention head. Similarly to LRP, we here consider as impact scores the relevance scores of the input layer, namely $j\left(l_k, \epsilon_{NN}(\mathbf{x}_i)\right) = R^{(1)}$.

## Integrated Gradient (HESS)

Motivated by the shortcomings of previously proposed gradient-based relevance score attribution mechanisms – such as GS and GI –, Sundararajan et al. [Sundararajan et al., 2017] propose a novel Integrated Gradient approach. The proposed approach aims at explaining the input sample components relevance by integrating the gradient along some trajectory of the input space, which links some baseline value $\mathbf{x}_i'$ to the sample under examination $\mathbf{x}_i$. Therefore, the relevance score of the input $k^{th}$ component of the input sample $\mathbf{x}_i$ is obtained following

$$j\left(l_k, \epsilon_{NN}(\mathbf{x}_i)\right) = \left(\mathbf{x}_{i,k} - \mathbf{x}_{i,k}'\right) \cdot \int_{a=0}^{1} \frac{\delta f(\mathbf{x}_i' + t \cdot (\mathbf{x}_i - \mathbf{x}_i'))}{\delta \mathbf{x}_{i,k}} \, dt, \tag{9.8}$$

where $\mathbf{x}_{i,k}$ represents the $k^{th}$ component of the input sample $\mathbf{x}_i$. By integrating the gradient along an input space trajectory, the authors aim at addressing the locality issue of gradient information. In our experiments we refer to the Integrated Gradient approach as HESS, as for its implementation we rely on the integrated hessian library available for hugging face models[1].

## SHapley Additive exPlanations (SHAP)

SHapley Additive exPlanations (SHAP) relies on Shapley values to identify the contribution of each component of the input sample toward the final prediction distribution. The Shapley value concept derives from game theory, where it represents a solution for a cooperative game, found assigning a distribution of a total surplus generated by the players coalition. SHAP computes the impact of an input component as its marginal contribution toward a label $\tau_m$, computed deleting the component from the input and evaluating the output discrepancy. Firstly defined for explaining simple NN models [Lundberg and Lee, 2017], in our experiments we leverage the extension of SHAP supporting transformer models such as BERT [Kokalj et al., 2021], available in the SHAP python library[2].

## Local Interpretable Model-agnostic Explanations (LIME)

Similarly to SHAP, Local Interpretable Model-agnostic Explanations (LIME) relies on input sample perturbation to identify its relevant components. Here, the predictions of the NN at hand are explained via learning an explainable surrogate model [Ribeiro et al., 2016]. In detail, in order to obtain its explanations LIME constructs a set of samples from the perturbation of the input observation under examination. The constructed samples are considered to be close to the

[1]https://github.com/suinleelab/path_explain
[2]https://github.com/slundberg/shap

observation to be explained from a geometric perspective, thus considering small perturbation of the input. The explainable surrogate model is then trained over the constructed set of samples, obtaining the corresponding local explanation. Given an input sentence, we here consider obtaining its perturbed version via words – or tokens – removal and words substitution. In our experiments, we rely on the already available LIME python library[3].

### 9.2.3 Aggregating Local Explanations

Once local explanations of the NN model are obtained for each input sentence, we aggregate them to obtain a global list of concept impact scores. Before aggregating the local impact scores, we convert the words composing local explanations into their corresponding lemmas – i.e., concepts – to avoid issues when aggregating different words expressing the same concept—e.g., hate and hateful. As no bullet-proof solution exists for the aggregation of different impact scores, we adopt four different approaches in our experiments, namely:

**Sum** A simple summation operation is leveraged to obtain the aggregated score for each lemma. While simple this aggregation approach is effective when dealing with additive impact scores such as SHAP values. However, it suffers from lemma frequency issues, as it tends to overestimate frequent lemmas with average low impact scores. Global impact scores are here defined as $J(l_k, \epsilon_{NN}) = \sum_{i=1}^{N} j\left(l_k, \epsilon_{NN}\left(\mathbf{x}_i\right)\right)$. Therefore, we define $\mathcal{A}$ as

$$\mathcal{A}\left(\{\epsilon_{NN}(\mathbf{x}_i) \text{ } \mathit{for\ each}\ \mathbf{x}_i \in \mathcal{S}\}\right) = \left\{\sum_{i=1}^{N} j\left(l_k, \epsilon_{NN}\left(\mathbf{x}_i\right)\right) \ \mathit{for\ each}\ l_k \in \mathcal{S}\right\}. \tag{9.9}$$

**Absolute sum** Here we sum the absolute values of the local impact scores – rather than their true values – to increase the awareness of global impact scores towards lemmas having both high positive and high negative impact over some sentences. Mathematically, we obtain aggregated scores as

[3]https://github.com/marcotcr/lime

$J(l_k, \epsilon_{NN}) = \sum_{i=1}^{N} |j\left(l_k, \epsilon_{NN}\left(\mathbf{x}_i\right)\right)|$.

$$\mathcal{A}\left(\{\epsilon_{NN}(\mathbf{x}_i) \text{ } \textit{for each } \mathbf{x}_i \in \mathcal{S}\}\right) = \left\{\sum_{i=1}^{N} |j\left(l_k, \epsilon_{NN}\left(\mathbf{x}_i\right)\right)| \text{ } \textit{for each } l_k \in \mathcal{S}\right\}. \tag{9.10}$$

**Average** Similar to the sum operation, here we obtain aggregated scores averaging local impact scores, thus avoiding possible overshooting issues arising when dealing with very frequent lemmas. Mathematically, we define $J(l_k, \epsilon_{NN}) = \frac{1}{N} \cdot \sum_{i=1}^{N} j\left(l_k, \epsilon_{NN}\left(\mathbf{x}_i\right)\right)$.

$$\mathcal{A}\left(\{\epsilon_{NN}(\mathbf{x}_i) \text{ } \textit{for each } \mathbf{x}_i \in \mathcal{S}\}\right) = \left\{\frac{1}{N} \cdot \sum_{i=1}^{N} j\left(l_k, \epsilon_{NN}\left(\mathbf{x}_i\right)\right) \text{ } \textit{for each } l_k \in \mathcal{S}\right\}. \tag{9.11}$$

**Absolute average** Similarly to absolute sum, here we average absolute values of local impact scores for better-managing lemmas with a skewed impact as well as tackling frequency issues. Global impact scores are here defined as $J(l_k, \epsilon_{NN}) = \frac{1}{N} \cdot \sum_{i=1}^{N} |j\left(l_k, \epsilon_{NN}\left(\mathbf{x}_i\right)\right)|$.

$$\mathcal{A}\left(\{\epsilon_{NN}(\mathbf{x}_i) \text{ } \textit{for each } \mathbf{x}_i \in \mathcal{S}\}\right) = \left\{\frac{1}{N} \cdot \sum_{i=1}^{N} |j\left(l_k, \epsilon_{NN}\left(\mathbf{x}_i\right)\right)| \text{ } \textit{for each } l_k \in \mathcal{S}\right\}. \tag{9.12}$$

Since the selection of the aggregation mechanism may influence the correlation between different LPEs, in our experiments we analyse LPEs correlation over the same aggregation scheme. Moreover, we also analyse how aggregation impacts the impact scores correlation over the same LPE, highlighting how leveraging the absolute value of impact score is highly similar to adopting its true value—see Section 9.3.3.

### 9.2.4 Comparing Explanations

Each aggregated global explanation $J$ depends on a corresponding label $\tau_m$ since LPEs produce either a scalar impact value for a single $\tau_m$ or a vector of impact scores for each $\tau_m$. Therefore, recalling Section 9.2.3, we can define the set of aggregated global scores depending on the label they refer to as following:

$$\mathcal{J}_{\tau_m}\left(\epsilon_{NN}, \mathcal{S}\right) = \left\{J\left(l_k, \epsilon_{NN}\right) | \tau_m \textit{ for each } l_k \in \mathcal{S}\right\}. \tag{9.13}$$

$\mathcal{J}_{\tau_m}\left(\epsilon_{NN}, \mathcal{S}\right)$ represents a distribution of impact scores over the set of lemmas – i.e., concepts – available in the samples set for a specific label. To compare the distributions of impact scores extracted using two LPEs – i.e., $\mathcal{J}_{\tau_m}\left(\epsilon_{NN}, \mathcal{S}\right)$ and $\mathcal{J}_{\tau_m}\left(\epsilon'_{NN}, \mathcal{S}\right)$ – we use Pearson correlation, which is defined as the ratio between the covariance of two variables and the product of their standard deviations, and it measures their level of linear correlation. The selected correlation metric is applied to the normalised impact scores. Indeed, different LPEs produce impact scores that may differ relevantly in terms of their magnitude. Normalising the impact scores, we map impact scores to a fixed interval, allowing for a direct comparison of $\mathcal{J}_{\tau_m}$ over different $\epsilon_{NN}$. Mathematically, we refer to the normalised global impact scores as $\|\mathcal{J}_{\tau_m}\|$. Therefore, we define the correlation score between two sets of global impact scores for a single label as:

$$\begin{aligned}\rho\left(\|\mathcal{J}_{\tau_m}\left(\epsilon_{NN}, \mathcal{S}\right)\|, \|\mathcal{J}_{\tau_m}\left(\epsilon'_{NN}, \mathcal{S}\right)\|\right) = \rho\Big(&\|\left\{J\left(l_k, \epsilon_{NN}\right) | \tau_m \textit{ for each } l_k \in \mathcal{S}\right\}\|, \\ &\|\left\{J\left(l_k, \epsilon_{NN}\right) | \tau_m \textit{ for each } l_k \in \mathcal{S}\right\}\|\Big)\end{aligned} \tag{9.14}$$

where $\rho$ refers to the Pearson correlation used to compare couples of $\mathcal{J}_{\tau_m}\left(\epsilon_{NN}, \mathcal{S}\right)$. Throughout our analysis we experimented with similar correlation metrics, such as Spearman correlation and simple vector distance – similarly to [Liscio et al., 2023] –, obtaining similar results. Therefore, to avoid redundancy we here show only the Pearson correlation results. Throughout our experiments, we consider a simple *min-max* normalisation process, scaling the scores to the range $[0, 1]$.

As we aim at obtaining a measure of similarity between LPEs applied over the same set of samples, we can average the correlation scores $\rho$ obtained for each label $\tau_m$ over the set of labels $\mathcal{T}$. Therefore, we mathematically define the correlation score of two LPEs, putting together Equations (9.2), (9.13) and (9.14) as:

$$\mathcal{C}\left(\epsilon_{NN}\left(\mathcal{S}\right), \epsilon'_{NN}\left(\mathcal{S}\right)\right) = \frac{1}{M} \cdot \sum_{m=1}^{M} \rho\left(\|\mathcal{J}_{\tau_m}\left(\epsilon_{NN}, \mathcal{S}\right)\|, \|\mathcal{J}_{\tau_m}\left(\epsilon'_{NN}, \mathcal{S}\right)\|\right) \tag{9.15}$$

where $M$ is the total number of labels, belonging to $\mathcal{T}$.

### 9.2.5 GELPE: Global Explanations from LPEs

Although useful, local explanations are limited, as they do not highlight the general reasoning principle of the underlying model, but rather focus solely on relevant input components for a specific prediction. Aiming at overcoming such limitations, we here present GELPE as the first – up to our knowledge – framework for extracting global explanations from LPEs. Relying on LPE outputs, GELPE allows for the adoption of reliable local extraction mecanisms, while extending their impact to the global reasoning process of the black-box model. More in detail, the aggregated explanations $\epsilon_{NN}(\mathcal{S})$ obtained from LPE's outputs are leveraged as a starting point for building a transparent surrogate models of the original LLM. GELPE relies on transparent-by-design models such as CART optimised over the LLM outputs, rather than the dataset considered.

As described in Equation (9.3), during the optimisation process of the CART model, input sentences are converted into a binary format, expressing the presence or absence of relevant lemmas and their combinations. In order to convert a sentence $\mathbf{x}_i$ into its binary format, we consider the $\mathcal{K}$ most valuable lemmas for each class identified during the aggregation process presented in Section 9.2.3. The $\mathcal{K}$ most valuable lemmas are the ones with the highest aggregated impact scores over a set of sample sentences. To avoid relying only on keywords, and accounting instead for more complex constructs, we also consider the set of skipgrams built from the combination of the single $\mathcal{K}$ most valuable lemmas. In this context, skipgrams define co-occurences of relevant lemmas over a span of limited tokens sequences [Nguyen and Grishman, 2016]. With such a procedure we build a set of valuable lemmas and sequences $\mathcal{L}$ defined as:

$$\mathcal{L} = \{(L_i), (L_i, L_j), (L_i, L_j, L_k), \ldots \ \forall\, i, j, k \in \mathcal{K}\}, \tag{9.16}$$

where $L_i$ represents the lemma in the $i^{th}$ position of the sorted lemmas list – in terms of relevance –, and $(L_i, \ldots, L_j)$ represent the concatenation of two or more lemmas. Once the set of most relevant lemmas and corresponding sequences $\mathcal{L}$ is available, we can define the binarized version of an input sentence as the binary vector that identify the presence or absence of each lemma and sequence in the considered sentence. Mathematically, the binarisation function can be defined as the following:

$$\mathbf{x}_{bin} = bin_{\epsilon_{NN}(\mathcal{S})}(\mathbf{x}_i) = \mathbb{1}(x_j \in \mathcal{L}) \,||\, \mathbb{1}(skip(x_{j-n}, \ldots, x_j) \in \mathcal{L}) \quad \forall\, j \in \mathbf{x}_i, \tag{9.17}$$

where $x_j$ represent the components – i.e., tokens or lemmas – of the input sentence $\mathbf{x}_i$, $skip(x_{j-n}, \ldots, x_j)$ the corresponding skipgrams built from the last $n$ input components, and $\mathbb{1}$ represents the indicator function, being equal to 1 if the lemma/skipgram belongs to $\mathcal{L}$ and 0 otherwise. Finally, $||$ represents the concatenation operation between vectors. As an example, consider the input sentence *the dog is an animal with four legs* and the set of most relevant lemmas extracted by a given LPE to be $\mathcal{L} = \{animal, face, legs\}$. Then the corresponding binarised version of the input sentence is shown in Figure 9.1, where the + symbol is used to identify the concatenation of two relevant lemmas inside a sentence—i.e., *lemma1* + *lemma2* can be interpreted as *lemma1 followed by lemma2.*

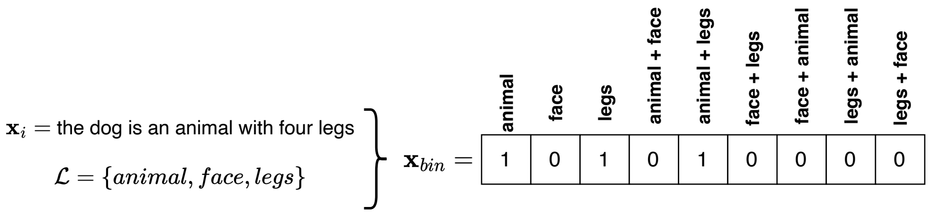


Figure 9.1: Sentence binarization approach in GELPE.

The binarised input is used to optimise the underlying CART model, from which it is possible to extract the equivalent logic program $\mathcal{P}$—see Equation (9.3). The logic program $\mathcal{P}$ obtained represents an explanation of the black-box LLM in the form of a set of Prolog-like rules containing lemmas, sequences of lemmas, and negations thereof. As GELPE relies on the CART model, the extracted rules can only identify the presence or absence of a specific set of keywords and sequences, which represents a limitation of such approach. However, varying the value of $\mathcal{K}$ and the length and expressiveness of the skipgram construction process, the GELPE extraction procedure can be tuned to consider sequences of lemmas as complex as it is needed to fit well the LLM reasoning process. To keep the complexity of the extraction process under control, throughout our experiments we consider relying at most on (2,5)-skipgrams—i.e., building sequences of lemmas of length at most two which are contained over the span of five input tokens. An example of the GELPE extracted knowledge, along with the analysis of its correctness is made available in Section 9.3.4.

## 9.3 Experiments

In this section we present the setup and results of our experiments. More in detail, we first analyse the set of datasets used in our experimental evaluation in Section 9.3.1, along with the model training details and its obtained performance in Section 9.3.2. We then focus on the comparison between the available LPEs, showing the correlation between their explanations in Section 9.3.3. Section 9.3.4 presents the knowledge extraction results, analysing the performance of the knowledge extractor model, along with the complexity of the extracted knowledge. Finally, we analyse the efficiency of the knowledge extraction model, showcasing the improvements in terms of time and energy consumption over the LLM counterpart. The source code of our framework and experiments is publicly available.[4]

### 9.3.1 Datasets

In our experiments, we aim at analysing the correlation among different LPEs and the feasibility of global knowledge extraction from LLM over a large set of scenarios. Therefore, we consider an heterogeneous set of datasets targetting text classification tasks, ranging from easy to complex setups. More in detail, we consider targetting the SMS [Almeida et al., 2011] and YOUTUBE [Alberto et al., 2015] spam classification datasets as easy setups, having two highly separable classes. Here, each sample represents a text – either obtained from text messages or from comment posted in the comments section of a youtube videos – manually labeled as spam or legitimate (ham). Although available, the metadata information – such as the author's name and publication date – is not used. As a slightly more complex setup, we consider the TREC [Li and Roth, 2002] dataset, containing 4,965

[4]https://github.com/AndAgio/SKE_NLP

labeled questions. In this context, each sample represents a question belonging to one of six classes – i.e., *Abbreviation, Entity, Description, Human, Location, Numeric-value* – to be semantically classified. Finally, as a complex setup we select the MFTC datasets as the target classification task.

The MFTC dataset is composed of 35,108 tweets – sentences –, which can be considered as a collection of different datasets. Each split of MFTC corresponds to a different context. Here, tweets corresponding to the dataset samples are collected following a certain event or target. As an example, tweets belonging to the Black Lives Matter (BLM) split were collected during the period of Black Lives Matter protests in the US. The list of all MFTC subjects considered in our experiments is the following: *(i)* All Lives Matter (ALM), *(ii)* Black Lives Matter (BLM), *(iii)* Baltimore protests (BLT), *(iv)* 2016 presidential election (ELE), *(v)* MeToo movement (MT), *(vi)* hurricane Sandy (SND). Each tweet in MFTC is labelled, following the same moral theory, with one or more of the following 11 moral values: *(i)* care/harm, *(ii)* fairness/cheating, *(iii)* loyalty/betrayal, *(iv)* authority/subversion, *(v)* purity/degradation, *(vi)* non-moral. Ten of the 11 available moral values are obtained as a moral concept and its opposite expression—e.g., fairness refers to the act of supporting fairness and equality, while cheating refers to the act of refraining from cheating or exploiting others. Given morality subjectivity, each tweet is labelled by multiple annotators, and the final moral labels are obtained via majority voting.

### 9.3.2 Model Training

The SMS, YOUTUBE, and TREC datasets represent standard classification tasks, thus not requiring particular setups. Meanwhile, tackling MFTC we follow state-of-the-art approaches for dealing with morality classification task [Kiesel et al., 2022, Alshomary et al., 2022]. Thus, we treat the morality classification problem as a multi-class multi-label classification task. Differently from recent approaches, we here do not rely on the *sequential training* paradigm for the MFTC datasets, but rather train each model solely on the MFTC split at hand. Indeed, in our experiments, we do not aim at obtaining strong transferability between domains, but rather we focus on analysing LPEs behaviour.

| | SMS | YOUTUBE | TREC | ALM | BLM | BLT | ELE | MT | SND |
|---|---|---|---|---|---|---|---|---|---|
| $F_1$ score | 98.71% | 95.81% | 97.18% | 63.04% | 82.59% | 64.51% | 63.14% | 52.16% | 56.85% |

Table 9.1: BERT performance over considered datasets.

For each dataset we leverage BERT as the LLM to be optimised [Devlin et al., 2019], and define one NN model for each dataset, optimising its parameters over the 70% of samples, leaving the remaining 30% for testing purposes. We leverage the pre-trained *bert-base-uncased* model – available in the Hugging Face python library[5] – as the starting point of our training process. Each model is trained for 3 epochs using a standard binary cross entropy loss [Zhang and Sabuncu, 2018], a learning rate of $5 \times 10^{-5}$, a batch size of 16 and a maximum sequence length of 64. We keep track of the macro F1-score for each model to identify its performance over the test samples. Table 9.1 shows the performance of the trained BERT model.

### 9.3.3 Local Post-hoc Explainers Comparison

We analyse the extent to which different LPEs are aligned in their process of identifying impactful concepts for the underlying NN model. With this aim, we train a BERT model over a specific dataset (following the approach described in Section 9.3.2) and compute the pairwise correlation $\mathcal{C}\left(\epsilon_{NN}\left(\mathcal{S}\right), \epsilon'_{NN}\left(\mathcal{S}\right)\right)$ (as described in Section 9.2) for each pair of LPEs in the selected set. To avoid issues caused by model overfitting over the training set, which would render explanations unreliable, we apply each $\epsilon_{NN}$ over the test set of the selected dataset.

#### Local Post-hoc Explainers Disagreement

Using the pairwise correlation values we construct the correlation matrices shown in Figures 9.2 and 9.3, which highlight how there exist a very weak correlation score between most LPEs over different datasets. Here, it is interesting to notice how few specific couples or clusters of LPEs exist which highly correlate with each other. For example, GS, GI, and LRP show moderate-to-high correlation score, mainly due to their reliance on computing the gradient of the prediction

[5]`https://github.com/huggingface`

to identify impactful concepts. However, this is not the case for all LPE couples relying on similar approaches. For example, GI and gradient integration – HESS in the matrices – show little to no correlation, although they both are gradient-based approach for producing local explanations. Similarly, SHAP and LIME show no correlation even if they both rely on input perturbation and are considered the state-of-the-art.

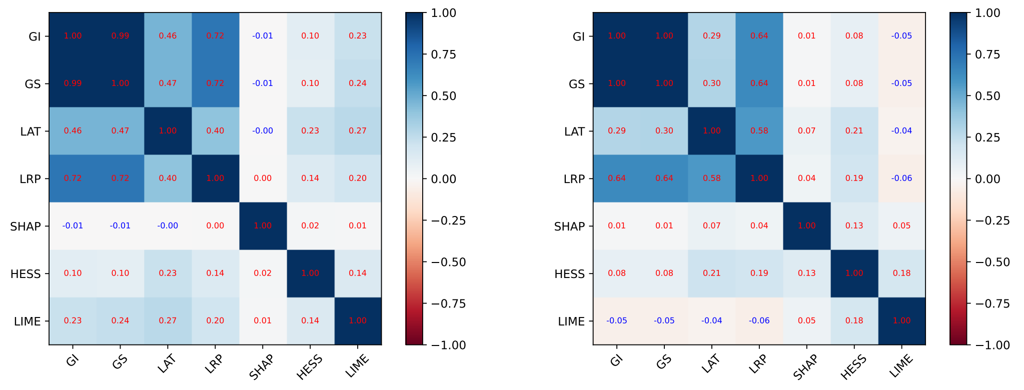


Figure 9.2: $\mathcal{C}\left(\epsilon_{NN}\left(\mathcal{S}\right), \epsilon'_{NN}\left(\mathcal{S}\right)\right)$ using average aggregation as $\mathcal{A}$ over the SMS (left) and YOUTUBE (right) dataset.

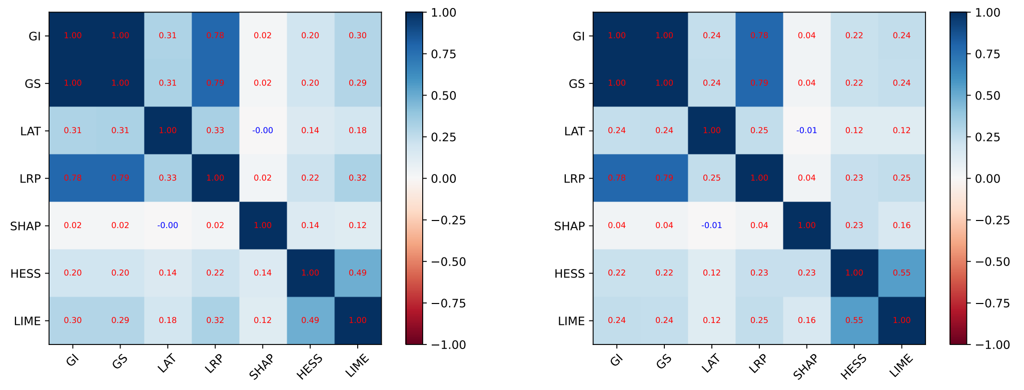


Figure 9.3: $\mathcal{C}\left(\epsilon_{NN}\left(\mathcal{S}\right), \epsilon'_{NN}\left(\mathcal{S}\right)\right)$ using average aggregation as $\mathcal{A}$ over the ALM (left) and BLM (right) dataset.

Figures 9.2 and 9.3 highlight how the vast majority of LPE pairs show very-small-to-no correlation at all, exposing how the selected approaches actually disagree. Interestingly enough, disagreement between LPEs holds true for every dataset studied in our analyses, no matter the complexity or simplicity of the learning task and the samples considered. This finding represents a fundamental result of our study, as it demonstrates how no accordance exists between LPEs even when they are applied to the same model and dataset, even on very simple classification tasks such as the one represented by the SMS dataset. The reason behind the large discrepancies among LPE might be various, but mostly bear down to the following:

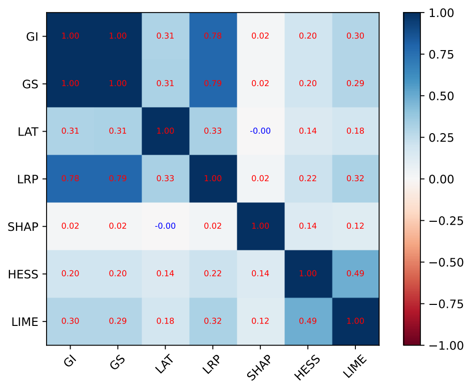
- Few of the LPEs considered in the literature do not represent reliable solutions for identifying the reasoning principles of LLMs.
- Each of the uncorrelated LPEs highlight a different set or subset of reasoning principles of the underlying model.

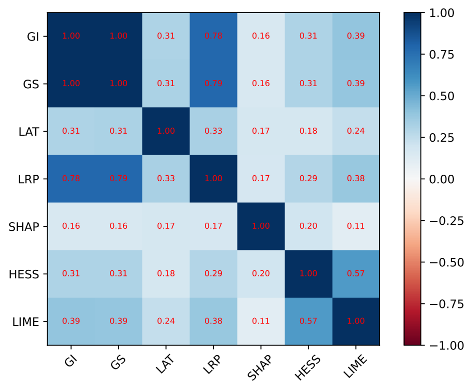
Therefore, our results show how complex it is to identify a set of fair and reliable metrics to spot the best LPE or even reliable LPEs, as they seem to gather uncorrelated explanations. Similar results to the ones shown in Figures 9.2 and 9.3 are obtained for all datasets and are made available at `https://tinyurl.com/QU4RR3L`.

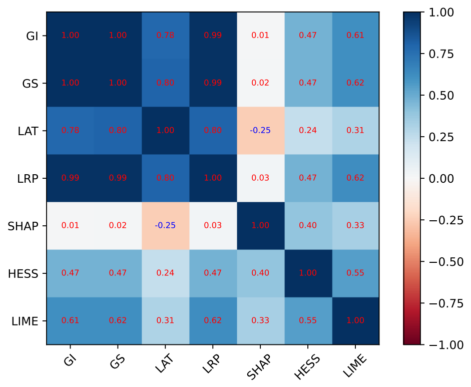
### Aggregation Affects Correlation

Since our LPE correlation metric is dependent on $\mathcal{A}$, we here analyse how the selection of different aggregation strategies impacts the correlation between LPEs. To understand the impact of $\mathcal{A}$ on $\mathcal{C}$, we plot the correlation matrices for a single dataset, varying the aggregation approach, thus obtaining the four correlation matrices shown in Figure 9.4.

From Figures 9.4c and 9.4d one could notice the strong correlation between different LPEs. This seems to be in contrast with the results found in Section 9.3.3. However, the reason behind the strong correlation achieved when relying on summation aggregation is not caused by the actual correlation between explanations, but rather on the susceptibility of summation to tokens frequency. Indeed, since the summation aggregation approaches do not take into account the occurrence frequency of lemmas in $\mathcal{S}$, they tend to overestimate the relevance of popular concepts. Intuitively, using this aggregations, a rather impactless lemma appearing


(a) Average aggregation


(b) Absolute average aggregation


(c) Sum aggregation

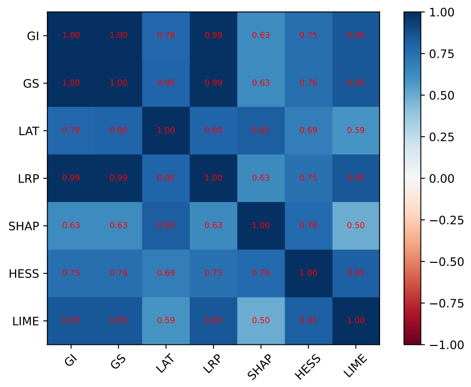

(d) Absolute sum aggregation

Figure 9.4: $\mathcal{C}\left(\epsilon_{NN}\left(\mathcal{S}\right), \epsilon'_{NN}\left(\mathcal{S}\right)\right)$ using different aggregations over the ALM dataset.

5000 times would obtain a global impact higher than a very impactful lemma appearing only 10 times. These results highlight the importance of relying on average based aggregation approaches when considering to construct global explanations from the LPE outputs.

Figure 9.4 also points out how leveraging the absolute value of LPEs incurs in higher correlation scores. The reason behind this is to be found in the impact scores distributions. While true local impact scores are distributed over the set of real numbers $\mathbb{R}$, computing the absolute value of local impacts $j$ shifts their distribution to $\mathbb{R}^+$, shrinking possible differences between positive and negative scores. Moreover, LPE outputs rely much more heavily on scoring positive contributions using positive impact scores, and typically give less focus to negative impact scores. Therefore, the output of LPEs is generally unbalanced towards positive impact scores, making negative impact scores mostly negligible.

### 9.3.4 Knowledge Extraction

We here analyse if and to what extent it is possible to extract a knowledge base representing the trained LLM from each LPE, and how much these are aligned in their process of explaining the underlying NN model. With this aim, we rely on the GELPE global explainer construction process presented in Section 9.2.5, extracting a set of rules representing the LLM decision process for each dataset at hand. As the building process is dependent on the number of most impactful lemmas, we consider varying the hyperparameter $\mathcal{K}$ to select the top-$\mathcal{K}$ relevant lemmas for each class. After the relevant lemmas are selected, we construct the skipgrams of relevant lemmas as the set of skipgrams occurring in the training set that are composed from relevant lemmas only. Skipgrams are considered to extend the capabilities of the extraction process to consider sequences of relevant concepts rather than blindly focusing only on single tokens. Once the relevant lemmas and skipgrams are available, we consider converting the samples of the training set into binary vectors describing the presence or absence of each lemma and skipgram. We optimise the CART model on the binary vectors representing the training samples and extract the corresponding knowledge from the tree as a set of ordered Prolog rules. To avoid incurring in an unbearable number of Prolog clauses – that would hinder the utility of the knowledge extraction process – we limit the depth of the CART model to be:

$$depth = \mu \cdot \frac{L}{\mathcal{K} * |C|}, \tag{9.18}$$

where $L$ represents the number of total relevant lemmas and skipgrams identified from the LPE, $|C|$ represents the number of classes of the classification task at hand, and $\mu$ represents an hyperparameter that we set to $\mu = 5$ empirically. Throughout the remainder of this chapter, we consider leveraging the average operation as the aggregation function $\mathcal{A}$, as it represents the least biased aggregation

Table 9.2: Fidelity of the extracted knowledge w.r.t. to the original BERT model over the SMS dataset. † identifies the best LPE over a single $\mathcal{K}$ values, while the green row(s) identify the overall best LPE.

| LPEs \ $\mathcal{K}$ | 50 | 100 | 150 | 200 | 250 |
|---|---|---|---|---|---|
| GI | 87.00% | 87.60% | 90.20%† | 91.80%† | 91.60%† |
| GS | 87.40%† | 87.80%† | 89.80% | 90.40% | 91.60%† |
| LAT | 87.40%† | 87.40% | 89.00% | 89.60% | 91.00% |
| LRP | 86.60% | 86.40% | 86.60% | 87.80% | 90.80% |
| SHAP | 86.40% | 86.60% | 86.60% | 86.40% | 86.40% |
| HESS | 86.20% | 86.40% | 86.80% | 86.80% | 86.40% |
| LIME | 86.20% | 86.20% | 86.20% | 86.60% | 86.80% |

process. However, we also experiment with other aggregation functions, such as sum, absolute sum, and absolute average, obtaining similar results. Therefore, in order to avoid redundancy we here show only the average aggregation results.

### Knowledge Fidelity

To asses the performance of the proposed knowledge extraction process from LPEs, we measure the fidelity of the predictions obtained using the Prolog rules against the corresponding LLM predictions. The fidelity metric measures the percentage of instances in which the Prolog rules predictions and model predictions are equivalent, thus measuring the accuracy of the knowledge extraction process. Tables 9.2 and 9.3 presents the fidelity of the GELPE extraction process over the SMS and YOUTUBE datasets. In those simple scenarios, the proposed approach extracts a set of accurate rules, representing with high fidelity the decision process of the underlying LLM. Using GELPE, we enable the extraction of simple and easy to understand rules from the complex black-box model.

Over more complex datasets, the performance of the extracted knowledge using GELPE varies depending on the dataset at hand. Table 9.4 shows the fidelity of GELPE over the BLT dataset, where the explanation model achives up to 95.09% fidelity. Meanwhile, Tables 9.5 and 9.6 presents the fidelity results over the ELE

Table 9.3: Fidelity of the extracted knowledge w.r.t. to the original BERT model over the YOUTUBE dataset. † identifies the best LPE over a single $\mathcal{K}$ values, while the green row(s) identify the overall best LPE.

| LPEs \ $\mathcal{K}$ | 50 | 100 | 150 | 200 | 250 |
|---|---|---|---|---|---|
| GI | 69.20% | 72.40% | 72.40% | 76.00% | 76.80% |
| GS | 69.20% | 72.40% | 72.40% | 78.00% | 76.40% |
| LAT | 66.00% | 64.40% | 70.80% | 80.00% | 84.40% |
| LRP | 65.20% | 65.20% | 68.80% | 70.00% | 70.00% |
| SHAP | 43.20% | 75.20% | 80.80% | 80.80% | 80.40% |
| HESS | 82.40% | 87.60% | 86.40% | 88.80% | 87.20% |
| LIME | 88.00%† | 92.00%† | 94.00%† | 93.20%† | 92.80%† |

and SND datasets respectively, where the proposed GELPE extraction seems to struggle to achieve high fidelity values. This is due to the underlying complexity of the dataset at hand. For some tasks – e.g. YOUTUBE, BLT –, considering the most relevant lemmas and their skipgram combinations is sufficient, while others – e.g. ELE, SND – require a more complex understanding of the inner sentence constructs.

As expected, increasing the number of relevant lemmas $\mathcal{K}$ considered to optimise GELPE results in higher fidelity, as the underlying CART model takes into account a broader set of meaningful features. However, increasing $\mathcal{K}$ over a certain threshold results in an unbearable rules complexity and in smaller fidelity gains. The increment on rule complexity also hiders the understandability of the extracted explanation, representing a fundamental concept to take into account. This phenomenon is clearly shown in Tables 9.5 and 9.6, where the fidelity grows up to 20% when $\mathcal{K}$ ranges from 50 to 250.

Interestingly, the disagreement between different LPEs seems to affect also the performance of the obtained global explainer model. Fidelity results highlight that GELPE explanations obtained from highly correlated LPEs such as GI and GS achieve comparable performance level. Meanwhile, Prolog rules obtained from uncorrelated LPEs result in different fidelity level. While expected, such a behaviour

Table 9.4: Fidelity of the extracted knowledge w.r.t. to the original BERT model over the BLT dataset. † identifies the best LPE over a single $\mathcal{K}$ values, while the green row(s) identify the overall best LPE.

| LPEs \ $\mathcal{K}$ | 50 | 100 | 150 | 200 | 250 |
|---|---|---|---|---|---|
| GI | 92.64% | 92.70% | 93.18% | 93.12% | 92.76% |
| GS | 93.60% | 92.28% | 93.18% | 93.24% | 92.82% |
| LAT | 90.19% | 91.74% | 92.28% | 92.34% | 92.46% |
| LRP | 92.28% | 93.12% | 93.00% | 93.48% | 92.88% |
| SHAP | 95.69% | 94.14% | 94.14% | 94.14% | 94.14% |
| HESS | 93.72% | 93.84% | 93.48% | 93.48% | 93.60% |
| LIME | 95.27%† | 95.27%† | 95.09%† | 95.09%† | 95.09%† |

represents a useful finding as it allows for the identification of more reliable LPEs, as the ones that results in a higher level of fidelity—e.g., LIME in most scenarios.

### Knowledge Complexity

The ideal extraction process is required to output a set of Prolog rules that is as faithful as possible w.r.t. the underlying LLM. However, the dimensionality of the extracted program should be kept small to limit the complexity burden of the analysis process. An overly complex knowledge base would not be useful for anlaysing the inner working principle of the explained LLM, as it would be mostly impossible to be processed by a human interpreter. To assess the complexity of the extracted knowledge, we consider tracking the length of the Prolog program and its cumbersomeness. In this context, the length $L$ represents the number of clauses in the obtained explanation, while the cumbersomeness $C$ represents the average number of atoms in each clause.

For each dataset considered we keep track of $L$ and $C$ and analyse their variability over each LPE and $\mathcal{K}$ value. Tables 9.7 and 9.8 show the complexity of the GELPE output over the YOUTUBE and ELE dataset respectively. The results highlight the relevant difference in terms of required complexity to extract reliable explanations when dealing with simple or complex classification tasks. Both $L$ and

Table 9.5: Fidelity of the extracted knowledge w.r.t. to the original BERT model over the ELE dataset. † identifies the best LPE over a single $\mathcal{K}$ values, while the green row(s) identify the overall best LPE.

| LPEs \ $\mathcal{K}$ | 50 | 100 | 150 | 200 | 250 |
|---|---|---|---|---|---|
| GI | 68.51% | 72.00% | 74.67% | 75.92% | 76.23% |
| GS | 68.95% | 73.93% | 74.74%† | 74.67% | 76.79%† |
| LAT | 58.93% | 61.36% | 66.77% | 67.45% | 69.14% |
| LRP | 72.81% | 74.74%† | 75.36% | 75.48% | 75.79% |
| SHAP | 67.64% | 68.70% | 69.51% | 70.50% | 70.50% |
| HESS | 68.33% | 73.80% | 74.05% | 74.11% | 74.11% |
| LIME | 73.61%† | 73.93% | 74.49% | 76.60%† | 76.73% |

$C$ are kept small for each LPE and $\mathcal{K}$ combination over the YOUTUBE dataset, while still being able to reach high fidelity (see Table 9.3). Meanwhile, the ELE moral classification task requires to consider higher values of $L$ and $C$ in order to achieve a satisfactory level of fidelity (see Table 9.5).

As expected, Table 9.7 also highlights a linear correlation between the complexity of the extracted explanation and the parameter $\mathcal{K}$. A higher value of $\mathcal{K}$ identifies a broader set of relevant lemmas considered during the optimization of the CART explainer, thus increasing the number of features available to construct Prolog clauses. The increased complexity of the obtained explanation represents a fundamental aspect to take into account when considering leveraging GELPE, as we need for the explanations to be bounded in complexity for them to be human-readable. The limitation of the CART depth (see Equation (9.18)) represents an helping tool from this perspective, as it allows to keep the complexity of the explainer under control in complex setup, such as the ELE dataset. This phenomenon can be seen in Table 9.8, where the complexity of the extracted explanations remains stable over $\mathcal{K}$. However, depth limitation is not drawback free, as it hinders the achievement of high fidelity values.

Table 9.6: Fidelity of the extracted knowledge w.r.t. to the original BERT model over the SND dataset. $^\dagger$ identifies the best LPE over a single $\mathcal{K}$ values, while the green row(s) identify the overall best LPE.

| LPEs \ $\mathcal{K}$ | 50 | 100 | 150 | 200 | 250 |
|---|---|---|---|---|---|
| GI | 45.39% | 58.39% | 60.20% | 59.84% | 61.73% |
| GS | 46.48% | 57.59% | 59.84% | 61.00%$^\dagger$ | 62.60%$^\dagger$ |
| LAT | 38.63% | 45.53% | 49.82% | 49.89% | 57.30% |
| LRP | 40.02% | 49.67% | 59.98% | 59.62% | 61.00% |
| SHAP | 57.01% | 57.23% | 57.23% | 57.23% | 57.23% |
| HESS | 60.49% | 60.13%$^\dagger$ | 58.75% | 58.53% | 58.61% |
| LIME | 61.15%$^\dagger$ | 60.06% | 60.28%$^\dagger$ | 60.20% | 60.13% |

**Knowledge Visualisation**

We visualise the logic program obtained from the knowledge extraction process to analyse their correctness and understandability. Figure 9.5 shows the logic program $\mathcal{P}$ obtained from the GELPE extraction process when leveraging LIME as LPE and $\mathcal{K} = 50$ on the YOUTUBE dataset. The extracted knowledge is characterised by a manageable complexity, having a small number of relatively short clauses. In this context, the summation symbol $+$ is used to identify the concatenation of two relevant lemmas inside a sentence—*lemma1* + *lemma2* can be interpreted as *lemma1 followed by lemma2*. Interestingly, the extracted knowledge also shows some relevant properties, such as the identification of spam comments as those containing certain hyperlinks (*org* lemma), subscription related lemmas (*sub* and *subscribe*), as well as grammatical errors (*suscribe* rather than subscribe and *withing* rather than within).

Figure 9.6 shows the extracted knowledge when GELPE is used with SHAP and $\mathcal{K}$ over the BLM dataset. Here, it is also possible to notice relevant concepts being extracted from the LLM decision process. For example, the proposed extraction process allows to identify that the combination of keywords *obey* and *rape* result in the text being considered as harmful, as well as the keyword *murder*. Meanwhile, the sequence *standing* + *injustice* along with the *justice* keyword identify that the

Table 9.7: Complexity of the extracted knowledge over the YOUTUBE dataset. $L$ represents the length of the obtained explanation – i.e., the number of clauses –, while $C$ represents the cumbersomeness—i.e., the average number of atoms in each clause. The green row(s) identify the overall simplest LPE.

| LPEs \ $\mathcal{K}$ | 50 | 100 | 150 | 200 | 250 |
|---|---|---|---|---|---|
| GI | $L = 30$ $C = 6.73$ | $L = 41$ $C = 7.71$ | $L = 40$ $C = 7.65$ | $L = 40$ $C = 7.05$ | $L = 66$ $C = 10.38$ |
| GS | $L = 30$ $C = 6.63$ | $L = 41$ $C = 7.71$ | $L = 42$ $C = 7.79$ | $L = 37$ $C = 7.51$ | $L = 73$ $C = 10.59$ |
| LAT | $L = 20$ $C = 5.65$ | $L = 43$ $C = 6.42$ | $L = 116$ $C = 10.28$ | $L = 75$ $C = 8.84$ | $L = 64$ $C = 11.23$ |
| LRP | $L = 37$ $C = 6.51$ | $L = 46$ $C = 7.15$ | $L = 36$ $C = 7.17$ | $L = 32$ $C = 7.09$ | $L = 48$ $C = 7.81$ |
| SHAP | $L = 14$ $C = 5.21$ | $L = 25$ $C = 7.04$ | $L = 34$ $C = 7.62$ | $L = 36$ $C = 7.78$ | $L = 33$ $C = 7.61$ |
| HESS | $L = 48$ $C = 9.67$ | $L = 53$ $C = 10.36$ | $L = 55$ $C = 12.05$ | $L = 39$ $C = 11.67$ | $L = 52$ $C = 11.88$ |
| LIME | $L = 32$ $C = 6.72$ | $L = 56$ $C = 9.29$ | $L = 57$ $C = 9.42$ | $L = 60$ $C = 11.30$ | $L = 68$ $C = 13.01$ |

sentiment is fairness. These results highlight the goodness of the proposed GELPE framework than enables the extraction of meaningful Prolog rules from the LLM reasoning principle with high fidelity.

### Resource Effeciency

The proposed GELPE framework allows for the extraction of Prolog rules from LLM starting from LPEs outputs. In an ideal scenario, the Prolog program obtained as a result of the GELPE process contains a handful of simple – i.e., short – clauses. The execution of such simple program – surrogate of the original LLM model – requires few computational power, as it does not rely on complex operations such as convolutions that require GPU or hardware-specific solution. However, the complexity of the GELPE output can grow quickly depending on the set of considered lemmas and skipgrams, thus hindering its efficiency. Therefore, it is fundamental to assess the ability of the proposed GELPE framework to produce a resource-friendly surrogate model of the original LLM. To this end, we consider measuring the time and energy efficiency of the original LLM model against few of

Table 9.8: Complexity of the extracted knowledge over the ELE dataset. $L$ represents the length of the obtained explanation – i.e., the number of clauses –, while $C$ represents the cumbersomeness—i.e., the average number of atoms in each clause. The green row(s) identify the overall simplest LPE.

| LPEs \ $\mathcal{K}$ | 50 | 100 | 150 | 200 | 250 |
|---|---|---|---|---|---|
| GI | $L = 430$ $C = 19.78$ | $L = 363$ $C = 19.68$ | $L = 353$ $C = 18.21$ | $L = 273$ $C = 15.76$ | $L = 296$ $C = 15.29$ |
| GS | $L = 422$ $C = 19.58$ | $L = 335$ $C = 19.54$ | $L = 350$ $C = 18.27$ | $L = 269$ $C = 15.61$ | $L = 369$ $C = 17.94$ |
| LAT | $L = 639$ $C = 20.52$ | $L = 487$ $C = 19.93$ | $L = 373$ $C = 19.11$ | $L = 379$ $C = 20.00$ | $L = 360$ $C = 20.47$ |
| LRP | $L = 390$ $C = 19.85$ | $L = 433$ $C = 22.08$ | $L = 391$ $C = 18.45$ | $L = 375$ $C = 18.19$ | $L = 364$ $C = 17.93$ |
| SHAP | $L = 16$ $C = 4.06$ | $L = 15$ $C = 3.93$ | $L = 16$ $C = 4.06$ | $L = 16$ $C = 4.06$ | $L = 16$ $C = 4.06$ |
| HESS | $L = 17$ $C = 4.12$ | $L = 64$ $C = 7.84$ | $L = 71$ $C = 8.04$ | $L = 71$ $C = 8.03$ | $L = 72$ $C = 8.08$ |
| LIME | $L = 64$ $C = 7.75$ | $L = 68$ $C = 7.94$ | $L = 71$ $C = 8.06$ | $L = 130$ $C = 10.72$ | $L = 131$ $C = 10.76$ |

the Prolog programs obtained using GELPE. More in detail, we consider running the original BERT model both in a GPU enabled scenario – using a Tesla V100S-PCIE with 32GB of RAM – and a CPU only scenario – using an Intel(R) Xeon(R) Gold 6226R CPU @ 2.90GHz. We consider comparing the BERT efficiency performance against the most faithful Prolog program – i.e., the one obtained with LIME as LPE – and against the simplest one—i.e., the one obtained with SHAP as LPE. For each LPE, we consider two setups, having the lowest and highest value of $\mathcal{K}$—i.e., $\mathcal{K} = 50$ and $\mathcal{K} = 250$, respectively. The Prolog programs obtained from GELPE from each LPE are run using only the CPU device. We keep track of the average time $\bar{t}$ required to infer the prediction over a single sample and the corresponding average energy consumed $\overline{E}$. Table 9.9 shows the obtained results over all datasets.

The results highlight how over simple setups such as SMS and YOUTUBE, the surrogate model obtained using GELPE always outperforms the BERT counterpart. This is due to the small task complexity, enabling the proposed framework to extract a small set of simple clauses to mimic the model behaviour. Indeed, the

```
ham  :-  ¬ channel ∧ ¬ check ∧ ¬ subscribe ∧ ¬ playlist ∧ ¬ share ∧ ¬ billion ∧ ¬ subscriber ∧ ¬ org ∧ ¬ hackfbaccountlive ∧ ¬ sub.
spam :-  ¬ channel ∧ ¬ check ∧ ¬ subscribe ∧ ¬ playlist ∧ ¬ share ∧ ¬ billion ∧ ¬ subscriber ∧ ¬ org ∧ ¬ hackfbaccountlive ∧ sub.
spam :-  ¬ channel ∧ ¬ check ∧ ¬ subscribe ∧ ¬ playlist ∧ ¬ share ∧ ¬ billion ∧ ¬ subscriber ∧ ¬ org ∧ hackfbaccountlive.
spam :-  ¬ channel ∧ ¬ check ∧ ¬ subscribe ∧ ¬ playlist ∧ ¬ share ∧ ¬ billion ∧ ¬ subscriber ∧ org.
spam :-  ¬ channel ∧ ¬ check ∧ ¬ subscribe ∧ ¬ playlist ∧ ¬ share ∧ ¬ billion ∧ subscriber ∧ ¬ subscriber + million ∧ ¬ reason ∧ ¬ suscribe.
spam :-  ¬ channel ∧ ¬ check ∧ ¬ subscribe ∧ ¬ playlist ∧ ¬ share ∧ ¬ billion ∧ subscriber ∧ ¬ subscriber + million ∧ ¬ reason ∧ suscribe.
spam :-  ¬ channel ∧ ¬ check ∧ ¬ subscribe ∧ ¬ playlist ∧ ¬ share ∧ ¬ billion ∧ subscriber ∧ ¬ subscriber + million ∧ reason.
ham  :-  ¬ channel ∧ ¬ check ∧ ¬ subscribe ∧ ¬ playlist ∧ ¬ share ∧ ¬ billion ∧ subscriber ∧ subscriber + million.
ham  :-  ¬ channel ∧ ¬ check ∧ ¬ subscribe ∧ ¬ playlist ∧ ¬ share ∧ billion.
spam :-  ¬ channel ∧ ¬ check ∧ ¬ subscribe ∧ ¬ playlist ∧ share ∧ ¬ million.
ham  :-  ¬ channel ∧ ¬ check ∧ ¬ subscribe ∧ ¬ playlist ∧ share ∧ million.
spam :-  ¬ channel ∧ ¬ check ∧ ¬ subscribe ∧ playlist.
spam :-  ¬ channel ∧ ¬ check ∧ subscribe.
spam :-  ¬ channel ∧ check ∧ ¬ 2x10 ∧ ¬ billion ∧ ¬ subscribe ∧ ¬ million ∧ ¬ soundcloud ∧ ¬ comment ∧ ¬ subscriber ∧ ¬ survival.
spam :-  ¬ channel ∧ check ∧ ¬ 2x10 ∧ ¬ billion ∧ ¬ subscribe ∧ ¬ million ∧ ¬ soundcloud ∧ ¬ comment ∧ ¬ subscriber ∧ survival.
spam :-  ¬ channel ∧ check ∧ ¬ 2x10 ∧ ¬ billion ∧ ¬ subscribe ∧ ¬ million ∧ ¬ soundcloud ∧ ¬ comment ∧ subscriber.
spam :-  ¬ channel ∧ check ∧ ¬ 2x10 ∧ ¬ billion ∧ ¬ subscribe ∧ ¬ million ∧ ¬ soundcloud ∧ comment.
spam :-  ¬ channel ∧ check ∧ ¬ 2x10 ∧ ¬ billion ∧ ¬ subscribe ∧ ¬ million ∧ soundcloud.
ham  :-  ¬ channel ∧ check ∧ ¬ 2x10 ∧ ¬ billion ∧ ¬ subscribe ∧ million.
spam :-  ¬ channel ∧ check ∧ ¬ 2x10 ∧ ¬ billion ∧ subscribe.
ham  :-  ¬ channel ∧ check ∧ ¬ 2x10 ∧ billion.
ham  :-  ¬ channel ∧ check ∧ 2x10.
spam :-  ¬ check ∧ ¬ subscribe ∧ ¬ sub + channel ∧ ¬ comment ∧ ¬ withing + channel ∧ ¬ share ∧ ¬ shaking ∧ ¬ burn ∧ ¬ suscribe + channel.
spam :-  ¬ check ∧ ¬ subscribe ∧ ¬ sub + channel ∧ ¬ comment ∧ ¬ withing + channel ∧ ¬ share ∧ ¬ shaking ∧ ¬ burn ∧ suscribe + channel.
spam :-  ¬ check ∧ ¬ subscribe ∧ ¬ sub + channel ∧ ¬ comment ∧ ¬ withing + channel ∧ ¬ share ∧ ¬ shaking ∧ burn.
spam :-  ¬ check ∧ ¬ subscribe ∧ ¬ sub + channel ∧ ¬ comment ∧ ¬ withing + channel ∧ ¬ share ∧ shaking.
spam :-  ¬ check ∧ ¬ subscribe ∧ ¬ sub + channel ∧ ¬ comment ∧ ¬ withing + channel ∧ share.
spam :-  ¬ check ∧ ¬ subscribe ∧ ¬ sub + channel ∧ ¬ comment ∧ withing + channel.
spam :-  ¬ check ∧ ¬ subscribe ∧ ¬ sub + channel ∧ comment.
spam :-  ¬ check ∧ ¬ subscribe ∧ sub + channel.
spam :-  ¬ check ∧ subscribe.
spam.
```

Figure 9.5: Logic program $\mathcal{P}$ obtained from the GELPE extraction process when leveraging LIME as LPE and $\mathcal{K} = 50$ on the YOUTUBE dataset.

efficiency of the Prolog program obtained is proportional to the complexity of the clauses to be analysed to achieve a prediction. Meanwhile, over more complex setups, such as the ELE dataset, in which GELPE outputs a large set of long clauses, it is possible to outperform the BERT counterpart only when considering a small value of $\mathcal{K}$. However, noticeably it is always possible to find a surrogate Prolog model obtained via GELPE representing a more efficient solution than running the LLM model over the CPU. These results highlight the advantage of leveraging a simple rule-based approach over sub-symbolic models when hardware acceleration is not available. As such, the proposed model represents a feasible solution for those scenarios where the deployment setup is composed of resource-constrained devices, such as embedded devices and micro-controllers. In this scenarios, running the original LLM would not be acceptable, due to latency and memory issues, while GELPE's output results in a resource efficient transparent program easily deployable. Therefore, the obtained results show that the GELPE surrogate model does not represent just an explainable and transparent twin of the LLM original

```
non-moral   :-  ¬ solidarity ∧ ¬ justice ∧ ¬ obey ∧ ¬ injustice ∧ ¬ compassion.
care        :-  ¬ solidarity ∧ ¬ justice ∧ ¬ obey ∧ ¬ injustice ∧ compassion.
cheating    :-  ¬ solidarity ∧ ¬ justice ∧ ¬ obey ∧ injustice ∧ ¬ compassion.
care        :-  ¬ solidarity ∧ ¬ justice ∧ ¬ obey ∧ injustice ∧ compassion.
authority   :-  ¬ solidarity ∧ ¬ justice ∧ obey ∧ ¬ rape ∧ ¬ corrupt.
cheating    :-  ¬ solidarity ∧ ¬ justice ∧ obey ∧ ¬ rape ∧ corrupt.
harm        :-  ¬ solidarity ∧ ¬ justice ∧ obey ∧ rape.
fairness    :-  ¬ solidarity ∧ justice ∧ ¬ injustice ∧ ¬ disobedience ∧ ¬ murder.
harm        :-  ¬ solidarity ∧ justice ∧ ¬ injustice ∧ ¬ disobedience ∧ murder.
subversion  :-  ¬ solidarity ∧ justice ∧ ¬ injustice ∧ disobedience.
cheating    :-  ¬ solidarity ∧ justice ∧ injustice ∧ ¬ standing + injustice ∧ ¬ racist + injustice.
cheating    :-  ¬ solidarity ∧ justice ∧ injustice ∧ ¬ standing + injustice ∧ racist + injustice.
fairness    :-  ¬ solidarity ∧ justice ∧ injustice ∧ standing + injustice.
loyalty     :-  ¬ injustice ∧ ¬ disobedience ∧ ¬ pseudoscience ∧ ¬ justice.
loyalty     :-  ¬ injustice ∧ ¬ disobedience ∧ ¬ pseudoscience ∧ justice.
harm        :-  ¬ injustice ∧ ¬ disobedience ∧ pseudoscience.
subversion  :-  ¬ injustice ∧ disobedience.
cheating    :-  ¬ solidarity + injustice.
loyalty.
```

Figure 9.6: Logic program $\mathcal{P}$ obtained from the GELPE extraction process when leveraging SHAP as LPE and $\mathcal{K} = 100$ on the BLM dataset.

model, but also an efficient one.

## Chapter Synopsis

In this chapter we analyse the possible efficiency benefits arising from SKE approaches. We focus on the NLP domain, proposing to build *global explainers* surrogates from the outputs of LPE approaches. The explanation extraction process – namely GELPE – distills the LLM knowledge under the form of a logic program leveraging the popular CART model. We test GELPE over a broad set of scenarios, highlighting its fidelity against the sub-symbolic model and the simplicity of the extracted knowledge. Moreover, we analyse the efficiency of the extracted logic programs, showing how it is possible to extract a logic program that is equivalent to the original LLM and is *faster* and *less energy wasteful* in scenarios where hardware acceleration is not available. Therefore, our experiments show how the extraction process can be leveraged to enable the deployment of NLP applications to resource-constrained environments, such as embedded devices and microcontrollers. These findings also highlights how – for some learning tasks – leveraging LLMs might represents an over complication, as it is possible to achieve similar performance using simple and small logic programs.

Table 9.9: Resource efficiency comparison of BERT against GELPE for each dataset. For each dataset, we highlight in blue the most energy efficient model, in brown the least energy efficient one, in green the quickest model and in red the slowest one.

| Model \ Dataset | SMS | YOUTUBE | TREC | ALM | BLM | BLT | ELE | MT | SND |
|---|---|---|---|---|---|---|---|---|---|
| $\text{BERT}_{GPU}$ | $\bar{t} = 0.017s$ | $\bar{t} = 0.009s$ | $\bar{t} = 0.008s$ | $\bar{t} = 0.006s$ | $\bar{t} = 0.006s$ | $\bar{t} = 0.006s$ | $\bar{t} = 0.006s$ | $\bar{t} = 0.007s$ | $\bar{t} = 0.006s$ |
| | $\overline{E} = 2.841J$ | $\overline{E} = 2.350J$ | $\overline{E} = 0.987J$ | $\overline{E} = 1.181J$ | $\overline{E} = 1.209J$ | $\overline{E} = 1.481J$ | $\overline{E} = 1.196J$ | $\overline{E} = 1.961J$ | $\overline{E} = 1.148J$ |
| $\text{BERT}_{CPU}$ | $\bar{t} = 0.047s$ | $\bar{t} = 0.066s$ | $\bar{t} = 0.023s$ | $\bar{t} = 0.027s$ | $\bar{t} = 0.028s$ | $\bar{t} = 0.029s$ | $\bar{t} = 0.026s$ | $\bar{t} = 0.049s$ | $\bar{t} = 0.026s$ |
| | $\overline{E} = 5.008J$ | $\overline{E} = 7.893J$ | $\overline{E} = 2.421J$ | $\overline{E} = 2.940J$ | $\overline{E} = 3.037J$ | $\overline{E} = 3.141J$ | $\overline{E} = 2.906J$ | $\overline{E} = 5.576J$ | $\overline{E} = 2.719J$ |
| $\text{SHAP}_{50}$ | $\bar{t} = 0.009s$ | $\bar{t} = 0.004s$ | $\bar{t} = 0.004s$ | $\bar{t} = 0.008s$ | $\bar{t} = 0.011s$ | $\bar{t} = 0.005s$ | $\bar{t} = 0.006s$ | $\bar{t} = 0.008s$ | $\bar{t} = 0.008s$ |
| | $\overline{E} = 0.574J$ | $\overline{E} = 0.223J$ | $\overline{E} = 0.269J$ | $\overline{E} = 0.492J$ | $\overline{E} = 0.595J$ | $\overline{E} = 0.307J$ | $\overline{E} = 0.383J$ | $\overline{E} = 0.456J$ | $\overline{E} = 0.490J$ |
| $\text{LIME}_{50}$ | $\bar{t} = 0.004s$ | $\bar{t} = 0.004s$ | $\bar{t} = 0.010s$ | $\bar{t} = 0.021s$ | $\bar{t} = 0.026s$ | $\bar{t} = 0.005s$ | $\bar{t} = 0.015s$ | $\bar{t} = 0.020s$ | $\bar{t} = 0.013s$ |
| | $\overline{E} = 0.208J$ | $\overline{E} = 0.260J$ | $\overline{E} = 0.592J$ | $\overline{E} = 1.189J$ | $\overline{E} = 1.455J$ | $\overline{E} = 0.283J$ | $\overline{E} = 0.891J$ | $\overline{E} = 1.121J$ | $\overline{E} = 0.777J$ |
| $\text{SHAP}_{250}$ | $\bar{t} = 0.010s$ | $\bar{t} = 0.012s$ | $\bar{t} = 0.019s$ | $\bar{t} = 0.032s$ | $\bar{t} = 0.035s$ | $\bar{t} = 0.018s$ | $\bar{t} = 0.025s$ | $\bar{t} = 0.026s$ | $\bar{t} = 0.034s$ |
| | $\overline{E} = 0.556J$ | $\overline{E} = 0.694J$ | $\overline{E} = 1.115J$ | $\overline{E} = 1.736J$ | $\overline{E} = 1.968J$ | $\overline{E} = 1.015J$ | $\overline{E} = 1.403J$ | $\overline{E} = 1.432J$ | $\overline{E} = 1.859J$ |
| $\text{LIME}_{250}$ | $\bar{t} = 0.016s$ | $\bar{t} = 0.034s$ | $\bar{t} = 0.084s$ | $\bar{t} = 0.144s$ | $\bar{t} = 0.235s$ | $\bar{t} = 0.024s$ | $\bar{t} = 0.087s$ | $\bar{t} = 0.106s$ | $\bar{t} = 0.078s$ |
| | $\overline{E} = 0.866J$ | $\overline{E} = 1.789J$ | $\overline{E} = 4.696J$ | $\overline{E} = 7.990J$ | $\overline{E} = 13.047J$ | $\overline{E} = 1.388J$ | $\overline{E} = 4.808J$ | $\overline{E} = 5.797J$ | $\overline{E} = 4.364J$ |

# Chapter 10

# Robustness of Symbolic Knowledge Injection as a Proxy for Efficientisation

While measuring directly the *efficiency* improvements achieved via SKI represents a valid approach, we here stress the importance of measuring the injection mechanism *robustness*. Robust mechanisms can effectively incorporate new knowledge into neural networks without causing errors or deteriorating their overall performance. In safety-critical applications, like autonomous vehicles or medical diagnosis systems, a non-robust mechanism may introduce errors, leading to poor performances. A robust mechanism can seamlessly adapt to new knowledge, ensuring the quality of performance and enhancing the trustworthiness of NN systems. Moreover, a robust mechanism represents a more efficient solution by design, as it ensures quick adaptability to different knowledge and setups, thus avoiding expensive re-optimization procedures. Therefore, we here consider measuring the robustness of injection approaches as a valuable proxy for the identification of their efficiency.

Robustness can be assessed by evaluating the injected model's ability to maintain its performance in the presence of data perturbations, such as noisy or corrupted training samples. Measure the robustness of injection mechanisms is a crucial stage in developing neuro-symbolic systems, and it should be included in the process of evaluating any SKI approach. Accordingly, in this chapter, we present a comprehensive modelling of a new robustness metric for SKI, as well as an empirical evaluation of this metric through PSyKI [Magnini et al., 2022b]. Our findings provide compelling evidence that the introduction of a robustness metric for SKI is a crucial step towards enhancing the reliability and transparency of AI systems. Finally, the proposed robustness measure directly links with the efficiency of SKI, confirming the findings obtained in Chapter 8.

## 10.1 Robustness Score for SKI

In counterfactual analysis [Verma et al., 2020], it is a common practice to exploit controlled perturbation on the training data to assess the robustness of predictors. Taking inspiration from this technique, in this section we introduce the notion of *statistical robustness* of SKI procedure. Accordingly, we consider an injection mechanism as *robust* if the predictive performance of the educated predictors is poorly affected by *perturbations* of the training data—as long as the perturbation *magnitude* is small. The remainder of this section provides a general description of robustness, before delving into the details of which perturbations types can be applied to training data and how their magnitude may be measured.

**Data perturbation** We define *data perturbation* as altering a training dataset $D$ by adding, removing, or editing its entries, and denote it by $\Delta D$. Accordingly, we denote the perturbed dataset as $D' = D \circ \Delta D$, where $(\cdot \circ \cdot)$ is the perturbation *application* operator. We also denote by $\|\Delta D\| \in \mathbb{R}_{\geq 0}$ the *magnitude of the perturbation*—i.e., the scalar value quantifying the amount of changes induced by the perturbation.

**Robustness score** Let $\mathbf{D} = \{\Delta D_1, \ldots, \Delta D_n\}$ be a set of potential data perturbations to be applied to some dataset $D$, let $N$ be a predictor of any sort – trained on $D$ –, and let $N_{\Delta D}$ be the predictor having the same hyperparameters of $N$, yet being trained upon the perturbed dataset $D \circ \Delta D$. Under such hypotheses, we define the *robustness score* of $N$ w.r.t. $\mathbf{D}$, as follows:

$$\rho_{N,D}(\mathbf{D}) = \frac{1}{n} \sum_{\Delta D \in \mathbf{D}} \|\Delta D\| \cdot \frac{\pi(N_{\Delta D}, D \circ \Delta D)}{\pi(N, D)} \tag{10.1}$$

where $\pi$ is a performance metric of choice, such as accuracy, computing the performance of the input predictor w.r.t. the input dataset.

As the reader may notice, robustness is *directly* proportional to *(i)* the magnitude of the perturbations, and *(ii)* the ratio among the performance of the perturbed predictors and the performance of the unperturbed predictor. In other words, the robustness of a predictor increases when relatively big perturbations in the training data have a relatively small effect on the performance of the predictor.

The robustness of some injection mechanism can be measured by applying Equation (10.1) to some educated predictor $\hat{N} = \mathcal{I}(N, K, D)$, attained by injecting the knowledge $K$, on some uneducated predictor $N$, then trained upon $D$—which we denote denote by $\rho_{\hat{N},D}(\mathbf{D})$. One may also be interested in understanding whether some injection mechanism makes the predictor more or less robust than its uneducated counterpart. To this end, we define the *robustness gain* score as follows:

$$R_{N,D}(\mathcal{I}) = \frac{\rho_{\hat{N},D}(\mathbf{D})}{\rho_{N,D}(\mathbf{D})} \tag{10.2}$$

Here, the robustness gain is a positive measure that indicates if the injection mechanism $\mathcal{I}$ produces a more robust predictor ($R_{N,D}(\mathcal{I}) > 1$) w.r.t. its uneducated counterpart over data perturbation. Meanwhile, injection mechanisms suffering data perturbations result in $R_{N,D}(\mathcal{I}) < 1$, as they produce an educated model $\hat{N}$ less effective for dealing with perturbed data. Therefore, $R_{N,D}(\mathcal{I})$ is an easy-to-understand measure for analysing the robustness quality of a selected SKI mechanism.

### 10.1.1 Measuring Data Perturbations

Equation (10.1) defines robustness by relying on the possibility of measuring the magnitude $\|\Delta D\|$ of any given perturbation $\Delta D$. While being easy to model in theory, the problem is hard to tackle in practice. Therefore, we focus on the simpler goal of measuring the *difference* among any two datasets $A, B$, leveraging the Kullback-Leibler (KL) divergence [Joyce, 2011] to serve this purpose.

The KL-divergence is a statistical operator aimed at measuring the difference among any two probability distributions $\alpha$ and $\beta$. In the particular case where *(i)* $\alpha$ and $\beta$ are multivariate normal distributions having the same dimensionality, and *(ii)* two datasets $A, B$ are sampled from $\alpha$ and $\beta$ respectively, the KL-divergence can be computed as follows:

$$\begin{aligned}\psi(A,B) = \tfrac{1}{2}\Big[&\mathrm{tr}(\boldsymbol{\sigma}_B^{-1}\boldsymbol{\sigma}_A) - \dim(A) \; + \ln\left(\tfrac{\det\boldsymbol{\sigma}_B}{\det\boldsymbol{\sigma}_A}\right) + \\ &+ (\boldsymbol{\mu}_B - \boldsymbol{\mu}_A)^\top \boldsymbol{\sigma}_B^{-1}(\boldsymbol{\mu}_B - \boldsymbol{\mu}_A)\Big]\end{aligned} \tag{10.3}$$

where $\boldsymbol{\mu}_A$ (resp. $\boldsymbol{\mu}_B$) represents the mean of $A$ (resp. $B$), and $\boldsymbol{\sigma}_A$ (resp. $\boldsymbol{\sigma}_B$) represents its covariance matrix, $\det\boldsymbol{\sigma}_A$ (resp. $\det\boldsymbol{\sigma}_B$) is its determinant, and $\mathrm{tr}(\boldsymbol{\sigma}_A)$ (resp. $\mathrm{tr}(\boldsymbol{\sigma}_B)$) its trace. Finally, $\dim(A)$ represents the dimensionality of $A$—i.e., the amount of features each of its entries if characterised by.

The normality hypothesis may appear restrictive, as datasets $A$ and $B$ may, in the general case, be sampled from unknown distributions. However, any unknown probability distribution can be approximated by a *mixture* of normal distributions – as the latter are universal approximators for probability densities [Goodfellow et al., 2016, Sec. 3.9.6] –, and the computation of the KL-divergence for a mixture of normal distributions can be approximated to that of a single normal distribution—as discussed in [Joyce, 2011]. So, in practice, Equation (10.3) can be exploited to estimate the difference among any two datasets $A$ and $B$.

**KL-divergence for classification problems.** Similarly to what done in the previous chapters of this thesis, we here focus on multi-class classification tasks. Therefore, we need to extend the KL-divergence measure to the cases in which datasets comprise several classes of samples. In this context, the KL divergence is commonly computed per-class, as different classes may come with different distributions. In case $K$ classes are available, we consider datasets as partitioned into disjoint sub-sets, on a per-class basis: i.e., $A = A_1 \cup \ldots \cup A_K$ and $B = B_1 \cup \ldots \cup B_K$, where $A_k$ (resp. $B_k$) is the set of samples belonging to class $k$ in $A$ (resp. $B$).

We define the *overall* divergence score between $A$ and $B$ as the weighted average of class-specific Kullback-Leibler divergences, namely:

$$\Psi(A, B) = \frac{1}{|A|} \sum_{k=1}^{K} \psi(A_k, B_k) \cdot |A_k| \tag{10.4}$$

where $|A|$ (resp. $|A_i|$) represents the cardinality of $A$ (resp. $A_i$). The weighted average is necessary to tackle very unbalanced datasets with diverging distributions over unpopular labels.

Accordingly, we estimate the magnitude of a perturbation $\Delta D$ transforming $D$ into $D' = D \circ \Delta D$ by means of the overall divergence among $D$ and $D'$:

$$\|\Delta D\| = \Psi(D, D') \tag{10.5}$$

The proposed metric lays in $\mathbb{R}_{\geq 0}$: it is equal to zero for two identical datasets – i.e., when the perturbation $\Delta D$ has no effect –, and it gets higher in value as $D'$ differs from $D$—i.e., proportionally to the effect of $\|\Delta D\|$.

It is worth highlighting that the proposed perturbation metric $\|\cdot\|$ can be applied to any sort of perturbation, as it does not take into account *how* the perturbation $\Delta D$ affects the dataset or its samples. For this reason, the metric can be exploited to measure the many sorts of perturbations discussed in Section 10.1.2.

## 10.1.2 Perturbation Strategies

Here we discuss three major perturbation strategies for altering a dataset, namely: *(i)* sample drop, *(ii)* noise addition, and *(iii)* label flipping. These strategies mimic common issues that can degrade a dataset's quality, as they involve randomly deleting samples, adding random noise, and randomly changing labels.

### Sample Drop

This perturbation strategy mimics the effect of lacking training data. Hence, it aims at selectively mutilating a dataset in a *controlled* way. This strategy is commonly exploited to test if and to what extent neuro-symbolic approaches are effective to deal with data scarcity [Xu et al., 2018, Deschamps and Sahbi, 2022].

The basic idea behind sample drop is to randomly remove some samples from $D$ to obtain $D'$. The whole process is stochastic and controlled by a single parameter, namely the *dropping probability* – denoted by $p \in [0, 1[$ –, which is shared among all data entries in $D$. Generally speaking, $p = \mathbb{E}[X_d]$ represents the mean of the Bernoulli-distributed random variable $X_d \sim \mathcal{B}(p)$, dictating whether the data entry $d \in D$ should ($\mathbb{P}(X_d = 0)$) or should not ($\mathbb{P}(X_d = 1)$) be included in $D'$. Under such hypotheses, the perturbed dataset $D'$ is such that $D' = \{d \mid \forall d \in D \text{ s.t. } X_d = 0\}$.

We consider constructing the set of data perturbations $\mathbf{D} = \{\Delta D_1, \ldots, \Delta D_n\}$ by applying the sample drop process on $D$ multiple times – namely, $n$ times – with increasing dropping probabilities $0 < p_1 < \ldots < p_n < 1$.

### Noise Addition

This perturbation strategy mimics the situation where the data sampling / acquisition process is affected by error—e.g., due to sensor noise, or human error. Hence, it aims at degrading a dataset in a *controlled* way. This strategy is commonly exploited to test if and to what extent neuro-symbolic approaches are effective to deal with unreliable (e.g., corrupted, inconsistent) data [Raissi et al., 2019, Yazdani et al., 2020].

The basic idea behind noise addition is to add some random noise to the data entries in $D$, to obtain $D'$. The whole process is stochastic, and it is controlled by a single parameter, namely the *noise intensity* – denoted by $v \in \mathbb{R}_{\geq 0}$ –, which is shared among all data entries in $D$. Generally speaking[1], $v \cdot \mathbf{1} = \text{cov}(V_d)$ represents the covariance matrix of the 0-mean, multi-variate, normally distributed random variable $V_d \sim \mathcal{N}(\mathbf{0}, v \cdot \mathbf{1})$, representing the random noise to be applied to each entry $d \in D$. Under such hypotheses, the perturbed dataset $D'$ can be described as $D' = \{d + V_d \mid \forall d \in D\}$. As the reader may notice, the perturbed dataset is characterised by the same number of samples of $D$—meaning that $|D| = |D'|$.

We consider constructing the set of data perturbations $\mathbf{D} = \{\Delta D_1, \ldots, \Delta D_n\}$ by applying the noise addition process on $D$ multiple times – namely, $n$ times – with increasing noise intensities $0 < v_1 < \ldots < v_n$.

[1] $\mathbf{1}$ is the $m \times m$ identity matrix, $\mathbf{0}$ is the $m \times 1$ zero vector, where $m$ is dimensionality of the dataset.

**About discrete features.** Normal noise can be applied to any *continuous* feature in a dataset. However, when dealing with datasets with ordinal or categorical features, it is necessary to adjust the noise addition process to account for discontinuity. Accordingly, to deal with discrete features, the additive normal noise is discretised by replacing the distribution $\mathcal{N}(\mathbf{0}, v \cdot \mathbf{1})$ with its discrete counterpart $\bar{\mathcal{N}}$ [Canonne et al., 2020, Kairouz et al., 2021]. Overall, this means that for *ordinal* features, additive noise is more likely to choose an additional ordinal value close to the original one, whereas for *categorical* features, additive noise may choose an additional ordinal value with uniform probability—analogously to the label flipping case described below.

### Label Flipping

This perturbation strategy mimics the situation where some data labelling process is affected by error—e.g., human operator misabelling some data entries. Hence, it aims at selectively flipping the labels of some entries in a dataset. The underlying assumption is that the dataset consists of input and output features, and the output ones are either binary or categorical. Therefore, this perturbation strategy mostly makes sense for classification tasks.

The basic idea behind label flipping is to randomly alter the output features of some data entries in $D$, to produce $D'$. Flipping, in particular, requires each output feature to be randomly re-assigned with some value in the output domain—of course different from the original one.

Without loss of generality, we consider the case of a single output feature having $K$ admissible values—i.e., $K$ classes represented as a single categorical attribute. Hence, for an $m$-dimensional dataset $D$, we let $d \equiv (\mathbf{x}_d, y_d) \in D$ denote the generic data entry in $D$, where $\mathbf{x}_d$ is the input sample, and $y_d \in \{1, \ldots, K\}$ is the value of its output feature.

Similarly to the cases above, we model label flipping as a stochastic process controlled by the *flipping probability* parameter – denoted by $f \in [0, 1[$ –, representing the likelihood that the label of any given data entry may flip. Parameter $f$ controls the probability distribution of the random variable $Y_d$, which represents the novel value of the output feature for data entry $d \in D$, after flipping. In particular, we define the probability density of $Y_d$ as follows: $\mathbb{P}(Y_d = k) = f/(K-1)$ if $k \not\equiv y_d$; otherwise $\mathbb{P}(Y_d = y_d) = (1-f)$. In other words, the value of the output feature may flip with probability $f$, and in that case each value different than the original one is equally likely to be selected.

Under such hypotheses, the perturbed dataset $D'$ is such that

$$D' = \{(\mathbf{x}_d, y'_d) \mid \forall d \in D : Y_d = y'_d\}. \tag{10.6}$$

We consider constructing the set of data perturbations $\mathbf{D} = \{\Delta D_1, \ldots, \Delta D_n\}$ by applying the label flipping process on $D$ multiple (i.e., $n$) times, with increasing flipping probability $0 < f_1 < \ldots < f_n < 1$.

## 10.2 Experiments

In this section we present the setups and results of the experiments for evaluating the effectiveness of the proposed robustness metric.

### 10.2.1 Datasets

We rely on three different datasets from the UCI repository[2], namely the "Breast Cancer Wisconsin" (BCW) dataset [Wolberg, 1992], the "Primate Splice Junction Gene Sequences" (PSJGS) dataset [Towell and Shavlik, 1994], and "Census Income" (CI) dataset [Kohavi and Becker, 1996]. As the datasets are the same used in the experimental evaluation of Chapter 8, we avoid a detailed description of the datasets and refer the reader to Section 8.4.1.

### 10.2.2 Injected Knowledge

For our experiments we utilize specific knowledge for each dataset, using Prolog syntax [Körner et al., 2022] for classification logic rules The PSJGS dataset already has a knowledge base in the literature, with biological rules provided by human domain experts [Towell et al., 1990]. We use this very knowledge as well, with the

[2]https://archive.ics.uci.edu/ml/index.php

addition of an explicit third rule to classify the `n` class:

$$class(\bar{X}, \texttt{n}) \leftarrow \neg class(\bar{X}, \texttt{ei}) \wedge \neg class(\bar{X}, \texttt{ie}) \tag{10.7}$$

where $\bar{X}$ is an abbreviation for the full sequence of variables $X_{-30}, \ldots, X_{+30}$ that represent the input data of a 60-basis long DNA sequence. The rule simply states that if a record is not classified as `ei` or `ie` then it is classified as `n`. For the remaining datasets – BCW and CI – there are no predefined rules in the literature. Therefore, we define a custom set of logic rules for both of them.

### 10.2.3 Injectors

Similarly to what done in Chapter 8, to implement our experimental evaluation we rely on PSyKI. Therefore, the set of injection algorithms consider in this section are *KINS*, *KILL*, and *KBANN*. To avoid redundancy, we refer the reader to Section 8.3 for a detailed introduction of such algorithms.

### 10.2.4 Experimental Setup

For our experiments, we leverage three datasets, three input knowledge bases, three different injectors, and three perturbation strategies. For each dataset we train an uneducated model, which is then applied to the three injectors using the corrisponding knowledge base. The uneducated model has two hidden layers, one input, and one output layer, with rectified linear units (ReLU) as the activation function, and softmax for the output layer. The number of neurons per hidden layer changes depending on the dataset—namely, $[16, 8]$ for BCW, $[32, 16]$ for CI, and $[64, 32]$ for PSJGS. Training involves *categorical cross-entropy* loss function, a batch size of 32 elements, and an epoch limit of 100.

Data perturbation is applied multiple times for each dataset-model configuration, using sample drop, noise addition, and label flipping. More precisely, the drop probability ($d$) parameter varies from 0.0 to 0.95 with a step of 0.05, the flipping probability ($f$) parameter varies from 0.0 to 0.90 with a step of 0.08, and the noise intensity ($v$) parameter varies from 0.0 to 1.0 with a step of 0.1. Each experiment is repeated 30 times to draw statistical comparisons.

### 10.2.5 Results and Discussion

Here we present the results of our experiments (Figure 10.1) over different perturbation strategies, grouped by the datasets and the predictors they are applied to.

To better understand the real differences in performance between educated and uneducated predictors—which are fairly comparable—we compare the average accuracy of each predictor w.r.t. the uneducated one. For this purpose, we employ the *Mann-Whitney U Test* [McKnight and Najab, 2010], a non-parametric test for differences between two groups. In this setting, a *p-value* $\geq 0.05$ indicates that there are no significant differences between the two average accuracy distributions, whereas a *p-value* $< 0.05$ indicates the opposite.

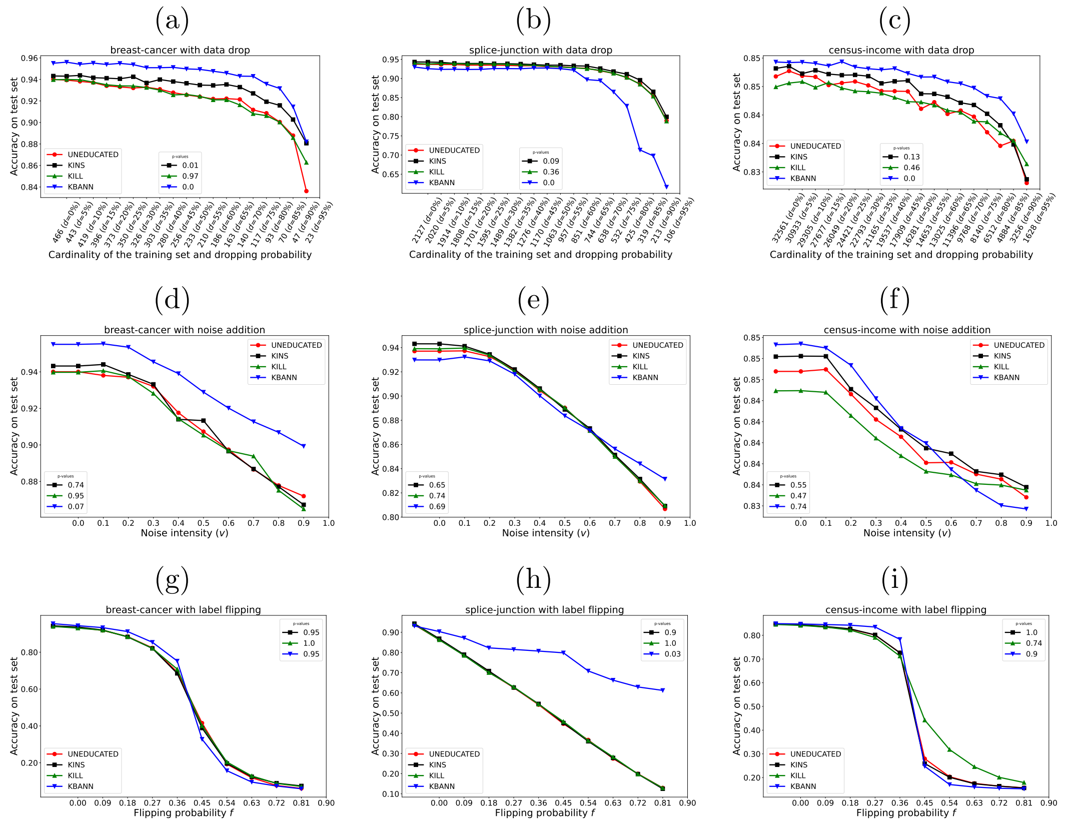


Figure 10.1: Average accuracy over different datasets with different perturbation strategies

Table 10.1: Robustness relative scores. Bold numbers are the ones greater than 1 (i.e., the educated model is more robust than the uneducated one).

| Dataset | $R_{N,D}(\mathcal{I})$ drop | | | $R_{N,D}(\mathcal{I})$ noise | | | $R_{N,D}(\mathcal{I})$ flip | | |
|---|---|---|---|---|---|---|---|---|---|
| | KINS | KILL | KBANN | KINS | KILL | KBANN | KINS | KILL | KBANN |
| BCW | **1.0493** | **1.0318** | **1.0382** | 0.9960 | 0.9985 | **1.0109** | 0.9994 | **1.0184** | 0.9520 |
| PSJGS | **1.0045** | 0.9968 | 0.8425 | 0.9950 | 0.9984 | **1.0145** | 0.9962 | **1.0026** | **1.6749** |
| CI | 0.9998 | **1.0039** | **1.0043** | 0.9992 | **1.0012** | 0.9965 | 0.9897 | **1.1703** | 0.9815 |

### Drop

In the data drop experiments (Figure 10.1 a-c), KBANN is the only predictor that shows significant differences w.r.t. the uneducated one. Specifically, KBANN demonstrates improved performance on the BCW (Figure 10.1a) and CI (Figure 10.1c) datasets, but its performance on the PSJGS (Figure 10.1b) dataset rapidly declined when 60% of the data are dropped ($d = 0.6$). For the other educated predictors, KINS shows slightly better performances than the uneducated model. In the BCW dataset its *p-value* $= 0.01$ indicates that there are significant differences in terms of performance compared to the uneducated model.

Looking at Table 10.1, one can see the influence of the performance values on the robustness score, e.g. for KBANN in the PSJGS dataset. Apart from this, the educated models improve robustness in 6 out of 9 experiments.

### Noise

The noise experiments (Figure 10.1 d-f) reveal that SKI mechanisms are more sensitive to noisy data than missing data. In fact, the average accuracy of predictors trained over noisy data drops more quickly than the data drop scenario. This is due to the downward trends exhibited by all predictors since the early stages of the curves. Indeed, Table 10.1 confirms that educated models enhance robustness in only 3 experiments out of 9.

KBANN outperforms other SKI methods and the uneducated model in the BCW (Figure 10.1d), but performs poorly in the CI (Figure 10.1f) dataset. In PSJGS (Figure 10.1e) dataset, it has low performances too w.r.t. the uneducated predictor —except for high noise levels, where it is subject to relatively slower performance decline. This explain why in Table 10.1 KBANN's robustness score shows a gain over the uneducated one. The opposite trend, on the other hand, can be observed for the CI scenario.

Along this line, it is worth of notice observing that KINS – despite being a SKI approach based on neural structuring like KBANN – performs noticeably lower than KBANN in terms of robustness, regardless of the dataset. The discrepancy may be caused by the trainable nature of KINS' neural modules, which makes them more prone to overfitting noisy data. It is noticeable that KILL consistently exhibits similar (or worse) behaviour across all datasets than the uneducated model. This may imply that the penalties imposed by KILL during training for performing injection are not compensating for noise perturbations.

### Label-flipping

The label flipping experiments (Figure 10.1 g-i) show that predictors behave similarly in both BCW (Figure 10.1g) and CI (Figure 10.1i) datasets. This similarity between all predictors can also be observed when looking at *p-values* which are all close to 1—i.e. no significant difference between educated and uneducated predictors. In both, BCW and CI, there is a quick degradation in performance starting from 54% of flipped labels ($f = 0.54$). Conversely, for the PSJGS dataset (Figure 10.1h), the performance decline pattern is linear. One notable exception is KBANN, that exhibits an impressive performance, retaining nearly 70% accuracy even for highest flipping probability values ($f = 0.9$)—as opposed to other models whose accuracies drop to 20%. Such behavior is further emphasised by Table 10.1, where KBANN demonstrates a remarkable improvement in robustness w.r.t. the uneducated model.

Except for KBANN, the other predictors exhibit similar performance trends w.r.t. the uneducated model across all three scenarios. Looking at Table 10.1, KBANN generally experiences a slight decrease in robustness compared to the uneducated model—except for the PSJGS case. On the other hand, KILL emerges as the predictor that best withstands the perturbations caused by label flipping.

**Discussion** These experiments demonstrate that among the introduced perturbations, data drop is the one where SKI methods exhibit the greatest gains in robustness, indicating their ability to compensate for lacking data through integration of prior knowledge and confirming the findings of Chapter 8. The PSJGS dataset is an exception, indeed the injected knowledge is far from optimal, being good for classifying only one class out of three. In this case, poor prior knowledge combined with missing data may hinder the model from learning underlying patterns. For noise perturbations, the SKI methods employed prove insufficient to improve robustness. For label flipping perturbations, the constraining method—KILL—demonstrates good robustness across all three datasets, suggesting that penalty added during training is sufficient to compensate for label flipping. The exception of KBANN on PSJGS likely stems from its strong adherence to the injected knowledge, which provides useful joint information on splice junctions and mitigates the influence of flipped labels.

## Chapter Synopsis

In this chapter we propose a novel metric to assess the robustness of SKI mechanisms. As such, we define three different types of data perturbation strategies and a way to assess their intensity. We also provide a formula to calculate the robustness score of a predictor, assessing its robustness when the data is perturbed. Furthermore, we introduce a comparison to better understand whether the SKI predictor is better (or not) than its uneducated counterpart from a robustness perspective.

Overall, our experiments indicate that our proposed metric can be used to effectively determine the robustness of the performance variations obtained through SKI. The experimental findings highlight how SKI approaches represent a valuable approach for achieving robustness in the data-drop scenario. Moreover, SKI approaches seem to achieve better results when part of the data labels are not fully reliable—i.e., label flipping scenario. These findings highlight once again the strenght of SKI models for learning from fewer data and/or mislabeled data, thus identifying a more *data efficient* solution with respect to standard NN models.

# Chapter 11

# Measuring Other Relevant Properties of Neuro-Symbolic Integration

This chapter contains contributions from [Agiollo and Omicini, 2023].

Resource efficiency represents one of the pillars of AI trustworthiness as defined by the Ethics Guidelines for Trustworthy AI[1] released by the European Union (EU) as a part of its AI strategy. However, these ethics guidelines apparently focus on popular ML solutions in their definition process. Indeed, most trust requirements are clearly linked with the black-box nature of ML and DL solutions—such as the need for transparency, explanations, human interaction, and many others. The result is the current lack of suitable definitions of the notion of trustworthiness in terms of NeSy systems. This is why in this chapter we expand the umbrella of our analysis and deal with the definition of trustworthiness for NeSy systems, focusing specifically on SKI and SKE approaches. Although this analysis does not directly help in identifying more efficient NeSy models, it represents a fundamental effort to identify the missing components for the definition of fully trustworthy NeSy systems. Therefore, since full trustworthiness requires maximum efficiency, we include this chapter to shed some light on the foreseeable future developments of

[1] https://digital-strategy.ec.europa.eu/en/library/ethics-guidelines-trustworthy-ai

NeSy systems, and to help the reader understand the open gaps in the literature and the available opportunities.

The definition of requirements for trustworthy NeSy systems represents a fundamental step towards their safe adoption. However, requirements definition by itself can not be considered as an exhaustive measure to ensure and calibrate the trustworthiness of NeSy systems. Instead, it is of utmost significance to define NeSy *trustworthiness metrics* that allow to actually measure the level of a system trust, possibly enabling an in-depth analysis of the components raising trust concerns. Whereas a few trustworthiness metrics definition already exist, tackling specific components of NeSy models – such as accuracy, robustness and efficiency as seen in the previous Chapters 8 to 10 –, the vast majority of NeSy most relevant aspects are still unexplored. This is why in this chapter we:

- define how the AI trustworthiness requirements translate to the NeSy realm, analysing in detail each pillar of trust and its implication on NeSy models.
- analyse the available metrics for each of the novel NeSy trust requirements as well as the potential future directions to explore in the analysis of NeSy trust;
- suggest some novel metrics to measure specific NeSy elements, focussing on SKI and SKE.

## 11.1 From Trustworthy AI to Trustworthy NeSy

As a fundamental step of its AI strategy, the EU has defined seven key trustworthiness criteria to meet during the development, deployment, and use of AI systems, namely: *(i) human agency and oversight*, as the need for oversight mechanisms enabling the informed interaction between the AI system(s) and the human(s) counterpart; *(ii) robustness and safety*, as the need for accuracy, reliability, resilience and security of AI system(s); *(iii) privacy and data governance*, as the need for ensuring legitimised access to data, while taking into account data quality and integrity; *(iv) transparency*, as the need for providing human users with explanations of the AI system(s)'s decision process; *(v) diversity, non-discrimi-*

*nation and fairness*, as the need for avoiding unfair bias while enable everyone's access to AI technology; *(vi) environmental and societal well-being*, as the need for sustainability of AI system(s) and the transition to their environmentally friendly development; *(vii) accountability*, as the need for mechanisms that ensure responsibility and accountability for the behaviour and outcomes of AI systems. The above requirements define a broad umbrella of concepts and means to identify relevant components in the deployment of AI systems and ensure their trustworthiness. However, being designed to be general enough to be applicable to any – or at least as most as possible – AI systems, they are actually too general to be used to define actual metrics to effectively measure every sort of AI systems. Therefore, to make them actually working, a more detailed specification of trustworthiness requirements is needed: in particular, the general EU pillars should be *translated* into domain-specific pillars, promoting the definition of trustworthiness metrics for each specific AI domain. Such a translation should also account for the current bias of EU trustworthiness pillars towards subsymbolic AI systems—where, for instance, the black-box nature of all components is given as understood when dealing with issues such as transparency, explainability, human interaction, even though it mostly concerns subsymbolic components only.

Thus, in the remainder of this chapter we define the pillars of trustworthiness for AI systems based on Neuro-Symbolic integration. We analyse the seven EU-defined trustworthiness criteria for AI, and translate each of them into its NeSy counterpart, leveraging on the aspects of NeSy that promote fairness and explainability by design. On the other hand, leveraging both symbolic and subsymbolic paradigms, NeSy systems may be affected by robustness, safety, and bias issues from both sides – i.e., symbolic and subsymbolic –, hindering their overall trustworthiness. Therefore, a fundamental issue in the NeSy context is to identify whether and to what extent the blending of symbolic and subsymbolic techniques can either help or hinder trustworthiness, in particular in the perspective of the definition of ad-hoc trustworthiness metrics.

## Human Agency and Oversight

In its original formulation this requirement stresses the need for the introduction of oversight mechanisms enabling informed interaction between AI systems and humans counterparts. The underlying assumption here is that humans can not understand AI system at all – or, can understand interaction with AI systems in a very limited way – so AI systems can never be considered as trustworthy, as humans are incapable to fix the AI system when issues arise. When taking into account NeSy mechanisms, the symbolic and subsymbolic fusion component clearly affects the interaction with with its human counterpart. Instead, the symbolic component could represent the enabling agent for meaningful interaction between human and the system, promoting human-in-the-loop, human-on-the-loop, and human-in-command approaches.

### NeSy version

> *The need for assessing to what extent the symbolic and subsymbolic* ***interaction*** *of NeSy components helps* ***improving*** *informed human-AI interaction and human oversight.*

## Technical Robustness and Safety

In its original formulation this requirement stresses the need for accuracy, reliability, resilience, and security of AI systems. Indeed, an inaccurate or unstable AI system can not be considered trustworthy, as its behaviour may fluctuate radically throughout its life cycle. Let us consider for instance adversarial examples [Zhang and Li, 2020, Serban et al., 2021], where slight perturbations of the input fed to the AI system result in radically different outcomes: AI system of that sort are inherently unreliable—thus untrustworthy. Even though this has motivated some research efforts focused on the identification of robustness issues of ML/DL systems, very small light has been shed on the robustness and safety issues of NeSy systems. NeSy relies on both symbolic and subsymbolic components, the former being – with some exception – verifiable and stable by design while the latter lacks of stability, verifiability or strong mathematical modeling of their behaviour and properties. The interaction of such elements introduces non-trivial behaviour in NeSy systems, where the symbolic components can be used as a helping tool for stabilising subsymbolic elements or the subsymbolic tools can be used to produce imperfect – thus unreliable – symbolic knowledge. Therefore, we consider relevant studying to what extent the verifiability of symbolic components alters during the integration process, and how the (in)stability of the subsymbolic element is impacted by the symbolic knowledge.

#### NeSy version

> *The need for assessing the impact of both symbolic (verifiable) and subsymbolic (not verifiable)* ***interaction*** *on the* ***stability*** *of the NeSy system.*

### Privacy and Data Governance

In its original formulation this requirement stresses the need for legitimate access to data, while taking into account data quality and integrity. This requirement identifies the untrustworthy nature of systems optimised over unreliable data, and promotes the introduction of open data for testing AI systems and their behaviour. To this end, NeSy systems differ quite heavily from their pure subsymbolic AI counterparts, as they – in most cases – require the processing of symbolic knowledge and data at the same time. Therefore, it is relevant to notice that data quality issues extend to knowledge quality issues when considering NeSy systems—even though symbolic knowledge is typically managed explicitly by AI programmers and is often verifiable in automatic way.

#### NeSy version

> *The need for ensuring the* ***quality*** *of both* ***data*** *and* ***symbolic knowledge*** *of a NeSy system, along with its accessibility.*

## Transparency

In its original formulation this requirement stresses the need for producing explanations of the AI systems' decision processes, deeming as untrustworthy those AI systems for which it is complex or unfeasible to obtain an explanation of its decision process. The definition of an AI system transparency depends on the complexity of the process of obtaining explanations, and their understandability. Indeed, in most AI scenarios multiple explanations can be drawn to render transparent the system at hand, depending on the level of detail needed and the process used. While being conceptually similar, the transparency level of NeSy systems – with respect to their pure subsymbolic AI counterparts – may differ a lot in terms of extraction complexity and understandability. Indeed, most NeSy systems represent a more transparent solution by design, as they leverage symbolic components, inputs or outputs, which are – to some extent – intrinsically understandable by humans. Therefore, we consider relevant to assess if – and to what extent – the integration components of NeSy impacts the transparency of the obtained system(s).

### NeSy version

> *The need for assessing the **gain** in terms of **transparency** obtained by a NeSy system with respect to its pure subsymbolic components.*

## Diversity, Non-Discrimination, and Fairness

In its original formulation this requirement stresses the need for avoiding unfair bias and enable everyone's access to the AI technology. Indeed, biased AI technologies must not be deployed as they have been proven to increase the chance of harmful events against humans. Given the relevance of fairness, several efforts have been put in place to investigate the nature of AI mechanisms' bias. However, biases of pure subsymbolic models and their NeSy counterparts differ conceptually in terms of their root causes: bias can rise in NeSy models as the consequence of any unexpected behaviour of their subsymbolic components, or their interaction with their symbolic elements. Indeed, similarly to what done for NeSy robustness, here it is relevant to highlight that the bias and fairness of symbolic components represent a verifiable and provable variable, while its interaction with subsymbolic elements does not, as it is not possible to define a-priori how the subsymbolic interaction impact the overall system behaviour. Therefore, it is fundamental for NeSy systems to consider possible biases rooted in each step of the fusion between symbolic and subsymbolic components. This is also valid for possible bias benefits that can be obtained from the interaction between symbolic and subsymbolic components in NeSy, as the symbolic elements can be used to tune the subsymbolic components to avoid biases that may arise during their optimisation.

#### NeSy version

> *The need for **measuring** biased and discriminative behaviour of NeSy systems rooted in the **interaction** between their symbolic and subsymbolic components.*

### Environmental and Societal Well-Being

In its original formulation this requirement stresses the need for sustainability of AI systems and the transition to their environmentally friendly development, deeming as untrustworthy those AI systems that do not benefit all human beings, including future generations. While measuring the impact of pure subsymbolic AI systems on the environment has been the focus of several works in the AI community, the in-depth analysis of how NeSy mechanism can help reducing the environmental impact of AI. The symbolic component of several NeSy mechanism can be leveraged as a helping tool for reducing the amount of resources required for the optimisation of its subsymbolic component. Moreover, it is also possible for some NeSy mechanism to leverage symbolic approaches to achieve comparable performance – w.r.t. pure subsymbolic AI – while requiring a smaller memory footprint—resulting in smaller latency and energy consumption. On the other hand, the complex interaction between symbolic and subsymbolic components may introduce an overhead in the NeSy system, causing the waste of resources and thus decreasing the efficiency of the system. Therefore, it is necessary to define a novel resource efficiency requirement for NeSy.

#### NeSy version

> *The need for assessing the **gain** in terms of **sustainability** of NeSy systems with respect to their pure subsymbolic components.*

### Accountability

In its original formulation this requirement stresses the need for mechanisms that ensure responsibility and accountability for AI systems and their outcomes. At its core, accountability can be defined as an obligation to inform about, and justify the AI's conduct [Novelli et al., 2023]. Therefore, the fundamental property for AI's accountability is represented by *answerability*, which is the property of an AI system to allow for *interrogation* concerning a decision process. Accountability is closely tight to transparency, as it requires for an AI system to produce justification – a.k.a. explanations – for its actions. Therefore, a similar analysis to the one done for transparency applies to this context, where we stress the relevance of analysing the accountability gains obtained through symbolic and subsymbolic integration in NeSy systems over ML/DL counterparts.

#### NeSy version

> *The need for assessing the **gain** in terms of **answerability** obtained by a NeSy system with respect to its pure subsymbolic components.*

## 11.2 On the Relevance of Trustworthiness Metrics

The trustworthy requirements proposed for both general AI and NeSy represent a general umbrella of concepts that should be covered in the system at hand. Indeed, none of the requirements defined so far give a specific characterisation of a target level – e.g., target fairness – for that requirement to be considered satisfied. Such general characterisation of trustworthiness is mainly caused by two contributing factors, namely

- *High variability characterising AI systems.* AI systems optimised to solve different tasks are expected to differ largely in terms of inner working principles. Therefore, identifying a common trustworthiness definition with the due level of detail represents a complex task

- *Conceptual complexity of trustworthiness building blocks.* Trustworthiness is defined as a collection of diverse features of a systems to be achieved for it to be worthy of humans' trust. However, some – if not most – of the trustworthiness sub-components are not easy-to-grasp concepts in their definition. For example, taking into account bias, we immediately understand that bias must be one of the sub-components required to achieve trustworthiness. However, the definition of bias by itself represents a complex task that have bogged researchers with troublesome questions like what is bias?, when is a system biased?, what is the minimum amount of bias for a system to be considered as such?. Being complex in their definition, these building blocks are also complex to measure effectively, hindering the overall level of trust measurement.

The issues connected with the general characterisation of AI trustworthiness hinder the applicability of such trustworthiness requirements. Indeed, while representing a valid starting point for analysing AI trustworthiness, these requirements do not fully allow to comprehensively grasp the extent of a system's trustworthiness. To this end, the definition of trustworthiness metrics – rather than requirements or pillars – represents an open issue of the utmost importance. Trustworthiness metrics make it possible to evaluate the extent of a system trust, allowing for a more detailed classification of the AI components to be deployed and the ones to block. However, the definition of a single general, flexible, and ubiquitous trustworthiness metric is made almost impossible by the same issues that affect the generality of trustworthiness requirements. Therefore, we here consider to translate the trustworthiness requirements into a set of equivalent trustworthiness metrics, taking into account the *high variability characterising AI systems* and the *conceptual complexity of trustworthiness building blocks.*

We first consider the issue connected with the *high variability characterising AI systems.* To enable the definition of rigorous trustworthy metrics, we here propose to consider the transition from the general AI trustworthy requirements to the corresponding pillars for each AI branch. Section 11.1 presents a similar transition from trustworthy AI into trustworthy NeSy. A similar transition can be identified for each and every AI domain, obtaining domain-specific detailed trustworthiness requirements. This step enables a stricter definition of trustworthiness for each AI domain, making it possible to focus more specifically on the peculiar approaches, components, and aspects that characterise the domain under analysis.

To tackle the *conceptual complexity of trustworthiness building blocks*, we here propose to avoid focusing on the proposal of single, overly-complex trustworthy metrics with the aim of obtaining a general formulation applicable to any AI system. Rather, we suggest to tackle the measurement of systems' trustworthiness through the adoption of a broad set of highly-specialised metrics that analyse single components of the trustworthiness definition. In this context, we consider proposing a single metric or a set of metrics for each pillar/requirement of trustworthiness. The proposed metrics should focus on a specific issue or feature of the AI system at hand – such as its robustness to specific input perturbation, or the bias towards a specific group –, producing as output a single numeric value, describing its safety level—i.e., how much that issue is alarming for the system. Highly-specialised metrics can then be arbitrarily combined to obtain a dynamic trustworthiness score, depending on the trustworthiness components that are to be considered more relevant for the scenario under examination. This simplified process allows not just the easier definition of each set of trustworthiness metric – e.g., bias metrics, robustness metrics, etc. –, but also the evaluation of set based on a given relevance. Consider for example a scenario where the bias requirement should be considered as more relevant w.r.t. the human oversight requirement. Our approach allows a higher weight to be assigned to the bias metrics before its combination with the human oversight metrics to obtain the general trustworthy measurement. Therefore, we here propose to tackle the trustworthiness measurement issue by adopting a dynamic broad set of highly specific metrics that can be combined depending on the given measurement requirements.

## 11.3 NeSy Metrics for Trustworthiness

In this section we present the trustworthiness metrics (both available and missing ones) for NeSy systems, specifically focusing on SKI and SKE. We analyse each of the seven trustworthiness pillars/requirements separately to obtain a thorough representation of the state-of-the-art and future directions.

### 11.3.1 Human Oversight

NeSy version of human oversight requirement is defined as the need for assessing to what extent the symbolic and subsymbolic *interaction* of NeSy components helps *improving* informed human-AI interaction and human oversight.

#### Available metrics

Most approaches to measure human oversight in AI scenarios focus on aspects of human-AI interaction, where explanation of behaviours represents the most important component of the interaction process. As a results, much attention has been paid to the measurement of how explanations could guide people to respond to and predict the AI system behaviour [de Graaf and Malle, 2017]. A large number of studies exist in this realm, which mainly leverage on users to subjectively rate system predictability, likability, etc.[Huang and Mutlu, 2012] While useful in order to define systems predictability, these studies lack the assessment of human influence and control on the AI system at hand. The reason for this is to be found mainly on the black-box and data-driven nature of subsymbolic models that these works take into account. Indeed, most – if not all – subsymbolic models allow for limited control by the human users, given mostly by the data gathering and selection process.

#### Missing metrics

Unlike pure subsymbolic systems, NeSy models intrinsically enable higher level of human oversight via the integration of symbolic knowledge. However, the extent of such oversight capabilities should be studied in depth through the proposal of ad-hoc metrics that measure how much the behaviour of a NeSy system can be controlled by a human user. To this aim, in the SKI context, we consider proposing a novel metric assessing the impact of the injection process to the underlying model. The impact can be measured as the amount of injected knowledge that is effectively absorbed by the underlying model. The metric would assess the level of available human oversight in SKI systems, allowing for a precise definition of the extent of human control. Meanwhile, the SKE context emphasises the need for measuring the modifiability of the extracted symbolic knowledge from an initial subsymbolic predictor. Indeed, SKE approaches by themselves do not allow for an in-depth control of the model behaviour, but rather enable their inspection. In this context, a desirable solution is represented by refining the extracted knowledge and using it as input for a SKI system acting upon the same subsymbolic model. This process would enable a sort of debugging loop of NeSy systems leveraging both SKE and SKI, with an increased potential for human oversight. Here, we require the definition of an ad-hoc metric capable of assessing the portion of symbolic knowledge that can be extracted, refined and injected back in the system with it being correctly assimilated by the model.

### 11.3.2 Robustness

NeSy version of the robustness requirement is defined as the need for assessing the impact of symbolic (verifiable) and subsymbolic (not verifiable) *interaction* on the *stability* of the NeSy system.

#### Available metrics

The state-of-the-art picture of NeSy robustness emphasises the lack of a common agreement on the definition of robustness itself, thus leading to diverging works focusing on opposite aspects of NeSy systems. Indeed, in this context, several works focus on highlighting the robustness of NeSy models in terms of their performance over complex or out-of-distribution inputs [Li et al., 2022b, Wu et al., 2021a, Liu et al., 2023]. Although relevant for pointing out the potential of NeSy approaches, these works propose somehow misleading definitions of robustness, mostly focusing on NeSy flexibility rather than its stability. NeSy systems may perform well on complex and out-of-distribution samples, while suffering instability on small input perturbations—causing robustness collapse. Several other concepts have been taken into account when considering NeSy robustness such as prediction coherence and consistency [Nye et al., 2021], subsymbolic verification through neuro-symbolic integration [Xie et al., 2022], avoidance of reasoning shortcuts [Marconato et al., 2023] and many more. However, the majority of these approaches not only assess an ad-hoc concept of robustness, but also focus on its qualitative evaluation thus failing to assess the quantitative aspect required to achieve robustness metrics.

While it is true that there exists some confusion concerning the definition of NeSy robustness, there are few relevant works aiming at defining precise robustness metrics. More in detail, Yang et al. [Yang and Chaudhuri, 2022] present a novel learning approach for neuro-symbolic programs, showing its robustness against input perturbations in terms of provably safe portion of the learned model. In this context, NeSy robustness against adversarial attacks represents a popular area of research with several works aiming at proving either qualitatively [Vilamala et al., 2023] or quantitatively [Ibarra-Vázquez et al., 2022] the safety of NeSy approaches. Most of these works define robustness in terms of accuracy degradation over varying input perturbation intensity, independently of the input perturbation type and magnitude.

### Missing metrics

As a result of the mixed focus given to NeSy aspects when tackling robustness, several aspect of NeSy robustness and stability have not been thoroughly analysed, yet. Indeed, there exists the need to study if – and to what extent – the stability and verifiability of symbolic AI components is preserved throughout the integration process in NeSy models. In this context, focusing on the SKI realm, we suggest that a measure of integration stability – as the portion of symbolic elements that are correctly integrated in the injected model – is needed here. Such a metric would basically represent the portion of symbolic control that a NeSy system can attain during its integration step. Secondly, also those scenarios where the symbolic elements of NeSy models suffer from some sort of imperfection have to be taken into account. Here, it is important to measure the stability of SKI models when the injected knowledge is altered as a result of some imperfect automation process.

Finally, it is also relevant to measure the stability of NeSy systems over symbolic representation variability, to assess how different symbolic representations – e.g., logic formulæ, knowledge graphs, etc. – may impact the integration process. To this end, we propose to measure the performance of SKI integration when two syntactically different yet equivalent chunks of symbolic knowledge are exploited in the same integration process.

### 11.3.3 Data & Knowledge Quality

NeSy version of the data & knowledge quality requirement is defined as the need for ensuring the *quality* of both *data* and *symbolic knowledge* of a NeSy system, along with its accessibility.

#### Available metrics

Given the impact of data quality on the optimisation process of ML and DL systems, several quality metrics are available, namely: *(i)* class overlap [Denil and Trappenberg, 2010], *(ii)* boundary complexity [Lorena et al., 2019], *(iii)* label noise [Northcutt et al., 2021], *(iv)* class imbalance [Lu et al., 2020], *(v)* missing value analysis [Corrales et al., 2018], and many more. Although designed for subsymbolic AI models, these metrics translate to the data-driven component of NeSy systems without particular issues, especially in those systems that follow a neural to symbolic – neuro $\rightarrow$ symbolic [Sarker et al., 2021] – pipeline such as SKE approaches. In this context, these metrics makes it possible to check the correctness of the information that the subsymbolic components of NeSy gather from the data.

#### Missing metrics

Unlike pure subsymbolic approaches – which rely solely on data for optimisation –, NeSy models gather information from both a data-driven and a symbolic knowledge component. In this context, it is fundamental to assess the level of compatibility or overlap between the data and the symbolic knowledge to be combined. In most NeSy systems quite a strong overlap is required between data and symbolic knowledge in order to avoid optimisation drift issues, where the integrated knowledge contrasts concepts learnt from the data. Meanwhile, a perfect overlap would also not be ideal in NeSy systems, as the optimisation process would gather the same information from both data and symbolic knowledge. Therefore, we here stress the need for new metrics that could measure the conceptual and technical overlap between data and symbolic knowledge at hand. Another relevant aspect to measure in this context is represented by the quality of the symbolic component of the NeSy system. While symbolic AI approaches are verifiable and deemed trustworthy, several NeSy – especially SKI – approaches rely on the integration of knowledge bases given a-priori and defined by human experts. Although mostly reliable, knowledge bases may be either incomplete or imperfect due to the human-centred building process. Therefore, metrics are needed that would make it possible to score knowledge components exploited in NeSy systems.

### 11.3.4 Transparency

NeSy version of the transparency requirement is defined as the *transparency gain* obtained by a NeSy system with respect to its pure subsymbolic components.

#### Available metrics

When focusing on transparency, most of the available metrics for AI and NeSy models focus on explanations quality evaluation. Generally speaking, explanations quality is characterised by several key attributes [Hoffman et al., 2018], namely: *(i)* understandability – i.e., explanation complexity –; *(ii)* completeness – i.e., explanation coverage –; *(iii)* sufficiency of detail – i.e., explanations depth –; *(iv)* usefulness – i.e., explanation applicability –; and *(v)* feeling of satisfaction—i.e., explanation interactivity. By focusing on some of the above attributes, several works propose explainability and transparency metrics for AI and NeSy. [Nguyen and Martínez, 2020] introduce a set of metrics to evaluate interpretability methods through measurements of simplicity, broadness, and fidelity of explanations. Meanwhile, Holzinger et al. [Holzinger et al., 2020] introduce a system causability scale to measure explanations quality, based on the notion of causability [Holzinger et al., 2019] together with the notion of usability scale. Although designed for explanations in general, these metrics nicely fit in the SKE frame, where they can be used to assess the quality of the extraction mechanism, as done by [Lakkaraju et al., 2017] focusing on unambiguity, interpretability, and interactivity of explanations.

### Missing metrics

Available explainability metrics aim at measuring the quality of explanations in absolute terms—i.e., how good are my extracted explanations? Meanwhile, our definition of NeSy transparency requires to measure the *gain* in transparency obtained from symbolic and subsymbolic integration. Therefore, there is the need for novel metrics for NeSy systems comparing the quality of a system's explanations before and after symbolic and subsymbolic integration. Moreover, we here stress the unbalanced nature of explainability metrics, as most metrics focus solely on features of explanations that are automatically measurable – e.g., correctness, coverage, length, etc. –, whereas there are basically no metrics focusing on human oriented specifications. A relevant issue for future research in this are is the definition of metrics that account for the subjective human factor in explanations, assessing the level of explanations satisfaction and understandability via human-assisted experimentation. Finally, it should be noted that transparency should not just focus on measuring the quality of the explanations that can be obtained from a system, but should instead assess the complexity of the process for extracting those explanations, too. Indeed, explanations obtained from a DL model using SKE may be complete, understandable and useful, but require a high computational burden to be extracted, rendering the overall DL and SKE process less transparent.

## 11.3.5 Fairness

NeSy version of fairness requirement is defined as the need for *measuring* biased and discriminative behaviour of NeSy systems rooted in the *interaction* between their symbolic and subsymbolic components.

### Available metrics

Given the nuances characterising a context-dependent notion like fairness, developing quantitative formulations for fairness metrics is challenging [Chierichetti et al., 2019]. In the general context of AI systems, fairness is generally regarded as *outcome fairness*, which is the definition of equality of the decision making process outcomes. Here, fairness can be categorised into individual vs. group notions of fairness, and observational vs. causal approaches to assess fairness [Calegari et al., 2023]. Observational fairness approaches are characterised by a number of existing metrics, such as: *(i) independence metrics* – e.g., statistical parity, group fairness, demographic parity, etc. –; *(ii) separation metrics* – e.g., equal opportunity, equalised odds, predictive equality, etc. –; and *(iii) sufficiency metrics*—e.g., groups calibration, predictive parity, etc.

While representing a fundamental requirement, fairness in NeSy setups is yet to be explored in detail. Indeed, only a handful of works have investigated fairness in NeSy systems. [Wagner and d'Avila Garcez, 2021] propose to leverage the combination of symbolic knowledge extraction from Logic Tensor Networks [Badreddine et al., 2022] and injection of fairness constraints via continual learning to enforce fairness. Gao et al. [Gao et al., 2022] inject a fairness-based component in the loss function of subsymbolic models during their optimisation process to achieve higher fairness. Beyond their obvious relevance, these work focus solely on possible fairness benefits obtained through NeSy, as they rely on the application of SKI and SKE to reduce bias issues, leveraging the general AI fairness metrics. Therefore, available NeSy-specific fairness metrics are still missing that would aim at measuring just the impact of symbolic and subsymbolic integration upon fairness. This deficit is probably due to two aspects: *(i)* most observational fairness metrics are considered to be applicable to NeSy systems without modification; and *(ii)* most research focuses on measuring the fairness and assess it, rather than aiming at identifying its root causes.

### Missing metrics

In its NeSy version, the fairness requirement highlights the need to assess the possible fairness issues or improvements that arise from the use of symbolic and subsymbolic integration. It is clear that this requirement is not satisfied by available fairness metrics. Indeed, although most observational fairness metrics apply to NeSy systems, they do not allow for identification of the root causes of bias. One approach to tackle this issue would be to measure NeSy fairness as a differential of observational fairness between a SKI/SKE model and its ML/DL counterpart. However, such an approach would be over-simplistic, as it would not allow the specific sub-components of the integration process or of the symbolic knowledge that impact fairness to be captured. One possible solution would be to measure the fairness of NeSy systems over a set of symbolic knowledge bases, each representing a specific set of fairness goal. This process would allow fairness goal to be decomposed into its components/elements, then measure how well a NeSy system can enforce each fairness element.

### 11.3.6 Resource Efficiency

NeSy version of the resource efficiency requirement is defined as the need for assessing the *gain* in terms of *sustainability* with respect to pure subsymbolic counterparts.

#### Available metrics

When dealing with resource efficiency of AI systems in general, the detailed definition of the set of resources to take into account represents a fundamental aspect. Several elements of the system at hand can be identified as resources, ranging from the energy required by the system to be optimised to its scalability—e.g., overall complexity. In this context, the previous chapters of this thesis propose a rigorous definition of resource efficiency improvements achievable by SKI (see Chapters 8 and 10) and SKE (see Chapter 9) systems, focusing on energy, latency, memory, and data efficiency of these models. Similarly, several other works show the data efficiency of NeSy models – such as [Mao et al., 2019, Zhang et al., 2021b, Škrlj et al., 2021] – even though lacking a proper definition for efficiency.

#### Missing metrics

As data efficiency represents one of the declared advantages of NeSy systems, most of the literature focuses specifically on this aspect, leaving some space for investigation about other relevant aspects of resource efficiency. More in detail, detailed analysis of the environmental impact of AI and NeSy models development in terms of their carbon footprint are still mostly missings. Studying the energy consumption of the development of a single NeSy model is not enough, as the computation infrastructure used throughout this development – such as clusters and cloud infrastructures – strongly impact its environmental footprint. Moreover, whereas few metrics exist that assess the efficiency of NeSy under the SKI perspective, there are basically no metrics for resource efficiency in the SKE area. In this context, it would be desirable to have metrics similar to the ones obtained for SKI comparing the resource usage of the original subsymbolic model and its symbolic emulation. Depending on the SKE approach at hand, it is possible to consider extracting a small symbolic AI models mimicking the behaviour big DL frameworks. The small symbolic model obtained may help hugely reducing the amount of resources – especially energy, latency, and memory – required to deploy the AI system. Therefore, we here suggest as a future direction to investigate whether – and to what extent – SKE can produce small and fast counterparts of DL models. Here, the resource efficiency metric could be simply designed as the relative difference between the amount of resources required to run the original DL model and its symbolic emulation.

### 11.3.7 Accountability

NeSy version of the accountability requirement is defined as the need for assessing the *gain* in terms of *answerability* obtained by a NeSy system with respect to its pure subsymbolic components.

#### Available metrics

As it is represented by the answerability of an AI system, accountability is closely tight to transparency. Indeed, accountability requires the underlying system to be explainable, and the explanations to be correct, reliable, and comprehensible. Correctness and reliability of explanations depend on the precision of the AI system and its explanation construction counterpart. Therefore, most efforts in this field focus on the explainability of the AI/NeSy system at hand. As a result, the set of available AI and NeSy metrics for accountability is basically represented by the same set of metrics presented in Section 11.3.4.

#### Missing metrics

While being tightly linked with explainability, accountability also requires the extracted explanations to be correct and reliable. As correctness and reliability mostly depend on the precision of the AI/NeSy system, we here propose to define novel accountability metrics by opportunistically mixing transparency metrics (Section 11.3.4) and robustness metrics (Section 11.3.2). Therefore, accountability metrics should be defined as the result of explainability metrics applied over a set of input perturbations, measuring the rate of change of the obtained explanations.

## Chapter Synopsis

Resource efficiency represents one of the pillars of AI trustworthiness. Therefore, in this chapter we consider expanding the umbrella of our analysis and focus on the set of available and missing measures of trustworthiness for NeSy systems. Although it does not directly help in identifying more efficient NeSy models, the proposed analysis represents a fundamental effort to identify the missing components for the definition of fully trustworthy NeSy systems. As full trustworthiness requires maximum efficiency, we felt the need to include this chapter to shed some light on the foreseeable future developments of NeSy systems, and to help the reader understand the open gaps in the literature and the available opportunities.

The notion of Trustworthy AI as defined by the EU is mostly a general one, yet implicitly accounting for issues coming from popular ML and DL techniques—so it fits well subsymbolic AI systems. NeSy systems call for a more specific definition of trustworthiness, as they rely on the integration of subsymbolic and symbolic AI where the symbolic components may affect – either positively or negatively –

the trust level of the system. Accordingly, we show how the AI trustworthiness requirements defined by the EU translate to the NeSy realm, focusing on the relevant elements of the NeSy integration process impacting trust. First we analyse in detail each pillar of trust and its implication on NeSy models, then we focus on the available metrics for measuring such requirements. The state-of-the-art analysis highlights a lack of available metrics for most trustworthiness aspects when specifically considering NeSy systems. Therefore, we suggest potential future directions to explore in the analysis of NeSy trust along with related metrics definitions. We believe that the rigorous definition of novel trust metrics tailored to NeSy systems is going to represent an essential step towards measurably reliable and trustworthy AI systems based on neuro-symbolic integration.

# Part III

# Epilogue

There is no real ending. It's just
the place where you stop the story.

*Frank Herbert*

# Chapter 12

# Conclusions

In this chapter, conclusions are drawn – analysing the achievement of this thesis' goals – and some relevant future research directions are briefly discussed. As outlined in Chapter 1, this thesis tackles the complex problem of integrating powerful AI models – especially focusing on state-of-the-art NNs – into resource constrained devices. Coherently with goal 1, we kick off our investigation by analysing the set of limitations that affect the most popular embedded devices and eleciting the relationship between such limitations and a set of efficiency metrics defined for AI systems (see Chapter 2). Accordingly, we reframe the integration of AI into constrained/embedded devices problem as a NN efficientisation task, where efficientisation is defined as the issue of minimising the resource usage of NN models, either during their optimization process or their deployment phase. We tackle the integration/efficientisation task following a multi-faceted approach, in which we focus both on *(i)* the most popular techniques available in the state-of-the-art – aiming at overcoming some of their limitations –, as well as *(ii)* proposing to leverage a novel paradigm to efficientise NNs—namely NeSy systems. Accordingly, our contribution is split into two major parts.

**Embedding AI via Classic Efficientisation.** In Part I, we focus on the *classic* efficientisation approaches, targetting first goal 2. To this end, we provide a concise – yet insightful – overview of the available approaches for reducing the resource requirements of NN models, eliciting their advantages and shortcomings in Chapter 2. While there exists quite a few different approaches available in the literature, many of them consider focusing on one or few efficientisation issue at the time, thus representing limited solutions. For example, pruning, quantization, and architecturing approaches focus on minimising the amount of parameters of the model under analysis, while data coresets approaches focus solely on enabling a data-efficient learning setup, targetting the reduction of samples required to optimise a model. These findings highlight the lack of a silver-bullet solution for tackling the efficientisation task. Accordingly, we consider tackling goal 3 focusing on few different efficientisation paradigms (see Chapters 3 to 6). More in detail, we tackle the limitations of *architecturing* approaches which usually lack mechanisms to limit the architecture complexity – while promoting variability – and learn their inner working principles in Chapters 3 and 4. Subsequently, we focus on the shortcomings of the available *data coresets* approaches, proposing to leverage a set of simple and resource efficient properties of NN to identify *clean* samples to be used during training in Chapter 5. Finally, we consider energy and resource utilization optimisation in federated learning scenarios – which represents an overlooked issue –, presenting a novel energy and resource aware client selection approach and showcasing its resource friendliness in Chapter 6.

**Efficientisation via Neuro-Symbolic Integration.** In Part II, we explore the usage of NeSy systems to study their achievable efficiency improvements over classic NN models. NeSy approaches rely on the hybridisation of symbolic and sub-symbolic realms, where symbolic approaches usually focus on rational intelligence, while sub-symbolic approaches mostly focus on intuitive intelligence. In Chapter 7, we give a detailed overview of the set of NeSy approaches which can be leveraged as helping tools for tackling the efficientization of NNs task. To this end, we promote two complementary broad set of NeSy approaches, namely *symbolic knowledge extraction* and *injection* from and into sub-symbolic predictors.

Symbolic knowledge injection frameworks aim at steering the NN learning process towards a predefined goal, injecting complex logical information which is otherwise difficult to learn and elaborate from numerical data using bare NNs. Here, the simple intuition is that injection can be leveraged as a tool to remove part of the complexity burden from the underlying NN and obtain simpler and more efficient sub-symbolic models. In Chapter 8, we validate this intuition, showcasing how different SKI approaches can be leveraged to achieve more efficient systems w.r.t. purely sub-symbolic counterparts. Similarly, in Chapter 10, we define SKI robustness metrics which support the findings on the achievable efficiency improvements when leveraging SKI over uneducated counterparts. These findings represent the partial completion of goal 4. To further investigate the feasibility of leveraging NeSy systems for NN efficientisation, in Chapter 9, we analyse if – and to what extent – SKE can help efficientisation. Since SKE frameworks aim at producing symbolic knowledge that reflects the behaviour of the NN predictor with high fidelity, we propose to leverage them to achieve efficient replica of the starting NN model bounding the complexity of the extracted symbolic knowledge. Our empirical findings encouragingly highlight several efficiency gains when SKE approaches are applied to Natural Language Processing scenarios, once again showcasing the completion of goal 4.

Thorughout Part II, we consider tackling goal 5 via the proposal of several Quality-of-Service (QoS) metrics for NeSy mechanisms that enable the thorough analysis of their efficiency improvements (see Chapter 8) and other relevant properties such as robustness (see Chapter 10) and trustworthiness (see Chapter 11). The proposed QoS metrics represent a step forward in the fair measurement of the advantages and limitations of NeSy systems, filling an open gap in the research community.

## Future Directions

This thesis dives into the NN efficientisation perspective, proposing a multi-faceted approach in which multiple techniques are analysed, along with their limitations. As the NN efficientisation problem represents a fundamental open research question, several approaches have been proposed and tested in the literature. However, while these approaches show their effectiveness, they are seen as watertight compartments. In the future, a relevant research direction would be represented by the analysis of the techniques that can arise from the cross-contamination process of classic NN efficientisation approaches. For example, leveraging coreset construction approaches in combination with architecturing techniques can represent a relevant and successful mix, in which the exploration of efficient NN architectures is boosted by the data compression achievable via coresets. Similarly, it might represent a relevant solution to embed coresets into distributed learning approaches, where local information about data is partially shared via synthetic data – generated using coresets – to optimise the distributed learning process.

The extension of our analysis to a multitude of embedded devices which were not considered thorughout the thesis would represent another relevant future direction. The embedded devices considered in this thesis were selected mainly due to budget and/or time reasons. However, the IoT world includes many devices made of microcontrollers, such as the one based on STM32[1] or Expressif ESP32[2] that we feel would be a valuable addition to our resource-efficiency analysis. We consider this extension to be very valuable given the added heterogeneity of results it would bring. However, we also consider it as an engineering effort, rather than an innovative research-oriented problem. Therefore, we believe that the set of embedded devices considered in this thesis are sufficiently valuable to address the research questions and goals we identified in Chapter 1 and leave the deployment of our approaches on more embedded devices for the future.

Finally, concerning the efficientisation via neuro-symbolic integration perspective, we consider it to be a relevant future direction to be investigated by itself. Indeed, while this thesis showcases the achievable efficiency gains of symbolic knowledge injection and extraction mechanisms, it is important to stress that these results are not exhaustive. Comprehensive studies analysing the totality of neuro-symbolic approaches – thus not only focusing on SKI and SKE – are still missing, along with several metrics required to measure their trustworthiness. In this context, the cross contamination of the two views on efficientisation presented in this thesis may also represent a valuable future research direction. For example, one may consider leveraging neural architecture search approaches in the neuro-symbolic integration realm to explore effectively the available integration space. Similarly, integrating distributed learning and NeSy systems may represent an interesting – yet vastly unexplored – research direction, testing if the learning task distribution can be effective also for neuro-symbolic models. Therefore, we

[1] `https://www.st.com/en/microcontrollers-microprocessors/stm32-32-bit-arm-cortex-mcus.html`

[2] `https://www.espressif.com/en/products/socs/esp32`

consider the work presented in this thesis as a stepping stone towards the comprehensive adoption of NeSy for efficientisation and acknowledge the current lacks in this realm.

# Bibliography

[Abnar and Zuidema, 2020] Abnar, S. and Zuidema, W. (2020). Quantifying attention flow in transformers. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 4190–4197. Association for Computational Linguistics.

[Adadi and Berrada, 2018] Adadi, A. and Berrada, M. (2018). Peeking inside the black-box: A survey on explainable artificial intelligence (XAI). *IEEE Access*, 6:52138–52160.

[Agiollo et al., 2024a] Agiollo, A., Bardhi, E., Conti, M., Dal Fabbro, N., and Lazzeretti, R. (2024a). Anonymous federated learning via named-data networking. *Future Generation Computer Systems*, 152:288–303.

[Agiollo et al., 2024b] Agiollo, A., Bellavista, P., Mendula, M., and Omicini, A. (2024b). EneA-FL: Energy-aware orchestration for serverless federated learning. *Future Generation Computer Systems*, 154:219–234. Special Issue "Serverless Computing in the Cloud-to-Edge Continuum".

[Agiollo et al., 2021] Agiollo, A., Ciatto, G., and Omicini, A. (2021). *Shallow2Deep*: Restraining neural networks opacity through neural architecture search. In *Explainable and Transparent AI and Multi-Agent Systems. Third International Workshop, EXTRAAMAS 2021, Virtual Event, May 3–7, 2021, Revised Selected Papers*, volume 12688 of *Lecture Notes in Computer Science*, pages 63–82. Springer Nature, Basel, Switzerland.

[Agiollo and Omicini, 2021] Agiollo, A. and Omicini, A. (2021). Load classification: A case study for applying neural networks in hyper-constrained embedded devices. *Applied Sciences*, 11(24). Special Issue "Artificial Intelligence and Data Engineering in Engineering Applications".

[Agiollo and Omicini, 2022] Agiollo, A. and Omicini, A. (2022). GNN2GNN: Graph neural networks to generate neural networks. In *Uncertainty in Artificial Intelligence*, volume 180 of *Proceedings of Machine Learning Research*, pages 32–42, Maastricht, The Netherlands. ML Research Press. Proceedings of the Thirty-Eighth Conference on Uncertainty in Artificial Intelligence, UAI 2022, 1-5 August 2022, Eindhoven, The Netherlands.

[Agiollo and Omicini, 2023] Agiollo, A. and Omicini, A. (2023). Measuring trustworthiness in neuro-symbolic integration. In *Proceedings of the 18th Conference on Computer Science and Intelligence Systems*, volume 35 of *Annals of Computer Sciences and Information Systems*, pages 1–10.

[Agiollo et al., 2023a] Agiollo, A., Rafanelli, A., Magnini, M., Ciatto, G., and Omicini, A. (2023a). Symbolic knowledge injection meets intelligent agents: QoS metrics and experiments. *Autonomous Agents and Multi-Agent Systems*, 37(2):27:1–27:30.

[Agiollo et al., 2022] Agiollo, A., Rafanelli, A., and Omicini, A. (2022). Towards quality-of-service metrics for symbolic knowledge injection. In *WOA 2022 – 23rd Workshop "From Objects to Agents"*, volume 3261 of *CEUR Workshop Proceedings*, pages 30–47. Sun SITE Central Europe, RWTH Aachen University.

[Agiollo et al., 2023b] Agiollo, A., Siebert, L. C., Murukannaiah, P. K., and Omicini, A. (2023b). The quarrel of local post-hoc explainers for moral values classification in natural language processing. In *Explainable and Transparent AI and Multi-Agent Systems*, volume 14127 of *Lecture Notes in Computer Science*, chapter 6, pages 97–115. Springer.

[Ahmed et al., 2023] Ahmed, K., Chang, K., and Van den Broeck, G. (2023). Semantic strengthening of neuro-symbolic learning. *CoRR*, abs/2302.14207.

[Ajtai and Gurevich, 1994] Ajtai, M. and Gurevich, Y. (1994). Datalog vs first-order logic. *Journal of Computer and System Sciences*, 49(3):562–588.

[Alberto et al., 2015] Alberto, T. C., Lochter, J. V., and Almeida, T. A. (2015). Tubespam: Comment spam filtering on youtube. In *14th IEEE International Conference on Machine Learning and Applications, ICMLA 2015, Miami, FL, USA, December 9-11, 2015*, pages 138–143. IEEE.

[Ali et al., 2022] Ali, A., Schnake, T., Eberle, O., Montavon, G., Müller, K.-R., and Wolf, L. (2022). Xai for transformers: Better explanations through conservative propagation. In *International Conference on Machine Learning*, pages 435–451. PMLR.

[Allamanis et al., 2017] Allamanis, M., Chanthirasegaran, P., Kohli, P., and Sutton, C. (2017). Learning continuous semantic representations of symbolic expressions. In *Proceedings of the 34th International Conference on Machine Learning (ICML), Sydney, NSW, Australia, August 6-11 2017*, volume 70 of *Proceedings of Machine Learning Research*, pages 80–88. PMLR.

[Almeida et al., 2011] Almeida, T. A., Hidalgo, J. M. G., and Yamakami, A. (2011). Contributions to the study of SMS spam filtering: new collection and results. In *Proceedings of the 2011 ACM Symposium on Document Engineering, Mountain View, CA, USA, September 19-22, 2011*, pages 259–262. ACM.

[Aloqaily et al., 2022] Aloqaily, M., Ridhawi, I. A., and Guizani, M. (2022). Energy-Aware Blockchain and Federated Learning-Supported Vehicular Networks. *IEEE Transactions on Intelligent Transportation Systems*, 23(11):22641–22652.

[Alshomary et al., 2022] Alshomary, M., Baff, R. E., Gurcke, T., and Wachsmuth, H. (2022). The moral debater: A study on the computational generation of morally framed arguments. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics*

*(Volume 1: Long Papers)*, pages 8782–8797, Dublin, Ireland. Association for Computational Linguistics.

[Andrews et al., 1995] Andrews, R., Diederich, J., and Tickle, A. B. (1995). Survey and critique of techniques for extracting rules from trained artificial neural networks. *Knowledge-Based Systems*, 8(6):373–389.

[Andrews and Geva, 1995] Andrews, R. and Geva, S. (1995). Rulex & cebp networks as the basis for a rule refinement system. In *Hybrid Problems, Hybrid Solutions*, pages 1–12. IOS Press.

[Arouj and Abdelmoniem, 2022] Arouj, A. and Abdelmoniem, A. M. (2022). Towards Energy-Aware Federated Learning on Battery-Powered Clients. *CoRR*, abs/2208.04505.

[Augasta and Kathirvalavakumar, 2012] Augasta, M. G. and Kathirvalavakumar, T. (2012). Reverse engineering the neural networks for rule extraction in classification problems. *Neural Processing Letters*, 35(2):131–150.

[Baader, 2003] Baader, F. (2003). Basic description logics. In *The Description Logic Handbook: Theory, Implementation, and Applications*, pages 43–95, USA. Cambridge University Press.

[Bach et al., 2015] Bach, S., Binder, A., Montavon, G., Klauschen, F., Müller, K.-R., and Samek, W. (2015). On pixel-wise explanations for non-linear classifier decisions by layer-wise relevance propagation. *PloS ONE*, 10(7):e0130140.

[Bader, 2009] Bader, S. (2009). Extracting propositional rules from feedforward neural networks by means of binary decision diagrams. In *NeSy'09 – Neural-Symbolic Learning and Reasoning*, volume 481 of *CEUR Workshop Proceedings*, Pasadena, CA, USA. CEUR-WS.org.

[Bader et al., 2005] Bader, S., d'Avila Garcez, A. S., and Hitzler, P. (2005). Computing first-order logic programs by fibring artificial neural networks. In *Proceedings of the Eighteenth International Florida Artificial Intelligence Research Society Conference, Clearwater Beach, Florida, USA, AAAI Press*, pages 314–319.

[Bader et al., 2008] Bader, S., Hölldobler, S., and Marques, N. C. (2008). Guiding backprop by inserting rules. In *NeSy'08 – Neural-Symbolic Learning and Reasoning*, volume 366 of *CEUR Workshop Proceedings*, Patras, Greece. CEUR-WS.org.

[Bader et al., 2007] Bader, S., Hölldobler, S., and Mayer-Eichberger, V. (2007). Extracting propositional rules from feed-forward neural networks – A new decompositional approach. In *NeSy'07 – Neural-Symbolic Learning and Reasoning, 3rd International Workshop*, volume 230 of *CEUR Workshop Proceedings*. CEUR-WS.org.

[Badreddine et al., 2022] Badreddine, S., d'Avila Garcez, A., Serafini, L., and Spranger, M. (2022). Logic tensor networks. *Artificial Intelligence*, 303:103649.

[Bai et al., 2023] Bai, L., Yu, W., Chai, D., Zhao, W., and Chen, M. (2023). Temporal knowledge graphs reasoning with iterative guidance by temporal logical rules. *Information Sciences*, 621:22–35.

[Bai et al., 2019] Bai, Y., Wang, Y., and Liberty, E. (2019). Proxquant: Quantized neural networks via proximal operators. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net.

[Ballard, 1986] Ballard, D. H. (1986). Parallel logical inference and energy minimization. In *Proceedings of the 5th National Conference on Artificial Intelligence. Philadelphia, PA, USA, August 11-15, 1986. Volume 1: Science*, pages 203–209. Morgan Kaufmann.

[Banner et al., 2019] Banner, R., Nahshan, Y., and Soudry, D. (2019). Post training 4-bit quantization of convolutional networks for rapid-deployment. In *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, pages 7948–7956.

[Barakat and Bradley, 2007] Barakat, N. H. and Bradley, A. P. (2007). Rule extraction from support vector machines: A sequential covering approach. *IEEE Transactions on Knowledge and Data Engineering*, 19(6):729–741.

[Barakat and Diederich, 2005] Barakat, N. H. and Diederich, J. (2005). Eclectic rule-extraction from support vector machines. *International Journal of Computer and Information Engineering*, 2(5):1672–1675.

[Barbado et al., 2022] Barbado, A., Corcho, Ó., and Benjamins, R. (2022). Rule extraction in unsupervised anomaly detection for model explainability: Application to oneclass SVM. *Expert Systems with Applications*, 189:116100.

[Barbiero et al., 2022] Barbiero, P., Ciravegna, G., Giannini, F., Liò, P., Gori, M., and Melacci, S. (2022). Entropy-based logic explanations of neural networks. In *Thirty-Sixth AAAI Conference on Artificial Intelligence, AAAI 2022, Thirty-Fourth Conference on Innovative Applications of Artificial Intelligence, IAAI 2022, The Twelveth Symposium on Educational Advances in Artificial Intelligence, EAAI 2022 Virtual Event, February 22 - March 1, 2022*, pages 6046–6054. AAAI Press.

[Basilio et al., 2001] Basilio, R., Zaverucha, G., and Barbosa, V. C. (2001). Learning logic programs with neural networks. In *Inductive Logic Programming, 11th International Conference, ILP 2001, Strasbourg, France, September 9-11, 2001, Proceedings*, volume 2157 of *Lecture Notes in Computer Science*, pages 15–26. Springer.

[Bastos et al., 2021] Bastos, A., Nadgeri, A., Singh, K., Mulang', I. O., Shekarpour, S., Hoffart, J., and Kaul, M. (2021). RECON: relation extraction using knowledge graph context in a graph neural network. In *WWW '21: The Web Conference 2021, Virtual Event / Ljubljana, Slovenia, April 19-23, 2021*, pages 1673–1685. ACM / IW3C2.

[Bengio et al., 2013] Bengio, Y., Léonard, N., and Courville, A. C. (2013). Estimating or propagating gradients through stochastic neurons for conditional computation. *CoRR*, abs/1308.3432.

[Benítez et al., 1997] Benítez, J. M., Castro, J. L., and Requena, I. (1997). Are artificial neural networks black boxes? *IEEE Transactions on Neural Networks*, 8(5):1156–1164.

[Berenji, 1991] Berenji, H. R. (1991). Refinement of approximate reasoning-based controllers by reinforcement learning. In *Proceedings of the Eighth International Workshop (ML91), Northwestern University, Evanston, Illinois, USA*, pages 475–479. Morgan Kaufmann.

[Besold et al., 2017] Besold, T. R., d'Avila Garcez, A. S., Bader, S., Bowman, H., Domingos, P. M., Hitzler, P., Kühnberger, K., Lamb, L. C., Lowd, D., Lima, P. M. V., de Penning, L., Pinkas, G., Poon, H., and Zaverucha, G. (2017). Neural-symbolic learning and reasoning: A survey and interpretation. *CoRR*, abs/1711.03902.

[Bian et al., 2014] Bian, J., Gao, B., and Liu, T. (2014). Knowledge-powered deep learning for word embedding. In *Proceedings of the 25th Machine Learning and Knowledge Discovery in Databases - European Conference (ECML), Nancy, France, September 15-19, 2014*, volume 8724 of *Lecture Notes in Computer Science*, pages 132–148. Springer.

[Birjali et al., 2021] Birjali, M., Kasri, M., and Hssane, A. B. (2021). A comprehensive survey on sentiment analysis: Approaches, challenges and trends. *Knowledge-Based Systems*, 226:107134.

[Biswas et al., 2017] Biswas, S. K., Chakraborty, M., Purkayastha, B., Roy, P., and Thounaojam, D. M. (2017). Rule extraction from training data using neural network. *International Journal on Artificial Intelligence Tools*, 26(3):1750006:1–1750006:26.

[Blalock et al., 2020] Blalock, D. W., Gonzalez Ortiz, J. J., Frankle, J., and Guttag, J. V. (2020). What is the state of neural network pruning? In *Proceedings of Machine Learning and Systems 2020, MLSys 2020, Austin, TX, USA, March 2-4, 2020*. mlsys.org.

[Bologna, 2021] Bologna, G. (2021). A rule extraction technique applied to ensembles of neural networks, random forests, and gradient-boosted trees. *Algorithms*, 14(12):339.

[Bordes et al., 2012] Bordes, A., Glorot, X., Weston, J., and Bengio, Y. (2012). Joint learning of words and meaning representations for open-text semantic parsing. In *Proceedings of the Fifteenth International Conference on Artificial Intelligence and Statistics, AISTATS 2012, La Palma, Canary Islands, Spain, April 21-23, 2012*, volume 22 of *JMLR Proceedings*, pages 127–135. JMLR.org.

[Bordes et al., 2014] Bordes, A., Glorot, X., Weston, J., and Bengio, Y. (2014). A semantic matching energy function for learning with multi-relational data - application to word-sense disambiguation. *Machine Learning*, 94(2):233–259.

[Bordes et al., 2013] Bordes, A., Usunier, N., García-Durán, A., Weston, J., and Yakhnenko, O. (2013). Translating embeddings for modeling multi-relational data. In *Proceedings of 27th Annual Conference on Neural Information Processing Systems (NeurIPS), Lake Tahoe, Nevada, United States, December 5-8, 2013*, pages 2787–2795.

[Bordes et al., 2011] Bordes, A., Weston, J., Collobert, R., and Bengio, Y. (2011). Learning structured embeddings of knowledge bases. In *Proceedings of the Twenty-Fifth AAAI Conference on Artificial Intelligence, AAAI 2011, San Francisco, California, USA, August 7-11, 2011*. AAAI Press.

[Bosselut et al., 2021] Bosselut, A., Le Bras, R., and Choi, Y. (2021). Dynamic neuro-symbolic knowledge graph construction for zero-shot commonsense question answering. In *Thirty-Fifth AAAI Conference on Artificial Intelligence, AAAI 2021, Thirty-Third Conference on Innovative Applications of Artificial Intelligence, IAAI 2021, The Eleventh Symposium on Educational Advances in Artificial Intelligence, EAAI 2021, Virtual Event, February 2-9, 2021*, pages 4923–4931. AAAI Press.

[Boz, 2002] Boz, O. (2002). Converting a trained neural network to a decision tree DecText - decision tree extractor. In *Proceedings of the 2002 International Conference on Machine Learning and Applications - ICMLA 2002, June 24-27, 2002, Las Vegas, Nevada, USA*, pages 110–116. CSREA Press.

[Brachman and Levesque, 2004] Brachman, R. J. and Levesque, H. J. (2004). The tradeoff between expressiveness and tractability. In *Knowledge Representation and Reasoning*, The Morgan Kaufmann Series in Artificial Intelligence, pages 327–348. Morgan Kaufmann, San Francisco.

[Breiman et al., 1984] Breiman, L., Friedman, J., Stone, C. J., and Olshen, R. A. (1984). *Classification and Regression Trees*. CRC Press.

[Browne et al., 2004] Browne, A., Hudson, B. D., Whitley, D. C., Ford, M. G., and Picton, P. (2004). Biological data mining with neural networks: implementation and application of a flexible decision tree extraction algorithm to genomic problem domains. *Neurocomputing*, 57:275–293.

[Brunk and Pazzani, 1991] Brunk, C. and Pazzani, M. J. (1991). An investigation of noise-tolerant relational concept learning algorithms. In *Proceedings of the Eighth International Workshop (ML91), Northwestern University, Evanston, Illinois, USA*, pages 389–393. Morgan Kaufmann.

[Cai et al., 2020] Cai, H., Gan, C., Zhu, L., and Han, S. (2020). TinyTL: Reduce memory, not parameters for efficient on-device learning. In *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*.

[Cai et al., 2019] Cai, H., Zhu, L., and Han, S. (2019). Proxylessnas: Direct neural architecture search on target task and hardware. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net.

[Caldiera and Rombach, 1994] Caldiera, V. R. B. G. and Rombach, H. D. (1994). The goal question metric approach. In *Encyclopedia of software engineering*, pages 528–532. John Wiley & Sons, Inc.

[Calegari et al., 2023] Calegari, R., Castañé, G. G., Milano, M., and O'Sullivan, B. (2023). Assessing and enforcing fairness in the AI lifecycle. In *32nd International Joint Conference on Artificial Intelligence (IJCAI 2023)*, Macau, China. IJCAI.

[Calegari et al., 2020] Calegari, R., Ciatto, G., and Omicini, A. (2020). On the integration of symbolic and sub-symbolic techniques for XAI: A survey. *Intelligenza Artificiale*, 14(1):7–32.

[Calegari and Federico, 2022] Calegari, R. and Federico, S. (2022). The PSyKE technology for trustworthy artificial intelligence. In *International Conference of the Italian Association for Artificial Intelligence*, volume 13796 of *Lecture Notes in Computer Science*, pages 3–16. Springer.

[Calheiros et al., 2010] Calheiros, R. N., Ranjan, R., Beloglazov, A., Rose, C. A. F. D., and Buyya, R. (2010). CloudSim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms. *Software: Practice and Experience*, 41(1):23–50.

[Canonne et al., 2020] Canonne, C. L., Kamath, G., and Steinke, T. (2020). The discrete gaussian for differential privacy. In *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*.

[Casale et al., 2019] Casale, F. P., Gordon, J., and Fusi, N. (2019). Probabilistic neural architecture search. *CoRR*, abs/1902.05116.

[Castillo et al., 2001] Castillo, L. A., González Muñoz, A., and Pérez, R. (2001). Including a simplicity criterion in the selection of the best rule in a genetic fuzzy learning algorithm. *Fuzzy Sets and Systems*, 120(2):309–321.

[Chakraborty et al., 2018] Chakraborty, M., Biswas, S. K., and Purkayastha, B. (2018). Recursive rule extraction from NN using reverse engineering technique. *New Generation Computing*, 36(2):119–142.

[Chakraborty et al., 2020] Chakraborty, M., Biswas, S. K., and Purkayastha, B. (2020). Rule extraction from neural network trained using deep belief network and back propagation. *Knowledge and Information Systems*, 62(9):3753–3781.

[Chan and Chan, 2017] Chan, V. and Chan, C. W. (2017). Towards developing the piece-wise linear neural network algorithm for rule extraction. *International Journal of Cognitive Informatics and Natural Intelligence*, 11(2):57–73.

[Chan and Chan, 2020] Chan, V. K. H. and Chan, C. W. (2020). Towards explicit representation of an artificial neural network model: Comparison of two artificial neural network rule extraction approaches. *Petroleum*, 6(4):329–339. SI: Artificial Intelligence (AI), Knowledge-based Systems (KBS), and Machine Learning (ML).

[Chang et al., 2014] Chang, K.-W., Yih, W.-t., Yang, B., and Meek, C. (2014). Typed tensor decomposition of knowledge bases for relation extraction. In *2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1568–1579, Doha, Qatar. Association for Computational Linguistics.

[Chang et al., 2007] Chang, M., Ratinov, L., and Roth, D. (2007). Guiding semi-supervision with constraint-driven learning. In *Proceedings of the 45th Annual Meeting of the Association for Computational Linguistics (ACL), June 23-30, 2007, Prague, Czech Republic*. The Association for Computational Linguistics (ACL).

[Chang et al., 2017] Chang, M., Ullman, T., Torralba, A., and Tenenbaum, J. B. (2017). A compositional object-based approach to learning physical dynamics. In *Proceedings of the 5th International Conference on Learning Representations (ICLR), Toulon, France, April 24-26, 2017*. OpenReview.net.

[Chaves et al., 2005] Chaves, A. d. C. F., Vellasco, M. M. B. R., and Tanscheit, R. (2005). Fuzzy rule extraction from support vector machines. In *5th International Conference on Hybrid Intelligent Systems (HIS 2005), 6-9 November 2005, Rio de Janeiro, Brazil*, pages 335–340. IEEE Computer Society.

[Che et al., 2015] Che, Z., GuoEmnlp16, D. C., Li, W., Bahadori, M. T., and Liu, Y. (2015). Deep computational phenotyping. In *Proceedings of the 21th International Conference on Knowledge Discovery and Data Mining (KDD), Sydney, NSW, Australia, August 10-13, 2015*, pages 507–516. ACM.

[Chen, 2004] Chen, F. (2004). Learning accurate and understandable rules from SVM classifiers. Master's thesis, Simon Fraser University.

[Chen et al., 2020] Chen, S., Bateni, S., Grandhi, S., Li, X., Liu, C., and Yang, W. (2020). DENAS: automated rule generation by knowledge extraction from neural networks. In *ESEC/FSE '20: 28th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering, Virtual Event, USA, November 8-13, 2020*, pages 813–825. ACM.

[Chen et al., 2016] Chen, T., Xu, B., Zhang, C., and Guestrin, C. (2016). Training Deep Nets with Sublinear Memory Cost. *CoRR*, abs/1604.06174.

[Chen et al., 2018] Chen, Y., Meng, G., Zhang, Q., Zhang, X., Song, L., Xiang, S., and Pan, C. (2018). Joint neural architecture search and quantization. *CoRR*, abs/1811.09426.

[Chen et al., 2010] Chen, Y., Welling, M., and Smola, A. J. (2010). Super-samples from kernel herding. In *UAI 2010, Proceedings of the Twenty-Sixth Conference on Uncertainty in Artificial Intelligence, Catalina Island, CA, USA, July 8-11, 2010*, pages 109–116. AUAI Press.

[Chen et al., 2007] Chen, Z., Li, J., and Wei, L. (2007). A multiple kernel support vector machine scheme for feature selection and rule extraction from gene expression data of cancer tissue. *Artificial Intelligence in Medicine*, 41(2):161–175.

[Cheng et al., 2019] Cheng, H., Zhang, T., Yang, Y., Yan, F., Teague, H., Chen, Y., and Li, H. (2019). Msnet: Structural wired neural architecture search for internet of things. In *2019 IEEE/CVF International Conference on Computer Vision Workshops, ICCV Workshops 2019, Seoul, Korea (South), October 27-28, IEEE*, pages 2033–2036.

[Cheng et al., 2021] Cheng, K., Yang, Z., Zhang, M., and Sun, Y. (2021). Uniker: A unified framework for combining embedding and definite horn rule reasoning for knowledge graph inference. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing, EMNLP 2021, Virtual Event / Punta Cana, Dominican Republic, 7-11 November, 2021*, pages 9753–9771. Association for Computational Linguistics.

[Chierichetti et al., 2019] Chierichetti, F., Kumar, R., Lattanzi, S., and Vassilvitskii, S. (2019). Matroids, matchings, and fairness. In *22nd International Conference on Artificial Intelligence and Statistics, AISTATS 2019*, volume 89 of *Proceedings of Machine Learning Research*, pages 2212–2220. PMLR.

[Cho et al., 2020] Cho, Y. J., Wang, J., and Joshi, G. (2020). Client Selection in Federated Learning: Convergence Analysis and Power-of-Choice Selection Strategies. *CoRR*, abs/2010.01243.

[Choi et al., 2017] Choi, E., Bahadori, M. T., Song, L., Stewart, W. F., and Sun, J. (2017). GRAM: graph-based attention model for healthcare representation learning. In *Proceedings of the 23rd ACM International Conference on Knowledge Discovery and Data Mining (SIGKDD), Halifax, NS, Canada, August 13-17, 2017*, pages 787–795. ACM.

[Choi et al., 2018] Choi, Y., El-Khamy, M., and Lee, J. (2018). Learning low precision deep neural networks through regularization. *CoRR*, abs/1809.00095.

[Chu et al., 2020] Chu, X., Zhang, B., and Xu, R. (2020). Multi-objective reinforced evolution in mobile neural architecture search. In *Computer Vision - ECCV 2020 Workshops - Glasgow, UK, August 23-28, 2020, Proceedings, Part IV*, volume 12538 of *Lecture Notes in Computer Science*, pages 99–113. Springer.

[Ciatto et al., 2024] Ciatto, G., Sabbatini, F., Agiollo, A., Magnini, M., and Omicini, A. (2024). Symbolic knowledge extraction and injection with sub-symbolic predictors: A systematic literature review. *ACM Computing Surveys*, 56(6):161:1–161:35.

[Cimiano, 2006] Cimiano, P. (2006). *Ontology Learning and Population from Text*. Springer US.

[Ciravegna et al., 2023] Ciravegna, G., Barbiero, P., Giannini, F., Gori, M., Liò, P., Maggini, M., and Melacci, S. (2023). Logic explained networks. *Artificial Intelligence*, 314:103822.

[Clark and Niblett, 1989] Clark, P. and Niblett, T. (1989). The CN2 induction algorithm. *Machine Learning*, 3:261–283.

[Cohen, 1993] Cohen, W. W. (1993). Efficient pruning methods for separate-and-conquer rule learning systems. In *Proceedings of the 13th International Joint Conference on Artificial Intelligence. Chambéry, France, August 28 - September 3, 1993*, pages 988–994. Morgan Kaufmann.

[Cohen, 1995] Cohen, W. W. (1995). Fast effective rule induction. In *Machine Learning, Proceedings of the Twelfth International Conference on Machine Learning, Tahoe City, California, USA, July 9-12, 1995*, pages 115–123. Morgan Kaufmann.

[Corrales et al., 2018] Corrales, D. C., Corrales, J. C., and Ledezma, A. (2018). How to address the data quality issues in regression models: A guided process for data cleaning. *Symmetry*, 10(4):99.

[Craven and Shavlik, 1994] Craven, M. W. and Shavlik, J. W. (1994). Using sampling and queries to extract rules from trained neural networks. In *Machine Learning Proceedings 1994*, pages 37–45. Elsevier.

[Craven and Shavlik, 1996] Craven, M. W. and Shavlik, J. W. (1996). Extracting tree-structured representations of trained networks. In *Advances in Neural Information Processing Systems 8. Proceedings of the 1995 Conference*, pages 24–30. The MIT Press.

[Cui et al., 2022] Cui, Y., Cao, K., Cao, G., Qiu, M., and Wei, T. (2022). Client Scheduling and Resource Management for Efficient Training in Heterogeneous IoT-Edge Federated Learning. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 41(8):2407–2420.

[Cunnington et al., 2023] Cunnington, D., Law, M., Lobo, J., and Russo, A. (2023). FFNSL: feed-forward neural-symbolic learner. *Machine Learning*, 112(2):515–569.

[d'Amato et al., 2021] d'Amato, C., Quatraro, N. F., and Fanizzi, N. (2021). Injecting background knowledge into embedding models for predictive tasks on knowledge graphs. In *The Semantic Web - 18th International Conference, ESWC 2021, Virtual Event, June 6-10, 2021, Proceedings*, volume 12731 of *Lecture Notes in Computer Science*, pages 441–457. Springer.

[Daniele and Serafini, 2019] Daniele, A. and Serafini, L. (2019). Knowledge enhanced neural networks. In *PRICAI 2019: Trends in Artificial Intelligence - 16th Pacific Rim International Conference on Artificial Intelligence, Cuvu, Yanuca Island, Fiji, August 26-30, 2019, Proceedings, Part I*, volume 11670 of *Lecture Notes in Computer Science*, pages 542–554. Springer.

[Danilevsky et al., 2020] Danilevsky, M., Qian, K., Aharonov, R., Katsis, Y., Kawas, B., and Sen, P. (2020). A survey of the state of explainable AI for natural language processing. In *Proceedings of the 1st Conference of the Asia-Pacific Chapter of the Association for Computational Linguistics and the 10th International Joint Conference on Natural Language Processing*, pages 447–459, Suzhou, China. Association for Computational Linguistics.

[Dash et al., 2022] Dash, T., Srinivasan, A., and Baskar, A. (2022). Inclusion of domain-knowledge into gnns using mode-directed inverse entailment. *Machine Learning*, 111(2):575–623.

[Dash et al., 2021] Dash, T., Srinivasan, A., and Vig, L. (2021). Incorporating symbolic domain knowledge into graph neural networks. *Machine Learning*, 110(7):1609–1636.

[Dattachaudhuri et al., 2021] Dattachaudhuri, A., Biswas, S. K., Chakraborty, M., and Sarkar, S. (2021). A transparent rule-based expert system using neural network. *Soft Computing*, 25(12):7731–7744.

[d'Avila Garcez et al., 2001] d'Avila Garcez, A. S., Broda, K., and Gabbay, D. M. (2001). Symbolic knowledge extraction from trained neural networks: A sound approach. *Artificial Intelligence*, 125(1-2):155–207.

[d'Avila Garcez and Gabbay, 2004] d'Avila Garcez, A. S. and Gabbay, D. M. (2004). Fibring neural networks. In *Proceedings of the Nineteenth National Conference on Artificial Intelligence, Sixteenth Conference on Innovative Applications of Artificial Intelligence, July 25-29, San Jose, California, USA, AAAI Press / The MIT Press*, pages 342–347.

[d'Avila Garcez et al., 2019] d'Avila Garcez, A. S., Gori, M., Lamb, L. C., Serafini, L., Spranger, M., and Tran, S. N. (2019). Neural-symbolic computing: An effective methodology for principled integration of machine learning and reasoning. *Journal of Applied Logics—IfCoLog Journal of Logics and their Applications*, 6(4):611–632.

[d'Avila Garcez and Zaverucha, 1999] d'Avila Garcez, A. S. and Zaverucha, G. (1999). The connectionist inductive learning and logic programming system. *Applied Intelligence*, 11(1):59–77.

[de Campos Souza and Lughofer, 2022] de Campos Souza, P. V. and Lughofer, E. (2022). Efnn-nulluni: An evolving fuzzy neural network based on null-uninorm. *Fuzzy Sets and Systems*, 449:1–31.

[De Cao and Kipf, 2018] De Cao, N. and Kipf, T. (2018). Molgan: An implicit generative model for small molecular graphs. *CoRR*, abs/1805.11973.

[de Graaf and Malle, 2017] de Graaf, M. M. A. and Malle, B. F. (2017). How people explain action (and autonomous intelligent systems should too). In *2017 AAAI Fall Symposia, Arlington, Virginia, USA, November 9-11, 2017*, pages 19–26. AAAI Press.

[Demeester et al., 2016] Demeester, T., Rocktäschel, T., and Riedel, S. (2016). Lifted rule injection for relation embeddings. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP), Austin, Texas, USA, November 1-4, 2016*, pages 1389–1399.

[Deng and Ren, 2021] Deng, J. and Ren, F. (2021). A survey of textual emotion recognition and its challenges. *IEEE Transactions on Affective Computing*.

[Denil and Trappenberg, 2010] Denil, M. and Trappenberg, T. P. (2010). Overlap versus imbalance. In *Advances in Artificial Intelligence*, volume 6085 of *Lecture Notes in Computer Science*, pages 220–231. Springer.

[Deschamps and Sahbi, 2022] Deschamps, S. and Sahbi, H. (2022). Reinforcement-based display selection for frugal learning. In *26th International Conference on Pattern Recognition, ICPR 2022, Montreal, QC, Canada, August 21-25, 2022*, pages 1186–1193. IEEE.

[Devlin et al., 2019] Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. (2019). BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, MN, USA. Association for Computational Linguistics.

[Diao et al., 2022] Diao, H., Lu, Y., Deng, A., Zou, L., Li, X., and Pedrycz, W. (2022). Convolutional rule inference network based on belief rule-based system using an evidential reasoning approach. *Knowledge-Based Systems*, 237:107713.

[Diligenti et al., 2017a] Diligenti, M., Gori, M., and Saccà, C. (2017a). Semantic-based regularization for learning and inference. *Artificial Intelligence*, 244:143–165.

[Diligenti et al., 2017b] Diligenti, M., Roychowdhury, S., and Gori, M. (2017b). Integrating prior knowledge into deep learning. In *16th IEEE International Conference on Machine Learning and Applications, ICMLA 2017, Cancun, Mexico, December 18-21, 2017*, pages 920–923. IEEE.

[Ding et al., 2022] Ding, Y., Wu, Y., Huang, C., Tang, S., Wu, F., Yang, Y., Zhu, W., and Zhuang, Y. (2022). NAP: neural architecture search with pruning. *Neurocomputing*, 477:85–95.

[Dong et al., 2019] Dong, H., Mao, J., Lin, T., Wang, C., Li, L., and Zhou, D. (2019). Neural logic machines. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net.

[Dong et al., 2021a] Dong, L., Ye, X., and Yang, G. (2021a). Two-stage rule extraction method based on tree ensemble model for interpretable loan evaluation. *Information Sciences*, 573:46–64.

[Dong et al., 2014] Dong, X., Gabrilovich, E., Heitz, G., Horn, W., Lao, N., Murphy, K., Strohmann, T., Sun, S., and Zhang, W. (2014). Knowledge vault: a web-scale approach to probabilistic knowledge fusion. In *The 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD '14, New York, NY, USA - August 24 - 27, 2014*, pages 601–610. ACM.

[Dong et al., 2021b] Dong, X., Liu, L., Musial, K., and Gabrys, B. (2021b). NATS-Bench: Benchmarking NAS algorithms for architecture topology and size. *IEEE Transactions on Pattern Analysis and Machine Intelligence*.

[Ducoffe and Precioso, 2018] Ducoffe, M. and Precioso, F. (2018). Adversarial active learning for deep networks: a margin based approach. *CoRR*, abs/1802.09841.

[Elsken et al., 2019] Elsken, T., Metzen, J. H., and Hutter, F. (2019). Neural architecture search: A survey. *Journal of Machine Learning Research*, 20:55:1–55:21.

[Erdös et al., 1960] Erdös, P., Rényi, A., et al. (1960). On the evolution of random graphs. *Publications of the Mathematical Institute of the Hungarian Academy of Sciences*, 5(1):17–60.

[Espinosa Zarlenga et al., 2021] Espinosa Zarlenga, M., Shams, Z., and Jamnik, M. (2021). Efficient decompositional rule extraction for deep neural networks. *CoRR*, abs/2111.12628.

[Esteva et al., 2021] Esteva, A., Chou, K., Yeung, S., Naik, N., Madani, A., Mottaghi, A., Liu, Y., Topol, E., Dean, J., and Socher, R. (2021). Deep learning-enabled medical computer vision. *npj Digital Medicine*, 4(1):1–9.

[Etchells and Lisboa, 2006] Etchells, T. A. and Lisboa, P. J. G. (2006). Orthogonal search-based rule extraction (OSRE) for trained neural networks: a practical and efficient approach. *IEEE Transactions on Neural Networks*, 17(2):374–384.

[Evans and Grefenstette, 2018] Evans, R. and Grefenstette, E. (2018). Learning explanatory rules from noisy data. *Journal of Artificial Intelligence Research*, 61:1–64.

[Fan et al., 2014] Fan, M., Zhou, Q., Chang, E., and Zheng, T. F. (2014). Transition-based knowledge graph embedding with relational mapping properties. In *Proceedings of the 28th Pacific Asia Conference on Language, Information and Computation, PACLIC 28, Cape Panwa Hotel, Phuket, Thailand, December 12-14, 2014*, pages 328–337. The PACLIC 28 Organizing Committee and PACLIC Steering Committee / ACL / Department of Linguistics, Faculty of Arts, Chulalongkorn University.

[Fan et al., 2019] Fan, W., Ma, Y., Li, Q., He, Y., Zhao, Y. E., Tang, J., and Yin, D. (2019). Graph neural networks for social recommendation. In *The World Wide Web Conference, WWW 2019*, pages 417–426. ACM.

[Fang et al., 2017] Fang, Y., Kuan, K., Lin, J., Tan, C., and Chandrasekhar, V. (2017). Object detection meets knowledge graphs. In *Proceedings of the 26th International Joint Conference* *on Artificial Intelligence (IJCAI), Melbourne, Australia, August 19-25, 2017*, pages 1661–1667. IJCAI.

[Feldman, 2020] Feldman, D. (2020). Introduction to core-sets: an updated survey. *CoRR*, abs/2011.09384.

[Feldman and Zhang, 2020] Feldman, V. and Zhang, C. (2020). What neural networks memorize and why: Discovering the long tail via influence estimation. In *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual.*

[Feng et al., 2016] Feng, J., Huang, M., Wang, M., Zhou, M., Hao, Y., and Zhu, X. (2016). Knowledge graph embedding by flexible translation. In *Principles of Knowledge Representation and Reasoning: Proceedings of the Fifteenth International Conference, KR 2016, Cape Town, South Africa, April 25-29, 2016*, pages 557–560. AAAI Press.

[Ferreira et al., 2022] Ferreira, J., de Sousa Ribeiro, M., Gonçalves, R., and Leite, J. (2022). Looking inside the black-box: Logic-based explanations for neural networks. In *Proceedings of the 19th International Conference on Principles of Knowledge Representation and Reasoning, KR 2022, Haifa, Israel. July 31 - August 5, 2022.*

[Finkelstein et al., 2019] Finkelstein, A., Almog, U., and Grobman, M. (2019). Fighting quantization bias with bias. *CoRR*, abs/1906.03193.

[Fischer et al., 2019] Fischer, M., Balunovic, M., Drachsler-Cohen, D., Gehr, T., Zhang, C., and Vechev, M. T. (2019). DL2: training and querying neural networks with logic. In *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*, volume 97 of *Proceedings of Machine Learning Research*, pages 1931–1941. PMLR.

[Foret et al., 2021] Foret, P., Kleiner, A., Mobahi, H., and Neyshabur, B. (2021). Sharpness-aware minimization for efficiently improving generalization. In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021.* OpenReview.net.

[França et al., 2014] França, M. V. M., Zaverucha, G., and Garcez, A. S. d. (2014). Fast relational learning using bottom clause propositionalization with artificial neural networks. *Machine Learning*, 94(1):81–104.

[Frankle and Carbin, 2019] Frankle, J. and Carbin, M. (2019). The lottery ticket hypothesis: Finding sparse, trainable neural networks. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019.* OpenReview.net.

[Freitas, 2014] Freitas, A. A. (2014). Comprehensible classification models: a position paper. *ACM SIGKDD Explorations Newsletter*, 15(1):1–10.

[Fu, 1994] Fu, L. (1994). Rule generation from neural networks. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, 24(8):1114–1124.

[Fu et al., 2004] Fu, X., Ong, C., Keerthi, S., Hung, G. G., and Goh, L. (2004). Extracting the knowledge embedded in support vector machines. In *2004 IEEE International Joint Conference on Neural Networks (IEEE Cat. No.04CH37541)*, volume 1, pages 291–296.

[Fung et al., 2005] Fung, G., Sandilya, S., and Rao, R. B. (2005). Rule extraction from linear support vector machines. In *Proceedings of the Eleventh ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Chicago, Illinois, USA, August 21-24, 2005*, pages 32–40. ACM.

[Fung et al., 2021] Fung, V., Zhang, J., Juarez, E., and Sumpter, B. G. (2021). Benchmarking graph neural networks for materials chemistry. *npj Computational Materials*, 7(1):84.

[Fürnkranz and Widmer, 1994] Fürnkranz, J. and Widmer, G. (1994). Incremental reduced error pruning. In *Machine Learning, Proceedings of the Eleventh International Conference, Rutgers University, New Brunswick, NJ, USA, July 10-13, 1994*, pages 70–77. Morgan Kaufmann.

[Gale et al., 2019] Gale, T., Elsen, E., and Hooker, S. (2019). The state of sparsity in deep neural networks. *CoRR*, abs/1902.09574.

[Gamma et al., 1993] Gamma, E., Helm, R., Johnson, R. E., and Vlissides, J. M. (1993). Design patterns: Abstraction and reuse of object-oriented design. In *ECOOP'93 - Object-Oriented Programming, 7th European Conference, Kaiserslautern, Germany, July 26-30, 1993, Proceedings*, volume 707 of *Lecture Notes in Computer Science, Springer*, pages 406–431.

[Gao et al., 2022] Gao, X., Zhai, J., Ma, S., Shen, C., Chen, Y., and Wang, Q. (2022). FairNeuron: improving deep neural network fairness with adversary games on selective neurons. In *44th International Conference on Software Engineering, ICSE 2022*, pages 921–933. ACM.

[García-Durán et al., 2014] García-Durán, A., Bordes, A., and Usunier, N. (2014). Effective blending of two and three-way interactions for modeling multi-relational data. In *Proceeding of the 25th Machine Learning and Knowledge Discovery in Databases - European Conference (ECML), Nancy, France, September 15-19, 2014*, volume 8724 of *Lecture Notes in Computer Science*, pages 434–449. Springer.

[Garg and Roy, 2023] Garg, I. and Roy, K. (2023). Samples with low loss curvature improve data efficiency. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR 2023, Vancouver, BC, Canada, June 17-24, 2023*, pages 20290–20300. IEEE.

[Ghobaei-Arani et al., 2019] Ghobaei-Arani, M., Souri, A., and Rahmanian, A. A. (2019). Resource Management Approaches in Fog Computing: a Comprehensive Review. *Journal of Grid Computing*, 18(1):1–42.

[Giannini et al., 2023] Giannini, F., Diligenti, M., Maggini, M., Gori, M., and Marra, G. (2023). T-norms driven loss functions for machine learning. *Applied Intelligence*, pages 1–15.

[Giles and Omlin, 1993] Giles, C. L. and Omlin, C. W. (1993). Rule refinement with recurrent neural networks. In *Proceedings of International Conference on Neural Networks (ICNN'88), San Francisco, CA, USA, March 28 - April 1, 1993*, pages 801–806. IEEE.

[Giunchiglia and Lukasiewicz, 2021] Giunchiglia, E. and Lukasiewicz, T. (2021). Multi-label classification neural networks with hard logical constraints. *Journal of Artificial Intelligence Research*, 72:759–818.

[Glavas and Vulic, 2018] Glavas, G. and Vulic, I. (2018). Explicit retrofitting of distributional word vectors. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (ACL), Melbourne, Australia, July 15-20, 2018*, pages 34–45. Association for Computational Linguistics.

[Golovko et al., 2017] Golovko, V., Egor, M., Brich, A., and Sachenko, A. (2017). A Shallow Convolutional Neural Network for Accurate Handwritten Digits Classification. In *Pattern Recognition and Information Processing*, Communications in Computer and Information Science, pages 77–85, Cham. Springer.

[Goodfellow et al., 2016] Goodfellow, I., Bengio, Y., and Courville, A. (2016). *Deep Learning*. MIT Press. `http://www.deeplearningbook.org`.

[Goodfellow et al., 2014] Goodfellow, I. J., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A. C., and Bengio, Y. (2014). Generative adversarial nets. In *Advances in Neural Information Processing Systems 27: Annual Conference on Neural Information Processing Systems 2014*, pages 2672–2680.

[Goodwin and Demner-Fushman, 2019] Goodwin, T. R. and Demner-Fushman, D. (2019). Bridging the knowledge gap: Enhancing question answering with world and domain knowledge. *CoRR*, abs/1910.07429.

[Gori, 2018] Gori, M. (2018). *Machine Learning: A Constraint Based Approach*. Morgan Kaufmann.

[Gou et al., 2021] Gou, J., Yu, B., Maybank, S. J., and Tao, D. (2021). Knowledge distillation: A survey. *Int. J. Comput. Vis.*, 129(6):1789–1819.

[Grafberger et al., 2021] Grafberger, A., Chadha, M., Jindal, A., Gu, J., and Gerndt, M. (2021). FedLess: Secure and Scalable Federated Learning Using Serverless Computing. In *2021 IEEE International Conference on Big Data (Big Data)*, pages 164–173.

[Green and Raphael, 1968] Green, C. C. and Raphael, B. (1968). The use of theorem-proving techniques in question-answering systems. In *Proceedings of the 23rd ACM national conference (ACM 1968)*, pages 169–181, New York, NY, USA. ACM.

[Grigorescu et al., 2020] Grigorescu, S. M., Trasnea, B., Cocias, T. T., and Macesanu, G. (2020). A survey of deep learning techniques for autonomous driving. *Journal of Field Robotics*, 37(3):362–386.

[Guidotti et al., 2018] Guidotti, R., Monreale, A., Ruggieri, S., Turini, F., Giannotti, F., and Pedreschi, D. (2018). A survey of methods for explaining black box models. *ACM Computing Surveys*, 51(5):1–42.

[Guo et al., 2022] Guo, C., Zhao, B., and Bai, Y. (2022). Deepcore: A comprehensive library for coreset selection in deep learning. In *Database and Expert Systems Applications - 33rd International Conference, DEXA 2022, Vienna, Austria, August 22-24, 2022, Proceedings, Part I*, volume 13426 of *Lecture Notes in Computer Science*, pages 181–195. Springer.

[Guo et al., 2020] Guo, S., Li, L., Hui, Z., Meng, L., Ma, B., Liu, W., Wang, L., Zhai, H., and Zhang, H. (2020). Knowledge graph embedding preserving soft logical regularity. In *CIKM '20: The 29th ACM International Conference on Information and Knowledge Management, Virtual Event, Ireland, October 19-23, 2020*, pages 425–434. ACM.

[Guo et al., 2016] Guo, S., Wang, Q., Wang, L., Wang, B., and Guo, L. (2016). Jointly embedding knowledge graphs and logical rules. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP), Austin, Texas, USA, November 1-4, 2016*, pages 192–202.

[Gupta et al., 2017] Gupta, H., Vahid Dastjerdi, A., Ghosh, S. K., and Buyya, R. (2017). iFogSim: A toolkit for modeling and simulation of resource management techniques in the Internet of Things, Edge and Fog computing environments. *Software: Practice and Experience*, 47(9):1275–1296.

[Hailesilassie, 2016] Hailesilassie, T. (2016). Rule extraction algorithm for deep neural networks: A review. *International Journal of Computer Science and Information Security*, 14(7):376–381.

[Halgamuge and Glesner, 1994] Halgamuge, S. K. and Glesner, M. (1994). Neural networks in designing fuzzy systems for real world applications. *Fuzzy Sets and Systems*, 65(1):1–12.

[Hamdi et al., 2022a] Hamdi, A. M. A., Hussain, F. K., and Hussain, O. K. (2022a). Task offloading in vehicular fog computing: State-of-the-art and open issues. *Future Generation Computer Systems*, 133:201–212.

[Hamdi et al., 2022b] Hamdi, R., Chen, M., Said, A. B., Qaraqe, M. K., and Poor, H. V. (2022b). Federated learning over energy harvesting wireless networks. *IEEE Internet Things J.*, 9(1):92–103.

[Hamilton et al., 2017] Hamilton, W. L., Ying, Z., and Leskovec, J. (2017). Inductive representation learning on large graphs. In *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017*, pages 1024–1034.

[Han et al., 2015] Han, S., Pool, J., Tran, J., and Dally, W. J. (2015). Learning both weights and connections for efficient neural network. In *Advances in Neural Information Processing Systems 28: Annual Conference on Neural Information Processing Systems 2015, December 7-12, 2015, Montreal, Quebec, Canada*, pages 1135–1143.

[Har-Peled and Mazumdar, 2004] Har-Peled, S. and Mazumdar, S. (2004). On coresets for k-means and k-median clustering. In *Proceedings of the 36th Annual ACM Symposium on Theory of Computing, Chicago, IL, USA, June 13-16, 2004*, pages 291–300. ACM.

[Hayashi, 1990] Hayashi, Y. (1990). A neural expert system with automated extraction of fuzzy if-then rules. In *Advances in Neural Information Processing Systems 3, [NIPS Conference, Denver, Colorado, USA, November 26-29, 1990]*, pages 578–584. Morgan Kaufmann.

[Hayashi and Takano, 2020] Hayashi, Y. and Takano, N. (2020). One-dimensional convolutional neural networks with feature selection for highly concise rule extraction from credit scoring datasets with heterogeneous attributes. *Electronics*, 9(8).

[He et al., 2021] He, C., Ceyani, E., Balasubramanian, K., Annavaram, M., and Avestimehr, S. (2021). SpreadGNN: Serverless Multi-task Federated Learning for Graph Neural Networks. *CoRR*, abs/2106.02743.

[He et al., 2006] He, J., Hu, H.-J., Harrison, R., Tai, P., and Pan, Y. (2006). Rule generation for protein secondary structure prediction with support vector machines and decision tree. *IEEE Transactions on NanoBioscience*, 5(1):46–53.

[He et al., 2016] He, K., Zhang, X., Ren, S., and Sun, J. (2016). Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016*, pages 770–778. IEEE Computer Society.

[He et al., 2015] He, S., Liu, K., Ji, G., and Zhao, J. (2015). Learning to represent knowledge graphs with gaussian embedding. In *Proceedings of the 24th ACM International Conference on Information and Knowledge Management, CIKM 2015, Melbourne, VIC, Australia, October 19 - 23, 2015*, pages 623–632. ACM.

[Hecht-Nielsen, 1988] Hecht-Nielsen, R. (1988). Theory of the backpropagation neural network. *Neural Networks*, 1(Supplement-1):445–448.

[Hinton et al., 2015] Hinton, G. E., Vinyals, O., and Dean, J. (2015). Distilling the knowledge in a neural network. *CoRR*, abs/1503.02531.

[Hoffman et al., 2018] Hoffman, R. R., Mueller, S. T., Klein, G., and Litman, J. (2018). Metrics for explainable AI: challenges and prospects. *CoRR*, abs/1812.04608.

[Holzinger et al., 2020] Holzinger, A., Carrington, A. M., and Müller, H. (2020). Measuring the quality of explanations: The system causability scale (SCS). *KI - Künstliche Intelligenz*, 34(2):193–198.

[Holzinger et al., 2019] Holzinger, A., Langs, G., Denk, H., Zatloukal, K., and Müller, H. (2019). Causability and explainability of artificial intelligence in medicine. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, 9(4):e1312:1–13.

[Hong and Chen, 1999] Hong, T. and Chen, J. (1999). Finding relevant attributes and membership functions. *Fuzzy Sets and Systems*, 103(3):389–404.

[Hong and Chen, 2000] Hong, T. and Chen, J. (2000). Processing individual fuzzy attributes for fuzzy rule induction. *Fuzzy Sets and Systems*, 112(1):127–140.

[Hong and Lee, 1996] Hong, T. and Lee, C. (1996). Induction of fuzzy rules and membership functions from training examples. *Fuzzy Sets and Systems*, 84(1):33–47.

[Hoover et al., 2020] Hoover, J., Portillo-Wightman, G., Yeh, L., Havaldar, S., Davani, A. M., Lin, Y., Kennedy, B., Atari, M., Kamel, Z., Mendlen, M., et al. (2020). Moral foundations Twitter corpus: A collection of 35k tweets annotated for moral sentiment. *Social Psychological and Personality Science*, 11(8):1057–1071.

[Horikawa et al., 1992] Horikawa, S., Furuhashi, T., and Uchikawa, Y. (1992). On fuzzy modeling using fuzzy neural networks with the back-propagation algorithm. *IEEE Transactions on Neural Networks*, 3(5):801–806.

[Horn, 1951] Horn, A. (1951). On sentences which are true of direct unions of algebras. *Journal of Symbolic Logic*, 16(1):14–21.

[Horta et al., 2021] Horta, V. A. C., Tiddi, I., Little, S., and Mileo, A. (2021). Extracting knowledge from deep neural networks through graph analysis. *Future Generation Computer Systems*, 120:109–118.

[Hossain and Lee, 2019] Hossain, S. and Lee, D. (2019). Deep learning-based real-time multiple-object detection and tracking from aerial imagery via a flying robot with gpu-based embedded devices. *Sensors*, 19(15):3371.

[Howard et al., 2019] Howard, A., Pang, R., Adam, H., Le, Q. V., Sandler, M., Chen, B., Wang, W., Chen, L., Tan, M., Chu, G., Vasudevan, V., and Zhu, Y. (2019). Searching for mobilenetv3. In *2019 IEEE/CVF International Conference on Computer Vision, ICCV 2019, Seoul, Korea (South), October 27 - November 2, 2019*, pages 1314–1324. IEEE.

[Howard et al., 2017] Howard, A. G., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., Weyand, T., Andreetto, M., and Adam, H. (2017). MobileNets: Efficient convolutional neural networks for mobile vision applications. *CoRR*, abs/1704.04861:1–9. `http://arxiv.org/abs/1704.04861`.

[Hu et al., 2016a] Hu, Z., Ma, X., Liu, Z., Hovy, E. H., and Xing, E. P. (2016a). Harnessing deep neural networks with logic rules. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics, ACL 2016, August 7-12, 2016, Berlin, Germany, Volume 1: Long Papers*.

[Hu et al., 2016b] Hu, Z., Yang, Z., Salakhutdinov, R., and Xing, E. P. (2016b). Deep neural networks with massive learned knowledge. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP), Austin, Texas, USA, November 1-4, 2016*, pages 1670–1679. The Association for Computational Linguistics.

[Huang and Mutlu, 2012] Huang, C. and Mutlu, B. (2012). Robot behavior toolkit: generating effective social behaviors for robots. In *International Conference on Human-Robot Interaction, HRI'12, Boston, MA, USA - March 05 - 08, 2012*, pages 25–32. ACM.

[Huang et al., 2018] Huang, G., Liu, S., van der Maaten, L., and Weinberger, K. Q. (2018). Condensenet: An efficient densenet using learned group convolutions. In *2018 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2018, Salt Lake City, UT, USA, June 18-22, Computer Vision Foundation / IEEE Computer Society*, pages 2752–2761.

[Huang et al., 2017] Huang, G., Liu, Z., van der Maaten, L., and Weinberger, K. Q. (2017). Densely connected convolutional networks. In *30th IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Honolulu, HI, USA, July 21-26, 2017*, pages 2261–2269. IEEE Computer Society.

[Huang and Wang, 2017] Huang, Z. and Wang, N. (2017). Like what you like: Knowledge distill via neuron selectivity transfer. *CoRR*, abs/1707.01219.

[Hussein and Mousa, 2020] Hussein, M. and Mousa, M. (2020). Efficient Task Offloading for IoT-Based Applications in Fog Computing Using Ant Colony Optimization. *IEEE Access*, 8:37191–37201.

[Huysmans et al., 2006a] Huysmans, J., Baesens, B., and Vanthienen, J. (2006a). ITER: An algorithm for predictive regression rule extraction. In *Data Warehousing and Knowledge Discovery (DaWaK 2006)*, pages 270–279. Springer.

[Huysmans et al., 2006b] Huysmans, J., Baesens, B., and Vanthienen, J. (2006b). Using rule extraction to improve the comprehensibility of predictive models. Tech. Report 0612, K.U. Leuven.

[Huysmans et al., 2011] Huysmans, J., Dejaeger, K., Mues, C., Vanthienen, J., and Baesens, B. (2011). An empirical evaluation of the comprehensibility of decision table, tree and rule based predictive models. *Decision Support Systems*, 51(1):141–154.

[Ibarra-Vázquez et al., 2022] Ibarra-Vázquez, G., Olague, G., Chan-Ley, M., Puente, C., and Soubervielle-Montalvo, C. (2022). Brain programming is immune to adversarial attacks: Towards accurate and robust image classification using symbolic learning. *Swarm and Evolutionary Computation*, 71:101059.

[Ibrahim et al., 2019] Ibrahim, M., Louie, M., Modarres, C., and Paisley, J. (2019). Global explanations of neural networks: Mapping the landscape of predictions. In *Proceedings of the 2019 AAAI/ACM Conference on AI, Ethics, and Society*, pages 279–287.

[Imteaj et al., 2022] Imteaj, A., Thakker, U., Wang, S., Li, J., and Amini, M. H. (2022). A Survey on Federated Learning for Resource-Constrained IoT Devices. *IEEE Internet Things Journal*, 9(1):1–24.

[Irfan et al., 2022] Irfan, M., Zheng, J., Iqbal, M., Masood, Z., and Arif, M. H. (2022). Knowledge extraction and retention based continual learning by using convolutional autoencoder-based learning classifier system. *Information Sciences*, 591:287–305.

[Ishibuchi et al., 1997] Ishibuchi, H., Nii, M., and Murata, T. (1997). Linguistic rule extraction from neural networks and genetic-algorithm-based rule selection. In *Proceedings of International Conference on Neural Networks (ICNN'97), Houston, TX, USA, June 9-12, 1997*, pages 2390–2395. IEEE.

[Iyer and Bilmes, 2013] Iyer, R. K. and Bilmes, J. A. (2013). Submodular optimization with submodular cover and submodular knapsack constraints. In *Advances in Neural Information Processing Systems 26: 27th Annual Conference on Neural Information Processing Systems 2013. Proceedings of a meeting held December 5-8, 2013, Lake Tahoe, Nevada, United States*, pages 2436–2444.

[Iyer et al., 2021] Iyer, R. K., Khargoankar, N., Bilmes, J. A., and Asanani, H. (2021). Submodular combinatorial information measures with applications in machine learning. In *Algorithmic Learning Theory, 16-19 March 2021, Virtual Conference, Worldwide*, volume 132 of *Proceedings of Machine Learning Research*, pages 722–754. PMLR.

[Jain et al., 2020] Jain, P., Jain, A., Nrusimha, A., Gholami, A., Abbeel, P., Keutzer, K., Stoica, I., and Gonzalez, J. (2020). Checkmate: Breaking the Memory Wall with Optimal Tensor Rematerialization. In *Proceedings of Machine Learning and Systems 2020, MLSys 2020, Austin, TX, USA, March 2-4, 2020*. mlsys.org.

[Ji et al., 2015] Ji, G., He, S., Xu, L., Liu, K., and Zhao, J. (2015). Knowledge graph embedding via dynamic mapping matrix. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing of the Asian Federation of Natural Language Processing, ACL 2015, July 26-31, 2015, Beijing, China, Volume 1: Long Papers*, pages 687–696. The Association for Computer Linguistics.

[Ji et al., 2016] Ji, G., Liu, K., He, S., and Zhao, J. (2016). Knowledge graph completion with adaptive sparse transfer matrix. In *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence, February 12-17, 2016, Phoenix, Arizona, USA*, pages 985–991. AAAI Press.

[Jia et al., 2020] Jia, S., Lin, P., Li, Z., Zhang, J., and Liu, S. (2020). Visualizing surrogate decision trees of convolutional neural networks. *Journal of Visualization*, 23(1):141–156.

[Jiang et al., 2020] Jiang, J., Wang, H., Xie, J., Guo, X., Guan, Y., and Yu, Q. (2020). Medical knowledge embedding based on recursive neural network for multi-disease diagnosis. *Artificial Intelligence in Medicine*, 103:101772.

[Jiang et al., 2021] Jiang, S., Lin, Z., Li, Y., Shu, Y., and Liu, Y. (2021). Flexible high-resolution object detection on edge devices with tunable latency. In *Proceedings of the 27th Annual International Conference on Mobile Computing and Networking (MobiCom), New Orleans, LA, USA, 25-29 October 2021*, pages 559–572, New York, NY, USA. Association for Computing Machinery (ACM).

[Johansson et al., 2022] Johansson, U., Sönströd, C., Löfström, T., and Boström, H. (2022). Rule extraction with guarantees from regression models. *Pattern Recognition*, 126:108554.

[Joyce, 2011] Joyce, J. M. (2011). Kullback-leibler divergence. In *International Encyclopedia of Statistical Science*, pages 720–722. Springer Berlin Heidelberg, Berlin, Heidelberg.

[Kairouz et al., 2021] Kairouz, P., Liu, Z., and Steinke, T. (2021). The distributed discrete gaussian mechanism for federated learning with secure aggregation. In *Proceedings of the 38th International Conference on Machine Learning, ICML 2021, 18-24 July 2021, Virtual Event*, volume 139 of *Proceedings of Machine Learning Research*, pages 5201–5212. PMLR.

[Kang et al., 2018] Kang, D., Kim, E., Bae, I., Egger, B., and Ha, S. (2018). C-good: C-code generation framework for optimized on-device deep learning. In *2018 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 1–8.

[Kautz, 2022] Kautz, H. A. (2022). The third AI summer: AAAI robert s. engelmore memorial lecture. *AI Mag.*, 43(1):93–104.

[Kaya et al., 2019] Kaya, Y., Hong, S., and Dumitras, T. (2019). Shallow-deep networks: Understanding and mitigating network overthinking. In *36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, CA, USA*, volume 97 of *Proceedings of Machine Learning Research*, pages 3301–3310.

[Kiesel et al., 2022] Kiesel, J., Alshomary, M., Handke, N., Cai, X., Wachsmuth, H., and Stein, B. (2022). Identifying the human values behind arguments. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 4459–4471.

[Killamsetty et al., 2021a] Killamsetty, K., Sivasubramanian, D., Ramakrishnan, G., De, A., and Iyer, R. K. (2021a). GRAD-MATCH: gradient matching based data subset selection for efficient deep model training. In *Proceedings of the 38th International Conference on Machine Learning, ICML 2021, 18-24 July 2021, Virtual Event*, volume 139 of *Proceedings of Machine Learning Research*, pages 5464–5474. PMLR.

[Killamsetty et al., 2021b] Killamsetty, K., Sivasubramanian, D., Ramakrishnan, G., and Iyer, R. K. (2021b). GLISTER: generalization based data subset selection for efficient and robust learning. In *Thirty-Fifth AAAI Conference on Artificial Intelligence, AAAI 2021, Thirty-Third Conference on Innovative Applications of Artificial Intelligence, IAAI 2021, The Eleventh Symposium on Educational Advances in Artificial Intelligence, EAAI 2021, Virtual Event, February 2-9, 2021*, pages 8110–8118. AAAI Press.

[Killamsetty et al., 2021c] Killamsetty, K., Zhao, X., Chen, F., and Iyer, R. K. (2021c). RETRIEVE: coreset selection for efficient and robust semi-supervised learning. In *Advances in Neural Information Processing Systems 34: Annual Conference on Neural Information Processing Systems 2021, NeurIPS 2021, December 6-14, 2021, virtual*, pages 14488–14501.

[Kim and Lee, 2000] Kim, D. and Lee, J. (2000). Handling continuous-valued attributes in decision tree with neural network modeling. In *Machine Learning: ECML 2000*, pages 211–219, Berlin, Heidelberg. Springer Berlin Heidelberg.

[Kim and Wu, 2021] Kim, Y. G. and Wu, C. (2021). AutoFL: Enabling Heterogeneity-Aware Energy Efficient Federated Learning. In *MICRO '21: 54th Annual IEEE/ACM International Symposium on Microarchitecture, Virtual Event, Greece, October 18-22, 2021*, pages 183–198. ACM.

[Kim et al., 2024] Kim, Y. I., Agrawal, P., Royset, J. O., and Khanna, R. (2024). On memorization and privacy risks of sharpness aware minimization.

[Kimmig et al., 2012] Kimmig, A., Bach, S., Broecheler, M., Huang, B., and Getoor, L. (2012). A short introduction to probabilistic soft logic. In *Proceedings of the NIPS Workshop on Probabilistic Programming: Foundations and Applications December 07, 2012*, pages 1–4. Mansinghka, Vikash.

[Kindermans et al., 2019] Kindermans, P.-J., Hooker, S., Adebayo, J., Alber, M., Schütt, K. T., Dähne, S., Erhan, D., and Kim, B. (2019). The (un)reliability of saliency methods. In *Explainable AI: Interpreting, Explaining and Visualizing Deep Learning*, pages 267–280. Springer.

[Kipf and Welling, 2017] Kipf, T. N. and Welling, M. (2017). Semi-supervised classification with graph convolutional networks. In *Proceedings of the 5th International Conference on Learning Representations (ICLR), Toulon, France, April 24-26, 2017*. OpenReview.net.

[Kohavi and Becker, 1996] Kohavi, R. and Becker, B. (1996). Census income data set. `https://archive.ics.uci.edu/ml/datasets/census+income`.

[Kokalj et al., 2021] Kokalj, E., Škrlj, B., Lavrač, N., Pollak, S., and Robnik-Šikonja, M. (2021). Bert meets shapley: Extending shap explanations to transformer-based classifiers. In *Proceedings of the EACL Hackashop on News Media Content Analysis and Automated Report Generation*, pages 16–21.

[König et al., 2008] König, R., Johansson, U., and Niklasson, L. (2008). G-REX: A versatile framework for evolutionary data mining. In *2008 IEEE International Conference on Data Mining Workshops (ICDM 2008 Workshops)*, pages 971–974.

[Körner et al., 2022] Körner, P., Leuschel, M., Barbosa, J., Costa, V. S., Dahl, V., Hermenegildo, M. V., Morales, J. F., Wielemaker, J., Diaz, D., Abreu, S., and Ciatto, G. (2022). 50 years of prolog and beyond. *CoRR*, abs/2201.10816.

[Krishnan et al., 1999a] Krishnan, R., Sivakumar, G., and Bhattacharya, P. (1999a). Extracting decision trees from trained neural networks. *Pattern Recognition*, 32(12):1999–2009.

[Krishnan et al., 1999b] Krishnan, R., Sivakumar, G., and Bhattacharya, P. (1999b). A search technique for rule extraction from trained neural networks. *Pattern Recognition Letters*, 20(3):273–280.

[Krizhevsky and Hinton, 2009] Krizhevsky, A. and Hinton, G. (2009). Learning multiple layers of features from tiny images. Technical report, Department of Computer Science – University of Toronto.

[Lai et al., 2021] Lai, F., Zhu, X., Madhyastha, H. V., and Chowdhury, M. (2021). Oort: Efficient Federated Learning via Guided Participant Selection. In *15th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2021, July 14-16, 2021*, pages 19–35. USENIX Association.

[Lakkaraju et al., 2017] Lakkaraju, H., Kamar, E., Caruana, R., and Leskovec, J. (2017). Interpretable & explorable approximations of black box models. *CoRR*, abs/1707.01154.

[Lamb et al., 2020] Lamb, L. C., d'Avila Garcez, A. S., Gori, M., Prates, M. O. R., Avelar, P. H. C., and Vardi, M. Y. (2020). Graph neural networks meet neural-symbolic computing: A survey and perspective. In *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, IJCAI 2020*, pages 4877–4884, Yokohama, Japan. ijcai.org.

[LeCun et al., 2010] LeCun, Y., Cortes, C., and Burges, C. (2010). Mnist handwritten digit database. *ATT Labs [Online]. Available: http://yann.lecun.com/exdb/mnist*, 2.

[Lehmann et al., 2010] Lehmann, J., Bader, S., and Hitzler, P. (2010). Extracting reduced logic programs from artificial neural networks. *Applied Intelligence*, 32(3):249–266.

[Levesque and Brachman, 1987] Levesque, H. J. and Brachman, R. J. (1987). Expressiveness and tractability in knowledge representation and reasoning. *Computational Intelligence*, 3:78–93.

[Li et al., 2017] Li, H., Kadav, A., Durdanovic, I., Samet, H., and Graf, H. P. (2017). Pruning filters for efficient convnets. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net.

[Li et al., 2020] Li, H., Zhou, K., Mo, L., Zain, A. M., and Qin, F. (2020). Weighted fuzzy production rule extraction using modified harmony search algorithm and BP neural network framework. *IEEE Access*, 8:186620–186637.

[Li et al., 2018a] Li, J., Liang, X., Shen, S., Xu, T., Feng, J., and Yan, S. (2018a). Scale-aware fast R-CNN for pedestrian detection. *IEEE Transactions on Multimedia*, 20(4):985–996.

[Li et al., 2022a] Li, W., Peng, R., and Li, Z. (2022a). Improving knowledge graph completion via increasing embedding interactions. *Applied Intelligence*, 52(8):9289–9307.

[Li et al., 2023a] Li, W., Peng, R., and Li, Z. (2023a). Knowledge graph completion by jointly learning structural features and soft logical rules. *IEEE Transactions on Knowledge and Data Engineering*, 35(3):2724–2735.

[Li et al., 2023b] Li, W., Zhu, L., Mao, R., and Cambria, E. (2023b). Skier: A symbolic knowledge integrated model for conversational emotion recognition. In *Proceedings of the AAAI Conference on Artificial Intelligence.*

[Li and Roth, 2002] Li, X. and Roth, D. (2002). Learning question classifiers. In *19th International Conference on Computational Linguistics, COLING 2002, Howard International House and Academia Sinica, Taipei, Taiwan, August 24 - September 1, 2002.*

[Li et al., 2018b] Li, Y., Vinyals, O., Dyer, C., Pascanu, R., and Battaglia, P. W. (2018b). Learning deep generative models of graphs. *CoRR*, abs/1803.03324.

[Li et al., 2022b] Li, Z., Wang, X., Stengel-Eskin, E., Kortylewski, A., Ma, W., Durme, B. V., and Yuille, A. L. (2022b). Super-CLEVR: A virtual benchmark to diagnose domain robustness in visual reasoning. *CoRR*, abs/2212.00259.

[Liang et al., 2021] Liang, T., Glossner, J., Wang, L., Shi, S., and Zhang, X. (2021). Pruning and quantization for deep neural network acceleration: A aurvey. *Neurocomputing*, 461:370–403.

[Liang et al., 2018] Liang, X., Hu, Z., Zhang, H., Lin, L., and Xing, E. P. (2018). Symbolic graph reasoning meets convolutions. In *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems (NeurIPS), Montréal, Canada, December 3-8, 2018*, pages 1858–1868.

[Liberis et al., 2021] Liberis, E., Dudziak, L., and Lane, N. D. (2021). $\mu$NAS: Constrained neural architecture search for microcontrollers. In *1st Workshop on Machine Learning and Systems (EuroMLSys@EuroSys 2021)*, pages 70–79, Edinburgh, Scotland, UK. ACM.

[Lin et al., 2023] Lin, Q., Mao, R., Liu, J., Xu, F., and Cambria, E. (2023). Fusing topology contexts and logical rules in language models for knowledge graph completion. *Information Fusion*, 90:253–264.

[Lin et al., 2015] Lin, Y., Liu, Z., Sun, M., Liu, Y., and Zhu, X. (2015). Learning entity and relation embeddings for knowledge graph completion. In *Proceedings of the 29th Conference on Artificial Intelligence (AAAI), Austin, Texas, USA, January 25-30, 2015*, pages 2181–2187. AAAI Press.

[Liscio et al., 2023] Liscio, E., Araque, O., Gatti, L., Constantinescu, I., Jonker, C. M., Kalimeri, K., and Murukannaiah, P. K. (2023). What does a text classifier learn about morality? an explainable method for cross-domain comparison of moral rhetoric. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics*, pages 1–12, Toronto. To appear.

[Liu et al., 2023] Liu, A., Xu, H., Van den Broeck, G., and Liang, Y. (2023). Out-of-distribution generalization by neural-symbolic joint training. In *AAAI Conference on Artificial Intelligence*, volume 37, pages 12252–12259.

[Liu et al., 2002] Liu, B., Abbass, H. A., and McKay, R. I. (2002). Density-based heuristic for rule discovery with ant-miner. In *The 6th Australia-Japan joint workshop on intelligent and evolutionary system*, volume 184.

[Liu et al., 2004] Liu, B., Abbass, H. A., and McKay, R. I. (2004). Classification rule discovery with ant colony optimization. *IEEE Intelligent Informatics Bulletin*, 3(1):31–35.

[Liu et al., 2018] Liu, C., Zoph, B., Neumann, M., Shlens, J., Hua, W., Li, L., Fei-Fei, L., Yuille, A. L., Huang, J., and Murphy, K. (2018). Progressive neural architecture search. In *Computer Vision - ECCV 2018 - 15th European Conference, Munich, Germany, September 8-14, 2018, Proceedings, Part I*, volume 11205 of *Lecture Notes in Computer Science*, pages 19–35. Springer.

[Liu et al., 2019a] Liu, H., Simonyan, K., and Yang, Y. (2019a). DARTS: differentiable architecture search. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net.

[Liu et al., 2017a] Liu, H., Wu, Y., and Yang, Y. (2017a). Analogical inference for multi-relational embeddings. In *Proceedings of the 34th International Conference on Machine Learning (ICML), Sydney, NSW, Australia, August 6-11, 2017*, volume 70 of *Proceedings of Machine Learning Research*, pages 2168–2178. PMLR.

[Liu et al., 2019b] Liu, J., Wen, D., Gao, H., Tao, W., Chen, T., Osa, K., and Kato, M. (2019b). Knowledge representing: Efficient, sparse representation of prior knowledge for knowledge distillation. In *IEEE Conference on Computer Vision and Pattern Recognition Workshops, CVPR Workshops 2019, Long Beach, CA, USA, June 16-20, 2019*, pages 638–646. Computer Vision Foundation / IEEE.

[Liu et al., 2020] Liu, L., Ouyang, W., Wang, X., Fieguth, P. W., Chen, J., Liu, X., and Pietikäinen, M. (2020). Deep learning for generic object detection: A survey. *Int. J. Comput. Vis.*, 128(2):261–318.

[Liu et al., 2016a] Liu, Q., Jiang, H., Ling, Z., Wei, S., and Hu, Y. (2016a). Probabilistic reasoning via deep learning: Neural association models. *CoRR*, abs/1603.07704.

[Liu et al., 2016b] Liu, W., Anguelov, D., Erhan, D., Szegedy, C., Reed, S. E., Fu, C., and Berg, A. C. (2016b). SSD: single shot multibox detector. In *Computer Vision - ECCV 2016 - 14th European Conference, Amsterdam, The Netherlands, October 11-14, 2016, Proceedings, Part I*, volume 9905 of *Lecture Notes in Computer Science*, pages 21–37. Springer.

[Liu et al., 2017b] Liu, Z., Li, J., Shen, Z., Huang, G., Yan, S., and Zhang, C. (2017b). Learning efficient convolutional networks through network slimming. In *IEEE International Conference* *on Computer Vision, ICCV 2017, Venice, Italy, October 22-29, 2017*, pages 2755–2763. IEEE Computer Society.

[Lloyd, 1990] Lloyd, J. W. (1990). *Computational Logic*. Springer.

[Loh, 2014] Loh, W.-Y. (2014). Fifty years of classification and regression trees. *International Statistical Review*, 82(3):329–348.

[Lorena et al., 2019] Lorena, A. C., Garcia, L. P. F., Lehmann, J., de Souto, M. C. P., and Ho, T. K. (2019). How complex is your classification problem?: A survey on measuring classification complexity. *ACM Computing Surveys*, 52(5):107:1–107:34.

[Lu et al., 2020] Lu, Y., Cheung, Y., and Tang, Y. Y. (2020). Bayes imbalance impact index: A measure of class imbalanced data set for classification problem. *IEEE Transactions on Neural Networks and Learning Systems*, 31(9):3525–3539.

[Lua et al., 2005] Lua, E. K., Crowcroft, J., Pias, M., Sharma, R., and Lim, S. (2005). A survey and comparison of peer-to-peer overlay network schemes. *IEEE Communications Surveys & Tutorials*, 7(2):72–93.

[Lundberg and Lee, 2017] Lundberg, S. M. and Lee, S.-I. (2017). A unified approach to interpreting model predictions. *Advances in neural information processing systems*, 30.

[Luo et al., 2020] Luo, R., Tan, X., Wang, R., Qin, T., Chen, E., and Liu, T. (2020). Neural architecture search with GBDT. *CoRR*, abs/2007.04785.

[Luo et al., 2018] Luo, R., Tian, F., Qin, T., Chen, E., and Liu, T. (2018). Neural architecture optimization. In *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, December 3-8, 2018, Montréal, Canada*, pages 7827–7838.

[Luo et al., 2021] Luo, S., Ivison, H., Han, C., and Poon, J. (2021). Local interpretations for explainable natural language processing: A survey. *CoRR*.

[Lyu et al., 2020] Lyu, L., Yu, H., and Yang, Q. (2020). Threats to Federated Learning: A Survey. *CoRR*, abs/2003.02133.

[Ma and Zhang, 2018] Ma, T. and Zhang, A. (2018). Multi-view factorization autoencoder with network constraints for multi-omic integrative analysis. In *Proceedings of the IEEE International Conference on Bioinformatics and Biomedicine (BIBM), Madrid, Spain, December 3-6, 2018*, pages 702–707. IEEE Computer Society.

[Ma et al., 2022] Ma, X., Zhang, J., Guo, S., and Xu, W. (2022). Layer-wised Model Aggregation for Personalized Federated Learning. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR 2022, New Orleans, LA, USA, June 18-24, 2022*, pages 10082–10091. IEEE.

[Madsen et al., 2022] Madsen, A., Reddy, S., and Chandar, S. (2022). Post-hoc interpretability for neural NLP: A survey. *ACM Computing Surveys*, 55(8):1–42.

[Magnini et al., 2022a] Magnini, M., Ciatto, G., and Omicini, A. (2022a). KINS: Knowledge injection via network structuring. In *CILC 2022 – Italian Conference on Computational Logic*, volume 3204 of *CEUR Workshop Proceedings*, pages 254–267.

[Magnini et al., 2022b] Magnini, M., Ciatto, G., and Omicini, A. (2022b). On the design of PSyKI: a platform for symbolic knowledge injection into sub-symbolic predictors. In *Explainable and Transparent AI and Multi-Agent Systems*, volume 13283 of *Lecture Notes in Computer Science*, chapter 6, pages 90–108. Springer, Cham, Switzerland.

[Magnini et al., 2022c] Magnini, M., Ciatto, G., and Omicini, A. (2022c). A view to a KILL: Knowledge injection via lambda layer. In *WOA 2022 – 23rd Workshop "From Objects to Agents"*, volume 3261 of *CEUR Workshop Proceedings*, pages 61–76. CEUR-WS.org.

[Mahdavifar and Ghorbani, 2020] Mahdavifar, S. and Ghorbani, A. A. (2020). DeNNeS: deep embedded neural network expert system for detecting cyber attacks. *Neural Computing and Applications*, 32(18):14753–14780.

[Makowsky, 1987] Makowsky, J. A. (1987). Why horn formulas matter in computer science: Initial structures and generic examples. *Journal of Computer and System Sciences*, 34(2):266–292.

[Manhaeve et al., 2021] Manhaeve, R., Dumancic, S., Kimmig, A., Demeester, T., and De Raedt, L. (2021). Neural probabilistic logic programming in deepproblog. *Artificial Intelligence*, 298:103504.

[Mao et al., 2019] Mao, J., Gan, C., Kohli, P., Tenenbaum, J. B., and Wu, J. (2019). The neuro-symbolic concept learner: Interpreting scenes, words, and sentences from natural supervision. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net.

[Marconato et al., 2023] Marconato, E., Bontempo, G., Ficarra, E., Calderara, S., Passerini, A., and Teso, S. (2023). Neuro symbolic continual learning: Knowledge, reasoning shortcuts and concept rehearsal. *CoRR*, abs/2302.01242.

[Margatina et al., 2021] Margatina, K., Vernikos, G., Barrault, L., and Aletras, N. (2021). Active learning by acquiring contrastive examples. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing, EMNLP 2021, Virtual Event / Punta Cana, Dominican Republic, 7-11 November, 2021*, pages 650–663. Association for Computational Linguistics.

[Marino et al., 2021] Marino, K., Chen, X., Parikh, D., Gupta, A., and Rohrbach, M. (2021). KRISP: integrating implicit and symbolic knowledge for open-domain knowledge-based VQA.

In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2021, virtual, June 19-25, 2021*, pages 14111–14121. Computer Vision Foundation / IEEE.

[Marino et al., 2017] Marino, K., Salakhutdinov, R., and Gupta, A. (2017). The more you know: Using knowledge graphs for image classification. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Honolulu, HI, USA, July 21-26, 2017*, pages 20–28. IEEE Computer Society.

[Markowska-Kaczmar and Chumieja, 2004] Markowska-Kaczmar, U. and Chumieja, M. (2004). Discovering the mysteries of neural networks. *International Journal of Hybrid Intelligent Systems*, 1(3-4):153–163.

[Markowska-Kaczmar and Trelak, 2003] Markowska-Kaczmar, U. and Trelak, W. (2003). Extraction of fuzzy rules from trained neural network using evolutionary algorithm. In *ESANN 2003, 11th European Symposium on Artificial Neural Networks, Bruges, Belgium, April 23-25, 2003, Proceedings*, pages 149–154.

[Marra et al., 2019] Marra, G., Giannini, F., Diligenti, M., and Gori, M. (2019). LYRICS: a general interface layer to integrate AI and deep learning. *CoRR*, abs/1903.07534.

[Marshakov, 2021] Marshakov, D. V. (2021). Rule extraction from the artificial neural network. In *IOP Conference Series: Materials Science and Engineering*, volume 1029, page 012127. IOP Publishing.

[Martens et al., 2009] Martens, D., Baesens, B., and Van Gestel, T. (2009). Decompositional rule extraction from support vector machines by active learning. *IEEE Transactions on Knowledge and Data Engineering*, 21(2):178–191.

[Martens et al., 2007] Martens, D., De Backer, M., Haesen, R., Vanthienen, J., Snoeck, M., and Baesens, B. (2007). Classification with ant colony optimization. *IEEE Transactions on Evolutionary Computation*, 11(5):651–665.

[Masuoka et al., 1990] Masuoka, R., Watanabe, N., Kawamura, A., Owada, Y., and Asakawa, K. (1990). Neurofuzzy systems – Fuzzy inference using a structured neural network. In *Proceedings of International Conference on Fuzzy Logic and Neural Networks, Iizuka Japan, July, 1990*, pages 173–177.

[Matthews and Jagielska, 1995] Matthews, C. and Jagielska, I. (1995). Fuzzy rule extraction from a trained multilayer neural network. In *Proceedings of ICNN'95 - International Conference on Neural Networks*, volume 2, pages 744–748 vol.2.

[McKnight and Najab, 2010] McKnight, P. E. and Najab, J. (2010). Mann-whitney u test. In *The Corsini Encyclopedia of Psychology*, pages 1–1.

[McMahan et al., 2017] McMahan, B., Moore, E., Ramage, D., Hampson, S., and y Arcas, B. A. (2017). Communication-Efficient Learning of Deep Networks from Decentralized Data. In

*Proceedings of the 20th International Conference on Artificial Intelligence and Statistics, AISTATS 2017, 20-22 April 2017, Fort Lauderdale, FL, USA*, volume 54 of *Proceedings of Machine Learning Research*, pages 1273–1282. PMLR.

[McNulty, 1977] McNulty, G. F. (1977). Fragments of first order logic, i: Universal Horn logic. *Journal of Symbolic Logic*, 42(2):221–237.

[Meidan et al., 2018] Meidan, Y., Bohadana, M., Mathov, Y., Mirsky, Y., Shabtai, A., Breitenbacher, D., and Elovici, Y. (2018). N-BaIoT - Network-Based Detection of IoT Botnet Attacks Using Deep Autoencoders. *IEEE Pervasive Computing*, 17(3):12–22.

[Meller et al., 2019] Meller, E., Finkelstein, A., Almog, U., and Grobman, M. (2019). Same, same but different: Recovering neural network quantization error through weight factorization. In *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*, volume 97 of *Proceedings of Machine Learning Research*, pages 4486–4495. PMLR.

[Meng et al., 2017] Meng, X., Wang, W., and Zhang, Z. (2017). Delay-Constrained Hybrid Computation Offloading With Cloud and Fog Computing. *IEEE Access*, 5:21355–21367.

[Miller et al., 1989] Miller, G. F., Todd, P. M., and Hegde, S. U. (1989). Designing neural networks using genetic algorithms. In *3rd International Conference on Genetic Algorithms*, pages 379–384, Fairfax, VA, USA. Morgan Kaufmann.

[Mintz et al., 2009] Mintz, M., Bills, S., Snow, R., and Jurafsky, D. (2009). Distant supervision for relation extraction without labeled data. In *Proceedings of the 47th Annual Meeting of the Association for Computational Linguistics (ACL) and the 4th International Joint Conference on Natural Language Processing of the AFNLP, Singapore, August 2-7, 2009*, pages 1003–1011. The Association for Computer Linguistics (ACL).

[Mirzasoleiman et al., 2020] Mirzasoleiman, B., Bilmes, J. A., and Leskovec, J. (2020). Coresets for data-efficient training of machine learning models. In *Proceedings of the 37th International Conference on Machine Learning, ICML 2020, 13-18 July 2020, Virtual Event*, volume 119 of *Proceedings of Machine Learning Research*, pages 6950–6960. PMLR.

[Mitra, 1994] Mitra, S. (1994). Fuzzy mlp based expert system for medical diagnosis. *Fuzzy Sets and Systems*, 65(2):285–296. Fuzzy Methods for Computer Vision and Pattern Recognition.

[Moreschini et al., 2022] Moreschini, S., Pecorelli, F., Li, X., Naz, S., Hästbacka, D., and Taibi, D. (2022). Cloud continuum: The definition. *IEEE Access*, 10:131876–131886.

[Mrksic et al., 2016] Mrksic, N., Ó Séaghdha, D., Thomson, B., Gasic, M., Rojas-Barahona, L. M., Su, P., Vandyke, D., Wen, T., and Young, S. J. (2016). Counter-fitting word vectors to linguistic constraints. In *Proceedings of the Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL), San*

*Diego California, USA, June 12-17, 2016*, pages 142–148. The Association for Computational Linguistics.

[Murphy and Pazzani, 1991] Murphy, P. M. and Pazzani, M. J. (1991). Id2-of-3: Constructive induction of m-of-n concepts for discriminators in decision trees. In *Machine Learning Proceedings 1991*, pages 183–187. Elsevier.

[Nagel et al., 2021] Nagel, M., Fournarakis, M., Amjad, R. A., Bondarenko, Y., van Baalen, M., and Blankevoort, T. (2021). A white paper on neural network quantization. *CoRR*, abs/2106.08295.

[Nagel et al., 2019] Nagel, M., van Baalen, M., Blankevoort, T., and Welling, M. (2019). Data-free quantization through weight equalization and bias correction. In *2019 IEEE/CVF International Conference on Computer Vision, ICCV 2019, Seoul, Korea (South), October 27 - November 2, 2019*, pages 1325–1334. IEEE.

[Nam and Han, 2016] Nam, H. and Han, B. (2016). Learning multi-domain convolutional neural networks for visual tracking. In *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016*, pages 4293–4302. IEEE Computer Society.

[Nash Jr, 1950] Nash Jr, J. F. (1950). Equilibrium points in n-person games. *Proceedings of the national academy of sciences*, 36(1):48–49.

[Nauck and Kruse, 1997] Nauck, D. D. and Kruse, R. (1997). A neuro-fuzzy method to learn fuzzy classification rules from data. *Fuzzy Sets and Systems*, 89(3):277–288.

[Nauck and Kruse, 1999] Nauck, D. D. and Kruse, R. (1999). Neuro-fuzzy systems for function approximation. *Fuzzy Sets and Systems*, 101(2):261–271.

[Nguyen and Martínez, 2020] Nguyen, A. and Martínez, M. R. (2020). On quantitative aspects of model interpretability. *CoRR*, abs/2007.07584.

[Nguyen et al., 2016] Nguyen, A. M., Yosinski, J., and Clune, J. (2016). Multifaceted feature visualization: Uncovering the different types of features learned by each neuron in deep neural networks. *CoRR*, abs/1602.03616.

[Nguyen et al., 2023] Nguyen, C., French, T., Liu, W., and Stewart, M. (2023). Cyle: Cylinder embeddings for multi-hop reasoning over knowledge graphs. In *Proceedings of the 17th Conference of the European Chapter of the Association for Computational Linguistics, EACL 2023, Dubrovnik, Croatia, May 2-6, 2023*, pages 1728–1743. Association for Computational Linguistics.

[Nguyen et al., 2021] Nguyen, D. C., Ding, M., Pathirana, P. N., Seneviratne, A., Li, J., and Poor, H. V. (2021). Federated Learning for Internet of Things: A Comprehensive Survey. *IEEE Communications Surveys & Tutorials*, 23(3):1622–1658.

[Nguyen and Grishman, 2016] Nguyen, T. H. and Grishman, R. (2016). Modeling skip-grams for event detection with convolutional neural networks. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing, EMNLP 2016, Austin, Texas, USA, November 1-4, 2016*, pages 886–891. The Association for Computational Linguistics.

[Nickel et al., 2016] Nickel, M., Rosasco, L., and Poggio, T. (2016). Holographic embeddings of knowledge graphs. In *Thirtieth AAAI Conference on Artificial Intelligence*, volume 30 of *Proceedings of the AAAI Conference on Artificial Intelligence*, Phoenix, AZ, USA. Association for the Advancement of Artificial Intelligence.

[Nickel et al., 2011] Nickel, M., Tresp, V., and Kriegel, H.-P. (2011). A three-way model for collective learning on multi-relational data. In *28th International Conference on Machine Learning (ICML 2011)*, pages 809–816, New York, NY, USA. ACM.

[Northcutt et al., 2021] Northcutt, C. G., Jiang, L., and Chuang, I. L. (2021). Confident learning: Estimating uncertainty in dataset labels. *Journal of Artificial Intelligence Research*, 70:1373–1411.

[Novelli et al., 2023] Novelli, C., Taddeo, M., and Floridi, L. (2023). Accountability in artificial intelligence: what it is and how it works. *AI & SOCIETY*, pages 1–12.

[Núñez et al., 2008] Núñez, H., Angulo, C., and Català, A. (2008). Rule extraction based on support and prototype vectors. In *Rule Extraction from Support Vector Machines*, volume 80 of *Studies in Computational Intelligence*, pages 109–134. Springer.

[Nye et al., 2021] Nye, M. I., Tessler, M. H., Tenenbaum, J. B., and Lake, B. M. (2021). Improving coherence and consistency in neural sequence models with dual-system, neuro-symbolic reasoning. In *Advances in Neural Information Processing Systems 34 (NeurIPS 2021)*, pages 25192–25204.

[Obregon and Jung, 2023] Obregon, J. and Jung, J. (2023). RuleCOSI+: Rule extraction for interpreting classification tree ensembles. *Information Fusion*, 89:355–381.

[Obregon et al., 2019] Obregon, J., Kim, A., and Jung, J. (2019). RuleCOSI: Combination and simplification of production rules from boosted decision trees for imbalanced classification. *Expert Systems with Applications*, 126:64–82.

[Odajima et al., 2008] Odajima, K., Hayashi, Y., Tianxia, G., and Setiono, R. (2008). Greedy rule generation from discrete data and its use in neural network rule extraction. *Neural Networks*, 21(7):1020–1028.

[Odense and d'Avila Garcez, 2020] Odense, S. and d'Avila Garcez, A. S. (2020). Layerwise knowledge extraction from deep convolutional networks. *CoRR*, abs/2003.09000.

[Parpinelli et al., 2001] Parpinelli, R. S., Lopes, H. S., and Freitas, A. A. (2001). An ant colony based system for data mining: applications to medical data. In *Proceedings of the genetic and evolutionary computation conference (GECCO-2001)*, pages 791–797. Citeseer.

[Passalis and Tefas, 2018] Passalis, N. and Tefas, A. (2018). Learning deep representations with probabilistic knowledge transfer. In *Computer Vision - ECCV 2018 - 15th European Conference, Munich, Germany, September 8-14, 2018, Proceedings, Part XI*, volume 11215 of *Lecture Notes in Computer Science*, pages 283–299. Springer.

[Paul et al., 2021] Paul, M., Ganguli, S., and Dziugaite, G. K. (2021). Deep learning on a data diet: Finding important examples early in training. In *Advances in Neural Information Processing Systems 34: Annual Conference on Neural Information Processing Systems 2021, NeurIPS 2021, December 6-14, 2021, virtual*, pages 20596–20607.

[Pawlicki et al., 2022] Pawlicki, M., Kozik, R., and Choras, M. (2022). A survey on neural networks for (cyber-) security and (cyber-) security of neural networks. *Neurocomputing*, 500:1075–1087.

[Peng et al., 2021] Peng, B., Li, C., Zhang, Z., Li, J., Zhu, C., and Gao, J. (2021). SYNERGY: building task bots at scale using symbolic knowledge and machine teaching. *CoRR*, abs/2110.11514.

[Peng et al., 2020] Peng, S., Ji, F., Lin, Z., Cui, S., Chen, H., and Zhang, Y. (2020). MTSS: learn from multiple domain teachers and become a multi-domain dialogue expert. In *AAAI Conference on Artificial Intelligence (AAAI-20 Technical Tracks 5)*, volume 34, pages 8608–8615. AAAI Press.

[Peters et al., 2019] Peters, M. E., Neumann, M., Logan IV, R. L., Schwartz, R., Joshi, V., Singh, S., and Smith, N. A. (2019). Knowledge enhanced contextual word representations. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP), Hong Kong, China, November 3-7, 2019*, pages 43–54. Association for Computational Linguistics.

[Pham et al., 2018] Pham, H., Guan, M. Y., Zoph, B., Le, Q. V., and Dean, J. (2018). Efficient neural architecture search via parameter sharing. In *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*, volume 80 of *Proceedings of Machine Learning Research*, pages 4092–4101. PMLR.

[Piedmont, 2014] Piedmont, R. L. (2014). Bias, statistical. In *Encyclopedia of Quality of Life and Well-Being Research*, pages 382–383. Springer Netherlands, Dordrecht.

[Pillutla et al., 2022] Pillutla, K., Kakade, S. M., and Harchaoui, Z. (2022). Robust Aggregation for Federated Learning. *IEEE Transactions on Signal Processing*, 70:1142–1154.

[Pinkas et al., 2013] Pinkas, G., Lima, P., and Cohen, S. (2013). Representing, binding, retrieving and unifying relational knowledge using pools of neural binders. *Biologically Inspired Cognitive Architectures*, 6:87–95. BICA 2013: Papers from the Fourth Annual Meeting of the BICA Society.

[Pinkus, 1991] Pinkus, G. (1991). Constructing proofs in symmetric networks. In *Advances in Neural Information Processing Systems 4, [NIPS Conference, Denver, Colorado, USA, December 2-5, 1991]*, pages 217–224. Morgan Kaufmann.

[Pop et al., 1994] Pop, E., Hayward, R., and Diederich, J. (1994). RULENEG: Extracting rules from a trained ANN by stepwise negation. Technical report, Neurocomputing Research Centre, Queensland University of Technology.

[Pourvali et al., 2023] Pourvali, M., Meng, Y., Sheng, C., and Du, Y. (2023). Taxoknow: Taxonomy as prior knowledge in the loss function of multi-class classification. *CoRR*, abs/2305.16341.

[Puliafito et al., 2020] Puliafito, C., Gonçalves, D. M., Lopes, M. M., Martins, L. L., Madeira, E., Mingozzi, E., Rana, O., and Bittencourt, L. F. (2020). MobFogSim: Simulation of mobility and migration for fog computing. *Simulation Modelling Practice and Theory*, 101:102062. Modeling and Simulation of Fog Computing.

[Puliafito et al., 2019] Puliafito, C., Mingozzi, E., Longo, F., Puliafito, A., and Rana, O. (2019). Fog Computing for the Internet of Things. *ACM Transactions on Internet Technology*, 19(2):1–41.

[Quinlan, 1986] Quinlan, J. R. (1986). Induction of decision trees. *Machine Learning*, 1(1):81–106.

[Quinlan, 1993] Quinlan, J. R. (1993). *C4.5: Programs for Machine Learning*. Morgan Kauffmann, San Mateo, CA, USA.

[Rabenstein and Volz, 2015] Rabenstein, B. and Volz, J. (2015). Prometheus: A next-generation monitoring system (talk). In *SRECon15 Europe*, Dublin. USENIX Association.

[Rabuñal et al., 2004] Rabuñal, J. R., Dorado, J., Pazos, A., Pereira, J., and Rivero, D. (2004). A new approach to the extraction of ANN rules and to their generalization capacity through GP. *Neural Computation*, 16(7):1483–1523.

[Raissi et al., 2019] Raissi, M., Perdikaris, P., and Karniadakis, G. E. (2019). Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *Journal of Computational Physics*, 378:686–707.

[Real et al., 2019] Real, E., Aggarwal, A., Huang, Y., and Le, Q. V. (2019). Regularized evolution for image classifier architecture search. In *AAAI Conference on Artificial Intelligence (AAAI-19, IAAI-19, EAAI-20)*, volume 33, pages 4780–4789. AAAI Press.

[Redmon and Farhadi, 2018] Redmon, J. and Farhadi, A. (2018). YOLOv3: An incremental improvement. *CoRR*, abs/1804.02767:1–6. `http://arxiv.org/abs/1804.02767`.

[Ren and Leskovec, 2020] Ren, H. and Leskovec, J. (2020). Beta embeddings for multi-hop logical reasoning in knowledge graphs. In *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual.*

[Ren et al., 2021] Ren, P., Xiao, Y., Chang, X., Huang, P., Li, Z., Chen, X., and Wang, X. (2021). A comprehensive survey of neural architecture search: Challenges and solutions. *ACM Computing Surveys*, 54(4):76:1–76:34.

[Ren et al., 2015] Ren, S., He, K., Girshick, R. B., and Sun, J. (2015). Faster R-CNN: towards real-time object detection with region proposal networks. In *Advances in Neural Information Processing Systems 28: Annual Conference on Neural Information Processing Systems (NeurIPS), Montreal, Quebec, Canada, December 7-12, 2015*, pages 91–99.

[Ren et al., 2017] Ren, S., He, K., Girshick, R. B., Zhang, X., and Sun, J. (2017). Object detection networks on convolutional feature maps. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 39(7):1476–1481.

[Ribeiro et al., 2016] Ribeiro, M. T., Singh, S., and Guestrin, C. (2016). "Why should I trust you?": Explaining the predictions of any classifier. In *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*, pages 1135–1144.

[Riegel et al., 2020] Riegel, R., Gray, A. G., Luus, F. P. S., Khan, N., Makondo, N., Akhalwaya, I. Y., Qian, H., Fagin, R., Barahona, F., Sharma, U., Ikbal, S., Karanam, H., Neelam, S., Likhyani, A., and Srivastava, S. K. (2020). Logical neural networks. *CoRR*, abs/2006.13155.

[Rjoub et al., 2022] Rjoub, G., Wahab, O. A., Bentahar, J., and Bataineh, A. (2022). Trust-driven reinforcement selection strategy for federated learning on IoT devices. *Computing*.

[Rocha et al., 2012] Rocha, A., Papa, J. P., and Meira, L. A. A. (2012). How far do we get using machine learning black-boxes? *International Journal of Pattern Recognition and Artificial Intelligence*, 26(02):1261001–(1–23).

[Rocktäschel and Riedel, 2017] Rocktäschel, T. and Riedel, S. (2017). End-to-end differentiable proving. In *Advances in Neural Information Processing Systems: Annual Conference on Neural Information Processing Systems, Long Beach, CA, USA, December 4-9, 2017*, pages 3788–3800.

[Rocktäschel et al., 2015] Rocktäschel, T., Singh, S., and Riedel, S. (2015). Injecting logical background knowledge into embeddings for relation extraction. In *The 2015 Conference of the North American Chapter of the Association for Computational Linguistics (NAACL): Human Language Technologies (HLT), Denver, Colorado, USA, May 31 - June 5, 2015*, pages 1119–1129. The Association for Computational Linguistics.

[Rodríguez et al., 2022] Rodríguez, N. D., Lamas, A., Sanchez, J., Franchi, G., Donadello, I., Tabik, S., Filliat, D., Cruz, P., Montes, R., and Herrera, F. (2022). Explainable neural-symbolic learning (*X-NeSyL*) methodology to fuse deep learning representations with expert knowledge graphs: The monumai cultural heritage use case. *Information Fusion*, 79:58–83.

[Rolnick and Tegmark, 2018] Rolnick, D. and Tegmark, M. (2018). The power of deeper networks for expressing natural functions. In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*. OpenReview.net.

[Rovnyagin et al., 2020] Rovnyagin, M. M., Hrapov, A. S., Guminskaia, A. V., and Orlov, A. P. (2020). Ml-based heterogeneous container orchestration architecture. In *2020 IEEE Conference of Russian Young Researchers in Electrical and Electronic Engineering (EIConRus)*, pages 477–481.

[Rudyy et al., 2019] Rudyy, O., Garcia-Gasulla, M., Mantovani, F., Santiago, A., Sirvent, R., and Vázquez, M. (2019). Containers in HPC: A Scalability and Portability Study in Production Biological Simulations. In *2019 IEEE International Parallel and Distributed Processing Symposium, IPDPS 2019, Rio de Janeiro, Brazil, May 20-24, 2019*, pages 567–577. IEEE.

[Saad and Wunsch II, 2007] Saad, E. W. and Wunsch II, D. C. (2007). Neural network explanation using inversion. *Neural Networks*, 20(1):78–93.

[Sabbatini and Calegari, 2022] Sabbatini, F. and Calegari, R. (2022). Symbolic knowledge extraction from opaque machine learning predictors: GridREx & PEDRO. In *Proceedings of the 19th International Conference on Principles of Knowledge Representation and Reasoning, KR 2022, Haifa, Israel. July 31 - August 5, 2022.*

[Sabbatini and Calegari, 2023] Sabbatini, F. and Calegari, R. (2023). Unveiling opaque predictors via explainable clustering: The CReEPy algorithm. In *Proceedings of the 2nd BEWARE workshop, BEWARE 2023, Rome, Italy. November 6, 2023 (accepted).*

[Sabbatini et al., 2022] Sabbatini, F., Ciatto, G., Calegari, R., and Omicini, A. (2022). Symbolic knowledge extraction from opaque ML predictors in PSyKE: Platform design & experiments. *Intelligenza Artificiale*, 16(1):27–48.

[Sabbatini et al., 2021] Sabbatini, F., Ciatto, G., and Omicini, A. (2021). GridEx: An algorithm for knowledge extraction from black-box regressors. In *Explainable and Transparent AI and Multi-Agent Systems. Third International Workshop, EXTRAAMAS 2021, Virtual Event, May 3–7, 2021, Revised Selected Papers*, volume 12688 of *LNCS*, pages 18–38. Springer Nature, Basel, Switzerland.

[Saito and Nakano, 1988] Saito, K. and Nakano, R. (1988). Medical diagnostic expert system based on PDP model. In *Proceedings of International Conference on Neural Networks (ICNN'88), San Diego, CA, USA, July 24-27, 1988*, pages 255–262. IEEE.

[Saito and Nakano, 1997] Saito, K. and Nakano, R. (1997). Law discovery using neural networks. In *Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence, IJCAI 97, Nagoya, Japan, August 23-29, 1997, 2 Volumes*, pages 1078–1083. Morgan Kaufmann.

[Saito and Nakano, 2002] Saito, K. and Nakano, R. (2002). Extracting regression rules from neural networks. *Neural Networks*, 15(10):1279–1288.

[Sajjad et al., 2020] Sajjad, M., Nasir, M., Muhammad, K., Khan, S., Jan, Z., Sangaiah, A. K., Elhoseny, M., and Baik, S. W. (2020). Raspberry pi assisted face recognition framework for enhanced law-enforcement services in smart cities. *Future Generation Computer Systems*, 108:995–1007.

[Salimi-Badr and Ebadzadeh, 2022] Salimi-Badr, A. and Ebadzadeh, M. M. (2022). A novel learning algorithm based on computing the rules' desired outputs of a TSK fuzzy neural network with non-separable fuzzy rules. *Neurocomputing*, 470:139–153.

[Samek et al., 2021] Samek, W., Montavon, G., Lapuschkin, S., Anders, C. J., and Müller, K.-R. (2021). Explaining deep neural networks and beyond: A review of methods and applications. *Proceedings of the IEEE*, 109(3):247–278.

[Sanchez-Iborra and Skarmeta, 2020] Sanchez-Iborra, R. and Skarmeta, A. F. (2020). TinyML-enabled frugal smart objects: Challenges and opportunities. *IEEE Circuits and Systems Magazine*, 20(3):4–18.

[Sarkar et al., 2023] Sarkar, S., Babar, M. F., Hassan, M. M., Hasan, M., and Santu, S. K. K. (2023). Exploring challenges of deploying bert-based NLP models in resource-constrained embedded devices. *CoRR*, abs/2304.11520.

[Sarker et al., 2021] Sarker, M. K., Zhou, L., Eberhart, A., and Hitzler, P. (2021). Neuro-symbolic artificial intelligence. *AI Communications*, 34(3):197–209.

[Sato and Tsukimoto, 2001] Sato, M. and Tsukimoto, H. (2001). Rule extraction from neural networks via decision tree induction. In *IJCNN'01. International Joint Conference on Neural Networks. Proceedings (Cat. No. 01CH37222)*, volume 3, pages 1870–1875. IEEE.

[Savazzi et al., 2021] Savazzi, S., Nicoli, M., Bennis, M., Kianoush, S., and Barbieri, L. (2021). Opportunities of federated learning in connected, cooperative, and automated industrial systems. *IEEE Communications Magazine*, 59(2):16–21.

[Schetinin et al., 2007] Schetinin, V., Fieldsend, J. E., Partridge, D., Coats, T. J., Krzanowski, W. J., Everson, R. M., Bailey, T. C., and Hernandez, A. (2007). Confident interpretation of bayesian decision tree ensembles for clinical applications. *IEEE Transactions on Information Technology in Biomedicine*, 11(3):312–319.

[Schmitz et al., 1999] Schmitz, G. P. J., Aldrich, C., and Gouws, F. S. (1999). ANN-DT: an algorithm for extraction of decision trees from artificial neural networks. *IEEE Transactions on Neural Networks*, 10(6):1392–1401.

[Sen et al., 2022] Sen, P., de Carvalho, B. W. S. R., Riegel, R., and Gray, A. G. (2022). Neuro-symbolic inductive logic programming with logical neural networks. In *Thirty-Sixth AAAI*

*Conference on Artificial Intelligence, AAAI 2022, Thirty-Fourth Conference on Innovative Applications of Artificial Intelligence, IAAI 2022, The Twelveth Symposium on Educational Advances in Artificial Intelligence, EAAI 2022 Virtual Event, February 22 - March 1, 2022*, pages 8212–8219. AAAI Press.

[Serban et al., 2021] Serban, A. C., Poll, E., and Visser, J. (2021). Adversarial examples on object recognition: A comprehensive survey. *ACM Computing Surveys*, 53(3):66:1–66:38.

[Sestito and Dillon, 1994] Sestito, S. and Dillon, T. S. (1994). *Automated knowledge acquisition.* Prentice Hall International series in computer science and engineering. Prentice Hall.

[Sethi et al., 2012] Sethi, K. K., Mishra, D. K., and Mishra, B. (2012). KDRuleEx: A novel approach for enhancing user comprehensibility using rule extraction. In *2012 Third International Conference on Intelligent Systems Modelling and Simulation*, pages 55–60.

[Setiono, 1997] Setiono, R. (1997). Extracting rules from neural networks by pruning and hidden-unit splitting. *Neural Computation*, 9(1):205–225.

[Setiono, 2000] Setiono, R. (2000). Extracting M-of-N rules from trained neural networks. *IEEE Transactions on Neural Networks and Learning Systems*, 11(2):512–519.

[Setiono et al., 2008] Setiono, R., Baesens, B., and Mues, C. (2008). Recursive neural network rule extraction for data with mixed attributes. *IEEE Transactions on Neural Networks*, 19(2):299–307.

[Setiono and Leow, 2000] Setiono, R. and Leow, W. K. (2000). FERNN: An algorithm for fast extraction of rules from neural networks. *Applied Intelligence*, 12(1-2):15–25.

[Setiono et al., 2002] Setiono, R., Leow, W. K., and Zurada, J. M. (2002). Extraction of rules from artificial neural networks for nonlinear regression. *IEEE Transactions on Neural Networks*, 13(3):564–577.

[Setiono and Liu, 1996] Setiono, R. and Liu, H. (1996). Symbolic representation of neural networks. *Computer*, 29(3):71–77.

[Setiono and Liu, 1997] Setiono, R. and Liu, H. (1997). Neurolinear: A system for extracting oblique decision rules from neural networks. In *Machine Learning: ECML-97, 9th European Conference on Machine Learning, Prague, Czech Republic, April 23-25, 1997, Proceedings*, volume 1224 of *Lecture Notes in Computer Science*, pages 221–233. Springer.

[Setiono and Thong, 2004] Setiono, R. and Thong, J. Y. L. (2004). An approach to generate rules from neural networks for regression problems. *European Journal of Operational Research*, 155(1):239–250.

[Sevilla et al., 2022] Sevilla, J., Heim, L., Ho, A., Besiroglu, T., Hobbhahn, M., and Villalobos, P. (2022). Compute Trends Across Three Eras of Machine Learning. In *International Joint Conference on Neural Networks, IJCNN 2022, Padua, Italy, July 18-23, 2022*, pages 1–8. IEEE.

[Shah and Lau, 2023] Shah, S. M. and Lau, V. K. N. (2023). Model compression for communication efficient federated learning. *IEEE Trans. Neural Networks Learn. Syst.*, 34(9):5937–5951.

[Shams et al., 2021] Shams, Z., Dimanov, B., Kola, S., Simidjievski, N., Terre, H. A., Scherer, P., Matjašec, U., Abraham, J., Liò, P., and Jamnik, M. (2021). REM: An integrative rule extraction methodology for explainable data analysis in healthcare. *medRxiv*, pages 2021–01.

[Shan et al., 2019] Shan, F., Luo, J., Jin, J., and Wu, W. (2019). Offloading delay constrained transparent computing tasks with energy-efficient transmission power scheduling in wireless iot environment. *IEEE Internet of Things Journal*, 6(3):4411–4422.

[Shumailov et al., 2021] Shumailov, I., Zhao, Y., Bates, D., Papernot, N., Mullins, R. D., and Anderson, R. (2021). Sponge Examples: Energy-Latency Attacks on Neural Networks. In *IEEE European Symposium on Security and Privacy, EuroS&P 2021, Vienna, Austria, September 6-10, IEEE*, pages 212–231.

[Simonyan and Zisserman, 2015] Simonyan, K. and Zisserman, A. (2015). Very deep convolutional networks for large-scale image recognition. In *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings.*

[Singh et al., 2022] Singh, P., Masud, M., Hossain, M. S., Kaur, A., Muhammad, G., and Ghoneim, A. (2022). Privacy-Preserving Serverless Computing Using Federated Learning for Smart Grids. *IEEE Transactions on Industrial Informatics*, 18(11):7843–7852.

[Škrlj et al., 2021] Škrlj, B., Martinc, M., Lavrač, N., and Pollak, S. (2021). autoBOT: evolving neuro-symbolic representations for explainable low resource text classification. *Machine Learning*, 110(5):989–1028.

[Smirnova et al., 2022] Smirnova, A., Yang, J., Yang, D., and Cudre-Mauroux, P. (2022). Nessy: A neuro-symbolic system for label noise reduction. *IEEE Transactions on Knowledge and Data Engineering.*

[Smullyan, 1968] Smullyan, R. M. (1968). First-order logic. preliminaries. *First-Order Logic.*

[Socher et al., 2013] Socher, R., Chen, D., Manning, C. D., and Ng, A. Y. (2013). Reasoning with neural tensor networks for knowledge base completion. In *Advances in Neural Information Processing Systems 26: 27th Annual Conference on Neural Information Processing Systems 2013. Proceedings of a meeting held December 5-8, 2013, Lake Tahoe, Nevada, United States*, pages 926–934.

[Sourek et al., 2018] Sourek, G., Aschenbrenner, V., Zelezný, F., Schockaert, S., and Kuzelka, O. (2018). Lifted relational neural networks: Efficient learning of latent relational structures. *Journal of Artificial Intelligence Research*, 62:69–100.

[Spillo et al., 2022] Spillo, G., Musto, C., de Gemmis, M., Lops, P., and Semeraro, G. (2022). Knowledge-aware recommendations based on neuro-symbolic graph embeddings and first-order logical rules. In *RecSys '22: Sixteenth ACM Conference on Recommender Systems, Seattle, WA, USA, September 18 - 23, 2022*, pages 616–621. ACM.

[Stamoulis et al., 2018] Stamoulis, D., Chin, T. R., Prakash, A. K., Fang, H., Sajja, S., Bognar, M., and Marculescu, D. (2018). Designing adaptive neural networks for energy-constrained image classification. In *Proceedings of the 37th International Conference on Computer-Aided Design (ICCAD), San Diego, CA, USA, November 05-08, 2018*, pages 1–8, New York, NY, USA. Association for Computing Machinery (ACM).

[Stewart and Ermon, 2017] Stewart, R. and Ermon, S. (2017). Label-free supervision of neural networks with physics and domain knowledge. In *Proceedings of the 31st Conference on Artificial Intelligence (AAAI), San Francisco, California, USA, February 4-9, 2017*, pages 2576–2582. AAAI Press.

[Stock et al., 2021] Stock, P., Fan, A., Graham, B., Grave, E., Gribonval, R., Jégou, H., and Joulin, A. (2021). Training with quantization noise for extreme model compression. In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net.

[Sun et al., 2019] Sun, Z., Deng, Z.-H., Nie, J.-Y., and Tang, J. (2019). RotatE: Knowledge graph embedding by relational rotation in complex space. *CoRR*, arXiv:1902.10197.

[Sundararajan et al., 2017] Sundararajan, M., Taly, A., and Yan, Q. (2017). Axiomatic attribution for deep networks. In *International conference on machine learning*, volume 70 of *Proceedings of Machine Learning Research*, pages 3319–3328. PMLR.

[Süzen et al., 2020] Süzen, A. A., Duman, B., and Şen, B. (2020). Benchmark analysis of jetson tx2, jetson nano and raspberry pi using deep-cnn. In *2nd International Congress on Human-Computer Interaction, Optimization and Robotic Applications (HORA), Ankara, Turkey, June 26-27, 2020*, pages 1–5.

[Szegedy et al., 2017] Szegedy, C., Ioffe, S., Vanhoucke, V., and Alemi, A. A. (2017). Inception-v4, inception-resnet and the impact of residual connections on learning. In *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence, February 4-9, 2017, San Francisco, California, USA*, pages 4278–4284. AAAI Press.

[Szegedy et al., 2016] Szegedy, C., Vanhoucke, V., Ioffe, S., Shlens, J., and Wojna, Z. (2016). Rethinking the inception architecture for computer vision. In *29th IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Las Vegas, NV, USA, June 27-30, 2016*, pages 2818–2826. IEEE Computer Society.

[Taha and Ghosh, 1999] Taha, I. A. and Ghosh, J. (1999). Symbolic interpretation of artificial neural networks. *IEEE Transactions on Knowledge and Data Engineering*, 11(3):448–463.

[Tallarida and Murray, 1987] Tallarida, R. J. and Murray, R. B. (1987). Area under a curve: Trapezoidal and simpson's rules. In *Manual of Pharmacologic Calculations*, pages 77–81. Springer New York.

[Tan, 1997] Tan, A. (1997). Cascade ARTMAP: integrating neural computation and symbolic knowledge processing. *IEEE Transaction on Neural Networks*, 8(2):237–250.

[Tan et al., 2019] Tan, M., Chen, B., Pang, R., Vasudevan, V., Sandler, M., Howard, A., and Le, Q. V. (2019). Mnasnet: Platform-aware neural architecture search for mobile. In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2019, June 16-20, 2019*, pages 2820–2828. Computer Vision Foundation / IEEE.

[Tan and Le, 2019] Tan, M. and Le, Q. V. (2019). Efficientnet: Rethinking model scaling for convolutional neural networks. In *36th International Conference on Machine Learning, ICML 2019*, volume 97 of *Proceedings of Machine Learning Research*, pages 6105–6114. PMLR.

[Tan et al., 2020] Tan, M., Pang, R., and Le, Q. V. (2020). Efficientdet: Scalable and efficient object detection. In *33rd IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), Seattle, WA, USA, June 13-19, 2020*, pages 10778–10787. Computer Vision Foundation / IEEE.

[Tang et al., 2022] Tang, X., Zhu, S., Liang, Y., and Zhang, M. (2022). Rule: Neural-symbolic knowledge graph reasoning with rule embedding. *CoRR*, abs/2210.14905.

[Tay et al., 2021] Tay, Y., Bahri, D., Metzler, D., Juan, D.-C., Zhao, Z., and Zheng, C. (2021). Synthesizer: Rethinking self-attention for transformer models. In *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pages 10183–10192. PMLR.

[Thrun, 1993] Thrun, S. B. (1993). Extracting provably correct rules from artificial neural networks. Technical report, University of Bonn.

[Tickle et al., 1996] Tickle, A. B., Orlowski, M., and Diederich, J. (1996). DEDEC: A methodology for extracting rules from trained artificial neural networks. In *Rules and Networks: Proceedings of the Rule Extraction from Trained Artificial Neural Networks Workshop*, pages 90–102. Neurocomputing Research Centre, Queensland University of Technology.

[Toneva et al., 2019] Toneva, M., Sordoni, A., des Combes, R. T., Trischler, A., Bengio, Y., and Gordon, G. J. (2019). An empirical study of example forgetting during deep neural network learning. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net.

[Torres and Rocco, 2005] Torres, D. E. D. and Rocco, C. M. S. (2005). Extracting trees from trained SVM models using a TREPAN based approach. In *5th International Conference on Hybrid Intelligent Systems (HIS 2005), 6-9 November 2005, Rio de Janeiro, Brazil*, pages 353–360. IEEE Computer Society.

[Towell and Shavlik, 1991] Towell, G. G. and Shavlik, J. W. (1991). Interpretation of artificial neural networks: Mapping knowledge-based neural networks into rules. In *Advances in Neural Information Processing Systems 4, [NIPS Conference, Denver, Colorado, USA, December 2-5, 1991]*, pages 977–984. Morgan Kaufmann.

[Towell and Shavlik, 1993] Towell, G. G. and Shavlik, J. W. (1993). Extracting refined rules from knowledge-based neural networks. *Machine Learning*, 13(1):71–101.

[Towell and Shavlik, 1994] Towell, G. G. and Shavlik, J. W. (1994). Knowledge-based artificial neural networks. *Artificial Intelligence*, 70(1-2):119–165.

[Towell et al., 1990] Towell, G. G., Shavlik, J. W., and Noordewier, M. O. (1990). Refinement of approximate domain theories by knowledge-based neural networks. *Proceedings of the 8th National Conference on Artificial Intelligence*, pages 861–866.

[Tran and Garcez, 2016] Tran, S. N. and Garcez, A. S. d. (2016). Deep logic networks: Inserting and extracting knowledge from deep belief networks. *IEEE Transaction on Neural Networks and Learning Systems*, 29(2):246–258.

[Tresp et al., 1992] Tresp, V., Hollatz, J., and Ahmad, S. (1992). Network structuring and training using rule-based knowledge. In *Advances in Neural Information Processing Systems 5, [NIPS Conference, Denver, Colorado, USA, November 30 - December 3, 1992]*, pages 871–878. Morgan Kaufmann.

[Trindade et al., 2021] Trindade, S., Bittencourt, L. F., and da Fonseca, N. L. S. (2021). Management of Resource at the Network Edge for Federated Learning. *CoRR*, abs/2107.03428.

[Trouillon et al., 2016] Trouillon, T., Welbl, J., Riedel, S., Gaussier, É., and Bouchard, G. (2016). Complex embeddings for simple link prediction. In *Proceedings of the 33rd International Conference on Machine Learning (ICML), New York City, NY, USA, June 19-24, 2016*, volume 48 of *JMLR Workshop and Conference Proceedings*, pages 2071–2080. JMLR.org.

[Tsukimoto, 2000] Tsukimoto, H. (2000). Extracting rules from trained neural networks. *IEEE Transactions on Neural Networks and Learning Systems*, 11(2):377–389.

[Van Benthem and Doets, 2001] Van Benthem, J. and Doets, K. (2001). Higher-order logic. In *Handbook of Philosophical Logic*, pages 189–243. Springer Netherlands, Dordrecht.

[Vasilev et al., 2020] Vasilev, N., Mincheva, Z., and Nikolov, V. (2020). Decision tree extraction using trained neural network. In *Proceedings of the 9th International Conference on Smart Cities and Green ICT Systems, SMARTGREENS 2020, Prague, Czech Republic, May 2-4, 2020*, pages 194–200. SCITEPRESS.

[Velickovic et al., 2018] Velickovic, P., Cucurull, G., Casanova, A., Romero, A., Liò, P., and Bengio, Y. (2018). Graph attention networks. In *Proceedings of the 6th International Conference on Learning Representations (ICLR), Vancouver, BC, Canada, April 30 - May 3, 2018*. OpenReview.net.

[Verma et al., 2020] Verma, S., Dickerson, J. P., and Hines, K. (2020). Counterfactual explanations for machine learning: A review. *CoRR*, abs/2010.10596.

[Vilamala et al., 2023] Vilamala, M. R., Xing, T., Taylor, H., Garcia, L., Srivastava, M., Kaplan, L. M., Preece, A. D., Kimmig, A., and Cerutti, F. (2023). DeepProbCEP: A neuro-symbolic approach for complex event processing in adversarial settings. *Expert Systems with Applications*, 215:119376:1–26.

[von Rueden et al., 2021] von Rueden, L., Mayer, S., Beckh, K., Georgiev, B., Giesselbach, S., Heese, R., Kirsch, B., Walczak, M., Pfrommer, J., Pick, A., Ramamurthy, R., Garcke, J., Bauckhage, C., and Schuecker, J. (2021). Informed machine learning – a taxonomy and survey of integrating prior knowledge into learning systems. *IEEE Transactions on Knowledge and Data Engineering*, 35(1):614–633.

[Wagner and d'Avila Garcez, 2021] Wagner, B. and d'Avila Garcez, A. (2021). Neural-symbolic integration for fairness in AI. In *AAAI-MAKE 2021 – Combining Machine Learning and Knowledge Engineering*, volume 2846 of *CEUR Workshop Proceedings*. CEUR-WS.org.

[Walawalkar et al., 2020] Walawalkar, D., Shen, Z., and Savvides, M. (2020). Online ensemble model compression using knowledge distillation. In *Computer Vision - ECCV 2020 - 16th European Conference, Glasgow, UK, August 23-28, 2020, Proceedings, Part XIX*, volume 12364 of *Lecture Notes in Computer Science*, pages 18–35. Springer.

[Wang et al., 2017] Wang, Q., Mao, Z., Wang, B., and Guo, L. (2017). Knowledge graph embedding: A survey of approaches and applications. *IEEE Transactions on Knowledge and Data Engineering*, 29(12):2724–2743.

[Wang et al., 2015] Wang, Q., Wang, B., and Guo, L. (2015). Knowledge base completion using embeddings and rules. In *Proceedings of the 24th International Joint Conference on Artificial Intelligence (IJCAI), Buenos Aires, Argentina, July 25-31, 2015*, pages 1859–1866. AAAI Press.

[Wang et al., 2019a] Wang, S., Tuor, T., Salonidis, T., Leung, K. K., Makaya, C., He, T., and Chan, K. (2019a). Adaptive Federated Learning in Resource Constrained Edge Computing Systems. *IEEE Journal on Selected Areas in Communications*, 37(6):1205–1221.

[Wang et al., 2020] Wang, S., Wang, Y., Wang, D., Yin, Y., Wang, Y., and Jin, Y. (2020). An improved random forest-based rule extraction method for breast cancer diagnosis. *Applied Soft Computing*, 86.

[Wang et al., 2019b] Wang, Y., Sun, Y., Liu, Z., Sarma, S. E., Bronstein, M. M., and Solomon, J. M. (2019b). Dynamic graph CNN for learning on point clouds. *ACM Transactions on Graphics*, 38(5):146:1–146:12.

[Wang et al., 2014] Wang, Z., Zhang, J., Feng, J., and Chen, Z. (2014). Knowledge graph embedding by translating on hyperplanes. In *Proceedings of the 28th Conference on Artificial*

*Intelligence (AAAI), Québec City, Québec, Canada, July 27 -31, 2014*, pages 1112–1119. AAAI Press.

[Wei et al., 2022] Wei, Z., Wang, Y., Li, J., Liu, Z., Yu, E., Tian, Y., Wang, X., and Chang, Y. (2022). Towards unified representations of knowledge graph and expert rules for machine learning and reasoning. In *Proceedings of the 2nd Conference of the Asia-Pacific Chapter of the Association for Computational Linguistics and the 12th International Joint Conference on Natural Language Processing, AACL/IJCNLP 2022 - Volume 1: Long Papers, Online Only, November 20-23, 2022*, pages 240–253. Association for Computational Linguistics.

[Wei et al., 2015] Wei, Z., Zhao, J., Liu, K., Qi, Z., Sun, Z., and Tian, G. (2015). Large-scale knowledge base completion: Inferring via grounding network sampling over selected instances. In *Proceedings of the 24th International Conference on Information and Knowledge Management (CIKM), Melbourne, VIC, Australia, October 19 - 23, 2015*, pages 1331–1340. ACM.

[Weiner et al., 2022] Weiner, J., Agarwal, N., Schatzberg, D., Yang, L., Wang, H., Sanouillet, B., Sharma, B., Heo, T., Jain, M., Tang, C., and Skarlatos, D. (2022). TMO: transparent memory offloading in datacenters. In *ASPLOS '22: 27th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Lausanne, Switzerland, 28 February 2022 - 4 March 2022*, pages 609–621. ACM.

[Werner et al., 2023] Werner, L., Layaïda, N., Genevès, P., and Chlyah, S. (2023). Knowledge enhanced graph neural networks. *CoRR*, abs/2303.15487.

[West et al., 2022] West, P., Bhagavatula, C., Hessel, J., Hwang, J. D., Jiang, L., Le Bras, R., Lu, X., Welleck, S., and Choi, Y. (2022). Symbolic knowledge distillation: from general language models to commonsense models. In *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL 2022, Seattle, WA, United States, July 10-15, 2022*, pages 4602–4625. Association for Computational Linguistics.

[Wistuba et al., 2019] Wistuba, M., Rawat, A., and Pedapati, T. (2019). A survey on neural architecture search. *CoRR*, abs/1905.01392.

[Wolberg, 1992] Wolberg, W. H. (1992). Breast cancer wisconsin (original) data set. `https://archive.ics.uci.edu/ml/datasets/breast+cancer+wisconsin+%28original%29`.

[Wu, 2019] Wu, B. (2019). Efficient deep neural networks. *CoRR*, abs/1908.08926.

[Wu et al., 2019] Wu, B., Dai, X., Zhang, P., Wang, Y., Sun, F., Wu, Y., Tian, Y., Vajda, P., Jia, Y., and Keutzer, K. (2019). Fbnet: Hardware-aware efficient convnet design via differentiable neural architecture search. In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2019, Long Beach, CA, USA, June 16-20, 2019*, pages 10734–10742. Computer Vision Foundation / IEEE.

[Wu et al., 2018a] Wu, B., Wan, A., Yue, X., Jin, P., Zhao, S., Golmant, N., Gholaminejad, A., Gonzalez, J., and Keutzer, K. (2018a). Shift: A zero FLOP, zero parameter alternative to spatial convolutions. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 9127–9135.

[Wu et al., 2021a] Wu, C. W., Wu, A. C., and Strom, J. (2021a). DeepTune: Robust global optimization of electronic circuit design via neuro-symbolic optimization. In *IEEE International Symposium on Circuits and Systems, ISCAS 2021, Daegu, South Korea, May 22-28, 2021*, pages 1–5. IEEE.

[Wu and Wang, 2022] Wu, H. and Wang, P. (2022). Node Selection Toward Faster Convergence for Federated Learning on Non-IID Data. *IEEE Transactions on Network Science and Engineering*, 9(5):3099–3111.

[Wu et al., 2018b] Wu, S., Li, G., Chen, F., and Shi, L. (2018b). Training and Inference with Integers in Deep Neural Networks. In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*. OpenReview.net.

[Wu et al., 2020] Wu, Z., Nguyen, T.-S., and Ong, D. C. (2020). Structured self-attention weights encode semantics in sentiment analysis. In *Proceedings of the Third Blackbox NLP Workshop on Analyzing and Interpreting Neural Networks for NLP*, pages 255–264. Association for Computational Linguistics.

[Wu et al., 2021b] Wu, Z., Pan, S., Chen, F., Long, G., Zhang, C., and Yu, P. S. (2021b). A comprehensive survey on graph neural networks. *IEEE Transactions on Neural Networks and Learning Systems*, 32(1):4–24.

[Xiao et al., 2015] Xiao, H., Huang, M., Hao, Y., and Zhu, X. (2015). Transa: An adaptive approach for knowledge graph embedding. *CoRR*, abs/1509.05490.

[Xiao et al., 2016a] Xiao, H., Huang, M., and Zhu, X. (2016a). From one point to a manifold: Knowledge graph embedding for precise link prediction. In *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence, IJCAI 2016, New York, NY, USA, 9-15 July 2016*, pages 1315–1321. IJCAI/AAAI Press.

[Xiao et al., 2016b] Xiao, H., Huang, M., and Zhu, X. (2016b). Transg : A generative model for knowledge graph embedding. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics, ACL 2016, August 7-12, 2016, Berlin, Germany, Volume 1: Long Papers*. The Association for Computer Linguistics.

[Xiao et al., 2017] Xiao, H., Rasul, K., and Vollgraf, R. (2017). Fashion-MNIST: a novel image dataset for benchmarking machine learning algorithms. *CoRR*, abs/1708.07747. `http://arxiv.org/abs/1708.07747`.

[Xie et al., 2019a] Xie, S., Kirillov, A., Girshick, R. B., and He, K. (2019a). Exploring randomly wired neural networks for image recognition. In *2019 IEEE/CVF International Conference on Computer Vision, ICCV 2019*, pages 1284–1293, Seoul, South Korea. IEEE.

[Xie et al., 2022] Xie, X., Kersting, K., and Neider, D. (2022). Neuro-symbolic verification of deep neural networks. In *Thirty-First International Joint Conference on Artificial Intelligence, IJCAI 2022*, pages 3622–3628. ijcai.org.

[Xie et al., 2019b] Xie, Y., Xu, Z., Meel, K. S., Kankanhalli, M. S., and Soh, H. (2019b). Embedding symbolic knowledge into deep networks. In *Advances in Neural Information Processing Systems 32 (NeurIPS 2019)*, pages 4235–4245, Vancouver, BC, Canada. Curran Associates, Inc.

[Xie et al., 2021] Xie, Y., Zhou, F., and Soh, H. (2021). Embedding symbolic temporal knowledge into deep sequential models. In *IEEE International Conference on Robotics and Automation, ICRA 2021, Xi'an, China, May 30 - June 5, 2021*, pages 4267–4273. IEEE.

[Xu et al., 2018] Xu, J., Zhang, Z., Friedman, T., Liang, Y., and Van den Broeck, G. (2018). A semantic loss function for deep learning with symbolic knowledge. In *Proceedings of the 35th International Conference on Machine Learning (ICML), Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*, volume 80 of *Proceedings of Machine Learning Research*, pages 5498–5507. PMLR.

[Xu et al., 2017] Xu, P., Shi, S., and Chu, X. (2017). Performance Evaluation of Deep Learning Tools in Docker Containers. In *2017 3rd International Conference on Big Data Computing and Communications (BIGCOM)*, pages 395–403.

[Xu et al., 2022] Xu, Y., Liao, Y., Xu, H., Ma, Z., Wang, L., and Liu, J. (2022). Adaptive control of local updating and model compression for efficient federated learning. *IEEE Transactions on Mobile Computing*.

[Xu et al., 2020] Xu, Y., Xie, L., Zhang, X., Chen, X., Shi, B., Tian, Q., and Xiong, H. (2020). Latency-aware differentiable neural architecture search. *CoRR*, abs/2001.06392.

[Yan et al., 2021] Yan, A., Chen, Z., Zhang, H., Peng, L., Yan, Q., Hassan, M. U., Zhao, C., and Yang, B. (2021). Effective detection of mobile malware behavior based on explainable deep neural network. *Neurocomputing*, 453:482–492.

[Yang et al., 2015] Yang, B., Yih, W., He, X., Gao, J., and Deng, L. (2015). Embedding entities and relations for learning and inference in knowledge bases. In *Proceedings of the 3rd International Conference on Learning Representations (ICLR), San Diego, CA, USA, May 7-9, 2015*.

[Yang and Chaudhuri, 2022] Yang, C. and Chaudhuri, S. (2022). Safe neurosymbolic learning with differentiable symbolic execution. In *The Tenth International Conference on Learning Representations, ICLR 2022, Virtual Event, April 25-29, 2022*. OpenReview.net.

[Yang et al., 2021a] Yang, C., Liu, J., and Shi, C. (2021a). Extract the knowledge of graph neural networks and go beyond it: An effective knowledge distillation framework. In *WWW '21: The Web Conference 2021, Virtual Event / Ljubljana, Slovenia, April 19-23, 2021*, pages 1227–1237. ACM / IW3C2.

[Yang et al., 2010] Yang, Y., Webb, G. I., and Wu, X. (2010). Discretization methods. In *Data Mining and Knowledge Discovery Handbook, 2nd ed.* Springer.

[Yang et al., 2021b] Yang, Z., Chen, M., Saad, W., Hong, C. S., and Shikh-Bahaei, M. (2021b). Energy efficient federated learning over wireless communication networks. *IEEE Trans. Wirel. Commun.*, 20(3):1935–1949.

[Yang et al., 2020] Yang, Z., Wang, Y., Chen, X., Shi, B., Xu, C., Xu, C., Tian, Q., and Xu, C. (2020). CARS: continuous evolution for efficient neural architecture search. In *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR 2020, Seattle, WA, USA, June 13-19, 2020*, pages 1826–1835. IEEE.

[Yazdani et al., 2020] Yazdani, A., Lu, L., Raissi, M., and Karniadakis, G. E. (2020). Systems biology informed deep learning for inferring parameters and hidden dynamics. *PLoS Computational Biology*, 16(11).

[Yedjour, 2021] Yedjour, D. (2021). Application of the genetic algorithm to the rule extraction problem. In *Artificial Intelligence and Renewables Towards an Energy Transition*, pages 604–611, Cham. Springer International Publishing.

[Yedjour and Benyettou, 2018] Yedjour, D. and Benyettou, A. (2018). Symbolic interpretation of artificial neural networks based on multiobjective genetic algorithms and association rules mining. *Applied Soft Computing*, 72:177–188.

[Yi et al., 2015] Yi, S., Li, C., and Li, Q. (2015). A Survey of Fog Computing: Concepts, Applications and Issues. In *Proceedings of the 2015 Workshop on Mobile Big Data*, Mobidata '15, pages 37–42. Association for Computing Machinery.

[Ying et al., 2019] Ying, C., Klein, A., Christiansen, E., Real, E., Murphy, K., and Hutter, F. (2019). NAS-bench-101: Towards reproducible neural architecture search. In *36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 7105–7114. PMLR.

[Yosinski et al., 2015] Yosinski, J., Clune, J., Nguyen, A. M., Fuchs, T. J., and Lipson, H. (2015). Understanding neural networks through deep visualization. *CoRR*, abs/1506.06579.

[You et al., 2018] You, J., Ying, R., Ren, X., Hamilton, W. L., and Leskovec, J. (2018). Graphrnn: Generating realistic graphs with deep auto-regressive models. In *35th International Conference on Machine Learning, ICML 2018*, volume 80 of *Proceedings of Machine Learning Research*, pages 5694–5703. PMLR.

[Yu et al., 2018] Yu, B., Yin, H., and Zhu, Z. (2018). Spatio-temporal graph convolutional networks: A deep learning framework for traffic forecasting. In *27th International Joint Conference on Artificial Intelligence, IJCAI 2018*, pages 3634–3640. ijcai.org.

[Yu et al., 2022] Yu, D., Yang, B., Wei, Q., Li, A., and Pan, S. (2022). A probabilistic graphical model based on neural-symbolic reasoning for visual relationship detection. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR 2022, New Orleans, LA, USA, June 18-24, 2022*, pages 10599–10608. IEEE.

[Yu and Liu, 2021] Yu, J. and Liu, G. (2021). Extracting and inserting knowledge into stacked denoising auto-encoders. *Neural Networks*, 137:31–42.

[Yu et al., 2020] Yu, K., Sciuto, C., Jaggi, M., Musat, C., and Salzmann, M. (2020). Evaluating the search phase of neural architecture search. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net.

[Yuan and Zhuang, 1996] Yuan, Y. and Zhuang, H. (1996). A genetic algorithm for generating fuzzy classification rules. *Fuzzy Sets and Systems*, 84(1):1–19.

[Zaw and Hong, 2021] Zaw, C. W. and Hong, C. S. (2021). A decentralized game theoretic approach for energy-aware resource management in federated learning. In *IEEE International Conference on Big Data and Smart Computing, BigComp 2021, Jeju Island, South Korea, January 17-20, 2021*, pages 133–136. IEEE.

[Zhang et al., 2022] Zhang, C., Cui, L., Yu, S., and Yu, J. J. Q. (2022). A communication-efficient federated learning scheme for iot-based traffic forecasting. *IEEE Internet Things J.*, 9(14):11918–11931.

[Zhang et al., 2021a] Zhang, C., Xie, Y., Bai, H., Yu, B., Li, W., and Gao, Y. (2021a). A survey on federated learning. *Knowledge Based Systems*, 216:106775.

[Zhang and Li, 2020] Zhang, J. and Li, C. (2020). Adversarial examples: Opportunities and challenges. *IEEE Transactions on Neural Networks and Learning Systems*, 31(7):2578–2593.

[Zhang et al., 2019a] Zhang, M., Jiang, S., Cui, Z., Garnett, R., and Chen, Y. (2019a). D-VAE: A variational autoencoder for directed acyclic graphs. In *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems (NeurIPS)*, pages 1586–1598.

[Zhang et al., 2021b] Zhang, Q., Wang, L., Yu, S., Wang, S., Wang, Y., Jiang, J., and Lim, E. (2021b). NOAHQA: Numerical reasoning with interpretable graph question answering dataset. In *Findings of the Association for Computational Linguistics: EMNLP 2021*, pages 4147–4161. ACL.

[Zhang et al., 2017] Zhang, W., Zhang, Z., and Chao, H. (2017). Cooperative Fog Computing for Dealing with Big Data in the Internet of Vehicles: Architecture and Hierarchical Resource Management. *IEEE Communications Magazine*, 55(12):60–67.

[Zhang et al., 2005] Zhang, Y., Su, H., Jia, T., and Chu, J. (2005). Rule extraction from trained support vector machines. In *Advances in Knowledge Discovery and Data Mining, 9th Pacific-Asia Conference, PAKDD 2005, Hanoi, Vietnam, May 18-20, 2005, Proceedings*, volume 3518 of *Lecture Notes in Computer Science*, pages 61–70. Springer.

[Zhang et al., 2020] Zhang, Z., Cai, J., Zhang, Y., and Wang, J. (2020). Learning hierarchy-aware knowledge graph embeddings for link prediction. In *Proceedings of the 34th Conference on Artificial Intelligence (AAAI), The 32nd Innovative Applications of Artificial Intelligence Conference (IAAI), The 10th AAAI Symposium on Educational Advances in Artificial Intelligence (EAAI), New York, NY, USA, February 7-12, 2020*, pages 3065–3072. AAAI Press.

[Zhang et al., 2019b] Zhang, Z., Han, X., Liu, Z., Jiang, X., Sun, M., and Liu, Q. (2019b). ERNIE: enhanced language representation with informative entities. In *Proceedings of the 57th Conference of the Association for Computational Linguistics (ACL), Florence, Italy, July 28 - August 2, 2019*, pages 1441–1451. Association for Computational Linguistics.

[Zhang and Sabuncu, 2018] Zhang, Z. and Sabuncu, M. (2018). Generalized cross entropy loss for training deep neural networks with noisy labels. In *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc.

[Zhao et al., 2016] Zhao, X., Zhao, L., and Liang, K. (2016). An Energy Consumption Oriented Offloading Algorithm for Fog Computing. In *Quality, Reliability, Security and Robustness in Heterogeneous Networks - 12th International Conference, QShine 2016, Seoul, Korea, July 7-8, 2016, Proceedings*, volume 199 of *Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering*, pages 293–301. Springer.

[Zhong et al., 2022] Zhong, Z., Xu, M., Rodriguez, M. A., Xu, C., and Buyya, R. (2022). Machine Learning-Based Orchestration of Containers: A Taxonomy and Future Directions. *ACM Computing Surveys*, 54(10s).

[Zhou et al., 2018] Zhou, H., Young, T., Huang, M., Zhao, H., Xu, J., and Zhu, X. (2018). Commonsense knowledge aware conversation generation with graph attention. In *Proceedings of the 27th International Joint Conference on Artificial Intelligence (IJCAI), Stockholm, Sweden, July 13-19, 2018*, pages 4623–4629. ijcai.org.

[Zhou et al., 2020] Zhou, J., Cui, G., Hu, S., Zhang, Z., Yang, C., Liu, Z., Wang, L., Li, C., and Sun, M. (2020). Graph neural networks: A review of methods and applications. *AI Open*, 1:57–81.

[Zhou et al., 2000] Zhou, Z., Chen, S., and Chen, Z. (2000). A statistics based approach for extracting priority rules from trained neural networks. In *Proceedings of the IEEE-INNS-ENNS International Joint Conference on Neural Networks, IJCNN 2000, Neural Computing: New Challenges and Perspectives for the New Millennium, Como, Italy, July 24-27, 2000, Volume 3*, pages 401–406. IEEE Computer Society.

[Zhou et al., 2003] Zhou, Z., Jiang, Y., and Chen, S. (2003). Extracting symbolic rules from trained neural network ensembles. *AI Communications*, 16(1):3–15.

[Zhu et al., 2022] Zhu, Z., Galkin, M., Zhang, Z., and Tang, J. (2022). Neural-symbolic models for logical queries on knowledge graphs. In *International Conference on Machine Learning, ICML 2022, 17-23 July 2022, Baltimore, Maryland, USA*, volume 162 of *Proceedings of Machine Learning Research*, pages 27454–27478. PMLR.

[Zilke et al., 2016] Zilke, J. R., Mencía, E. L., and Janssen, F. (2016). DeepRED – Rule extraction from deep neural networks. In *Discovery Science*, volume 9956 of *LNCS*, Bari, Italy. Springer.

[Zoph and Le, 2017] Zoph, B. and Le, Q. V. (2017). Neural architecture search with reinforcement learning. In *5th International Conference on Learning Representations (ICLR 2017)*, Toulon, France.