ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Dottorato di Ricerca in
Computer Science and Engineering

Ciclo 38

Settore Concorsuale: 09/H1 – Sistemi di Elaborazione delle Informazioni

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

Symbolic Knowledge Injection & Extraction for Autonomous Learning

Presentata da:
Matteo Magnini

Coordinatrice Dottorato:
Prof.ssa Paola Salomoni

Supervisore:
Prof. Andrea Omicini

Co-supervisore:
Prof. Enrico Denti

Esame finale anno 2026

Abstract

Artificial intelligence (AI) represents one of humanity's most transformative technological advancements, with roots in the mid-20th century and rapidly growing societal impact. Over the past decade, the field of neuro-symbolic (NeSy) AI – which seeks to integrate *symbolic* and *sub-symbolic* approaches – has gained increasing prominence as a promising paradigm for building intelligent systems. By combining the structured reasoning capabilities of symbolic AI with the adaptability and scalability of sub-symbolic methods, NeSy AI aims to overcome the inherent limitations of each paradigm, paving the way for more robust and interpretable systems.

Within this broad and multifaceted landscape, this thesis focuses on two complementary subfields of NeSy AI: symbolic knowledge injection (SKI) and symbolic knowledge extraction (SKE). These two areas, often seen as *two sides of the same coin*, provide foundational mechanisms for the development of hybrid intelligent systems. SKI enables the injection of structured domain knowledge into sub-symbolic models, guiding their behavior, improving generalization, and enhancing trustworthiness. SKE, conversely, facilitates the extraction of symbolic, human-understandable information from these models, bridging the gap between opaque internal representations and interpretability. Together, SKI and SKE form a synergistic loop: while SKI influences learning by introducing external structure, SKE enables inspection, verification, and refinement of that structure, contributing to the creation of systems that are explainable, reliable, and adaptable to real-world complexity.

This thesis explores the challenges and opportunities of NeSy AI, with a particular focus on SKI, SKE, and their role in the engineering of intelligent systems. After an overview of the background and state of the art, we identify key challenges in integrating symbolic and sub-symbolic paradigms. We then present original contributions in the form of methodologies, algorithms, and tools designed to advance the capabilities of NeSy systems.

The advent of large language models (LLMs) has further transformed the landscape of AI, offering unprecedented capabilities for understanding and generating natural language. This thesis investigates how LLMs can augment SKI and SKE,

providing new avenues for designing systems that learn and adapt autonomously. By leveraging these models, we demonstrate how hybrid approaches can address complex challenges – such as fairness and decision-making in healthcare – while ensuring interpretability and alignment with ethical principles.

Finally, we discuss how these advancements align with the broader vision of NeSy, while also contributing to the specific goal of this thesis: enabling the development of systems capable of fully autonomous learning. These systems integrate the structured reasoning of symbolic AI, the adaptability of sub-symbolic models, and the transformative potential of LLMs, opening pathways toward more versatile and intelligent applications. Such progress not only advances the field but also brings us closer to the distant horizon of general AI, where machines can learn, reason, and adapt autonomously.

Keywords – *artificial intelligence, neuro-symbolic AI, symbolic knowledge injection, symbolic knowledge extraction, large language models, intelligent systems, autonomous learning.*

"The work of each individual contributes to a totality, and so becomes an undying part of the totality. That totality of human lives – past and present and to come – forms a tapestry that has been in existence now for many tens of thousands of years and has been growing more elaborate and, on the whole, more beautiful in all that time. [...] An individual life is one thread in the tapestry and what is one thread compared to the whole? Daneel, keep your mind fixed firmly on the tapestry and do not let the trailing off of a single thread affect you."

—*Isaac Asimov,* Robots and Empire

Acknowledgements

First and foremost, I would like to thank my parents, Barbara and Marcello, for their unconditional love and support throughout my life. I thank my beloved wife, Aqila, for the never ending encouragement and for always pushing me to aim at greater goals. I am grateful to my entire family for their help during these years.

I want to thank my supervisor, Andrea, for his wise advice, continuous support, and for giving me the freedom to explore my ideas. I thank my close collaborator, and de-facto a co-supervisor, Giovanni, for his invaluable insights, constructive criticism, and for always being available to discuss ideas and challenges. I thank all the other researchers I collaborated with during these years, including Gianluca, Sara, Roberta, Andrea A., Andrea R., Federico and many others. I also want to mention – even if I do not explicitly list all the names – the researchers of the Lab 4.0 for the shared experiences and the fun times we had together. Finally, I thank Ana for being my supervisor during my visiting at the University of Oslo, and for her precious guidance and support.

Life is also outside Academia, actually, life is mostly outside Academia. At the very end of this journey I got married, and I want to thank all my friends who made this day unforgettable and help me in the organization. The wedding and the thesis together took most of my energy, and by far 2025 has been the most intense year of my life. Aside from the wedding, these years have been full of emotions, trips, cool experiences and tough moments. Eliciting everyone would be infeasible, but I want to mention a few of them. Alberto, Eugenio, Marco, Enrico, Michele, Federico, the researchers at the department of informatics of the Oslo university, the students and the people who frequent the pub nearby Cesena campus.

This paragraph has been added after the reviewers' comments. Research would not be possible without the peer review process, and it is good to acknowledge the effort of those who dedicate their time to review scientific works. At the same time it is also good to point out when a review is done just as a formality, without real care for the work being reviewed. I particularly thank the first reviewer for its time and for the detailed and constructive feedback, which helped me improve the quality of this thesis. Unfortunately, I cannot say the same for the second reviewer,

whose comments were very superficial and not much helpful. To be honest, it is a very rushed – and out of time – review, that me myself I would not have written even for a workshop paper. Is this a case of *Gollum effect*[1]?

Last but not least, I want give a special thanks to somebody that usually does not appear in the acknowledgements of a PhD thesis. In fact, since the very beginning of my academic career right after the Master's degree, a major event has shaken the world, and it is still going on: the invasion of Ukraine by fascist Russia. The free people of Ukraine are fighting every day to defend not only their homeland, but the entire Europe and also the values of freedom and democracy that we all cherish, and without which scientific research itself would not be possible as we know it.

Slava Ukraini!
Heroiam slava!

[1] `https://doi.org/10.1038/d41586-025-01588-w`

Contents

List of Figures

List of Tables

List of Listings

Glossary

$\mathcal{ALC}$ attributive concept language with complements.
$\mathcal{EL}$ existential language.

ABM agent-based model.
ABox assertional axiom.
ACCUEE Agencia Canaria de Calidad Universitaria y Evaluación Educativa.
AGI artificial general intelligence.
AI artificial intelligence.
AOP agent-oriented programming.
API application programming interface.
AST abstract syntax tree.
AutoML automated machine learning.

BCW breast cancer Wisconsin.
BERT bidirectional encoder representations from transformers.
BMI body mass index.
BWP blocks world problem.

CI census income.
CL computational logic.
CNN convolutional neural network.
CoL cooperative learning.
CoTL cooperative transfer learning.
CPS cyber-physical system.

DAG directed acyclic graph.
DI disparate impact.
DL description logic.
DNN deep neural network.
DP demographic parity.
DPP disparate predictive parity.
DT decision tree.
DTE decision tree ensambles.

EI exon-intron.
EO equalized odds.
EQ equal opportunity.
EXTRAAMAS EXplainable, Trustworthy, and Responsible AI and Multi-Agent Systems.

FairBayes-DPP fair bayes-optimal classifiers under predictive parity.
FaUCI fairness under constraints injection.
FLOP floating point operations per second.
FN false negative.
FNNC fairness through neural networks.
FOL first-order logic.
FP false positive.

GDI generalized disparate impact.
GDP generalized demographic parity.
GDPR General Data Protection Regulation.
GEO generalized equalized odds.
GNN graph neural network.
GRU gated recurrent unit.

HL Horn Logic.

IE intron-exon.
ILP inductive logic programming.

KB knowledge base.
KBANN knowledge-based artificial neural network.
KDE kernel density estimation.
KG knowledge graph.
KILL knowledge injection via lambda layers.
KINS knowledge injection via network structuring.
KNN k-nearest neighbor.
KR knowledge representation.

LeakyReLU leaky rectified linear unit.
LLM large language model.
LLM4KB large language model for knowledge bases.
LR logistic regression.
LSTM long short term memory.
LTN logic tensor network.

MAC multiplication addition computation.

MAR missing at random.
MARL multi-agent reinforcement learning.
MAS multi-agent system.
MCAR missing completely at random.
MCC Matthews correlation coefficient.
MDP Markov decision process.
ML machine learning.
MLP multilayer perceptron.
MNAR missing not at random.
MoE mixture of experts.
MTL multi-task learning.

NeSy neuro-symbolic.
NFL no free lunch.
NLG natural language generation.
NLP natural language processing.
NN neural network.
NSIF neuro-symbolic integration for fairness.

OWL web ontology language.

PAC probably approximately correct.
ParEGO pareto efficient global optimisation.
PCA principal component analysis.
PEBLS parallel exemplar-based learning system.
PHDS poker hand dataset.
PI prejudice index.
PID Pima Indians diabetes.
PP predictive parity.
PRR prejudice remover regularizer.
PS pervasive system.
PSJGS primate splice-junction gene sequence.
PSyKE platform for symbolic knowledge extraction.
PSyKI platform for symbolic knowledge injection.

QoS quality of service.

RAG retrieval-augmented generation.
ReLU rectified linear unit.
ReST representational state transfer.
RF random forest.
RL reinforcement learning.

RNN recurrent neural network.
RS recommender system.

SGD stocastic gradient descent.
SKE symbolic knowledge extraction.
SKI symbolic knowledge injection.
SLR systematic literature review.
STS socio-technical system.
SVM support vector machine.
SWOT strengths, weaknesses, opportunities and threats.

TAI trustworthy artificial intelligence.
TBox terminological axiom.
TF-IDF term frequency-inverse document frequency.
TL transfer learning.
TN true negative.
TP true positive.
TPR true positive rate.
TSP traveling salesman problem.

UI user interface.
UX user experience.

WDI weighted disparate impact.
WDP weighted demographic parity.
WEO weighted equalized odds.
WEQ weighted equal opportunity.

XAI explainable artificial intelligence.
XMPP extensible messaging and presence protocol.

Chapter 1

Introduction

Contents

1.1 Research background and context

Through the course of history, humanity has experienced several socio-technological revolutions that have changed the way we live. From the first industrial revolution that initiated the automation of manual labor, to the world-wide spread of computers that started the automation of processes and decision-making, we are now witnessing the artificial intelligence (AI) revolution. The advent of AI has already successfully automated cognitive tasks, and it is expected to go further by reaching – and possibly surpassing – *human-level intelligence*. AI is not a recent invention, it has been around since the 1950s. The reasons why only now (in the last decade to be more precise) AI has become ubiquitous are the presence of crucial elements that were missing in the past. Thanks to *(i)* the enormous amount of *data*, *(ii)* the improvement of *memory* and *computational power* – that still follows the Moore's law –, and *(iii)* the affordability of huge quantity of *energy*, AI finally flourished again.

The first kind of AI that was developed is *symbolic*. Symbolic means that there are *symbols* with specific *meanings* that are manipulated by algorithms. Symbolic AI follows the *deductive* process of reasoning, where the system starts from a set of axioms and applies rules. These kinds of AI programs are pretty effective in well-defined domains where there are clear rules that always hold, e.g., board games, traveling salesman problem (TSP), blocks world problem (BWP), etc. *Sub-symbolic* AI is based on the *inductive* process of reasoning. Conversely to symbolic AI, sub-symbolic AI does not rely on symbols that have meanings for humans, but on data *patterns*. Programs that uses sub-symbolic AI to solve a certain task are said to perform machine learning (ML), because a model needs to first learn from examples before being able to generalize to unseen data. Sub-symbolic models like neural networks (NNs) can reach *super-human performance* in pre-defined tasks like image recognition, natural language processing, etc., but they require a huge amount of data and hardware resources to be trained.

The natural evolution in AI research is to use both symbolic and sub-symbolic approaches together in order to increase the performance and face more challenging tasks. This is the idea behind *neuro-symbolic (NeSy) AI*, where the deductive reasoning of symbolic AI is combined with the inductive learning of sub-symbolic models, especially NNs. This branch of AI is relatively young; the first works that combined logic rules within a NN date back to the 90s [TSN90; TS94]. The last past years have been quite prolific both in the design of new NeSy techniques and in the development of intelligent systems that use them [Cia+24].

Finally, the advent of large language models (LLMs) has further transformed the landscape of AI, offering unprecedented capabilities for natural language (and also multimodal data) generation. LLMs are huge NN models up to *hundreds of billions* of parameters that are trained on a large corpus of text data. Despite the outstanding performance that LLMs have achieved in many tasks, their output is just a probability distribution over the vocabulary, therefore it is subject to errors (e.g., *hallucinations*) and biases (e.g., from training data, from prompt engineering). LLMs are still a great resource for NeSy AI because of their performance, versatility and customizability. Ultimately, the rapid progress in NeSy AI and the dazzling evolution of LLMs are significantly changing our world, leading to more and more intelligent systems, and possibly to the advent of the *singularity* [Sha15].

1.2 Overview and contributions

Engineering AI systems with characteristics approaching (super-)human intelligence remains an open and multifaceted challenge. Humans exhibit a wide range of cognitive capabilities: they can perform *deductive* and *inductive reasoning*, *plan*, *adapt* to changing conditions, *collaborate* with others, *self-organize*, and *acquire new knowledge* autonomously. Each of these skills contributes to what we broadly refer to as intelligence. Hence, an intelligent AI system – particularly one aiming toward artificial general intelligence (AGI) – must exhibit some of these abilities, with the capacity to learn autonomously being among the most fundamental.

To advance toward this long-term goal, we adopt a strategy based on the progressive decomposition of the overarching challenge into concrete, tractable subproblems. These include acquiring and applying domain knowledge, reasoning under uncertainty, adapting to new tasks, and ensuring interpretability and trustworthiness, among others. Rather than attempting to solve AGI in a monolithic way, we aim to address specific foundational problems that are critical building blocks for more general intelligence.

A key insight guiding this thesis is that many aspects of intelligent behavior rely on the interplay between two complementary paradigms: symbolic and sub-symbolic AI. Sub-symbolic models – such as those developed through ML, including NNs – excel at learning from raw data and generalizing inductively. Symbolic approaches, in contrast, are grounded in explicit, human-readable structures such as logic rules or ontologies, and support deductive reasoning. Humans seamlessly integrate both: they learn from examples and experience, but also reason with abstract, structured knowledge.

Bridging these two paradigms is the central objective of NeSy AI. In this context, two processes are particularly crucial: symbolic knowledge injection (SKI) and symbolic knowledge extraction (SKE). SKI refers to the *injection* of symbolic knowledge into sub-symbolic models, enabling them to benefit from prior domain expertise, improve generalization, enforce constraints, and operate more reliably under limited data regimes. Conversely, SKE is the process of *extracting* symbolic knowledge from trained sub-symbolic models, thus making their learned internal representations accessible, inspectable, and reusable in downstream reasoning

tasks. Together, SKI and SKE form the foundational pillars of advanced NeSy systems, enabling a virtuous cycle in which knowledge can flow in both directions fostering transparency, adaptability, and autonomy.

This thesis investigates both theoretical and practical aspects of SKI and SKE, proposing new methodologies, tools, and experimental frameworks to extend their applicability. By exploring how these techniques can be embedded in real-world AI systems, including in high-stakes domains such as healthcare, we aim to contribute toward the broader objective of constructing intelligent systems that learn and reason in a human-like, autonomous, and interpretable manner.

Research questions

RQ0 *Are SKI and SKE relevant in the context of modern AI?*
The increasing complexity and opacity of sub-symbolic models raise urgent needs for systems that are not only accurate, but also interpretable, robust, and adaptable. SKI and SKE address these needs by enabling, respectively, the integration of prior knowledge into learning systems and the extraction of insights from them. This research investigates the relevance and impact of these techniques in real-world scenarios, and evaluates how they contribute to broader AI goals such as trustworthiness, efficiency, and autonomy.

RQ1 *What are the characteristics of SKI and SKE techniques?*
There are different ways to perform SKI and SKE, depending on multiple dimensions. From these dimensions – such as the kind of supported sub-symbolic models, the formalism of the symbolic knowledge, the ways to inject/extract the knowledge, etc. – it is possible to refine the main research question into more detailed sub-questions. Ultimately, from the answers to these research questions it would be possible to define a comprehensive taxonomy of SKI and SKE techniques.

RQ2 *How can the effects of SKI and SKE be measured?*
Accuracy and other popular metrics in ML are not the only ones that should be considered when evaluating SKI and SKE techniques. There are many other aspects that a scientist or the final user of the technology wants to know. For instance, how robust is the model with injected knowledge to

data degradation, how well the extracted knowledge aligns with the actual behavior of the model, whether it is possible to reduce the size of the model by injecting knowledge without losing performance, and so on.

RQ3 *When and where should SKI and SKE be used?*
Traditionally, SKE originates from the context of explainable artificial intelligence (XAI), where the objective is to provide a human-understandable explanation of the model's behavior. SKI, on the other hand, was introduced primarily to improve model performance. However, both techniques have many other possible applications that deserve systematic investigation.

RQ4 *How to design and develop NeSy AI systems that leverage SKI and SKE?*
The use of SKI and SKE enables a variety of new applications and research directions. These new possibilities must be explored taking into account all the consequences and implications of the use of these techniques.

Contributions

The thesis mainly contributes to the field of NeSy AI and software engineering. In particular, it focuses on SKI and SKE methods and on the development of AI systems that leverage them. The *fil rouge* that binds all the contributions is the goal to design and develop intelligent systems, ultimately with capabilities of *autonomous learning*. The contributions are manifold, and they cover different aspects of NeSy AI including: SKI and SKE, software engineering, and social-technical systems. In this thesis, we present the following contributions:

(i) **SKI and SKE**

- *(1)* we systematically collect and organise into a taxonomy SKI and SKE methods and technologies (RQ0 and RQ1);
- *(2)* we design, implement and validate new SKI and SKE methods (RQ1 and RQ3);
- *(3)* we define new metrics to evaluate the performance of SKI techniques (RQ2 and RQ3).

(ii) **Software engineering**

(4) we design and develop software libraries to support the development and integration of SKI and SKE methods in AI systems (RQ4);

(5) we design and develop NeSy AI systems that leverage SKI and SKE techniques in real-world scenarios (RQ0, RQ3 and RQ4);

(iii) **Social-technical systems**

(6) we investigate how SKI techniques can be used to mitigate the risk of bias in AI systems (RQ0, RQ3 and RQ4);

1.3 Structure of the thesis

This thesis is structured as follows.

Part I gives the background necessary to understand all aspects of the research. In Chapter 2 where we introduce the broad topic of *intelligence*, starting from its declinations in living beings and then going deep into machines. A considerable part of the chapter is dedicated to *symbolic* and *sub-symbolic* AI (Sections 2.2.1 and 2.2.2), which is one of the main focus of this thesis. Chapter 3 follows with a presentation of NeSy AI, with particular focus to SKI and SKE by providing a comprehensive taxonomy of the existing methods (Sections 3.1 and 3.2).

Part II presents one of the main contributions of the thesis. Chapter 4 introduces new SKI methods that we designed and implemented. Chapter 5 presents platform for symbolic knowledge injection (PSyKI), a software library that we developed to support the design and development of NeSy AI systems that leverage SKI techniques. Chapter 6 investigates how SKI techniques can be used to mitigate bias in AI systems, thus contributing to the development of trustworthy artificial intelligence (TAI).

Part III presents NeSy AI systems that leverage SKI and SKE techniques. Chapter 7 presents three applications of NeSy AI systems that we designed and developed. In Chapter 8 we discuss how SKI and SKE techniques can be used to design and develop AI systems with capabilities of *autonomous learning*, thus contributing to the long-term goal of AGI. Finally, in Chapter 9 we summarise the main findings of the research, discuss its limitations, and outline directions for future work.

Part I

Background

Chapter 2

Artificial intelligent systems

What is *intelligence*?
What is *artificial intelligence*?
What are the characteristics of an *intelligent system*?
– **RQ0**

Contents

In this chapter, we introduce the concept of *intelligence* providing one of the possible definition (Section 2.1) starting from the sort of intelligence in animals and humans (Section 2.1.1). We then introduce intelligence in computer science with insights on reasoning, learning and agency (Section 2.1.2). Then, we focus on

the core background of this thesis: AI (Section 2.2) by providing all the necessary notions the reader needs to understand the rest of the thesis. Finally, we discuss the impact and implications of AI in modern society (Section 2.3).

2.1 What is intelligence?

Intelligence is a concept that encompasses a wide range of abilities and characteristics of single *individuals* or *groups*. From an *evolutionary* perspective, intelligence characterises some animal species – and organisms belonging to other biological kingdoms – from simple forms of life. In the history of our planet, carbon-based life evolved from unicellular organisms to multicellular organisms, and ultimately to complex organisms with specialized cells and tissues. Some of these organisms (e.g., insects) developed skills – such as *navigation*, *communication*, *self-organisation*, *adaptation*, and so on – that *are perceived* as intelligent abilities. More evolved individuals (e.g., mammals) developed even more complex forms of intelligence, such as *planning*, *reasoning*, *learning*, etc.

In much more recent history, the carbon-based life was not the only one to "manifest" intelligent behaviours. With the invention of the computer, also machines can perform tasks that we consider intelligent. To distinguish between the intelligence of living beings and that of machines, we refer to the former as *natural intelligence* and to the latter as *AI*.

Intelligence originally emerged as a result of the evolutionary process and from the interaction of organisms with each others and their environment. Later, we built machines that can perform tasks that require some sort of intelligent ability. However, it is us – as humans – *to attribute* the label of intelligent to someone or something and not to others. Indeed, it is said that *"intelligence is in the eye of the beholder"*. In this sense, we can say that intelligence is not an absolute concept, but it should be considered under a relative perspective.

Giving a single rigorous definition of intelligence is not a trivial task. Furthermore, there is not one single shape of intelligence, but it can be declined in many different ways (e.g., *emotional intelligence*, *social intelligence*, *spatial intelligence*, etc.). In this thesis we do not want to stay strictly dependant to a specific definition of intelligence, nevertheless it would be useful to provide a one. We give a broad

definition to help the reader to get more familiar with recurrent concepts that will appear later. We choose to adopt a definition inspired by a notorious satirical essay by Cipolla [Cip21]. In *"The basic laws of human stupidity"*, the author classifies *individuals* into four categories based on the *result* of their *actions*:

- **Stupid** → losses for others and for themselves;
- **Helpless** → benefits for others and losses for themselves;
- **Bandit** → losses for others and benefits for themselves;
- **Intelligent** → benefits for others and for themselves.

Now, what is a *loss* or a *benefit*? We can see losses and benefits as the failure or success in – fully or partially – achieving a certain *goal*. With this respect we can state that:

Definition 2.1 (Intelligence) *an individual (or a group) has an intelligent behaviour if it is able to* ***perform actions*** *that lead to the* ***achievement*** *of a given* ***goal****.*

Definition 2.1 intentionally adopts a broad notion of intelligent behaviour. Under this formulation, the ability to select actions that achieve a goal may also characterize simple reactive or feedback-based systems, as well as biological processes such as homeostasis. For this reason, the definition should not be interpreted as a comprehensive or exclusive account of intelligence. Rather, it serves as an operational definition, sufficient to delimit the class of systems considered in this work, namely artificial agents whose behaviour is guided by internal representations and decision mechanisms.

Definition 2.1 is also similar to the definitions of intelligence collected in the work of Legg and Hutter [LH07], such as the definition *"The capacity to learn or to profit by experience"* [Ste00]. Broadly speaking, a common feature of these definitions is that intelligence is seen as a property of an individual *interacting* with an external *environment*, problem, or situation. This interaction is fundamental to practically all proposed definitions of intelligence. Another shared characteristic is the relationship between intelligence and the ability to succeed or *profit* in achieving *objectives*. The notion of success or profit implies the existence of a specific *goal*, although the exact nature of the goal may vary across individuals

or contexts. What matters is the capacity of the individual to carefully select actions that lead to the accomplishment of these goals. The greater this capacity to succeed across a variety of goals, the greater the individual's intelligence. This perspective aligns with the idea that intelligence is not a fixed attribute but rather a dynamic ability to adapt and perform effectively in diverse scenarios.

2.1.1 Natural intelligence

In order to survive in their environment, animals – but also other organisms of different kingdoms such as plants or fungi – have developed a set of abilities and some of them are perceived as intelligent. For the sake of simplicity, we will consider just to animals, but many of the concepts we will talk about can be extended to other organisms. Animals have a *body*, and they are *situated* in the physical world. Through sensory organs, they can *perceive* the environment and with muscles they can *interact* with it. They are *autonomous* individuals, i.e., they are able to perform actions without the need of an external controller.

All animals are able to keep themselves alive (*self-sufficiency*) until natural death or an accident occurs. In addition to self-sufficiency, animals contribute to the survival of their species generating offspring. To do so, they need to navigate the environment, find food, avoid predators, reproduce, and so on.

Humans have developed two main characteristics about intelligence that are no match for any other animals: *reasoning* and *learning*.

Reasoning Logical reasoning, or simply reasoning, is a process of drawing conclusions from premises. There exists three main ways of reasoning:

- **Deductive reasoning** → it is a top-down approach that starts from a general statement and derives specific conclusions. For example, if we know that *all humans are mortal* and *Socrates is a human*, we can conclude that *Socrates is mortal*. Deductive reasoning can be compared to what Kahneman calls *System 2* in his book *Thinking, Fast and Slow* [Kah11]. Kahneman describes the second system as *slow thinking*, which is deliberate, effortful, logical and more rational.

- **Inductive reasoning** → it is a bottom-up approach that starts from specific observations and derives general conclusions. For example, if we observe that *the sun rises every day*, we can conclude that *the sun will rise tomorrow*. Inductive reasoning can be mapped in the *System 1* – i.e., *fast thinking* – of Kahneman, which is automatic, effortless, intuitive and emotional. However, this mapping should be interpreted as partial: while some forms of inductive reasoning may rely on rapid, intuitive mechanisms, *System 1* also includes innate or pre-attentive processes that are not the result of learning, and more complex inductive inferences may involve deliberative components typically associated with *System 2*.
- **Abductive reasoning** → it is a form of reasoning that starts from an observation and seeks the simplest and most likely explanation (i.e., *Occam's razor*). For example, if we observe that *the grass is wet*, we can conclude that *it rained last night*. This kind of reasoning is the same that is adopted by detectives to solve crimes. Abduction also requires the use of statistics and probabilities, and therefore it deals with uncertainty (*"Once you eliminate the impossible, whatever remains, no matter how improbable, must be the truth"—Arthur Conan Doyle*[1]).

Learning Human beings can perform a vast array of tasks, and their ability to learn new ones is a cornerstone of their intelligence. Learning is a process that allows individuals to *model reality*, adapt to their environment, and share knowledge with others. Broadly speaking, humans acquire new skills and knowledge in two primary ways: by generalising experience – e.g., via *inductive* reasoning –, or by *deductively* inferring new knowledge from what they already hold or can obtain from others—e.g., through direct communication (talking) or indirect communication (reading) [CO94]. In the former case, novel knowledge is formed in the learner's mind by interacting with the environment and interpreting sensory input. In the latter case, knowledge is represented symbolically (e.g., through words, gestures, or diagrams), enabling communication and the transfer of meaning between individuals. This symbolic representation is crucial for sharing complex ideas and building upon the collective knowledge of a society.

[1] a person must be aware of all the possibilities in order to not fall into a deductive error.

Animals, too, exhibit learning abilities, though the mechanisms and extent vary widely across species. For instance, many animals rely on experience to adapt to their environment, such as learning to navigate, find food, or avoid predators. Some species, like primates and certain birds, demonstrate the ability to use symbols or tools, which suggests a rudimentary form of modeling reality and reasoning about their surroundings [She09]. In both humans and animals, learning is deeply tied to the interaction with the environment, where experiences shape the internal models of reality that guide future actions.

In humans, the ability to reason about acquired knowledge is particularly advanced. This reasoning allows individuals to not only apply learned concepts but also to refine and expand them, often by integrating new information with existing symbolic representations. Such reasoning processes are essential for problem-solving, planning, and innovation, which are hallmarks of human intelligence [LH07]. Moreover, the ability to share knowledge through symbolic means, such as language, has enabled humans to build complex societies and advance technologically.

2.1.2 Intelligence in computer science

Since the automation of computation with the invention of the computer, intelligence has always been a central topic in computer science. Officially, the field of AI was born in 1956 at the Dartmouth Conference, where a group of researchers gathered to discuss the possibility of *computing towards intelligence*. In this section, we provide a high-level introduction of how machines can mimic the processes of reasoning and learning that we find in natural intelligence. Later, in Section 2.2 we give a detailed background on AI.

2.1.2.1 Reasoning

Every form of reasoning presupposes a structured connection between *reality* and its *representation*. In symbolic approaches to knowledge representation (KR), this connection is mediated through *symbols*, where the interplay between *signifier* and *signified* enables the encoding of meaning. Symbols thus act as the bridge between the external world and computational processes, allowing reasoning systems to ma-

nipulate abstract entities while still preserving a link to the underlying semantics of the domain. In this sense, reasoning is not only the mechanical application of rules, but also the interpretative activity that derives new knowledge by operating on representations whose significance is grounded in the symbolic layer of KR.

Symbolic KR has always been regarded as a key issue since the early days of AI, under the assumption that intelligent behavior requires explicit forms of knowledge and mechanisms to represent and manipulate it [NS76; BL04]. 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* [Llo90], that is, the exploitation of formal logic in computer science. Consider, for instance, *deductive databases* [GR68], *description logics* [Baa+03], *ontologies* [Cim06], *Horn* logic [McN77], *higher-order* logic [VD01].

Reasoning in computer science involves the process of deriving conclusions from a set of premises or facts, often encoded in a formal language. Logic reasoners, such as Prolog-based systems like tuProlog [DOR01], play a central role in automating this process. These reasoners typically follow a sequence of steps:

- **Parsing**, the reasoner first parses the input knowledge base, which consists of facts, rules, and queries, into an internal representation;
- **Inference**, using inference rules, such as *modus ponens* [End72], the reasoner derives new facts or verifies the truth of a query.
- **Unification**, a key operation in logic reasoning is unification, where variables in predicates are matched with constants or other variables to satisfy logical constraints;
- **Search**: the reasoner explores the search space of possible solutions, often using strategies like depth-first search or breadth-first search, depending on the implementation;
- **Result generation**, finally, the reasoner provides the results of the reasoning process, which could be a solution to the query, a proof, or a set of derived facts.

For example, in a Prolog-based system, a query is resolved by attempting to unify it with facts or the heads of rules in the knowledge base, recursively applying the

rules' bodies until a solution is found or all possibilities are exhausted. Logic reasoners are widely used in various applications, including expert systems, semantic web technologies, and automated theorem proving, due to their ability to handle complex symbolic reasoning tasks efficiently.

2.1.2.2 Learning

The learning process performed by machines is called machine learning (ML). ML is a wide umbrella term that encompasses a variety of different ways of learning and different learning tasks. A widely adopted definition of ML is the one given by Tom Mitchell in 1997 [Mit97]:

Definition 2.2 (Machine Learning) *a computer program is said to learn from experience E with respect to some class of tasks T and performance measure P if its performance at tasks in T, as measured by P, improves with experience E.*

The software component deputed to learning is referred as *model*, *learner* or *predictor* (and possibly other names). The experience E can be represented as a given dataset, i.e., a collection of input data, but there could be other ways (e.g., through reinforcement learning (RL)). The input can come in different shapes, e.g., *tabular data*, *images*, *text*, etc. Each input type is represented in different ways to be interpreted by a machine: an entry in a table is represented as a vector of features, an image is usually represented as a matrix, or a tensor, and a text can be represented as a vector of word embeddings. There can be a variety of different tasks T and performance measures P. In the rest of this section, we will provide a brief overview of the most common tasks and performance measures.

The most common macro ML tasks are:

- **Classification** $\rightarrow$ given the input data $X = x_1, x_2, \ldots, x_n$, the task is to predict a label y (a.k.a., output) from a finite set of labels $Y = \{y_1, y_2, \ldots, y_m\}$. The input is made of numerical features x_i that can either be binary, categorical, or continuous. Ultimately, the goal is to learn a prediction function $\pi^* : X \rightarrow Y$;
- **Regression** $\rightarrow$ the task consists in predicting a continuous value y from the input data X. Similarly to classification, the goal is to learn the optimal predictor that maps the input X to the output Y (usually $Y = \mathbb{R}$);

- **Clustering** $\rightarrow$ the task is to group the input data X into k clusters according to some strategy (e.g., usually a distance function). A cluster is a subset of the input data X that is *similar to each other* and *dissimilar from the data in other clusters*. Clustering predictors can be considered as classifiers upon anonymous classes.

To approximate π^*, a learning algorithm is used. Depending on the type of learning algorithm, the learning process can be *supervised*, *unsupervised*, or *reinforcement*.

Supervised A ML process is *supervised* if it is trained on a labelled dataset, where each data point is associated with a corresponding label that represents the desired output. The training set is made of pairs (x_i, y_i), where x_i is the input data and y_i is the corresponding label. The goal of the learning process is to learn a function f that maps the input data x_i to the corresponding label y_i, such that $f(x_i) \approx y_i$ for almost all training examples [2]. The function f is usually a mathematical model that is trained on the training set by minimizing a loss function, which quantifies the error between the predicted output $\hat{y}_i = f(x_i)$ and the true label y_i. For example, in a binary classification task, the loss function could be the binary cross-entropy, while in a regression task, it could be the mean squared error. Once trained, the model can generalize to unseen data, predicting labels for new inputs based on the patterns it has learned from the training set. An example of supervised learning is email spam detection, where the input x_i could be the content of an email, and the label y_i could be either *spam* or *not spam*. The model learns to classify emails by analyzing a dataset of labelled examples and can then predict whether a new email is spam or not.

Unsupervised In unsupervised learning, the learning task consists of discovering patterns, structures, or relationships within a dataset without relying on labeled outputs. The goal is to find an optimal representation or grouping of the data based on an objective criterion, such as minimizing intra-cluster distances or maximizing data variance. Common tasks in unsupervised learning include *clustering*, where data points are grouped into clusters based on similarity, and

[2] the learning process can sacrifice accuracy on some training examples to improve generalization on unseen data.

dimensionality reduction, where the data is transformed into a lower-dimensional space while preserving its essential structure. For example, in clustering, the *k-means* algorithm partitions a dataset into k clusters by iteratively assigning data points to the nearest cluster centroid and updating the centroids to minimize the total intra-cluster variance [Llo82]. Another example is principal component analysis (PCA), a dimensionality reduction technique that identifies the directions (principal components) in which the data varies the most, enabling efficient data compression and visualization [Pea01].

Reinforcement learning In reinforcement learning, the learning task consists of letting an agent estimate *optimal plans or policies* by interacting with an environment and receiving feedback in the form of rewards or penalties. The agent's goal is to maximize the cumulative reward over time by learning which actions to take in different states of the environment. This process is typically modeled as a Markov decision process (MDP), which is defined by a set of states, a set of actions, a transition function that describes the probabilities of moving between states, and a reward function that assigns a numerical value to each state-action pair [SB98]. For example, consider a ground robot that performs phototaxis, i.e., it moves towards a light source. The robot can perceive its environment through sensors that detect the light intensity and can perform actions such as moving forward, turning left, or turning right. When the robot moves closer to the light source, it receives a positive reward, while moving away results in a negative reward. Over time, the robot learns a policy that maps states to actions, allowing it to navigate effectively towards the light source by maximizing its cumulative reward.

2.2 Artificial intelligence

The wide range of AI techniques are divided into many categories, including two major ones: *symbolic* and *sub-symbolic* AI. Figure 2.1 provides an overview of AI fields, highlighting the distinction between symbolic and sub-symbolic approaches. The thing that distinguishes these two categories is the way they *represent knowledge* and how they process it. No intelligence can exist without knowledge and no

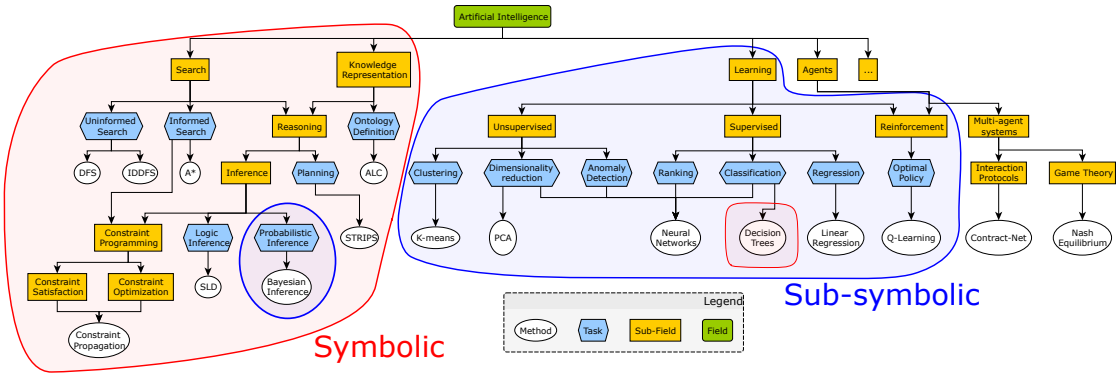


Figure 2.1: AI map (not exhaustive) illustrating the main subfields with a focus on symbolic and sub-symbolic approaches. On the left side, symbolic AI is represented by search algorithms, KRs, and reasoning. On the right side, sub-symbolic AI is represented by learning technique families, such as unsupervised learning, supervised learning, and reinforcement learning. The figure is a simplified representation to give the reader a general idea of the landscape of AI. Notably, learning-based approaches are not necessarily subsymbolic. For instance, inductive logic programming (ILP) and DT learning induce explicit symbolic structures whose components retain a well-defined and compositional semantics, despite being learned from data.

computation can occur in lack of representation. In the rest of the thesis, we will use the term symbolic (resp., sub-symbolic) AI and symbolic (resp., sub-symbolic) KR almost interchangeably. Symbolic AI is based on *symbols*, which come with a *meaning* and could be manipulated according to the formalism and rules of a given AI system. On the other hand, sub-symbolic AI is based on a numerical representation – a.k.a., sub-symbolic – where the numbers are not directly interpretable. Numbers are technically symbols, but numbers, arrays and their functions are not recognised as means for symbolic KR. According to Van Gelder [Gel90], in order to be considered symbolic, KR approaches must:

R1 involve a set of symbols;
R2 the symbols can be combined following a set of grammatical rules;
R3 elementary symbols and combinations of symbols can be assigned a meaning.

Local vs. distributed Multidimensional arrays are the fundamental building block of sub-symbolic data representation. Formally, a D-order array is an ordered container of real numbers, where D indicates the number of indices required to ac-

cess each element. We refer to 1-order arrays as *vectors*, 2-order arrays as *matrices*, and arrays of order greater than two as *tensors*. In sub-symbolic tasks based on arrays, information is typically conveyed both by the values stored in the array and their position within it. The dimensions of the array – denoted as $(d_1 \times \cdots \times d_D)$ – also play a crucial role, as sub-symbolic systems are usually designed to operate on arrays of fixed shape. That is, the values of $d_1, \ldots, d_D$ are chosen at design time and remain unchanged thereafter. This violates R2 above; accordingly, we define sub-symbolic KR as the task of encoding information into rigid numeric arrays. *Local* and *distributed* representations are two key modes for encoding data into such arrays. In local representations, each entry in the array corresponds to a well-defined concept from the target domain—its semantic meaning is clear and independent. In distributed representations, by contrast, individual values carry little or no standalone meaning: their interpretation depends on the configuration of values across a neighbourhood in the indexing space. Consequently, while the exact location of values is largely irrelevant in local representations, it becomes essential in distributed ones. Notably, distributed representations violate R3, and for this reason, recent literature often labels as *sub-symbolic* those predictors that rely on distributed encoding of data.

2.2.1 Symbolic AI

Symbolic AI has been regarded as crucial since AI's inception. Symbolic KR offers enhanced flexibility, expressiveness, and intelligibility, being interpretable both by machines and by humans.

Intentional vs. extensional In formal logic, one may define concepts either *extensionally* or *intensionally*. Extensional definitions are direct representations of data. For example, the set of square numbers admits the extensional definition $\{0, 1, 4, 9, 16, \ldots\}$ by listing every member explicitly. Conversely, an *intensional* definition is an indirect representation of data. In first-order logic (FOL), this corresponds to defining a relation via a formula; for instance, the set of square numbers can be defined as $\{\, x \mid \exists n \in \mathbb{Z}\,(x = n^2)\}$ which succinctly encodes an infinite extension with a single schema. Recursive intensional predicates further

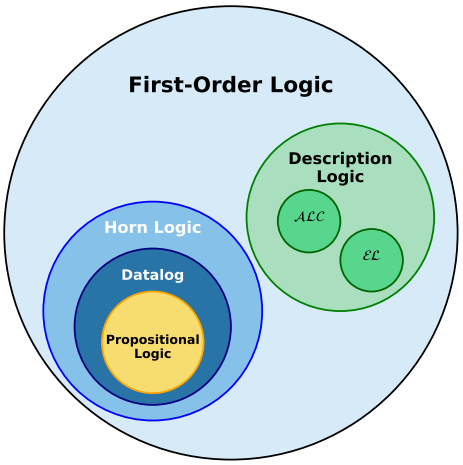


Figure 2.2: Venn diagram of different logic families, illustrating the trade-off between expressiveness and tractability. FOL is the most expressive logic – not the most expressive logic in general – encompassing all others. On the other hand, propositional logic is the least expressive, as it can only represent atomic propositions and their combinations. All the other logics fall somewhere in between, with varying degrees of expressiveness and tractability.

enhance expressivity: for example, the ancestor relation can be axiomatized by $Ancestor(x, y) \Leftrightarrow Parent(x, y) \vee \exists z\,[\,Parent(x, z) \wedge Ancestor(z, y)\,]$ allowing a compact representation of an infinite set of pairs with a finite rule. In formal logic, intensional definitions are prized for their ability to model potentially unbounded domains within finite logical formalisms.

Expressiveness vs. tractability Tractability addresses the theoretical question of whether a logic reasoner can determine the truth of a given formula within feasible time and space bounds. The answer is deeply tied to the specific reasoning algorithm and the logic's formal properties. Depending on the features a logic provides – such as quantifiers, function symbols, or recursive definitions – it may be more or less expressive. The higher the expressiveness, the more complex the problems that can be represented and reasoned about, but this also increases the computational burden. This well-known phenomenon is often referred to as the expressiveness/tractability trade-off [CS93a; BL04; LB87]. In practice, highly expressive logics make it easier for human users to model rich domains, often requiring fewer and more concise formulas. However, this comes at the cost of automated inference, which may become computationally intractable, undecidable, or

non-terminating in the general case. To mitigate this issue, various fragments and extensions of FOL have been identified, each providing different tradeoffs between what can be expressed and what can be decided efficiently. Figure 2.2 illustrates the relationships among different logic families, highlighting the trade-off between expressiveness and tractability.

2.2.1.1 First-order logic

FOL is a general-purpose formalism that underpins most symbolic KR systems. It enables both human and computational agents to model entities and their interrelations through predicates and terms within a defined domain of discourse. Its syntax comprises variables (quantified explicitly or implicitly), constants, function symbols, and predicate symbols, which are combined via logical operators such as conjunction ($\wedge$), disjunction ($\vee$), implication ($\rightarrow$), and equivalence ($\leftrightarrow$). FOL allows for both *extensional* and *intensional* definitions. Recursive intensional definitions, in particular, are powerful, enabling finite representations of infinite sets. Despite its flexibility, FOL is semi-decidable in general: there is no algorithm that can determine the truth of every FOL formula in finite time, which limits its use in systems requiring guaranteed termination [Baa+03].

2.2.1.2 Horn logic

Horn logic is a significant subset of FOL, offering a balanced trade-off between theoretical expressiveness and practical tractability [Mak87]. It is built around the concept of *Horn clauses* [Hor51], which are formulas in FOL that exclude quantifiers and consist of a disjunction of predicates, with at most one non-negated literal. Alternatively, a Horn clause can be expressed as an implication where the consequent is a single predicate and the antecedent is a conjunction of predicates: $h \leftarrow b_1, \ldots, b_n$. Here, $\leftarrow$ denotes logical implication (from right to left), commas represent logical conjunctions, and b_i as well as h are predicates of arbitrary arity, potentially containing FOL terms such as variables, constants, or functions.

Horn clauses can be interpreted as *if-then* rules written in reverse order, where only conjunctions of predicates are allowed in the antecedent. In essence, Horn logic is a constrained subset of FOL characterized by the following limitations:

(i) formulas are reduced to clauses, containing only predicates, conjunctions, and a single implication operator; *(ii)* operators such as $\vee$, $\leftrightarrow$, or $\neg$ (negation) are not allowed; *(iii)* variables are implicitly quantified; and *(iv)* terms behave as they do in FOL.

2.2.1.3 Datalog

Datalog is a declarative query language and a restricted subset of FOL, designed for deductive databases and knowledge representation [AG94]. It represents knowledge using function-free Horn clauses, as defined in Section 2.2.1.2. This restriction eliminates the use of function symbols, thereby forbidding structured terms such as recursive data structures. As a result, Datalog is well-suited for applications requiring finite and decidable reasoning, as the absence of function symbols ensures termination of inference algorithms. Similar to Horn logic, Datalog's knowledge bases consist of sets of function-free Horn clauses, which are interpreted as rules and facts. Rules in Datalog follow the form $h \leftarrow b_1, \ldots, b_n$, where h is the head of the rule and $b_1, \ldots, b_n$ are the body predicates. Unlike general FOL, Datalog does not allow disjunctions, negations, or explicit quantifiers, as variables are implicitly universally quantified. Datalog is widely used in areas such as knowledge graphs (KGs), semantic web technologies, and database systems, where efficient reasoning over large datasets is required. Its simplicity and computational efficiency make it a practical choice for symbolic AI tasks that demand tractable reasoning.

2.2.1.4 Description logic

Description logic (DL) are a family of subsets of FOL, typically involving limited or no quantifiers, no structured terms, and no n-ary predicates where $n \geq 3$ [Baa+17]. In essence, DL represents knowledge using constants and variables, along with atomic, unary, and binary predicates.

The differences among specific variants of DL lie in the set of supported logical connectives and whether negation is allowed. The wide variety of DL stems from the well-known trade-off between expressiveness and tractability. Depending on the application, one may prefer a more expressive DL variant, which offers richer features at the cost of reduced tractability or even decidability of algorithms

manipulating the knowledge, or vice versa.

In DL, it is common practice to use specific terminology for different elements of knowledge representation:

- Constant terms are referred to as *individuals*, as each constant represents a single entity within a domain.
- Unary predicates are called *classes* or *concepts*, grouping sets of individuals for which the predicate holds true.
- Binary predicates are referred to as *properties* or *roles*, connecting pairs of individuals.

Using this nomenclature, knowledge in DL can be represented by associating entities with constants (e.g., URLs) and defining concepts and properties accordingly. Binary predicates are particularly significant as they enable the connection of pairs of entities. This is typically achieved through subject-predicate-object triplets, represented as ground binary predicates of the form $\langle a\, f\, b \rangle$ or $f(a, b)$, where a is the subject, f is the predicate, and b is the object.

Collections of such triplets form KGs, which are directed graphs where vertices represent individuals and arcs represent binary properties connecting these individuals. KGs may explicitly or implicitly instantiate a specific ontology, which is a formal description of classes characterizing a domain, their relationships (e.g., inclusion, exclusion, intersection, equivalence), and the properties they must or must not include.

DLs are widely used in applications such as semantic web [GM03] and ontology engineering [Hit+16], where efficient reasoning and knowledge representation are essential. Their ability to balance expressiveness and computational efficiency makes them a cornerstone of symbolic reasoning systems.

2.2.1.5 Ontologies and knowledge graph

An ontology is a formal and explicit specification of a shared conceptualisation of a domain [Gri10]. It provides a structured vocabulary to describe the entities relevant in that domain, along with their attributes and the relationships among them. This organisation enables both human understanding and machine-based reasoning.

Ontologies are typically expressed using DLs, a family of logic-based formalisms for knowledge representation. DLs define three main components: *(i) concepts* (or *classes*), which group entities sharing similar features; *(ii) individuals* (or *instances*), which are the concrete elements of the domain; *(iii) roles* (or *properties*), which describe binary relationships between individuals. Different DLs vary in their expressive power: for example, existential language ($\mathcal{EL}$) supports only conjunction and existential quantification to ensure efficient reasoning, while more expressive DLs like attributive concept language with complements ($\mathcal{ALC}$) allow for full Boolean operators and universal quantification.

Concepts are typically denoted using capital italic letters, such as *Animal* or *Cat*. These can be combined using logical constructors like intersection ($\sqcap$), union ($\sqcup$), or negation ($\neg$) to form more complex classes. A statement like $Cat \sqsubseteq Animal$ expresses that all cats are animals.

Individuals are constants representing specific entities in the domain and are usually written in monospaced lowercase, for example `tom`. Membership of an individual in a concept is denoted using the "is-a" relation, written as $\texttt{tom} : Cat$, meaning "Tom is a cat." Each individual may belong to multiple concepts.

Roles represent binary relations between individuals and are written in lowercase sans-serif font, such as eats. They connect pairs of individuals, and their domain and range can be restricted using expressions such as $\mathsf{eats} \sqsubseteq Animal \times Edible$. Assertions like $\mathsf{eats}(\texttt{tom}, \texttt{mouse})$ state that Tom eats the mouse.

The subsumption relation ($\sqsubseteq$) is used to express inclusion between concepts or roles. For instance, $Cat \sqsubseteq Animal$ means that every cat is also an animal, and $\mathsf{predatorOf} \sqsubseteq \mathsf{eats}$ means that every predator-prey relationship implies eating. Special concepts such as $\top$ and $\bot$ are used to denote the most general and the most specific concepts, respectively.

Collections of such axioms form an ontology. Terminological axiom (TBox) define concepts and roles and their interrelations, while assertional axiom (ABox) specify which individuals belong to which concepts or are related via which roles.

KGs also provide a structured way to represent knowledge as graphs. They consist of triplets (or *facts*) of the form (s, p, o), where s is the subject, p is the predicate (or property), and o is the object. These triplets form a directed graph where nodes represent individuals and edges represent relationships.

Unlike ontologies, KGs do not necessarily impose formal constraints on the structure or semantics of the triplets. This flexibility allows for representing heterogeneous and incomplete data. However, some KGs are explicitly grounded in an ontology, and may follow its vocabulary and logical constraints.

In summary, ontologies and knowledge graphs both aim to formally capture structured knowledge. Ontologies provide formal semantics and enable logical reasoning, while knowledge graphs emphasise scalability and flexibility in representing factual data.

2.2.1.6 Propositional Logic

Propositional logic is a restricted subset of FOL in which quantifiers, terms, and non-atomic predicates are absent. Its language consists solely of atomic propositions – also called 0-ary predicates – combined using standard logical connectives such as conjunction ($\wedge$), disjunction ($\vee$), negation ($\neg$), and implication ($\rightarrow$). Each proposition can be interpreted as a Boolean variable that takes a truth value in $\{\texttt{true}, \texttt{false}\}$. The semantics of propositional logic aligns with Boolean algebra, making it straightforward to evaluate the truth of a formula given the truth values of its atomic components.

For example, a propositional formula such as $p \wedge \neg q \rightarrow r$ can be interpreted as follows: p might stand for the proposition "it is raining," q for "there is a roof," and r for "the floor is wet." In this case, the formula asserts that if it is raining and there is no roof, then the floor will be wet.

Compared to FOL, propositional logic is significantly less expressive. The absence of quantifiers means that general statements over a domain cannot be made. Similarly, the lack of terms prevents direct reference to entities in the domain. As a consequence, each relevant aspect of a scenario must be explicitly encoded as an individual proposition.

This limitation in expressiveness has important computational implications. In particular, determining the satisfiability of a propositional formula is a decidable problem. This makes propositional logic attractive in scenarios where tractable reasoning is required.

Despite its apparent simplicity, propositional logic can model a surprising range

of situations. Expressions involving numerical constants, variables, and comparisons (e.g., $x > 5$ or $y = 3$) can be encoded propositionally by introducing a distinct Boolean variable for each comparison. This reduction allows the use of propositional logic in settings where the original problem does not explicitly involve logical variables or quantifiers, but can be decomposed into atomic truth conditions.

2.2.1.7 Decision trees

Decision trees (DTs) are a family of predictors that support both classification and regression tasks. During the learning phase, the input space is recursively partitioned into multiple regions through a series of splits (i.e., decisions) based on the input data X. Each region is associated with a constant prediction, and the splits are designed to minimize the error with respect to the expected outputs Y, while keeping the total number of regions low. The learning process synthesizes a hierarchical structure of decision rules, which are followed to compute predictions for any $x \in X$. In the inference phase, these decision rules are evaluated sequentially, starting from the root of the tree and proceeding to a leaf node. Each leaf corresponds to a specific region of the input space, resulting in a single prediction for each x.

Unlike other predictor families, DTs produce a tree of decision rules as the outcome of the learning process. This tree is straightforwardly interpretable by humans and can be graphically represented in two-dimensional charts. This property is particularly valuable when the internal workings of an automatic predictor need to be understood by a human agent. DTs are often used in applications where interpretability is crucial, as their structure provides clear insights into the decision-making process. They are often transformed into sets of logical rules for easier comprehension and analysis.

2.2.1.8 Limits of symbolic AI

Symbolic AI has been a foundational approach in the field of AI, providing a structured way to represent and reason about knowledge. However, it also has several limitations.

Symbolic systems can be *delicate*, meaning they may fail to handle unexpected situations or incomplete information [McD90]. This is because symbolic systems rely on predefined rules and representations, which may not cover all possible scenarios.

As the complexity of the domain increases, the number of rules and symbols required to represent knowledge can grow significantly. This can make it difficult to manage and maintain large symbolic systems; they do not *scale* easily [LF91].

Symbolic systems typically require *manual encoding* of knowledge, which can be time-consuming and error-prone. This makes it challenging to adapt to new information or learn from experience. This limitation is usually referred in the literature as the *knowledge engineering bottleneck* [RN20].

Moreover, symbolic systems often struggle to handle *uncertainty* and probabilistic reasoning. This is because symbolic representations are typically deterministic, making it difficult to model real-world situations that involve uncertainty.

To sum up, symbolic models struggle to capture the nuances and complexities of real-world data, particularly when dealing with noisy or ambiguous information. Also, they struggle when the domain is *dynamic* and constantly changing, as they may require frequent updates to the rules and representations.

2.2.2 Sub-symbolic AI

Sub-symbolic AI encompasses a wide range of techniques that rely on numerical representations. Contributions come from varius fields, including ML, statistical learning, and data mining. The huge number of approaches is also motivated by the well known no free lunch (NFL) theorem [WM97] that states that no single learning algorithm can outperform all others across all possible tasks. The most common predictor families are linear models, DTs, random forests (RFs), support vector machines (SVMs), NNs (including LLMs), and many others.

Predictive performance vs. interpretability The *complexity* of a sub-symbolic predictor can vary significantly. By complexity we mean the overall structure of the model, including the number of parameters, the operations that are performed, and possibly the way the model is trained. DTs, for example, are

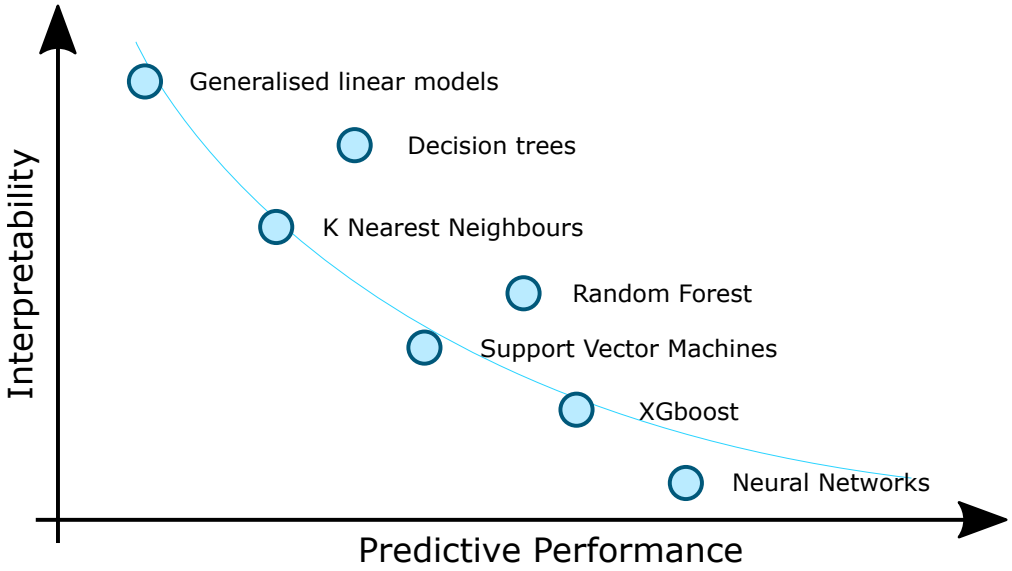


Figure 2.3: Trade-off between predictive performance and interpretability in common sub-symbolic predictors. The figure illustrates how different predictor families balance these two aspects, with simpler models being more interpretable but potentially less accurate.

relatively simple models that can be easily interpreted by humans. The downside of DTs is that they make predictions by linearly partitioning the input space, which can lead to poor generalisation on unseen data. On the other hand, NNs can be extremely complex, with millions of parameters and intricate architectures that are difficult to interpret. However, NNs can capture highly non-linear relationships in the data, often leading to superior predictive performance compared to simpler models. The definition of interpretability is not univocal, and it lacks measures that are widely accepted [Rud19]. Despite that, Figure 2.3 illustrates in an informal way the trade-off between predictive performance and interpretability in sub-symbolic predictors.

Sort of ML predictors There exist many sub-symbolic models in the literature. Some of them are relatively simple and easy to interpret, while others are more complex and difficult to understand. Among them, DT are relatively simple, interpretable and popular models. On the other hand, NNs are more complex and difficult to interpret, but they can capture highly non-linear relationships in

the data. In between these two extremes, there are many other models, such as RFs and SVMs, that offer a balance between interpretability and predictive performance. For the sake of conciseness, we will focus only on the models that are relevant for the rest of the thesis.

2.2.2.1 Neural networks

NNs are a family of predictors inspired by the structure and function of biological neural networks. They consist of interconnected nodes – a.k.a., neurons – interconnected into a directed acyclic graph (DAG) and organized into layers, where each neuron receives inputs, applies a non-linear activation function, and produces an output. Despite being very popular nowadays, the first idea of a NN has been proposed back in 1943 [MP43]. The so-called perceptron – a single-layer NN – has been introduced in 1958 [Ros58]. Due to the limited expressiveness of single-layer NNs, the perceptron could only learn linearly separable functions [MP87].

To overcome these limitations, researchers proposed to stack multiple layers of neurons, leading to the concept of multilayer perceptrons (MLPs). Although the theoretical foundations of such models were clear early on, the lack of an effective training algorithm hindered their practical use. This changed with the rediscovery and popularization of the *backpropagation* algorithm in the 1980s [RHW86], which allowed efficient computation of gradients through multiple layers using the chain rule of calculus. This innovation enabled MLPs to learn complex non-linear functions and marked the beginning of modern NN research.

Many different architectures have been proposed since then, including fully connected NNs, convolutional neural networks (CNNs), recurrent neural networks (RNNs), and more. Figure 2.4 provides an overview of some common NN architectures. These architectures are designed specifically to handle different types of data and tasks, such as image classification, natural language processing (NLP), and time series analysis.

Feedforward Neural Networks The simplest class of NNs is the family of feedforward networks, in which information flows unidirectional from the input layer through one or more hidden layers to the output layer, without forming cycles. Among these, MLPs are the canonical example, consisting of fully connected

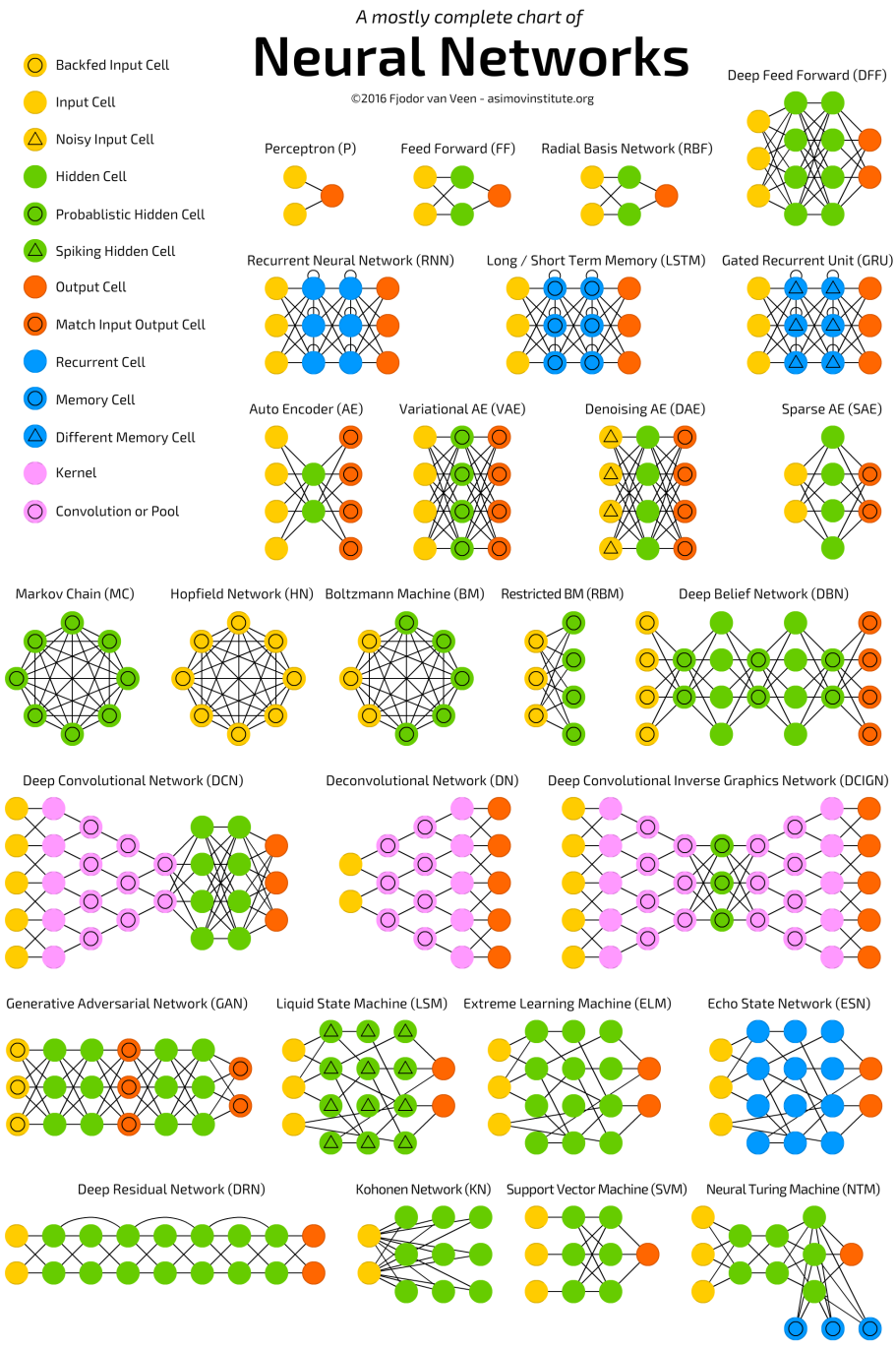


Figure 2.4: Overview of common NN architectures. Each architecture is designed to handle different types of data and tasks, leveraging specific structural features to improve performance.

layers interleaved with non-linear activation functions. Given sufficient capacity, MLPs are universal function approximators [HSW89], capable of representing any Borel measurable function under mild assumptions. Despite their theoretical expressiveness, MLPs are inefficient when processing structured inputs such as images or sequences, as they fail to exploit the inherent inductive biases of the data.

Convolutional Neural Networks To overcome the limitations of MLPs on grid-like data such as images, CNNs were introduced. Pioneering work on hierarchical feature extraction was done by Fukushima with the Neocognitron [Fuk80], followed by the successful application of convolutional networks to handwritten digit classification in LeNet-5 [LeC+98]. CNNs apply learnable convolutional filters that exploit local spatial correlations, reducing the number of parameters and enabling translation equivariance. Pooling layers further promote translation invariance by down sampling intermediate representations. These properties make CNNs particularly well-suited for tasks involving visual perception.

Recurrent Neural Networks RNNs are designed to process sequential data by incorporating recurrence in the form of feedback connections. More generally, recurrence in neural networks refers to the presence of cyclic connections that give rise to an internal state evolving over time, allowing the network to be interpreted as a discrete-time dynamical system. From this perspective, the current state of the network depends not only on the current input but also on its previous state, enabling the modelling of temporal dependencies and variable-length input sequences. In standard RNNs architectures, this recurrent behaviour is typically implemented by units that receive input both from the current time step and from their own past activations, effectively providing a form of memory across time. Despite their expressive power, standard RNNs suffer from the vanishing and exploding gradient problems [BSF94], which hinder their ability to learn long-range dependencies. To address these issues, more sophisticated variants have been developed, notably long short term memorys (LSTMs) [HS97] and gated recurrent units (GRUs) [Cho+14], which introduce gating mechanisms to regulate information flow and preserve long-term context.

2.2.2.2 Large language model

Pre-trained LLMs, such as the GPT models developed by OpenAI [Rad+18], have achieved remarkable success in various NLP tasks. These models, based on the *transformer* architecture [Vas+17], are trained on vast amounts of textual data, enabling them to encode extensive knowledge about real-world entities, such as historical events and geographical facts [Jia+20]. They can perform tasks like lexical operations, common-sense reasoning, and solving mathematical problems [MS22; Gev+21; Wei+22]. However, the extent to which LLMs exhibit genuine reasoning abilities remains a topic of debate [HHS24]. Some studies suggest that these models often rely on statistical patterns in the data to simulate reasoning, rather than truly understanding the underlying concepts [Ben+21].

The transformer architecture, introduced by Vaswani et al. [Vas+17], was originally designed for machine translation. Over time, it has become the foundation for state-of-the-art NLP models. A transformer consists of two main components: the *Encoder*, which converts a text sequence into an embedded representation, and the *Decoder*, which generates output sequences in an autoregressive manner. Both components are composed of multiple layers, with each layer incorporating mechanisms like self-attention and feed-forward neural networks.

The self-attention mechanism is a key innovation of the transformer. It allows the model to focus on different parts of a sentence, capturing contextual relationships between words. Formally, given an embedded representation E, the model computes three vectors: *key* (K), *query* (Q), and *value* (V), using weight matrices W_k, W_q, and W_v. The output is then calculated as:

$$Z = \text{softmax}\left(\frac{QK^\top}{\sqrt{d_k}}\right)V,$$

where d_k is the dimensionality of the key vectors. This mechanism is extended to multiple attention heads in the *Multi-Head Attention* module, enhancing the model's ability to capture diverse relationships.

Two prominent transformer-based models are BERT and GPT. BERT, introduced in [Dev+19], is an encoder-based model trained using *masked language modeling*, where certain words in the input are masked, and the model predicts

them. In contrast, GPT [Rad+18] is a decoder-based model trained with *causal language modeling*, generating text sequentially from a given prefix. These models have set benchmarks in various NLP tasks, demonstrating the versatility and power of transformer architectures.

Learning from the Web LLMs can be viewed as extensive *knowledge bases* [Pet+19]. They store the textual data they were trained on and enable efficient information retrieval through free-text queries. The training data for these models is typically sourced from a wide range of publicly available Web content.

For example, the early BERT model [Dev+19] was trained on the BooksCorpus dataset [Zhu+15], a collection of over 11,000 books, and a snapshot of the English Wikipedia. Similarly, GPT [Rad+19; Bro+20] were trained on datasets scraped from the public Internet, using Reddit as an entry point for identifying high-quality content [Car+21]. The RoBERTa model [Liu+19] extended this approach by incorporating additional datasets, such as: *(i)* the Common Crawl News dataset[3], *(ii)* the OpenWebTextCorpus [GC19], and *(iii)* the Stories dataset [TL18].

The LLAMA family of models [Tou+23b; Tou+23a] utilized a combination of datasets, including the colossal clean crawled corpus (C4) [Raf+20], English Wikipedia, and other sources. Finally, the Mixtral-of-Experts model by MistralAI is also pre-trained on data extracted from the open Web[4].

Oracles for Data Generation Web-based pre-training allows LLMs to accumulate substantial domain-specific knowledge across a wide range of topics. This knowledge can be extracted by crafting appropriate prompts, enabling LLMs to act as *oracles* for information retrieval tasks. Such capabilities are particularly useful for generating synthetic data that incorporates domain-specific knowledge. In these cases, users can design prompts to guide the LLM in generating the desired content, which can then be converted into the required format.

Hallucinations It is important to note that LLMs are not perfect oracles. They may produce text that is factually incorrect or inconsistent with the context, a

[3] `https://commoncrawl.org`
[4] `https://mistral.ai/news/mixtral-of-experts`

phenomenon known as *hallucination* [Zha+23]. This limitation highlights the need for careful evaluation of the outputs generated by LLMs, as they cannot be blindly trusted.

Temperature To mitigate hallucinations, LLMs often include a parameter called *temperature*, which controls the randomness of their responses. The temperature parameter is a real value in the range $[0, 1]$. A lower temperature results in more deterministic outputs, as the model selects the most probable next word. Conversely, a higher temperature introduces more randomness by sampling from the probability distribution of possible next words. This mechanism allows users to balance the *creativity* of the model with the need for accuracy[5].

The Role of Prompts Another approach to address hallucinations is *prompt engineering*. Clear and unambiguous prompts can help align the LLM's output with user expectations. For instance, explicitly instructing the LLM to adopt the perspective of a domain expert has been shown to improve the accuracy of its responses [MCB24]. Additionally, asking the model to "work step-by-step" can enhance the quality of procedural descriptions [Yan+23].

2.2.2.3 Limits of sub-symbolic AI

Sub-symbolic AI has achieved remarkable success in various domains and scenarios, but it also has limitations.

One of the main limitations is the need for *large amounts* of training data if the predictive task is not trivial (e.g., easy to linearly separate). Sub-symbolic models, especially deep learning models, often require vast amounts of labeled data to learn effectively [LBH15]. This can be a significant challenge in domains where data is scarce or expensive to obtain.

Another limitation is the *lack of interpretability*. Sub-symbolic models, particularly deep learning models, are often considered black boxes because their internal

[5]A language model predicts the next word in a sequence based on a probability distribution. Setting the temperature to 0 ensures deterministic outputs by always selecting the most likely word, while a temperature of 1 introduces randomness by sampling from the distribution.

workings are not easily understandable by humans [Rud19; Lip18]. This is a significant drawback in applications where transparency and explainability are crucial, such as healthcare and finance.

Sub-symbolic models can also be sensitive to *adversarial attacks*, where small perturbations to the input data can lead to incorrect predictions [**DBLP:journals/corr/SzegedyZSBEGF13**]. This vulnerability raises concerns about the robustness and security of sub-symbolic AI systems.

Additionally, sub-symbolic models may struggle to *generalize to out-of-distribution data*, meaning they may perform poorly when faced with inputs that differ significantly from the training data [Rec+19]. Finally, sub-symbolic models often lack common-sense reasoning abilities, making it difficult for them to understand and reason about the world in a way that humans do.

2.3 AI and society

This last section of the chapter is needed to introduce AI sub-fields that have become increasingly important in recent years. In particular, the consequences of the wide adoption of AI systems in everyday life have raised concerns about their trustworthiness, fairness, and accountability.

2.3.1 Explainable artificial intelligence

Modern intelligent systems increasingly rely on sub-symbolic predictive models to support their intelligent behaviour. These models are typically trained using a data-driven approach, leveraging the vast availability of data generated in recent years. ML algorithms enable the semi-automatic detection of statistical patterns hidden within data. Such patterns can then be used to support decision-making, planning, and forecasting across various domains where data is available.

Despite their predictive capabilities, ML models face significant challenges in critical applications. One of the most notable issues is *algorithmic opacity*, which refers to the difficulty humans face in understanding how these models operate or make decisions. Opacity arises due to the complex interplay between high-dimensional datasets, the algorithms processing them, and the dynamic behaviour

of these algorithms during training [Bur16]. This lack of transparency is particularly problematic in domains such as healthcare, finance, and law, where human accountability and explainability are essential.

The term *black box* is often used to describe ML predictors, as their internal workings are not symbolically represented [Lip18]. Without symbolic representations, it becomes challenging for humans to comprehend the operation of these systems or the rationale behind their decisions. This can lead to a lack of trust and acceptance of AI-based solutions.

Regulatory frameworks, such as the General Data Protection Regulation (GDPR), have started to recognise the *right to explanation* [GF17]. This right mandates that intelligent systems must be understandable to ensure fairness, identify biases, and verify that they function as intended. However, the concept of *understandability* remains neither standardised nor systematically assessed [Mil19]. While some ML models are more interpretable than others, there is no consensus on what constitutes an adequate explanation.

Interpretability and explainability are desirable properties for intelligent systems. XAI aims to make sub-symbolic AI more interpretable for humans, often by automating the generation of explanations. Interpretability refers to the cognitive effort required by humans to assign meaning to the behaviour or outcomes of intelligent systems. It is often associated with properties such as algorithmic transparency, decomposability, and predictability.

An *explanation*, on the other hand, is a set of statements or accounts that clarify an object or justify an action. Explanations can involve constructing more interpretable representations of a black-box model. For instance, *model simplification* techniques translate a complex model into a simpler one with high fidelity [Tol+17; Gui+19]. Alternatively, symbolic knowledge can be extracted from sub-symbolic predictors, producing interpretable objects from less interpretable ones.

Another approach to improving interpretability involves injecting symbolic knowledge into sub-symbolic predictors. This process does not produce explanations but increases the model's transparency by aligning its behaviour with more interpretable systems.

Interpretability and explainability enhance trustworthiness in AI-based solutions. However, they also enable users to exercise finer control over these systems,

deciding whether to trust them or not [Rud+22]. The surveyed SKE and SKI methods should be regarded as tools for increasing user control over AI systems.

2.3.1.1 Sorts of Explanation

Two major approaches exist to bring explainability or interpretability to intelligent systems: *by design* and *post-hoc* [Cia+20a; Arr+20; Gui+19].

XAI by Design This approach aims to make systems interpretable or explainable from the outset, treating these features as primary design goals. Methods in this category can be further divided into:

- *Symbols as constraints*, where predictive models are constrained by symbolic rules, often expressed in subsets of FOL.
- *Transparent box design*, where models are inherently interpretable and require no further manipulation.

Post-hoc Explainability This approach involves manipulating pre-existing systems to make them interpretable or explainable. Methods in this category include:

- *Text explanation*, which generates textual descriptions of model behaviour.
- *Visual explanation*, which visualises model behaviour, often using dimensionality reduction techniques.
- *Local explanation*, which segments the solution space into simpler subspaces and explains them.
- *Explanation by example*, which extracts representative examples to illustrate internal relationships.
- *Model simplification*, which constructs a simplified system that optimises similarity to the original while reducing complexity.
- *Feature relevance*, which assigns relevance scores to features, revealing their importance in the model's output.

2.3.2 Fairness in AI

ML models can inherit and amplify biases present in the dataset D. For instance, if D contains features related to gender or race, and the data is not equally distributed across these groups, biases may arise. This often occurs in datasets about hiring decisions, where historical data may reflect fewer non-male or non-white candidates. A supervised model H, trained on such a dataset to predict a target feature Y, might incorrectly learn that gender or race influences the prediction. If the predictions of H are used for decision-making, such as hiring or loan approvals, the model may discriminate against certain groups. In this case, the model is said to be biased, as it replicates patterns of past discrimination.

Fairness interventions can mitigate these biases and are applied at different stages of the ML workflow. In the literature, these methods are categorized as:

- *Pre-processing methods*, which modify the dataset D to reduce bias before training;
- *In-processing methods*, which adjust the learning algorithm A to enforce fairness during training;
- *Post-processing methods*, which modify the predictions of the model H to achieve fairness after training.

A critical prerequisite for fairness interventions is the ability to measure fairness. Fairness metrics quantify the extent of bias in a dataset or model and evaluate the effectiveness of mitigation techniques. Formally, a fairness metric is a function that takes data as input and returns a numerical score. The input data can be a dataset $D = (X, Y)$ or the predictions $h(X)$ of a model on D. A lower score typically indicates higher bias, while a higher score reflects greater fairness.

Numerous fairness metrics have been proposed, differing in the type of data they accept and their interpretation of fairness or bias. These metrics are often categorized based on the fairness notion they adhere to [Meh+22]. The two most common notions are *group fairness* and *individual fairness*.

Group fairness Group fairness ensures that distinct groups of individuals, defined by sensitive attributes, are treated equally. Sensitive attributes, denoted as $S \subset X$, are features such as *gender*, *race*, *age*, or social status, which partition

the dataset D into protected groups. These groups can be defined by a single sensitive attribute, e.g., the group of women identified by a specific value of the *gender* feature, or by a combination of attributes, e.g., the group of Black women identified by both *gender* and *race*. The principle of group fairness is commonly evaluated by measuring the correlation between sensitive attributes S and the target variable Y. For instance, if Y represents loan approval outcomes and S represents the applicant's race, group fairness metrics assess whether the approval rates are consistent across racial groups. Such metrics are crucial for identifying and addressing biases in decision-making processes.

Individual fairness Individual fairness ensures that similar individuals receive similar treatment. The simplest approach to individual fairness is *fairness through awareness*. This method relies on a distance metric $\delta : X \times X \to \mathbb{R}$ in the input space (e.g., Euclidean distance). It requires that if two individuals x_1 and x_2 are similar up to a threshold ϵ, i.e., $\delta(x_1, x_2) < \epsilon$, then their corresponding target values y_1 and y_2 should also be similar. In practice, model predictions $h(x_1)$ and $h(x_2)$ are compared instead of true labels, and fairness scores are computed by averaging the differences in predictions for similar instances. Another approach is *fairness through unawareness*, which requires that sensitive attributes are not used by the model during training or prediction. This can be achieved by removing sensitive features from the dataset or ensuring the model does not rely on them. Fairness scores in this case measure the extent to which the model depends on sensitive attributes and penalize such reliance. A more advanced formulation is *counterfactual fairness*. This requires that a model's prediction for an individual x remains the same, regardless of whether x belongs to a group s in the actual dataset or to a counterfactual group $s' \neq s$. Counterfactual fairness ensures that predictions are invariant to changes in sensitive attributes across hypothetical scenarios. As individual fairness is not the primary focus of this work, we do not delve into the specific metrics used to evaluate it. For a detailed discussion, the reader is referred to [Meh+22].

Chapter 3

Neuro-symbolic AI

What are *SKI* and *SKE*?
What are their *characteristics*?
– **RQ0, RQ1 and RQ3**

Contents

The origins of NeSy AI can be traced back to the 1950s and 1960s, during the early days of AI research [You23]. At that time, the focus was on developing

systems based on symbolic reasoning and rule-based problem-solving. However, by the 1980s, the limitations of purely symbolic approaches became evident. For instance, symbolic AI struggled with tasks such as NLP and computer vision, where the complexity of real-world data posed significant challenges. To address these shortcomings, researchers began incorporating principles inspired by neuroscience into AI systems. This shift marked the beginning of efforts to integrate sub-symbolic methods, such as neural networks, into the field.

In the early 21st century, the idea of combining symbolic reasoning with neural networks gained traction. This led to the emergence of NeSy AI, a hybrid approach that leverages the strengths of both symbolic and sub-symbolic paradigms.

NeSy AI has since been applied to a wide range of domains, including healthcare, robotics, and NLP. It represents one of the most promising directions in AI research, aiming to create systems capable of both learning from data and reasoning logically, much like humans. The "neuro" component of NeSy AI is inspired by the human brain's ability to process information and learn from experience. This is achieved through the use of neural networks, which enable the system to adapt and generalize from data. On the other hand, the "symbolic" component relies on logical reasoning and structured representations of knowledge. This allows the system to perform tasks such as deductive reasoning and to represent knowledge in a human-interpretable form, using symbols and rules. By combining these two components, NeSy AI aims to overcome the limitations of purely symbolic or sub-symbolic approaches. It enables the development of intelligent systems that can reason, learn, and adapt in complex environments.

According to a recent survey [Bhu+24] there exist at least six different types of integrating symbolic AI and sub-symbolic ML predictors. Approaches to NeSy can be categorised w.r.t. how the connectionist and symbolic components are integrated.

1. *Symbolic neuro-symbolic* (Type 1): utilizes neural networks for representational learning, which subsequently feeds into symbolic logic and inference mechanisms. This approach is particularly effective in tasks such as language translation and classification, where neural-based semantic representations enhance symbolic reasoning;

2. *symbolic[neuro]* (Type 2): incorporates neural networks as subcomponents within predominantly symbolic frameworks. This integration improves pattern recognition capabilities within symbolic reasoning systems;
3. *neuro—symbolic* (Type 3): involves a collaborative interaction between neural and symbolic components, which remain distinct and separate. This type of integration is commonly applied in reinforcement learning and probabilistic logic systems;
4. *neuro-symbolic→neuro* (Type 4): embeds symbolic knowledge directly into neural network architectures or training processes. This influences the decision-making and learning pathways of the neural networks;
5. *neuro-symbolic* (Type 5): encodes symbolic constraints within neural architectures, often using soft logic constraints to directly affect neural computations;
6. *neuro[symbolic]* (Type 6): represents a deeply integrated approach where symbolic reasoning capabilities are embedded directly within neural architectures. However, achieving robust combinatorial reasoning in this type remains a significant challenge.

In the rest of this chapter, we focus on two sub-fields of NeSy AI – namely SKI and SKE – that can be partially mapped into some of the aforementioned types of integration. In particular, SKI can be mapped into Type 4, 5 and 6, whereas SKE can be mapped into Type 2 and Type 3. SKI and SKE are two popular way to integrate symbolic AI and sub-symbolic ML predictors. In the literature, there are hundreds of works that propose methods for these two tasks coming from different communities. Here, we present both SKI and SKE in a unified way, providing a clear taxonomy of the methods, their properties, and their limitations. What follows is a re-elaboration of the result of an systematic literature review (SLR) that we conducted on the two topics: *Symbolic Knowledge Extraction and Injection with Sub-symbolic Predictors: A Systematic Literature Review* [Cia+24], published in the *ACM Computing Surveys* journal in 2024. The study surveyed more than 200 papers on SKI – *117 works* – and SKE – *132 works* –, providing a comprehensive overview of the state of the art in these two fields.

3.1 Symbolic knowledge injection

SKI is a wide sub-field of NeSy, which encompasses all the methods that in some way *inject* symbolic knowledge into sub-symbolic predictors. More precisely, we define SKI as:

Definition 3.1 (SKI) *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 [Cia+24].*

We adopt this broad definition because the amount of works in the literature is vast and varied, furthermore the contributions come from different communities (e.g., ML, AI, NLP, XAI, logics, etc.), and they often use different terminologies. This definition highlights several key aspects of SKI that merit further discussion. SKI is conceptualized as a class of algorithms, and it involves procedures that take symbolic knowledge as input and produce ML predictors as output.

Regarding the inputs of SKI procedures, the primary requirement is that the knowledge must be symbolic and provided by the user, ensuring it is human interpretable. Additionally, since the knowledge must be processed algorithmically, it is implicitly required to be machine interpretable. This necessitates the use of formal languages, such as FOL or DTs, for knowledge representation, while avoiding free text or natural language.

Another implicit requirement is that the input knowledge must align functionally with the task of the predictor undergoing injection. For instance, if a predictor is designed to classify customer profiles as either creditworthy or un-creditworthy, the symbolic knowledge should encode decision rules that serve the same purpose and utilize the same input features.

In terms of outcomes, SKI procedures can be categorized into two non-exclusive scenarios. First, they may enable sub-symbolic predictors to accept symbolic knowledge as input. This is achieved through pre-processing algorithms that encode symbolic knowledge into sub-symbolic forms, allowing predictors to compute predictions based on this knowledge. Second, SKI procedures may modify sub-symbolic predictors to ensure their predictions are consistent with the symbolic

knowledge. This involves altering the structure or training process of the predictors so that they incorporate the symbolic knowledge into their inference process. Regardless of the specific outcome, SKI procedures are typically applied during the early stages of the ML workflow, influencing both pre-processing and training phases.

Consistency plays a central role in SKI. To measure this, a consistency score is used to evaluate how effectively the predictor utilizes the injected knowledge in relation to the domain and task it was trained for. For example, if a knowledge base specifies that loans should be granted to individuals from a specific minority group if their annual income exceeds a certain threshold, the predictor should generate predictions that adhere to this rule or minimize violations of it.

In the rest of this thesis, we will refer to *uneducated predictor* to indicate a sub-symbolic predictor that does not use symbolic knowledge – i.e., before the injection takes place –, and to *educated predictor* to indicate a sub-symbolic predictor that uses symbolic knowledge—i.e., after the injection takes place. Sometimes, in the literature the term *enriched predictor* is also used instead of *educated predictor*.

3.1.1 Motivations and goals

SKI can be used for several reasons, such as: *(i) improving the model's predictive performance*, by leveraging symbolic knowledge to guide their learning or inference; *(ii) improving the model's interpretability*, by making their predictions consistent with symbolic knowledge; *(iii) increase the robustness* of sub-symbolic predictors, by making them less sensitive to data perturbations (e.g., noise, data scarcity, etc.); *(iv) reduce the model complexity* of the models, by shaping their structure or by constraining their parameters; *(v)* and possibly many more.

Item *(i)* is one of the most common motivations for SKI. The idea is simple: if there is already some (symbolic) knowledge about a particular domain or task, then it is reasonable to expect that the predictor can benefit from it. In this way the model learns both from the data – inductively – and from the symbolic knowledge—mimicking deductive reasoning.

Another common reason to use SKI is to increase the *interpretability* of the model, as stated in Item *(ii)*. In the context of XAI, this is usually referred as

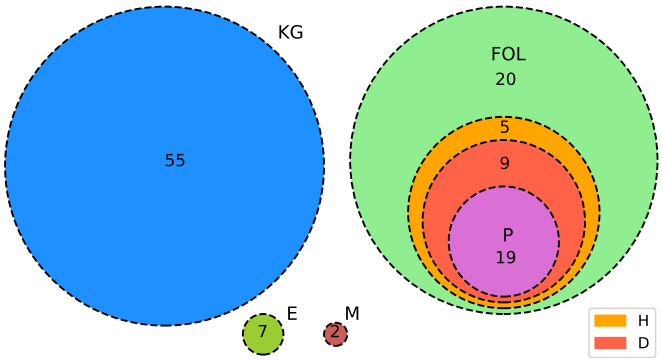


Figure 3.1: Venn diagram categorising SKI methods w.r.t. the *input knowledge* type: knowledge graphs (KG), propositional logic (P), first-order logic (FOL), expert knowledge (E), Datalog (D), Horn logic (H), or modal logic (M).

XAI *by design* (Section 2.3.1.1). The intuition is simple: the model is made to be consistent – up to a certain extent – with the symbolic knowledge, which is usually more interpretable than the model itself. This can be done in two ways: either by using *symbols as constraints* or by *transparent box design*. More details about these two approaches are provided in Section 3.1.4 and Section 3.1.4, respectively.

Predictive performances and XAI are the main motivations for SKI, but not the only ones. The *robustness* (Item *(iii)*) of a predictive model is another important challenge [Liu+18], and it relates to predictors' ability to maintain performance despite the presence of input perturbations. A metric of robustness in the context of SKI is defined in the work "An Empirical Study on the Robustness of Knowledge Injection Techniques Against Data Degradation" [Raf+24]. The content of the paper is presented in Section 5.3. Along with robustness, there are other metrics – often neglected – that play a crucial role in the design of intelligent systems, such as *memory footprint* (Item *(iv)*), *latency*, data efficiency, and so on. These quality of service (QoS) metrics are presented in the work "Symbolic Knowledge Injection Meets Intelligent Agents: QoS metrics and experiments" [Agi+23], which is discussed in Section 5.2.

3.1.2 What to inject

A key distinction in SKI methods lies in whether the chosen formalism is *machine interpretable*, *human interpretable*, or both. SKI methods can be categorized into two primary groups based on the formalism used to represent input knowledge [Cia+24]:

- **Logic formulas or knowledge bases (KBs):** These adhere to FOL or its subsets, making them interpretable by both humans and machines. The sub-categories (most of them already introduced in Section 2.2.1), ordered by decreasing expressiveness, include:
  - *FOL formulas:* These encompass recursive terms, variables, predicates of any arity, and various logic connectives, potentially expressing definitions.
  - *Horn logic:* Often referred to as Prolog-like logic, this formalism consists of head–body rules involving predicates and terms of any kind.
  - *Datalog:* A restricted subset of Horn logic that excludes recursive terms, allowing only constants or variables as terms.
  - *Modal logics:* These extend the above logics with modal operators (e.g., $\Box$ and $\Diamond$), which express modalities such as necessity or possibility.
  - *Knowledge graphs:* A practical application of description logics designed to represent entity–relation graphs.
  - *Propositional logic:* This involves Boolean variables and logical connectives, offering a simpler yet effective formalism.
- **Expert knowledge:** This category includes human-interpretable knowledge that is not inherently machine-readable. Examples include physics equations, syntactical rules, or domain-specific expertise. Since expert knowledge is not directly machine interpretable, it often requires transformation into tensorial form through data generation, a process that typically involves human engineers and can be labor-intensive.

Figure 3.1 illustrates the distribution of surveyed SKI methods based on their formalism of choice. KGs emerge as the most prevalent category, representing nearly half of the surveyed methods. In contrast, modal logics constitute the smallest

group. Methods based on FOL or its subsets (excluding KGs) form another significant cluster, with propositional logic being particularly prominent due to its relative simplicity and widespread use. The specific logic formalism employed in the surveyed papers is reported where available. However, this information is rarely explicitly stated by the authors. Instead, the logic is often inferred from the constraints and descriptions provided in the respective works.

Practical examples of symbolic knowledge that can be injected into sub-symbolic predictors are the public health guidelines on type-2 diabetes of the National Institute of Diabetes and Digestive and Kidney Diseases[1]. These guidelines have been encoded into logic formulas [Kun+10] and used in works related to SKI [Mag+25]. The first guideline state that if a patient has a glucose value greater or equal to 125 mg/dL and a body mass index (BMI) greater or equal to 30, then the patient is considered diabetic. The second one says that if a patient has a glucose value lower or equal to 100 mg/dL and a BMI lower or equal to 25, then the patient is considered non-diabetic. In FOL, the first statement can be encoded as:

$$\forall x.(\text{glucose}(x) \geq 125 \wedge \text{bmi}(x) \geq 30 \rightarrow \text{Diabetic}(x)) \tag{3.1}$$

whereas the second statement can be encoded as:

$$\forall x.(\text{glucose}(x) \leq 100 \wedge \text{bmi}(x) \leq 25 \rightarrow \neg\text{Diabetic}(x)) \tag{3.2}$$

3.1.3 Where to inject

SKI methods can target many different types of sub-symbolic predictors. The vast majority of them are NNs. Among NNs, there are different kinds of architectures that one can use to inject symbolic knowledge including. In addition to NNs, SKI methods can also target other types of sub-symbolic predictors. The predictors targeted by the surveyed SKI methods (most of them already introduced in Section 2.2.2) include:

- **feed-forward NNs** multi-layered NNs in which neurons from layer i are

[1] `https://www.niddk.nih.gov/health-information/diabetes/`

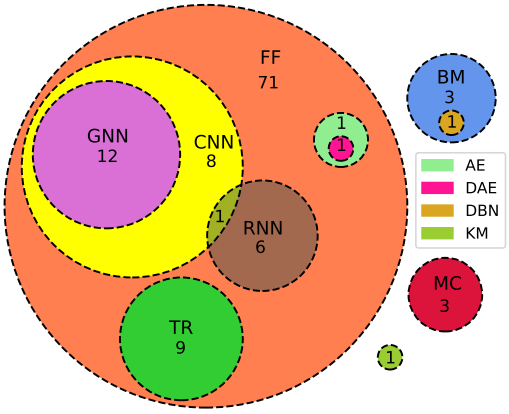


Figure 3.2: Venn diagram categorising SKI methods w.r.t. the type of sub-symbolic predictor: feed-forward (FF), convolutional (CNN), graph (GNN) or recurrent (RNN) neural networks, Boltzmann machines (BM), Markov chains (MC), transformers (TR), auto-encoders (AE), deep belief networks (DBN), denoising auto-encoders (DAE), kernel machines (KM).

only connected with layer i + 1, and multiple (≥ 2) layers may exist;

- **convolutional NNs** (CNNs) particular case of FF, NNs that use convolutional layers to extract features from the input data, typically used in computer vision tasks;
- **graph NNs** (GNNs) particular case of CNNs, NNs that operate on graph-structured data, allowing for the injection of symbolic knowledge in the form of graph structures;
- **recurrent NNs** (RNNs) NNs that use recurrent connections to process sequential data, such as time series or natural language;
- **transformers** (TR) a type of NNs architecture that uses self-attention mechanisms to process sequences, widely used in natural language processing tasks;
- **auto-encoders** (AE) a type of NNs that learns to encode input data into a lower-dimensional representation and then decode it back to the original input, often used for dimensionality reduction or feature extraction;
- **denoising auto-encoders** (DAE) a type of auto-encoder that learns to reconstruct the original input from a corrupted version, often used for noise reduction or data augmentation;

- **Boltzmann machines** (BM) a type of stochastic NNs that can learn complex distributions over their inputs, often used for generative tasks;
- **Markov chains** (MC) probabilistic models that represent systems that transition between states based on certain probabilities, often used for sequence prediction;
- **deep belief networks** (DBN) a type of generative model that consists of multiple layers of stochastic, latent variables, often used for unsupervised learning tasks;
- **kernel machines** (KM) a type of sub-symbolic predictor that uses kernel functions to map input data into a higher-dimensional space, allowing for non-linear decision boundaries.

Figure 3.2 illustrates the distribution of surveyed SKI methods based on the type of sub-symbolic predictor they target.

One reason why the vast majority of methods rely on NNs is straightforward: methods tailored to graph neural networks (GNNs) (resp., CNNs) assume the networks to accept specific kinds of data as input, e.g., graphs (resp., images), while ordinary feed-forward NNs accept raw vectors of real numbers. Another reason is that NNs are the most popular sub-symbolic predictors in the ML community, and they are often used as a default choice for many tasks. Finally, NNs are highly flexible and easy to manipulate, making them ideal for SKI methods.

3.1.4 How to inject

How can symbolic knowledge, such as the ones in Equations (3.1) and (3.2), be injected into sub-symbolic predictors? To answer this question, scientists have designed several strategies, which can be broadly grouped into three main categories [Cia+24]: structuring, guided learning, and embedding. It may occur that a method uses more than one strategy at a time. Figure 3.3 illustrates the distribution of surveyed SKI methods based on their strategy of choice. The survey revealed that all three strategies are widely used, with learning being the most frequent but not dominant.

The main entities involved in the SKI process – and that are common to virtually all SKI methods – are:

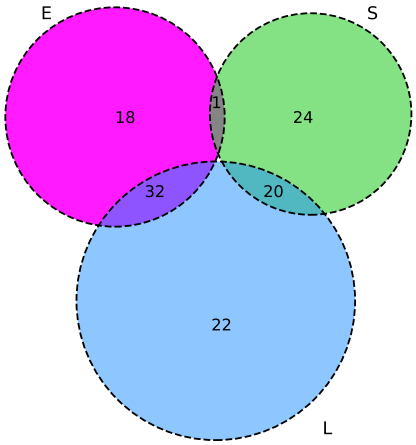


Figure 3.3: Venn diagram categorising SKI methods w.r.t. the *injection* strategy type: structuring (S), embedding (E), or guided learning (L).

- **symbolic knowledge:** in the vast majority of cases, this is a set of logic formulas or a KB that encodes the symbolic knowledge to be injected;
- **parsed knowledge:** a syntactic structure that represents the *parsed* symbolic knowledge, often in the form of a abstract syntax tree (AST) or graph;
- **sub-symbolic component:** a sub-symbolic instance built upon the parsed symbolic knowledge;
- **uneducated sub-symbolic predictor:** a sub-symbolic predictor that has not been enriched with the symbolic knowledge;
- **educated sub-symbolic predictor:** the final sub-symbolic predictor that has been enriched with the symbolic knowledge.

Structuring By predictor structuring, or simply structuring, we identify all those methods that use a part (or the whole) of a sub-symbolic predictor to *mirror* the symbolic knowledge via its own internal structure. A predictor is either created or extended to mimic the behavior of the symbolic knowledge. For instance, when the predictors are NNs, their internal structure is crafted to represent logic predicates via neurons, and logic connectives via synapses. Figure 3.4 illustrates the workflow of the structuring strategy. Usually, in the structuring strategy, the symbolic knowledge is in the form of logic formulas (ϕ) (e.g., subsets of FOL). The formulas are then parsed (Π) into a syntactic structure (e.g., AST). At this point, a *fuzzification* (ζ) step is performed to convert the *boolean* (also the term *crisp* is used in the literature) – because a logic predicate is either *true* or *false* – symbolic

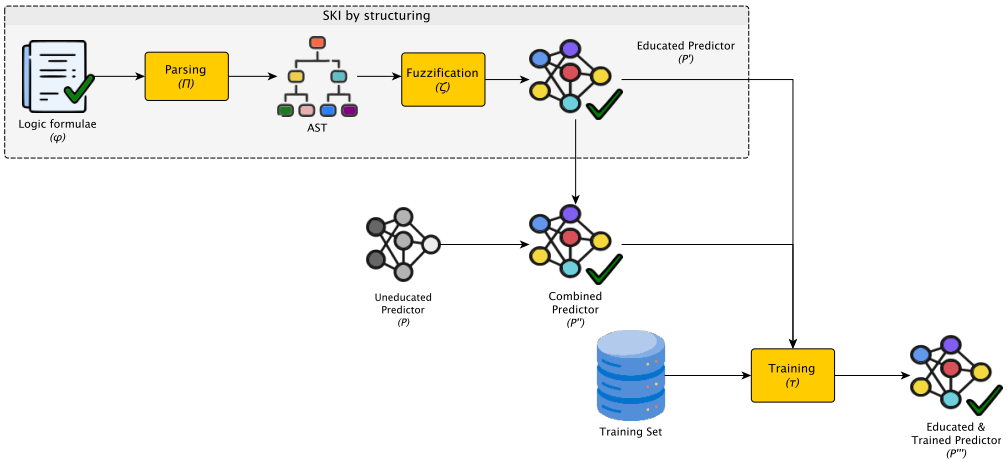


Figure 3.4: Workflow of the structuring strategy. The symbolic knowledge is parsed and used to create a sub-symbolic predictor – or part of it – that mimics the symbolic knowledge. The resulting predictor can then either be trained on data as is, or combined with an uneducated predictor to create a new educated predictor that is then trained on data.

knowledge into a *fuzzy* interpretation of it, which is more suitable for sub-symbolic predictors. This fuzzy interpretation is often *continuous* and bounded to a predefined range, such as $[0, 1]$. The property of continuity is a key requirement for training sub-symbolic structures via gradient descent (e.g., NNs or part of them). In practice, ζ is implemented as a mapping function that transforms logic operators into continuous functions (e.g., see Sections 4.2.2 and 4.3.2). The result of the fuzzification process is an *educated* predictor (P') that mimics the symbolic knowledge ϕ. At this point, P' can be used as is to make predictions, or it can be further trained (τ) to improve its predictive performance (P'''). Another option is to combine P' with an *uneducated* predictor (P) to create a new predictor P'' that embodies both the symbolic knowledge and the *uneducated* predictor's capabilities. After τ, the predictor is now both *educated* P''' with knowledge and trained on data.

Guided learning By guided learning, we refer to all those methods that affect the sub-symbolic predictor's training process by using the symbolic knowledge as a *constraint/guidance*. Usually, SKI algorithms based on this strategy target sub-symbolic predictors that can be trained with gradient descent, such as NNs.

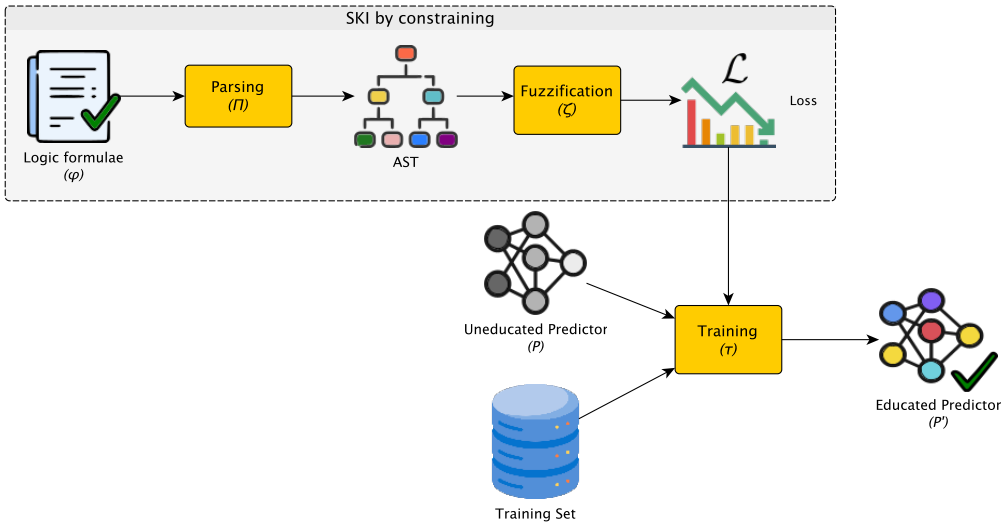


Figure 3.5: SKI workflow of the guided learning strategy. The symbolic knowledge is used to guide the training of a sub-symbolic predictor, which is then trained on data.

Figure 3.5 illustrates the workflow of the guided learning strategy. Like in the case of structuring, the symbolic knowledge is parsed and fuzzified. Differently from structuring, the result of the fuzzification process is not a subcomponent of the predictor's architecture (e.g., neurons, connections), but rather a set of continuous numeric functions. Conceptually, these functions represent the degree of violation of the symbolic knowledge by the predictor's predictions. The higher the degree of violation, the higher the value of the function. The symbolic-derived functions are then included in the loss function of the predictor, which is then trained on the training set. The combination between the "classical" loss function (e.g., cross-entropy, mean squared error, etc.) and the symbolic-derived functions can be done in many ways, but it is usually done via a weighted sum.

Embedding In the embedding strategy, the symbolic knowledge is converted – i.e., *embedded*– into numeric-array representations. Figure 3.6 illustrates the workflow of the embedding strategy. The output of the fuzzification process is a set of numeric arrays (e.g., vectors, matrices, and tensors) that represent the symbolic knowledge. These arrays are then provided as input to the sub-symbolic predictor. This is the common strategy exploited by the KG [Lam+20] embedding community as well as by GNNs [Wan+17].

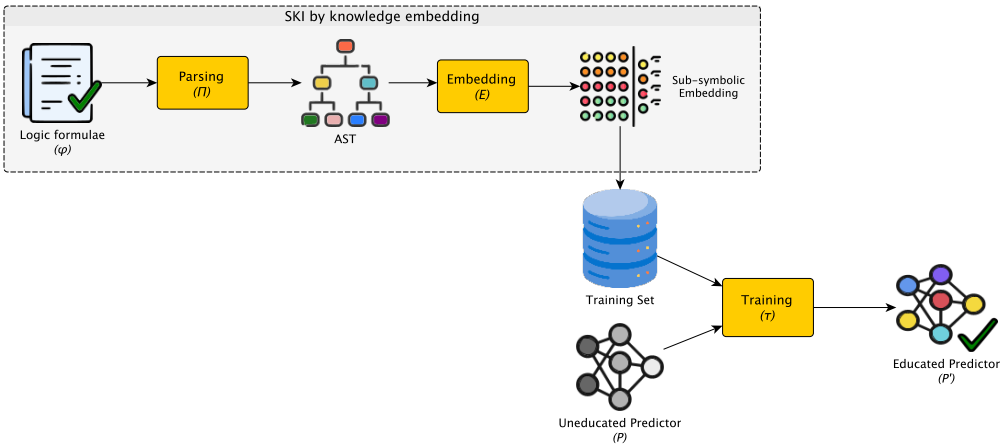


Figure 3.6: SKI workflow of the embedding strategy. The symbolic knowledge is parsed and embedded into a sub-symbolic predictor, which is then trained on data.

3.1.5 Limitations and challenges of SKI

Despite the numerous advantages of SKI, there are several limitations and challenges that need to be addressed. Here, we elicit the most common and relevant limitations and challenges of SKI that we encountered in the literature:

- **Lack of generality**, many SKI methods are tailored to domain-specific tasks. Often, even if they are presented as general-purpose, their implementation is not and it requires significant effort to adapt them to different tasks or domains;
- **Lack of reproducibility**, the experiments presented – if any – in the papers are often not reproducible. This happens because the authors do not provide enough details about the implementation of the method (e.g., they do not mention the value of some key hyperparameters), or they do not share the code and data used in the experiments. Only 49 out of 117 surveyed methods provide public code base for the experiments[2];
- **Lack of availability**, the vast majority of SKI methods is not available as open-source software. Only 11 out of 117 surveyed methods provide a public library for the method[2]. Authors usually do not mention any code base in their papers, and if they do, the code can lack the necessary dependencies,

[2]see Table B of the supplementary material of [Cia+24]

lack documentation, or it could have been removed from public repositories;

- **Limits of the symbolic knowledge**, despite the variety of symbolic knowledge that can be injected (see Figure 3.1), there are still many limitations. Indeed, the claimed expressiveness of the logic formalism supported by a method is often not fully exploited in practice. In particular, FOL allows for the representation of recursive terms, but most SKI methods do not support them because of the target predictors. Target predictors are almost always NNs, which cannot support recursion in their architecture—modern NNs are DAGs.

3.2 Symbolic knowledge extraction

SKE is the process of extracting symbolic knowledge from sub-symbolic predictors. Here, we provide the broad definition of SKE that we adopt in this thesis:

Definition 3.2 (SKE) *any* ***algorithmic*** *procedure accepting* ***trained*** *sub-symbolic predictors as input and producing* ***symbolic*** *knowledge as output so that the extracted knowledge reflects the behavior of the predictor with high* ***fidelity*** *[Cia+24].*

Once again, the definition is broad because the amount of contributions in the literature is vast and varied, they often use different terminologies and come from different communities. SKE is conceptualized as a class of algorithms, which are finite-step procedures defined by their inputs and outputs.

The input to SKE procedures must be trained MLs predictors. There are no restrictions on the type of predictor, meaning that SKE methods can, in principle, be applied to any predictor family. However, this requirement implies that the predictor must have already undergone training and achieved satisfactory performance for its intended task. Thus, within an ML workflow, SKE is performed after the training and validation phases are complete.

The output of SKE procedures is symbolic knowledge, which is broadly defined as human-intelligible information. This can include logic formulas, decision trees, or even plain human-readable text. For an algorithm to qualify as a valid SKE procedure, the extracted knowledge must closely reflect the behavior of the original

predictor within the domain it was trained for. This fidelity is typically measured using a fidelity score, which evaluates how well the extracted knowledge replicates the predictor's behavior for the given task and domain.

The extracted knowledge should ideally function as a predictor itself, allowing it to be queried in the same way as the original predictor. For example, if the original predictor is an image classifier, the extracted knowledge should enable an intelligent agent to classify similar images and produce equivalent results. This agent could be either computational (e.g., a software program) or human, depending on whether the extracted knowledge is machine or human interpretable. The use of logic-based knowledge as the target of SKE is particularly advantageous, as it supports both machine and human interpretability.

3.2.1 Motivations and goals

SKE serves multiple purposes, including: *(i)* the inspection of the internal operations of opaque predictors, which are otherwise treated as black boxes, *(ii)* the automation and acceleration of the process of acquiring symbolic knowledge, eliminating the need for manually crafting KBs, *(iii)* providing valuable capabilities within the scope of XAI to analyze black-box predictors.

Item *(i)* is particularly important for understanding the behavior of opaque predictors. By inspecting their internal operations, SKE allows researchers and practitioners to identify patterns in mispredicted inputs or even detect correctly predicted patterns that rely on unethical or biased decision processes. This capability is crucial for ensuring the ethical and transparent use of ML systems, especially in sensitive domains such as healthcare or finance.

Item *(ii)* highlights another key advantage of SKE: the ability to automate the extraction of symbolic knowledge. This eliminates the labor-intensive process of manually crafting KBs, which often requires significant domain expertise and time. By automating this process, SKE not only accelerates knowledge acquisition but also enables the reuse of extracted knowledge across different tasks or systems, provided the knowledge is represented in a compatible format.

Finally, Item *(iii)* emphasizes the role of SKE in XAI. The extracted knowledge can be used to construct explanations for the behavior of black-box predic-

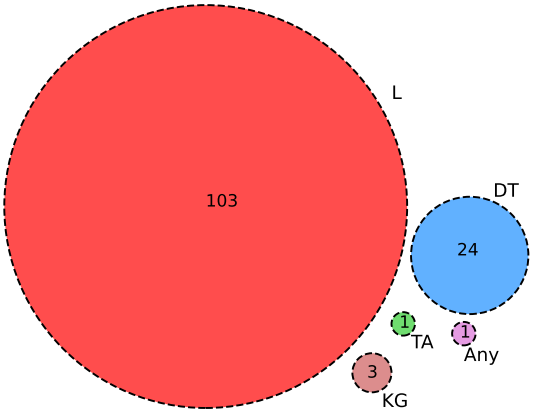


Figure 3.7: Venn diagram categorising SKE methods w.r.t. the formalism (*shape*) used to represent the extracted symbolic knowledge: rule lists (L), decision trees (DT), decision tables (TA), or knowledge graphs (KG).

tors, making their decision-making processes more interpretable. In some cases, this knowledge may even serve as an interpretable surrogate model, replacing the original predictor if it achieves a high fidelity score [Cia+20b]. Additionally, SKE enables the comparison of multiple black-box predictors performing the same task, highlighting their differences and commonalities. It also facilitates the merging of knowledge extracted from various predictors, even if they belong to different families, as long as the same representation format is used [Cia+19].

3.2.2 What to extract

What kind of symbolic knowledge can be extracted? Symbolic knowledge extracted from MLs predictor should ideally mimic the behavior of the predictor itself. For supervised ML, this means that the extracted knowledge should represent a function mapping input features to output features, such as classes in classification tasks. These functions can be expressed in symbolic formats, which vary in both syntax and semantics.

Shape of the extracted knowledge From a syntactic perspective, the formalism used to represent the extracted symbolic knowledge – also referred as *shape* for ease of reference in the rest of the thesis – is crucial for its interpretability. The

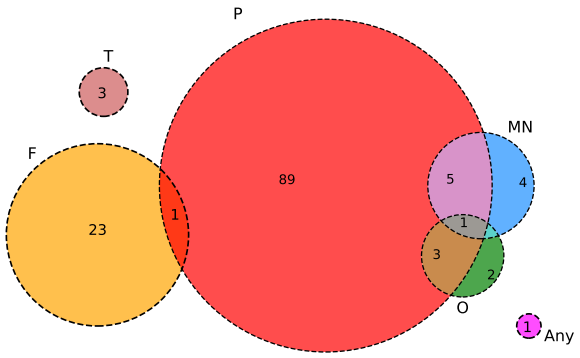


Figure 3.8: Pie chart categorising SKE methods w.r.t. the semantic *expressiveness* of the extracted symbolic knowledge: propositional rules (P), fuzzy rules (F), oblique rules (O), m-of-n rules (MN), or triplets (T).

most common formats include decision rules and decision trees, as they are widely recognized for their human-comprehensibility. Other formats, such as decision tables and knowledge graphs, are also used, though less frequently. Regardless of the format, the extracted knowledge typically uses symbols to represent the same input and output features as the original ML predictor.

We identify four primary syntactic shapes for extracted symbolic knowledge:

- **Lists of rules:** Sequences of logic rules presented in a predefined order.
- **Decision trees:** Hierarchical structures where each node represents a decision rule (see Section 2.2.1.7).
- **Decision tables:** Tabular representations specifying conclusions for different sets of conditions. These can be exhaustive, listing all possible combinations, or incomplete, covering only a subset of cases. Each row (or column) corresponds to a rule, with cells indicating variable values. An example is provided in the supplementary material.
- **Knowledge graphs:** Graph-based representations of entities and their relationships (see Section 2.2.1.5).

Figure 3.7 illustrates the distribution of these shapes across surveyed SKE methods. Rule lists are the most prevalent, likely due to their simplicity and algorithmic tractability.

Expressiveness of the extracted knowledge From a semantic perspective, the expressiveness of the extracted symbolic knowledge depends on the logical constructs it employs. Statements may include predicates, relations, logic connectives, and arithmetic comparators, which determine what can be expressed. We categorize the semantic formats of extracted symbolic knowledge as follows:

- **Propositional rules:** Statements involving Boolean input/output features connected by logic operators such as negation, conjunction, and disjunction. Arithmetic comparisons between continuous features and constants are also included in this category.
- **Fuzzy rules:** Extensions of propositional rules where truth values range continuously between 0 and 1.
- **Oblique rules:** Rules with conditions expressed as inequalities involving linear combinations of input variables, allowing comparisons between features.
- **m-of-n rules:** Rules that are true if at least m out of n conditions are satisfied. These are a concise way of representing disjunctions of conjunctions.
- **Triplets:** Representations commonly used in knowledge graphs, consisting of subject-predicate-object triples (see Section 2.2.1.5).

Figure 3.8 summarizes the occurrence of these semantic formats in surveyed SKE methods. Propositional rules dominate due to their simplicity and tractability. They allow a divide-and-conquer approach, focusing on one input feature at a time, which simplifies the extraction process.

3.2.3 Where to extract

The sort of ML predictors from which symbolic knowledge can be extracted is strongly related to the *translucency* of the SKE algorithms. The translucency of a SKE algorithm refers to its ability to access the internal structure of the predictor, such as its parameters, weights, and architecture. So, in other words, the question of where to extract symbolic knowledge is related to the question of how SKE algorithms work. There are two ways for SKE methods to deal with sub-symbolic predictors:

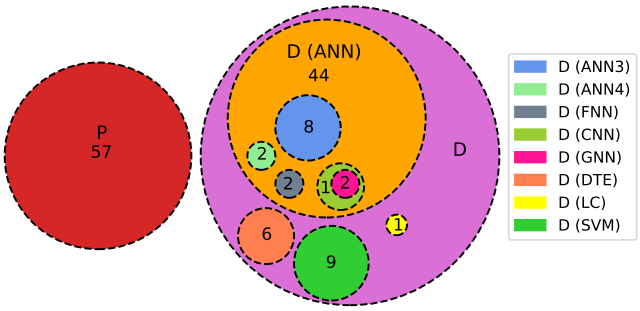


Figure 3.9: Venn diagram categorising SKE methods w.r.t. the *translucency* of the algorithm: decompositional (D) or pedagogical (P). For decompositional methods, we report the target predictor type: ANN¡n¿ = artificial NN (possibly having exactly ¡n¿ layers), CNN = convolutional NN, GNN = graph NN, FNN = fuzzy NN, SVM = support vector machines, DTE = decision tree ensembles, LC = linear classifiers.

- **Decompositional** → if the method needs to inspect – even partially – the internal structure of the predictor, such as its parameters, weights, and architecture;
- **Pedagogical** → if the method does not need to inspect the internal structure of the predictor, but rather it can treat it as a black box and it relies on the input/output behavior of the predictor.

The different nature of decompositional SKE methods is detailed in Figure 3.9. We observe that for what concern decompositional methods, the vast majority of them are designed to work with NN of various types and different number of hidden layers. Also SVMs and decision tree ensambless (DTEs) are targeted by some decompositional methods. Around 43% of the surveyed methods are pedagogical, meaning that these methods can work with virtually any type of sub-symbolic predictor, as long as it can be queried for input/output pairs. Therefore, in general SKE methods can be applied to any type of predictor, but for the decompositional ones, NNs are definitely the most targeted type.

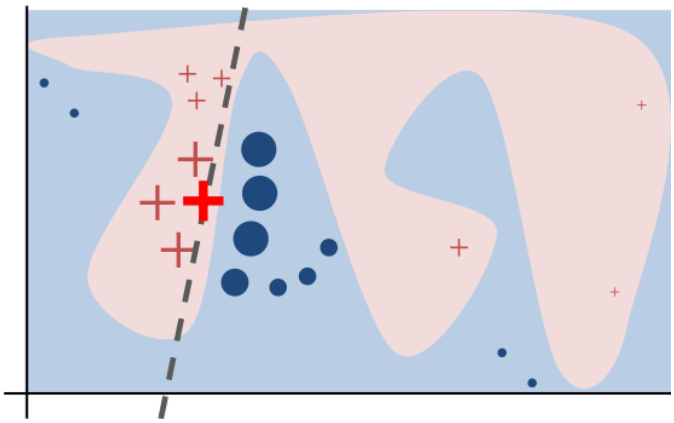


Figure 3.10: Simple example to of how a local explanation can be provided for a specific input. The sub-symbolic model's decision function f is represented by the blue and pink background, which cannot be approximated well by a linear model. The bold red cross is the instance being explained. A SKE method can sample inctances, get prediction using f, and weights them by the proximity to the instance being explained (the bigger the cross/circle, the closer to the instance). The dashed line is the trained interpretable model that is locally faithful to the sub-symbolic model.

3.2.4 How to extract

The way SKE methods extract symbolic knowledge highly depends on the translucency of the method itself [Cia+24]. *Pedagogical* methods treat the underlying predictor as an oracle to be queries for predictions the symbolic knowledge should mimic. In most of these cases, the SKE method exploits a *surrogate* model, which is a simpler, more interpretable model that approximates the behavior of the original predictor. This model can be used as the final output of the SKE process if it is already in a symbolic or interpretable (e.g., a linear model) format, or it can be further processed to extract symbolic knowledge. *Decompositional* methods, on the other hand, require access to the internal structure of the predictor and each method has its own way of extracting symbolic knowledge. Another important aspect of SKE methods when it comes to how symbolic knowledge is extracted is the *granularity* of the extraction.

Local explanations When the interest in the extracted symbolic knowledge is focused on a specific input, i.e., we want to know why a predictor produced a particular outcome, the SKE method is said to produce *local explanations*. A local

explanation is symbolic knowledge that explains only the behavior of the predictor for a specific input. The knowledge is not necessary encoded into logic rules, it can be provided for instance through *feature importance* scores, which indicate the relevance of each input feature for the prediction [RSG16]. Figure 3.10 illustrates a simple example of how a local explanation can be provided for a specific input using a pedagogical SKE method.

Global explanations Conversely, when the interest is in understanding the overall behavior of the predictor, we say that the SKE method produces *global explanations*. Global explanations aim to capture the general patterns and rules that govern the predictor's behavior across all inputs, rather than focusing on individual instances. For example, this can be done by training a surrogate model – if we want to use a pedagogical SKE method – using for instance the training set used to train the predictor. However, instead of using the labels of the training set, the surrogate model is trained using the predictions of the original predictor.

3.2.5 Limitations and challenges of SKE

SKE methods predominantly focus on supervised learning tasks, such as classification and regression. However, they do not yet address unsupervised learning tasks, like clustering, or reinforcement learning tasks, such as optimal policy search. This limitation restricts their applicability to a broader range of ML paradigms.

One might argue that certain clustering algorithms, such as k-nearest neighbors, are inherently interpretable. Despite this, they still operate on numerical data, which can obscure the underlying decision-making process. *Pedagogical* SKE methods could potentially be adapted to clustering predictors with minimal adjustments. This is because trained clustering predictors can be treated as classifiers over anonymous classes.

Similarly, it is conceivable to apply SKE techniques to predictors trained using reinforcement learning. For instance, symbolic knowledge could be extracted to explain the learned policies or value functions. This would provide insights into the decision-making process of reinforcement learning agents.

Future research should explore the extension of SKE methods to unsupervised

and reinforcement learning tasks. Such advancements would significantly broaden the scope and utility of SKE in AI systems.

The majority of SKI algorithms rely on symbolic knowledge represented as KGs or propositional logic (Figure 3.1). While these representations are widely used, they are significantly less expressive compared to FOL. In particular, they lack support for recursion and function symbols, which limits the complexity and depth of the knowledge that can be injected into sub-symbolic predictors. This limitation arises because most ML predictors, such as NNs, are inherently acyclic structures. As a result, integrating recursive logic or arbitrarily deep data structures into these predictors is challenging and often leads to severe information loss due to approximations. Addressing these limitations requires innovative approaches to enable the injection of more expressive logics, such as recursive clauses or function symbols, into sub-symbolic models. Future research should explore methods to overcome these challenges, potentially by designing predictors capable of handling recursive structures or by developing advanced approximation techniques.

Part II

Methods, metrics and platform for SKI & SKE

Chapter 4

SKI: methods and contributions

How to design and implement *SKI methods*?
– **RQ0 and RQ3**

Contents

In this chapter we present two main algorithmic contributions to the field of SKI. First, we provide the motivations behind the design and development of these contributions in Section 4.1. Second, we detail the contributions themselves in Sections 4.2 and 4.3, and we report the results of the empirical evaluation of these contributions.

4.1 Motivations

Aware by the limitations reported in Section 3.1.5, we wanted to develop novel SKI methods that could be used in many different domains. For this reason, these methods include all the elements described in Section 3.1.4. In particular, a *parser* is provided to convert symbolic knowledge into a format that can be injected into a neural network. In this way, we are not bounded to a specific knowledge base nor to a specific domain. The transformation of the parsed rules into a numeric interpretation is automatically performed by a *fuzzification step*. Finally, the sub-symbolic component is injected into a neural network, which is trained to learn the symbolic knowledge. Another limitation that we wanted to address is the expressiveness of the symbolic knowledge that can be injected. What we target in these two works is knowledge expressed in *stratified Datalog with negation*.

4.1.1 A good tradeoff language for SKI

Stratified Datalog with negation is a variant of Datalog [Mai+18] with no recursion – neither direct nor indirect – yet supporting negated atoms. We choose Datalog because of its expressiveness (strictly higher than propositional logic) and its acceptable limitations. The lack of recursion, in particular, prevents issues when it comes to convert formulas into neural structures (which are DAGs). Since we rely on Datalog with negation, we allow atoms in the bodies od clauses to be negated. Using the same notation as in Section 2.2.1.3, in case the i^{th} atom in the body of some clause is negated, we write $\neg b_i$. There, each atom h, b_1, b_2, ... may be a predicate of arbitrary arity.

An l-ary predicate p denotes a relation among l entities: $p(t_1, \ldots, t_l)$, where each t_i is a term, i.e., either a constant (denoted in `monospace`) representing a particular entity, or a logic variable (denoted by *Capitalised Italic*) representing some unknown entity or value. Well-known binary predicates – e.g., $>$, $<$, $=$ – are admissible, too, and retain their usual semantics from arithmetic. For the sake of readability, we may write these predicates in infix form—hence $>(X, 1) \equiv X > 1$.

To support injection into a particular NN, we further assume the input knowledge base defines one (and only one) outer relation – say `output` or `class` –

involving as many variables as the input and output features the NN has been trained upon. That relation must be defined via one clause per output neuron. Yet, each clause may contain other predicates in their bodies, in turn defined by one or more clauses. In that case, since we rely on stratified Datalog, we require the input knowledge to not include any (directly or indirectly) recursive clause definition.

For example, for a 3-class classification task, any provided knowledge base should include a clause such as the following one:

$$
\begin{aligned}
\texttt{class}(\bar{X}, y_1) &\leftarrow \texttt{p}_1(\bar{X}) \wedge \texttt{p}_2(\bar{X}) \\
\texttt{p}_1(\bar{X}) &\leftarrow \ldots \\
\texttt{p}_2(\bar{X}) &\leftarrow \ldots \\
\texttt{class}(\bar{X}, y_2) &\leftarrow \texttt{p}'_1(\bar{X}) \wedge \texttt{p}'_2(\bar{X}) \\
\texttt{p}'_1(\bar{X}) &\leftarrow \ldots \\
\texttt{p}'_2(\bar{X}) &\leftarrow \ldots \\
\texttt{class}(\bar{X}, y_3) &\leftarrow \texttt{p}''_1(\bar{X}) \wedge \texttt{p}''_2(\bar{X}) \\
\texttt{p}''_1(\bar{X}) &\leftarrow \ldots \\
\texttt{p}''_2(\bar{X}) &\leftarrow \ldots
\end{aligned}
$$

where $\bar{X}$ is a tuple having as many variables as the neurons in the output layer, y_i is a constant denoting the i-th class, and $\texttt{p}_1$, $\texttt{p}_2$, $\texttt{p}'_1$, $\texttt{p}'_2$, $\texttt{p}''_1$, $\texttt{p}''_2$ are ancillary predicates defined via Horn clauses as well.

4.2 An example of a guided learning SKI algorithm

In this section we present the paper "A view to a KILL: Knowledge Injection via Lambda Layer" [MCO22a], presented at the 23rd Workshop "From Objects to Agents" (WOA 2022) [1]. The paper introduces a novel SKI method, called knowledge injection via lambda layers (KILL), which allow to inject symbolic

[1] https://sites.google.com/view/woa2022/

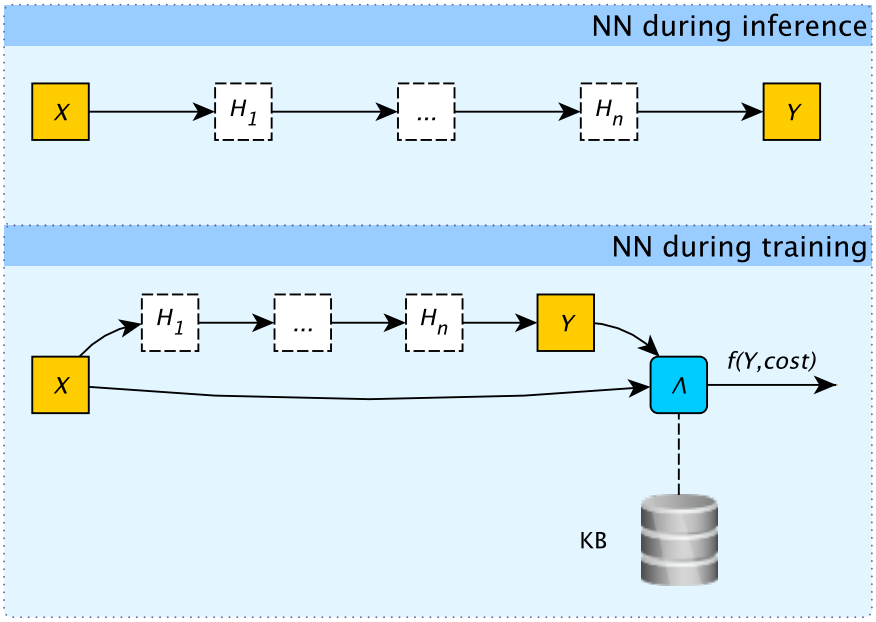


Figure 4.1: General architecture of a NN with n hidden layers, X and Y are respectively the input and output layers. Before training, KILL is applied to the predictor so that a new model constrained by the knowledge base (KB) is obtained. The Λ-layer is added to the output layer, and then removed for the inference phase.

knowledge in stratified Datalog with negation into NNs of any shape. KILL does not require the input formulas to be *ground*, and it does not impose any constraint on the NN undergoing injection. The method acts directly at the backpropagation level, by increasing the penalty to be back-propagated whenever the NN output is violating the knowledge to be injected. For this reason, KILL falls under the *guided learning* strategy, as described in Section 3.1.4.

4.2.1 Λ-layer

The idea behind KILL is to perform injection during training. The injection is performed by appending one further layer – the Λ-*layer* – at the output end of the NN, and by training the overall network as usual (e.g., via gradient descent or similar). The Λ-layer introduces an error whenever the prediction from the network's output layer violates the symbolic knowledge being injected. This error influences the gradient descent or other optimization functions, discouraging violations of

the symbolic knowledge. In essence, the NN learns inductively to avoid wrong predictions by penalizing violations during training. To achieve this, the Λ-layer employs a custom activation function that modifies the output of the network's original output layer. Additionally, the symbolic knowledge must be numerically interpreted, converting logical formulas into real-valued functions to compute error values. Once the training phase is complete, the Λ-layer can be removed, leaving the original network architecture intact.

To elaborate on the Λ-layer, we consider a symbolic knowledge base, denoted as $\mathcal{K}$, to be injected into a feedforward NN of arbitrary depth, denoted as $\mathcal{N}$. Let $\mathcal{N}$ have an input layer $\mathbf{X}$ and an output layer $\mathbf{Y}$, where $\mathbf{Y}$ is assumed to be of shape $n \times 1$. This discussion applies to layers of any shape, including multidimensional ones. No assumptions are made about the activation function of $\mathbf{Y}$, the topology or nature of the hidden layers, or the shape of $\mathbf{X}$. The output of $\mathbf{Y}$ is denoted as $\mathbf{y} = [y_1, \ldots, y_n]$, representing the network's prediction for a given input $\mathbf{x}$. The knowledge base $\mathcal{K}$ consists of n rules, $\mathcal{K} = \{\varphi_1, \ldots, \varphi_n\}$, where each rule φ_i constrains the relationship between $\mathbf{x}$ and y_i.

To inject $\mathcal{K}$, the Λ-layer is added to the network architecture, as illustrated in Figure 4.1. This layer is densely connected to both $\mathbf{X}$ and $\mathbf{Y}$, and its activation function introduces a penalty on y_i whenever the corresponding rule φ_i is violated for an input-output pair $(\mathbf{x}, \mathbf{y})$. The output of the Λ-layer, denoted as $\boldsymbol{\lambda}$, is defined as:

$$\boldsymbol{\lambda} = \mathbf{y} \times (1 + \mathbf{C}(\mathbf{x}, \mathbf{y}))$$

where $\mathbf{C}(\mathbf{x}, \mathbf{y})$ is a positive penalty vector representing the cost of modifying the network's output $\mathbf{y}$. The penalty vector $\mathbf{C}(\mathbf{x}, \mathbf{y})$ is given by:

$$\mathbf{C}(\mathbf{x}, \mathbf{y}) = [c_1(\mathbf{x}, y_1), \ldots, c_i(\mathbf{x}, y_i), \ldots, c_n(\mathbf{x}, y_n)]$$

where $c_i : \mathbf{X} \times \mathbf{Y} \to [0, 1]$ interprets the rule φ_i as a cost in the range $[0, 1]$ for the given values of $\mathbf{x}$ and y_i. A penalty is applied to the i-th neuron of $\mathbf{Y}$ whenever the corresponding rule φ_i is violated. The penalty is higher when the violation is severe and approaches zero when the violation is minimal or absent.

Formula	C. interpretation	Formula	C. interpretation
$[\![\neg\phi]\!]$	$\eta(1 - [\![\phi]\!])$	$[\![\phi \leq \psi]\!]$	$\eta([\![\phi]\!] - [\![\psi]\!])$
$[\![\phi \wedge \psi]\!]$	$\eta(max([\![\phi]\!], [\![\psi]\!]))$	$[\![class(\bar{X}, \mathtt{y}_i) \leftarrow \psi]\!]$	$[\![\psi]\!]^*$
$[\![\phi \vee \psi]\!]$	$\eta(min([\![\phi]\!], [\![\psi]\!]))$	$[\![\text{expr}(\bar{X})]\!]$	$\text{expr}([\![\bar{X}]\!])$
$[\![\phi = \psi]\!]$	$\eta(\lvert[\![\phi]\!] - [\![\psi]\!]\rvert)$	$[\![\mathtt{true}]\!]$	0
$[\![\phi \neq \psi]\!]$	$[\![\neg(\phi = \psi)]\!]$	$[\![\mathtt{false}]\!]$	1
$[\![\phi > \psi]\!]$	$\eta(0.5 - [\![\phi]\!] + [\![\psi]\!])$	$[\![X]\!]$	x
$[\![\phi \geq \psi]\!]$	$\eta([\![\psi]\!] - [\![\phi]\!])$	$[\![\mathtt{k}]\!]$	k
$[\![\phi < \psi]\!]$	$\eta(0.5 + [\![\phi]\!] - [\![\psi]\!])$	$[\![p(\bar{X})]\!]^{**}$	$[\![\psi_1 \vee \ldots \vee \psi_k]\!]$

* encodes the penalty for the i^{th} neuron

** assuming predicate p is defined by k clauses of the form:
$p(\bar{X}) \leftarrow \psi_1, \ \ldots, \ p(\bar{X}) \leftarrow \psi_k$

Table 4.1: Logic formulas' encoding into real-valued functions. There, X is a logic variable, while x is the corresponding real-valued variable, whereas is $\bar{X}$ a tuple of logic variables. Similarly, $\mathtt{k}$ is a numeric constant, and k is the corresponding real value, whereas $\mathtt{k}_i$ is the constant denoting the i^{th} class of a classification problem. Finally, $\text{expr}(\bar{X})$ is an arithmetic expression involving the variables in $\bar{X}$. The η function is a scaling function described in Equation (4.1).

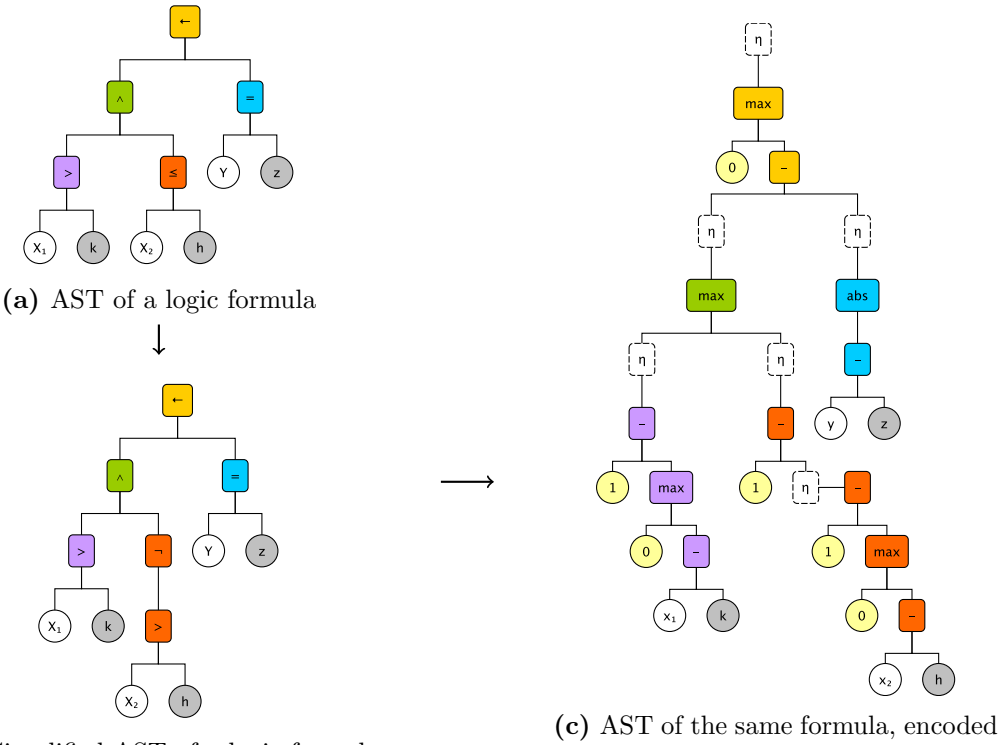
4.2.2 KILL fuzzifier

Before injecting symbolic knowledge into a neural network, each formula associated with an output neuron must be converted into a real-valued function to compute the cost of violating that formula.

This conversion relies on a multivalued interpretation of logic inspired by Łukasiewicz's logic [Hay63]. Each formula is encoded using the $[\![\cdot]\!]$ function, which maps logical formulas to real-valued functions. These functions accept real vectors of size $m+n$ as input and return scalars in $\mathbb{R}$ as output. The resulting scalars are clipped to the $[0, 1]$ range using the $\eta : \mathbb{R} \to [0, 1]$ function, defined as:

$$\eta(x) = \begin{cases} 0 & \text{if } x \leq 0, \\ x & \text{if } 0 < x < 1, \\ 1 & \text{if } x \geq 1. \end{cases} \tag{4.1}$$

The values obtained from $\eta(x)$ represent penalties, as discussed in Section 4.2.1. The penalty for the i^{th} neuron violating rule φ_i is expressed as $c_i(\mathbf{x}, y_i) =$



(a) AST of a logic formula

(b) Simplified AST of a logic formula

(c) AST of the same formula, encoded as real-valued function

Figure 4.2: Example of the encoding process of logic formulas into real-valued functions. Only AST are depicted. Boxes coloured in the same way represent the encoding of a given operator through each encoding step. For instance, operator $<$ (red) is first converted into a negated $\geq$, and then into a combination of *max* and subtractions.

$\eta([\![\varphi_i]\!](\mathbf{x}, y_i))$.

The $[\![\cdot]\!]$ encoding function is recursively defined in Table 4.1. To compute the penalty $c_i(\mathbf{x}, y_i)$ for the i^{th} neuron, KILL identifies the Datalog rule of the form:

$$\texttt{class}(\bar{X}, y_i) \leftarrow \psi.$$

The method focuses on the body ψ of the rule, ignoring its head, as the head specifies the expected output for the rule. If ψ contains predicates p_1, p_2, defined by one or more clauses in the knowledge base, these predicates are replaced by the disjunction of the bodies of all clauses defining them. This process is repeated until ψ contains only binary expressions involving input variables, constants, arithmetic operators, and logical connectives. Finally, operators and connectives are replaced

Class	**Train. Instances**	**Train. Freq. (%)**	**Test Instances**	**Test Freq. (%)**
high card	12,493	49.95	501,209	50.12
pair	10,599	42.38	422,498	42.25
two pairs	1,206	4.82	47,622	4.76
three of a kind	513	2.05	21,121	2.11
straight	93	0.37	3,885	0.39
flush	54	0.22	1,996	0.2
full house	36	0.14	1,424	0.14
four of a kind	6	0.024	230	0.023
straight flush	5	0.02	12	0.001
royal flush	5	0.02	3	$3 \cdot 10^{-4}$
Total	25,010	100	1,000,000	100

Table 4.2: Poker hand dataset (PHDS) statistics per class. The dataset is heavily imbalanced, with the *high card* class being the most frequent. Considering together the two most frequent classes, *high card* and *pair*, they account for over 92% of the training instances.

by continuous functions, as detailed in Table 4.1. The entire process produces a real-valued interpretation of the original formula, which KILL uses to compute $c_i(\mathbf{x}, y_i)$.

Figure 4.2 illustrates an example of the encoding process. The example formula is:

$$\texttt{class}(X_1, X_2, z) \leftarrow (X_1 \geq k) \wedge (X_2 \geq h),$$

where k, h, and z are numeric constants, X_1 and X_2 are input variables, and Y is an output variable. Figure 4.2a shows the AST of this formula. Figure 4.2b depicts the same AST after replacing the $\leq$ operator with a negated $>$ operator. Finally, Figure 4.2c shows the AST of the encoded function. A public implementation of KILL is available as part of the PSyKI [MCO22d] library[2].

4.2.3 Validation

The method has been validated on the poker hand dataset (PHDS) [CO02], a dataset for poker hand classification. To ensure reproducibility, the code used for our experiments is publicly available[3]. The task consists in a multiclass classification problem on a finite – yet very large – discrete domain. Classes are overlapped and heavily imbalanced, but exact classification rules can be written in logic formulas. PHDS has $1,025,010$ records[4], each with 10 features, and 1 output class. Each record represents a poker hand of 5 cards, each card is identified by two features: a rank and a suit. Suit is a categorical feature (e.g., `spades`, `hearts`, `diamonds`, `clubs`), while rank is a numeric feature (e.g., 1 for Ace, 2 for Two, ..., 13 for King). Each hand can be classified as one of 10 different classes denoting the poker hand type (e.g., `high card`, `pair`, `two pairs`, `three of a kind`, etc.). Table 4.2 reports the statistics of the dataset per class for both training and test sets.

PHDS logic rules We define a *class* rule for each class, encoding the correct way of classifying a poker hand. For instance, let $\{R_1, S_1, \ldots, R_5, S_5\}$ be the logic variables representing a poker hand (i.e., R for rank and S for suit), then for class `flush` we have:

$$\begin{aligned} \mathit{class}(R_1, S_1, \ldots, R_5, S_5, \texttt{flush}) &\leftarrow \mathit{flush}(R_1, S_1, \ldots, R_5, S_5) \\ \mathit{flush}(R_1, S_1, \ldots, R_5, S_5) &\leftarrow S_1 = S_2 \wedge S_1 = S_3 \wedge S_1 = S_4 \wedge S_1 = S_5 \end{aligned} \tag{4.2}$$

All other rules have the same structure as Equation (4.2): the left-hand side declares the expected class, while the right-hand side describes the necessary conditions for that class—possibly, via some ancillary predicates such as *flush*. Table 4.3 provides an overview of all the rules we rely upon in our experiments.

Methodology We adopt the same data partitioning strategy proposed by the dataset authors [CO02]. Specifically, the training set consists of $25,010$ samples,

[2] `https://github.com/psykei/psyki-python`

[3] `https://github.com/MatteoMagnini/kill-experiments-woa-2022`

[4] the size of the input space is the amount 5-permutations of 52 cards, i.e., $\frac{52!}{(52-5)!}$

Class	Logic Formulation
Pair	$class(R_1,\ldots,S_5,\texttt{pair}) \leftarrow pair(R_1,\ldots,S_5)$
	$pair(R_1,\ldots,S_5) \leftarrow R_1 = R_2$
	$pair(R_1,\ldots,S_5) \leftarrow R_1 = R_3$
	$pair(R_1,\ldots,S_5) \leftarrow R_1 = R_4$
	$pair(R_1,\ldots,S_5) \leftarrow R_1 = R_5$
	$pair(R_1,\ldots,S_5) \leftarrow R_2 = R_3$
	$pair(R_1,\ldots,S_5) \leftarrow R_2 = R_4$
	$pair(R_1,\ldots,S_5) \leftarrow R_2 = R_5$
	$pair(R_1,\ldots,S_5) \leftarrow R_3 = R_4$
	$pair(R_1,\ldots,S_5) \leftarrow R_3 = R_5$
	$pair(R_1,\ldots,S_5) \leftarrow R_4 = R_5$
Two Pairs	$class(R_1,\ldots,S_5,\texttt{two}) \leftarrow two(R_1,\ldots,S_5)$
	$two(R_1,\ldots,S_5) \leftarrow R_1 = R_2 \wedge R_3 = R_4$
	$two(R_1,\ldots,S_5) \leftarrow R_1 = R_3 \wedge R_2 = R_4$
	$two(R_1,\ldots,S_5) \leftarrow R_1 = R_4 \wedge R_2 = R_3$
	$two(R_1,\ldots,S_5) \leftarrow R_1 = R_2 \wedge R_3 = R_5$
	$two(R_1,\ldots,S_5) \leftarrow R_1 = R_3 \wedge R_3 = R_5$
	$two(R_1,\ldots,S_5) \leftarrow R_1 = R_5 \wedge R_2 = R_3$
	$two(R_1,\ldots,S_5) \leftarrow R_1 = R_2 \wedge R_4 = R_5$
	$two(R_1,\ldots,S_5) \leftarrow R_1 = R_4 \wedge R_2 = R_5$
	$two(R_1,\ldots,S_5) \leftarrow R_1 = R_5 \wedge R_2 = R_4$
	$two(R_1,\ldots,S_5) \leftarrow R_1 = R_3 \wedge R_4 = R_5$
	$two(R_1,\ldots,S_5) \leftarrow R_1 = R_4 \wedge R_3 = R_5$
	$two(R_1,\ldots,S_5) \leftarrow R_1 = R_5 \wedge R_3 = R_4$
	$two(R_1,\ldots,S_5) \leftarrow R_2 = R_3 \wedge R_4 = R_5$
	$two(R_1,\ldots,S_5) \leftarrow R_2 = R_4 \wedge R_3 = R_5$
	$two(R_1,\ldots,S_5) \leftarrow R_2 = R_5 \wedge R_3 = R_4$
Three of a Kind	$class(R_1,\ldots,S_5,\texttt{three}) \leftarrow three(R_1,\ldots,S_5)$
	$three(R_1,\ldots,S_5) \leftarrow R_1 = R_2 \wedge R_1 = R_3$
	$three(R_1,\ldots,S_5) \leftarrow R_1 = R_2 \wedge R_1 = R_4$
	$three(R_1,\ldots,S_5) \leftarrow R_1 = R_2 \wedge R_1 = R_5$
	$three(R_1,\ldots,S_5) \leftarrow R_1 = R_3 \wedge R_1 = R_4$
	$three(R_1,\ldots,S_5) \leftarrow R_1 = R_3 \wedge R_1 = R_5$
	$three(R_1,\ldots,S_5) \leftarrow R_1 = R_4 \wedge R_1 = R_5$
	$three(R_1,\ldots,S_5) \leftarrow R_2 = R_3 \wedge R_2 = R_4$
	$three(R_1,\ldots,S_5) \leftarrow R_2 = R_3 \wedge R_2 = R_5$
	$three(R_1,\ldots,S_5) \leftarrow R_2 = R_4 \wedge R_2 = R_5$
	$three(R_1,\ldots,S_5) \leftarrow R_3 = R_4 \wedge R_3 = R_5$
Straight	$class(R_1,\ldots,S_5,\texttt{straight}) \leftarrow royal(R_1,\ldots,S_5)$
	$class(R_1,\ldots,S_5,\texttt{straight}) \leftarrow straight(R_1,\ldots,S_5)$
	$straight(R_1,\ldots,S_5) \leftarrow (R_1+R_2+R_3+R_4+R_5) = (5*\min(R_1,\ldots,R_5)+10) \wedge \neg pair(R_1,\ldots,S_5)$
	$royal(R_1,\ldots,S_5) \leftarrow \min(R_1,\ldots,R_5) = 1 \wedge (R_1+R_2+R_3+R_4+R_5 = 47) \wedge \neg pair(R_1,\ldots,S_5)$
Flush	$class(R_1,\ldots,S_5,\texttt{flush}) \leftarrow flush(R_1,\ldots,S_5)$
	$flush(R_1,\ldots,S_5) \leftarrow S_1 = S_2 \wedge S_1 = S_3 \wedge S_1 = S_4 \wedge S_1 = S_5$
Four of a Kind	$class(R_1,\ldots,S_5,\texttt{four}) \leftarrow four(R_1,\ldots,S_5)$
	$four(R_1,\ldots,S_5) \leftarrow R_1 = R_2 \wedge R_1 = R_3 \wedge R_1 = R_4$
	$four(R_1,\ldots,S_5) \leftarrow R_1 = R_2 \wedge R_1 = R_3 \wedge R_1 = R_5$
	$four(R_1,\ldots,S_5) \leftarrow R_1 = R_2 \wedge R_1 = R_4 \wedge R_1 = R_5$
	$four(R_1,\ldots,S_5) \leftarrow R_1 = R_3 \wedge R_1 = R_4 \wedge R_1 = R_5$
	$four(R_1,\ldots,S_5) \leftarrow R_2 = R_3 \wedge R_2 = R_4 \wedge R_2 = R_5$
Full House	$class(R_1,\ldots,S_5,\texttt{full}) \leftarrow three(S_1,\ldots,R_5) \wedge two(S_1,\ldots,R_5) \wedge \neg four(S_1,\ldots,R_5)$
Straight Flush	$class(R_1,\ldots,S_5,\texttt{straight\_flush}) \leftarrow straight(R_1,\ldots,S_5) \wedge flush(R_1,\ldots,S_5)$
	$class(R_1,\ldots,S_5,\texttt{straight\_flush}) \leftarrow royal(R_1,\ldots,S_5) \wedge flush(R_1,\ldots,S_5)$
Royal Flush	$class(R_1,\ldots,S_5,\texttt{royal}) \leftarrow royal(R_1,\ldots,S_5) \wedge flush(R_1,\ldots,S_5)$
High Card	$class(R_1,\ldots,S_5,\texttt{nothing}) \leftarrow \neg pair(R_1,\ldots,S_5) \wedge \neg flush(R_1,\ldots,S_5) \wedge \neg straight(R_1,\ldots,S_5) \wedge \neg royal(R_1,\ldots,S_5)$

Table 4.3: Stritified datalog with negation formulas describing poker hands. For the sake of readability variables in the head of formulas and in the arguments of predicates are abbreviated.

Metric	Uneducated	Educated	Metric	Uneducated	Educated
Accuracy	0.962	0.978	**Acc. Straight**	0.415	0.509
Macro-F1	0.512	0.538	**Acc. Flush**	0.002	0.002
Weighted-F1	0.96	0.977	**Acc. Full**	0.628	0.69
Acc. High Card	0.977	0.989	**Acc. Four**	0.186	0.19
Acc. Pair	0.968	0.985	**Acc. Straight F.**	0.003	0
Acc. Two Pairs	0.867	0.914	**Acc. Royal F.**	0	0
Acc. Three	0.913	0.922			

Table 4.4: Test set accuracy, macro-F1 and weighted-F1 on all classes and single mean class accuracies. All measures represent the mean over the experiment population (30). The Wilcoxon signed-rank test shows that for the accuracy, macro-F1 and weighted-F1 metrics the educated model is significantly better than the uneducated one ($p < 0.05$). Also, the educated model is significantly better than the uneducated one for the single mean class accuracies, except for *high card*, *pair* and *straight* classes.

while the test set includes $1,000,000$ samples. This unusual ratio between training and test set sizes makes the task more challenging but ensures more reliable results.

For all experiments, we use a fully connected NN with three layers. The first two layers consist of 128 neurons each, while the output layer has 10 neurons, corresponding to the number of classes. The activation function for the first two layers is rectified linear unit (ReLU), whereas the output layer uses softmax. The network is trained using categorical cross-entropy as the loss function.

We set the batch size to 32 and train the network for up to 100 epochs. To evaluate the network's performance, we rely on metrics such as accuracy, macro-F1, and weighted-F1 scores. After training, the Λ-layer is removed, and the resulting NN is used for inference.

To prevent overfitting, we employ three stopping criteria during training. First, for 99% of the training examples, the activation of every output unit must be within 0.25 of the correct value. Second, training stops after 100 epochs. Third, the predictor must achieve at least 90% accuracy on the training examples but show no improvement in classification performance for five consecutive epochs. These criteria are inspired by previous work [TSN90]. Finally, we conduct 30 runs for each configuration to obtain a statistically significant population for comparisons.

Results and discussion We define two different configurations for experiments: *(i)* "uneducated", where we use the NN described in Section 4.2.3, *(ii)* "educated",

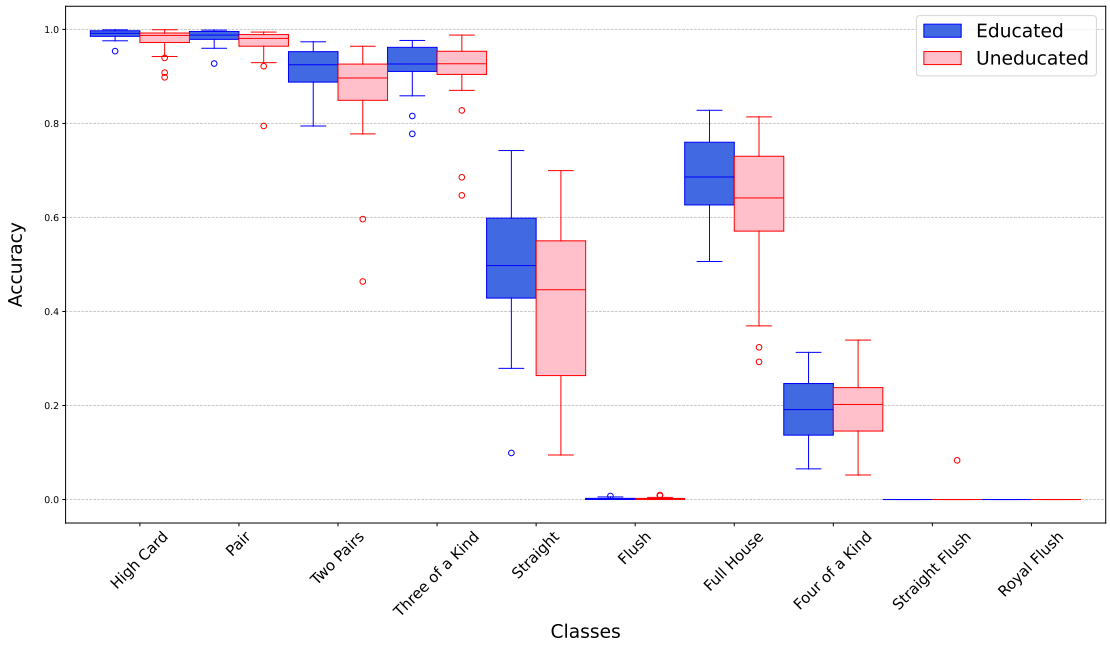


Figure 4.3: Per class results of the experiments on the PHDS dataset using KILL. The KILL method outperforms the uneducated NN in the prediction of almost all classes. For all class prediction, the Wilcoxon signed-rank test has been applied to compare the two configurations. With a p-value threshold of 0.05, we reject the null hypothesis in the case of *high card*, *pair*, and *straight* classes.

where we apply KILL on the same network architecture. For both configurations we use the same hyperparameters and perform 30 runs. Results are reported in Table 4.4 and in Figure 4.3. In general, experiments show that both predictors are very good at classifying frequent classes such as *high card* and *pair*. Also for less frequent classes, *two pairs* and *three of a kind*, accuracy is still high. The remaining six classes represent the 0.8% of the training set and therefore much more difficult to correctly predict. For *full house* and *straight*, accuracy has middle values, while for the remaining classes accuracy is pretty close to 0. All of this is quite expected due to the strong imbalance in the distribution of classes. The only remarkable fact is that accuracy is extremely low for class *flush* even if less frequent classes such as *full house* and *four of a kind* have higher accuracy. We hypothesise that this happens because the vast majority of data consists in classes which depend only on the values of rank and therefore the network tends to consider it much more. Indeed, only *flush*, *straight flush* and *royal flush* (about the 0.26% of the training set) depend on the values of suit. Concerning the comparison between the *uneducated* predictor – with no additional knowledge – and the *educated* predictor – obtained by applying KILL algorithm – results show that the latter has higher performances with statistic significance. The Wilcoxon signed-rank test [Wil45] has been applied to compare the two configurations on all the metrics reported in Table 4.4. In particular, because of the imbalanced nature of the dataset, the test on macro-F1 and weighted-F1 metrics is more reliable than the test on accuracy. The test shows that the *educated* predictor is significantly better than the *uneducated* one for the overall accuracy, macro-F1 and weighted-F1 metrics (with $p < 0.05$). Also, the *educated* predictor is significantly better than the *uneducated* one for the single mean class accuracies, except for *high card*, *pair* and *straight* classes.

4.3 An example of a structuring SKI algorithm

In this section we present the paper "KINS: Knowledge Injection via Network Structuring" [MCO22c], presented at the 37th Italian Conference on Computa-

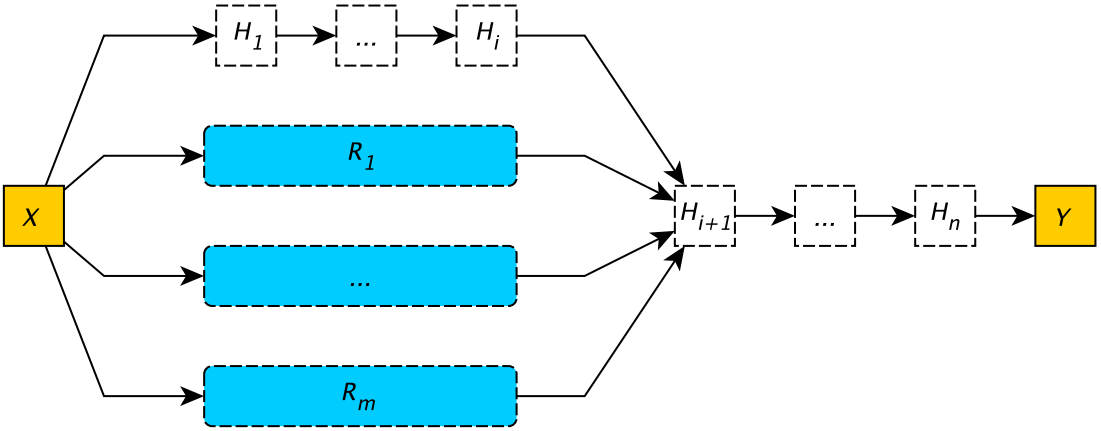


Figure 4.4: General architecture of a NN with m modules injected, X and Y are respectively the input and output layers. Each module is a subnetwork that evaluates a logic formula under a continuous interpretation. The modules can be injected at any layer H_i or at the output layer Y.

tional Logic (CILC 2022) [5]. The paper introduces a novel SKI methods, named knowledge injection via network structuring (KINS), which similarly to KILL allows to inject symbolic knowledge in stratified Datalog with negation into NNs of any shape. Differently from KILL, KINS does not act at the backpropagation level, but it modifies the structure of the NN to inject symbolic knowledge. Hence, KINS falls under the *structuring* strategy, as described in Section 3.1.4.

4.3.1 KINS architecture

A NN architecture can be extended with additional neural *modules* designed to reflect and mimic symbolic knowledge provided by designers. Each module is a subnetwork that shares the same input layer as the original network and outputs a value representing the evaluation of a logic formula under a continuous interpretation. The primary purpose of these modules is twofold: *(i)* to evaluate a specific logic formula against the current input, and *(ii)* to compute the degree of truth of that formula, expressed as a value in the range $[0, 1]$, which complements the network's output. During the feed-forward phase, variables in the formulas are dynamically grounded with respect to the current input. This dynamic grounding

[5] `http://cilc2022.apice.unibo.it/`

allows non-ground formulas to be directly exploited for SKI, eliminating the need for prior groundisation steps, which may be infeasible for complex domains. It is important to note that the provided formulas are not required to cover all possible scenarios. For instance, in classification problems, rules may only cover a subset of all possible classes.

Figure 4.4 illustrates the general architecture of a NN after the injection of m modules, represented as blue rectangles. Each module corresponds to one of the m rules to be injected. Modules can be arbitrarily complex subnetworks that share the same input and final outputs with the original NN. White boxes represent arbitrary hidden layers $H_1, \ldots, H_n$ of the original NN, while X and Y denote the input and output layers, respectively. Knowledge injection can be performed at any hidden layer H_i or at the output layer Y. For example, in networks that first extract features from the input (e.g., convolutional NNs) and then perform classification, knowledge can be injected between these two phases. The injection procedure involves three main steps. First, formulas are encoded into real-valued functions, as described in Section 4.2.2. Second, a neural module is constructed to approximate each real-valued function, following the strategy outlined in Section 4.2.2. Finally, the module is added to the original NN, as depicted in Figure 4.4. The inner synapses of the modules can be either immutable, meaning their weights and biases remain fixed during training, or mutable, meaning their weights are trainable. All other synapses, including those within hidden layers $H_1, \ldots, H_i$, as well as the ingoing synapses of layer H_{i+1} and subsequent layers, remain trainable. This design allows the NN to leverage both prior knowledge and information gathered from data during training. The synapses connecting each module and the final hidden layer to the output layer are also trainable. This enables the NN to adjust the relative importance of the injected knowledge during training. The rationale is that logic rules may not hold universally across all patterns in a given domain but may generally be true with a certain degree of confidence.

KINS does not impose constraints on the network architecture, such as the number of layers, number of neurons, types of activation functions, or initialization status (e.g., random weights or partially trained). It can be applied to both untrained and partially trained networks. However, it requires the network to have an input and output layer and to be trainable via gradient descent or simi-

Formula	C. interpretation	Formula	C. interpretation
$[\![\neg\phi]\!]$	$\eta(1 - [\![\phi]\!])$	$[\![\phi \leq \psi]\!]$	$\eta(1 + [\![\psi]\!] - [\![\phi]\!])$
$[\![\phi \wedge \psi]\!]$	$\eta(min([\![\phi]\!], [\![\psi]\!]))$	$[\![class(\bar{X}, \mathtt{y}_i) \leftarrow \psi]\!]$	$[\![\psi]\!]^*$
$[\![\phi \vee \psi]\!]$	$\eta(max([\![\phi]\!], [\![\psi]\!]))$	$[\![\text{expr}(\bar{X})]\!]$	$\text{expr}([\![\bar{X}]\!])$
$[\![\phi = \psi]\!]$	$\eta([\![\neg(\phi \neq \psi)]\!])$	$[\![\mathtt{true}]\!]$	1
$[\![\phi \neq \psi]\!]$	$\eta(\lvert[\![\phi]\!] - [\![\psi]\!]\rvert)$	$[\![\mathtt{false}]\!]$	0
$[\![\phi > \psi]\!]$	$\eta(max(0, \frac{1}{2} + [\![\phi]\!] - [\![\psi]\!]))$	$[\![X]\!]$	x
$[\![\phi \geq \psi]\!]$	$\eta(1 + [\![\phi]\!] - [\![\psi]\!])$	$[\![\mathtt{k}]\!]$	k
$[\![\phi < \psi]\!]$	$\eta(max(0, \frac{1}{2} + [\![\psi]\!] - [\![\phi]\!]))$	$[\![p(\bar{X})]\!]^{**}$	$[\![\psi_1 \vee \ldots \vee \psi_k]\!]$

* encodes the value for the i^{th} output

** assuming p is defined by k clauses of the form:
$p(\bar{X}) \leftarrow \psi_1, \ \ldots, \ p(\bar{X}) \leftarrow \psi_k$

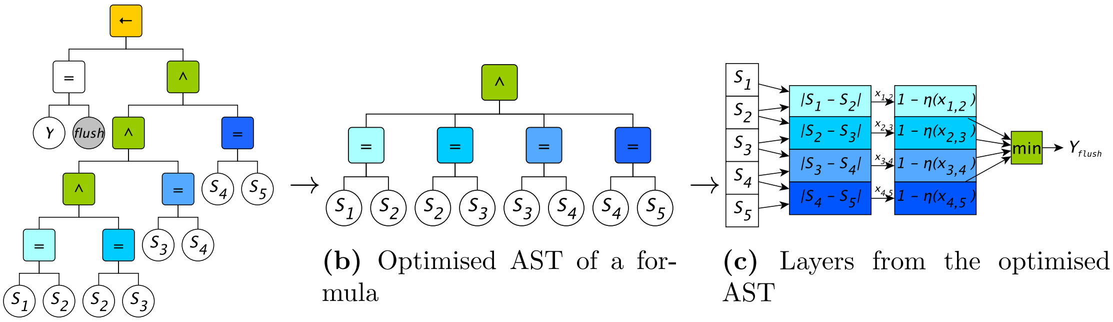
Table 4.5: Logic formulas' encoding into real-valued functions. There, X is a logic variable, while x is the corresponding real-valued variable, whereas is $\bar{X}$ a tuple of logic variables. Similarly, $\mathtt{k}$ is a numeric constant, and k is the corresponding real value, whereas $\mathtt{k}_i$ is the constant denoting the i^{th} class of a classification problem. Finally, $\text{expr}(\bar{X})$ is an arithmetic expression involving the variables in $\bar{X}$. The η function is a scaling function described in Equation (4.1).

lar algorithms. Additionally, symbolic knowledge must be expressed using one or more formulas in Datalog form, and logic statements about the network's input or output features must be encoded.

4.3.2 KINS fuzzifier

To inject symbolic knowledge into a neural network, each formula associated with an output neuron must first be converted into a real-valued function to compute the degree of truth of that formula. This conversion relies on a multivalued interpretation of logic – inspired by Łukasiewicz's logic [Hay63] – similarly to the one used for KILL. Differently from KILL, the fuzzification function is not used to compute a penalty, but rather to compute the degree of truth of the formula. For this reason, `true` and `false` are interpreted as 1 and 0, respectively, rather than as 0 and 1. While Table 4.5 describes the mapping between formulas and their fuzzy interpretation, we now discuss how such interpretation can be further encoded into neural modules to be added to the NN undergoing injection.

By considering the same domain of the PHDS used in Section 4.2.3, we can



(a) AST of a formula

(b) Optimised AST of a formula

(c) Layers from the optimised AST

Figure 4.5: Example of the encoding process of formulas into module network. Box coloured in the same way represent the encoding of a given operator through each encoding step.

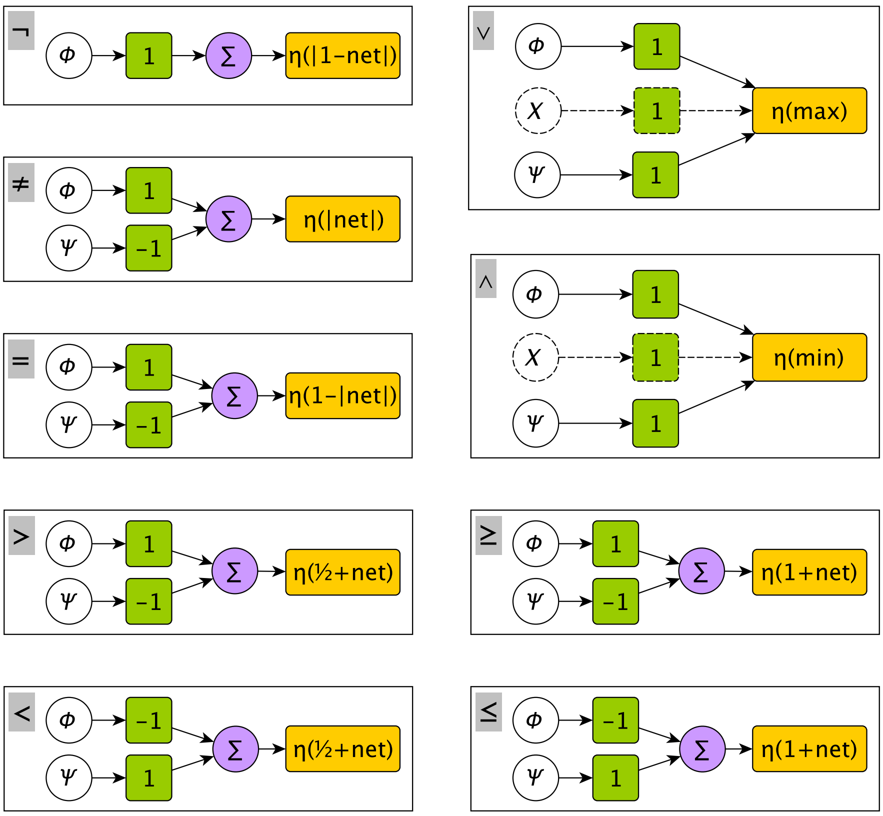


Figure 4.6: KINS mapping of formulas into neurons. White circles are input variables (I), green boxes represent the corresponding weights (W), purple circles are the sum of the weighted inputs (WxI). Yellow rectangles are activation functions, net is the output of WxI, maz and min respectively the maximum and minimum of input values, η is the function described in Equation (4.1).

Symbol	Adenine	Cytosine	Guanine	Thymine	Logic form
d	•		•	•	$(X_i = \mathtt{d}) \equiv (X_i = \mathtt{a} \vee X_i = \mathtt{g} \vee X_i = \mathtt{t})$
m	•	•			$(X_i = \mathtt{m}) \equiv (X_i = \mathtt{a} \vee X_i = \mathtt{c})$
r	•		•		$(X_i = \mathtt{r}) \equiv (X_i = \mathtt{a} \vee X_i = \mathtt{g})$
s		•	•		$(X_i = \mathtt{s}) \equiv (X_i = \mathtt{c} \vee X_i = \mathtt{g})$
y		•		•	$(X_i = \mathtt{y}) \equiv (X_i = \mathtt{c} \vee X_i = \mathtt{t})$

Table 4.6: Mapping of aggregative symbols and the four nucleotides. Each symbol can be substituted with one base on the right that has a dot.

take Equation (4.2) as an example of a formula to be injected. The procedure for encoding a logic formula into a dedicated neural module is illustrated in Figure 4.5. In this example, Equation (4.2) is transformed into a neural module through three distinct phases. First, the logic formula is parsed to construct its AST, as shown in Figure 4.5a. Second, the AST is simplified by merging commutative binary operators, as depicted in Figure 4.5b. Third, the simplified AST is encoded into a neural network, where each operator is mapped to a neuron that implements the corresponding operation, as specified in Table 4.5. During the final step, the encoding rules defined in Figure 4.6 are recursively applied to convert operators into neurons. Input variables, such as S_i, are represented as input neurons, while constants appearing in the formulas are mapped to neurons with fixed outputs. Additionally, algebraic operations, including addition and multiplication, are implemented using single neurons that perform the respective operations. This approach ensures that the symbolic knowledge is seamlessly integrated into the neural network structure. A public implementation of the KINS algorithm is available as part of the PSyKI [MCO22d] library[2].

4.3.3 Validation

In this section, we evaluate the performance of KINS in improving the predictive capabilities of NNs through SKI. To ensure reproducibility, the code used for our experiments is publicly available[6]. The validation is conducted using the well-known primate splice-junction gene sequence (PSJGS) dataset [91]. This dataset consists of 3190 records, each representing a sequence of 60 DNA nucleotides. The nucleotides are adenine (a), cytosine (c), guanine (g), and thymine (t). Auxiliary

[6] https://github.com/MatteoMagnini/kins-experiments-cilc-2022

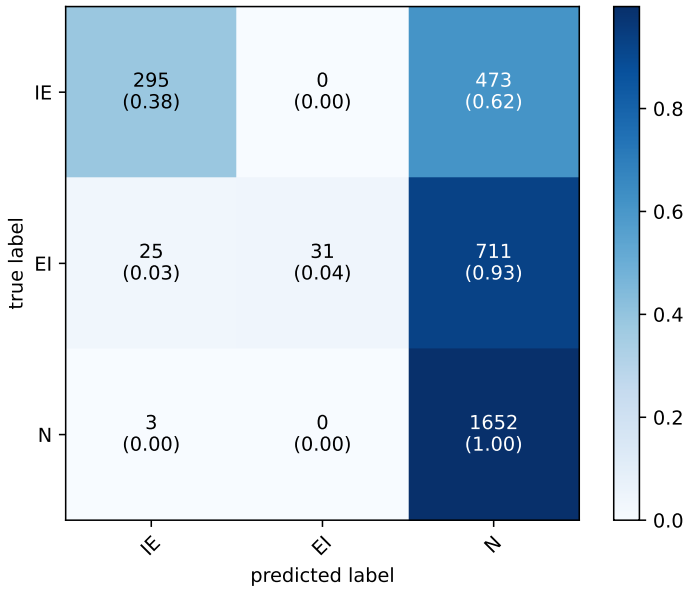


Figure 4.7: Confusion matrix of the knowledge for the PSJGS dataset. The matrix shows the classification results of the knowledge rules on the dataset if applied directly.

symbols are used to denote multiple possible nucleotides in a compact way, as shown in Table 4.6. Each sequence spans positions from -30 to +30, excluding position 0. The classification task involves identifying whether a sequence corresponds to an intron-exon (IE) boundary, an exon-intron (EI) boundary, or neither (N). The dataset is imbalanced, with class frequencies distributed as follows: 50% for N, and 25% each for IE and EI. This imbalance poses challenges for accurate classification, making it a suitable benchmark for testing KINS.

PSJGS logic rules The PSJGS dataset includes a set of textual logical rules designed by human experts to classify DNA sequences. Table 4.7 presents these rules converted into Datalog form, which is equivalent to the original custom formalism but machine-interpretable. In Datalog, variables are indexed from -30 to 30, excluding position 0. Each variable $X_{\pm i}$ represents the nucleotide at position $\pm i$, encoded using constants such as `a`, `c`, `g`, and `t`. For simplicity, the notation $\bar{X}$ is used to represent the full sequence of variables $X_{-30}, \ldots, X_{-1}, X_{+1}, \ldots, X_{+30}$.

The original rules also utilize additional symbols to compactly represent multi-

Class	Logic Formulation
EI	$class(\bar{X}, \texttt{ei}) \leftarrow X_{-3} = \texttt{m} \wedge X_{-2} = \texttt{a} \wedge X_{-1} = \texttt{g} \wedge X_{+1} = \texttt{g} \wedge X_{+2} = \texttt{t} \wedge X_{+3} = \texttt{a} = \texttt{r} \wedge X_{+4} = \texttt{a} \wedge X_{+5} = \texttt{g} \wedge X_{+6} = \texttt{t} \wedge \neg(ei\_stop(\bar{X}))$ $ei\_stop(\bar{X}) \leftarrow X_{-3} = \texttt{t} \wedge X_{-2} = \texttt{a} \wedge X_{-1} = \texttt{a}$ $ei\_stop(\bar{X}) \leftarrow X_{-3} = \texttt{t} \wedge X_{-2} = \texttt{a} \wedge X_{-1} = \texttt{g}$ $ei\_stop(\bar{X}) \leftarrow X_{-3} = \texttt{t} \wedge X_{-2} = \texttt{g} \wedge X_{-1} = \texttt{a}$ $ei\_stop(\bar{X}) \leftarrow X_{-4} = \texttt{t} \wedge X_{-3} = \texttt{a} \wedge X_{-2} = \texttt{a}$ $ei\_stop(\bar{X}) \leftarrow X_{-4} = \texttt{t} \wedge X_{-3} = \texttt{a} \wedge X_{-2} = \texttt{g}$ $ei\_stop(\bar{X}) \leftarrow X_{-4} = \texttt{t} \wedge X_{-3} = \texttt{g} \wedge X_{-2} = \texttt{a}$ $ei\_stop(\bar{X}) \leftarrow X_{-5} = \texttt{t} \wedge X_{-4} = \texttt{a} \wedge X_{-3} = \texttt{a}$ $ei\_stop(\bar{X}) \leftarrow X_{-5} = \texttt{t} \wedge X_{-4} = \texttt{a} \wedge X_{-3} = \texttt{g}$ $ei\_stop(\bar{X}) \leftarrow X_{-5} = \texttt{t} \wedge X_{-4} = \texttt{g} \wedge X_{-3} = \texttt{a}$
IE	$class(\bar{X}, \texttt{ie}) \leftarrow pyramidine\_rich(\bar{X}) \wedge \neg(ie\_stop(\bar{X})) \wedge X_{-3} = \texttt{y} \wedge X_{-2} = \texttt{a} \wedge X_{-1} = \texttt{g} \wedge X_{+1} = \texttt{g}$ $pyramidine\_rich(\bar{X}) \leftarrow 6 \leq (X_{-15} = \texttt{y} + \ldots + X_{-6} = \texttt{y})$ $ie\_stop(\bar{X}) \leftarrow X_{+2} = \texttt{t} \wedge X_{+3} = \texttt{a} \wedge X_{+4} = \texttt{a}$ $ie\_stop(\bar{X}) \leftarrow X_{+2} = \texttt{t} \wedge X_{+3} = \texttt{a} \wedge X_{+4} = \texttt{g}$ $ie\_stop(\bar{X}) \leftarrow X_{+2} = \texttt{t} \wedge X_{+3} = \texttt{g} \wedge X_{+4} = \texttt{a}$ $ie\_stop(\bar{X}) \leftarrow X_{+3} = \texttt{t} \wedge X_{+4} = \texttt{a} \wedge X_{+5} = \texttt{a}$ $ie\_stop(\bar{X}) \leftarrow X_{+3} = \texttt{t} \wedge X_{+4} = \texttt{a} \wedge X_{+5} = \texttt{g}$ $ie\_stop(\bar{X}) \leftarrow X_{+3} = \texttt{t} \wedge X_{+4} = \texttt{g} \wedge X_{+5} = \texttt{a}$ $ie\_stop(\bar{X}) \leftarrow X_{+4} = \texttt{t} \wedge X_{+5} = \texttt{a} \wedge X_{+6} = \texttt{a}$ $ie\_stop(\bar{X}) \leftarrow X_{+4} = \texttt{t} \wedge X_{+5} = \texttt{a} \wedge X_{+6} = \texttt{g}$ $ie\_stop(\bar{X}) \leftarrow X_{+4} = \texttt{t} \wedge X_{+5} = \texttt{g} \wedge X_{+6} = \texttt{a}$

Table 4.7: Datalog formulas describing DNA classification criteria generated from the original one.

ple possible nucleotides. Table 4.6 provides the mapping of these symbols to their corresponding nucleotide combinations.

When applying the rules in Table 4.7 to classify sequences in the PSJGS dataset, sequences of type IE are correctly classified 295 times as true positive (TP). However, the same rule also incorrectly classifies 25 EI sequences and 3 N sequences as false positive (FP). In contrast, EI sequences are correctly classified 31 times, with no FP.

To address the classification of sequences of type N, a fictional rule is introduced, which negates both IE and EI rules:

$$class(\bar{X}, \texttt{n}) \leftarrow \neg class(\bar{X}, \texttt{ei}) \wedge \neg class(\bar{X}, \texttt{ie}). \tag{4.3}$$

Figure 4.7 shows the confusion matrix of the knowledge rules applied to the dataset. Although these rules do not perfectly describe the domain, they are sufficiently accurate to positively influence the training of the predictor. Further refinements or additional rules may improve the classification performance.

Methodology To ensure comparability with existing literature benchmarks, we adopt the methodology proposed by Towell and Shavlik [TS94]. Specifically, we perform 10-fold cross-validation using a training set of $1,000$ randomly selected records from the $3,190$ available ones, which corresponds to 31.3% of the dataset. For each fold, we train one instance of KINS. The test accuracy is computed on the remaining $2,190$ records by averaging the predictions of the 10 trained KINS instances. Unlike the original method, we repeat the experiment 30 times instead of 10 to improve statistical significance.

Each KINS instance is trained using a fully connected NN with three layers. The input layer consists of 60 neurons, the hidden layer has a configurable number of neurons, and the output layer includes 3 neurons corresponding to the classification task. To mitigate overfitting, we apply dropout [Sri+14] with a rate of 0.2 to each layer during training. The activation function for the hidden layers is ReLU, while the output layer uses the softmax function. The network is optimized using the Adam optimizer [KB15] and categorical cross-entropy as the loss function.

We employ the stopping criteria described in [TS94]. These criteria include

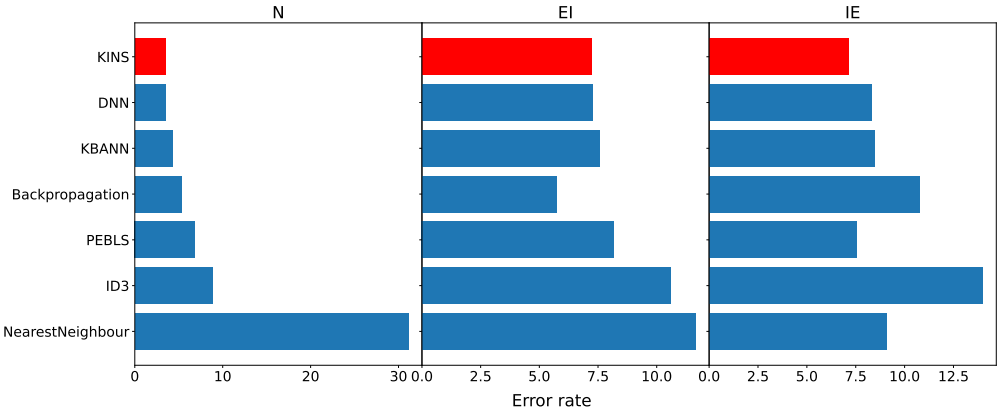


Figure 4.8: Error rate per class for the PSJGS dataset using KINS. The results are compared with other algorithms, including DNN (the network used in KINS without knowledge injection) and KBANN [TS94]. The error rate is significantly lower for KINS compared to DNN and KBANN.

ensuring that 99% of the training examples have output activations within 0.25 of the correct value, limiting training to a maximum of 100 epochs, and stopping when the predictor achieves at least 90% accuracy on the training examples but shows no improvement for five consecutive epochs.

The rules $ei\_stop(\bar{X})$ and $ie\_stop(\bar{X})$ are immutable, meaning their weights remain fixed during training. In contrast, the rules $class(\bar{X}, \texttt{ei})$, $class(\bar{X}, \texttt{ie})$, and $pyramidine\_rich(\bar{X})$ are mutable, allowing their weights to be adjusted during training. Immutable rules preserve their structure, while mutable rules adapt to the data to improve predictive performance.

Results and discussion We evaluate KINS by injecting prior knowledge into different layers of the NN described earlier. The best results are achieved when the injection is performed at the first hidden layer. Following the methodology outlined in Section 4.3.3, we train 10 predictors on 900 records each, resulting in a total of $1,000$ unique training records out of $3,190$ for each experiment. We conduct 30 experiments using 10-fold cross-validation with random weight initialization. After injecting prior knowledge into the first hidden layer and training, the mean accuracy across the 30 experiments on the test set is 94.73%. The mean class accuracies are as follows: IE 92.79%, EI 92.49%, and N 96.67%.

For comparison, we perform 30 additional runs using the same base NN architecture without injecting any knowledge. The results are: mean accuracy 94.45%, IE 91.67%, EI 92.73%, and N 96.54%. Using Student's t-test, we reject the null hypothesis, confirming that predictors generated with KINS achieve statistically significant improvements in accuracy. The accuracy improvement using KINS is notable even when the injected knowledge is imperfect.

Figure 4.8 illustrates the error rate per class for different algorithms. KINS represents our knowledge injection method, while deep neural network (DNN) refers to the same network used in KINS without knowledge injection. Knowledge-based artificial neural network (KBANN), as proposed in [TS94], performs slightly worse than both DNN and KINS. The primary reason for this difference is that KBANN fully reflects the provided knowledge in its structure, whereas KINS allows a portion of the network to adapt to the data. This flexibility is advantageous when the knowledge closely aligns with the domain rules but can be a limitation in the opposite scenario. Other algorithms, including standard backpropagation [RHW87], parallel exemplar-based learning system (PEBLS) [CS93b], ID3 [Qui86], and nearest neighbors, generally perform worse than KINS.

Chapter 5

Platform for symbolic knowledge injection

How to *facilitate* the engineering of SKI methods?
How to measure the *quality of service* of SKI methods?
– **RQ0, RQ2 and RQ3**

Contents

In this chapter, we present the PSyKI platform, which is a framework for SKI methods. PSyKI provides a unified way to implement SKI methods, aiming to facilitate the development and comparison of different approaches. It also provides a set of tools to evaluate the performance of SKI methods, including metrics for measuring the quality of the injected knowledge and the performance of the resulting models. In Section 5.1, we present the work "*On the Design of PSyKI: A Platform for Symbolic Knowledge Injection into Sub-symbolic Predictors*" [MCO22d], which describes the design and implementation of PSyKI. Then, in Sections 5.2 and 5.3, we present respectively two works that study the quality of service of SKI methods: in Section 5.2 the paper "*Symbolic Knowledge Injection Meets Intelligent Agents*" [Agi+23], which introduces metrics for evaluating the performance of SKI methods in the context of intelligent agents, and in Section 5.3 the paper "*An Empirical Study on the Robustness of Symbolic Knowledge Injection Techniques Against Data Degradation*" [Raf+24], which studies the robustness of SKI methods against data degradation.

5.1 One platform to bring them all

PSyKI is a (Python) platform for the development and evaluation of SKI methods. It is designed to provide a unified framework for implementing SKI methods, allowing researchers and practitioners to easily develop, test, and compare different approaches. In the following, we present a summary of the work "*On the Design of PSyKI: A Platform for Symbolic Knowledge Injection into Sub-symbolic Predictors*" [MCO22d], presented at the 4th international workshop on EXplainable, Trustworthy, and Responsible AI and Multi-Agent Systems (EXTRAAMAS), 2022[1]. The platform is meant to be an open-source library that can be easily extended with new SKI methods and evaluation metrics. The library is public

[1] `https://extraamas.ehealth.hevs.ch/archive.html`

available on GitLab and PyPi[2].

5.1.1 Motivations

This work is motivated by the need to address the common challenges that affect SKI methods (see Section 3.1.5). In particular, we want to address the following points: *(i) lack of generality*, by providing the proper tools to automatically translate symbolic knowledge of arbitrary domains into a format suitable for injection into sub-symbolic predictors, *(ii) lack of reproducibility*, by providing a unified framework for implementing SKI methods, allowing researchers to easily develop, test, and compare different approaches, and *(iii) lack of availability*, by encouraging the development of reusable software libraries for SKI methods. Concerning the last point mentioned in Section 3.1.5, we identified in the logic language of stratified Datalog with negation a suitable candidate for the symbolic knowledge representation. It is expressive enough to represent a wide range of symbolic knowledge, while being simple enough to be easily translated into a format suitable for injection into sub-symbolic predictors.

This work differs from previous recent efforts in the area of SKI software – e.g., logic tensor network (LTN) [Bad+22] and DeepProbLog [Man+21] – because our platform is not designed to support just one injection method, but it is meant to be an extendable library of SKI methods. Moreover, PSyKI provides a set of tools to evaluate the performance of SKI methods, including metrics for measuring the quality of the injected knowledge and the performance of the resulting models. Arguably, the most similar work – *but it came one year after the publication of PSyKI* – is Scallop [LHN23], a DataLog-based framework that supports differentiable logical and relational reasoning.

5.1.2 Design

All symbolic knowledge injection (SKI) methods implemented in PSyKI share a common transformation pipeline, illustrated in Figure 5.1. Symbolic knowledge ϕ cannot usually be injected directly into a sub-symbolic predictor. Instead, it

[2] `https://gitlab.com/psykei/psyki-python` and `https://pypi.org/project/psyki/`

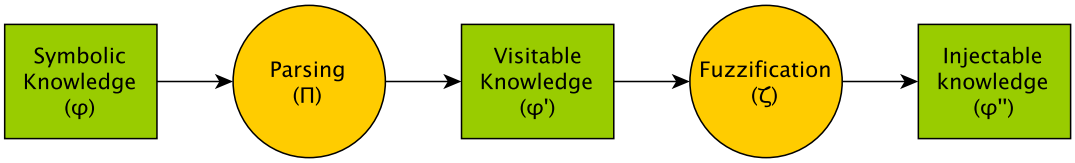


Figure 5.1: General workflow of symbolic knowledge transformation in PSyKI. The symbolic knowledge (ϕ), typically expressed as logical formulas, is first parsed into a visitable form (ϕ'), and then fuzzified into a machine-injectable representation (ϕ'').

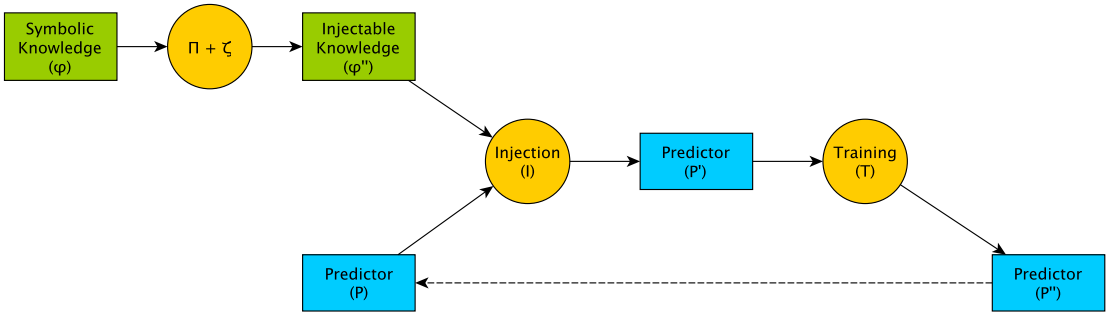


Figure 5.2: Complete workflow of structuring and guided learning SKI methods in PSyKI. The symbolic knowledge is transformed and injected into the sub-symbolic predictor, which is then trained on data.

undergoes a two-step transformation: *(i) Parsing* (Π): the knowledge is converted into a visitable data structure, such as an AST in the case of logic formulas, resulting in ϕ'; *(ii) Fuzzification* (ζ): the parsed representation is transformed into a sub-symbolic form ϕ'', suitable for injection. Fuzzification plays a key role in bridging the symbolic and sub-symbolic domains. It translates crisp Boolean logic into a form compatible with sub-symbolic models, often by relaxing discrete truth values into continuous-valued functions or generating neural components such as layers or entire NNs.

Figure 5.2 shows the overall injection process in PSyKI, including both the transformation of the symbolic knowledge and its integration into the learning system. After the transformation phase, the different SKI methods diverge: *(i) Structuring* and *guided learning* inject ϕ'' directly into the architecture or training process of the predictor; *(ii) Embedding*-based methods use symbolic knowledge to enrich or generate the input data, so they are not directly represented in Figure 5.2.

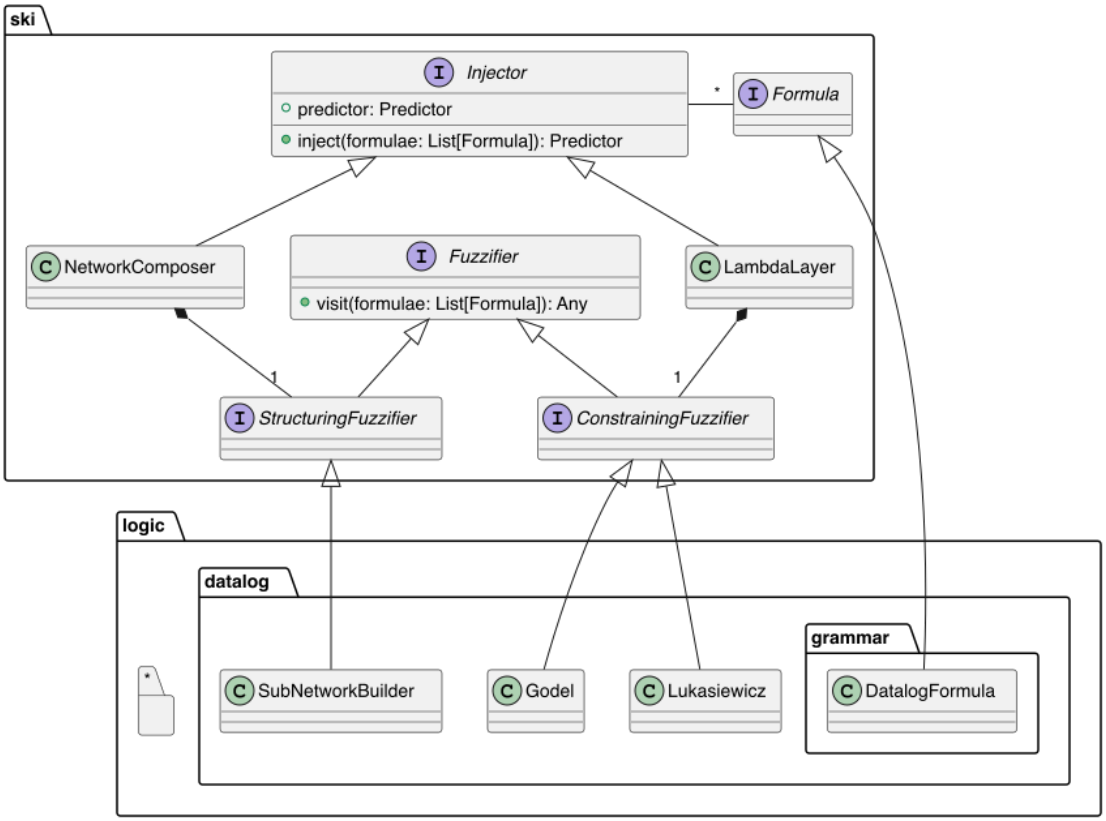


Figure 5.3: Class diagram of PSyKI. The main components are the *Injector*, the *Formula* and the *Fuzzifier*. The package *logic.datalog* is an examplification showing two *Injector* implementations and their relationships.

PSyKI supports all three approaches, although it is primarily designed for *structuring* and *guided learning*, where the injection occurs directly into the predictor. In general, SKI algorithms operate on a predictor P and symbolic knowledge ϕ, producing a new predictor P' as output. This modified predictor is then trained on data, resulting in a final model P'', which can be reused in further iterations with different knowledge or injection strategies.

Architecture Essentially, PSyKI is designed around the notion of injector. An injector is any algorithm accepting a ML predictor and prior symbolic knowledge – predominantly logic formulas – as input that produces a new predictor as output. In order to properly perform injection, injectors may require additional information such as algorithm specific hyperparameters.

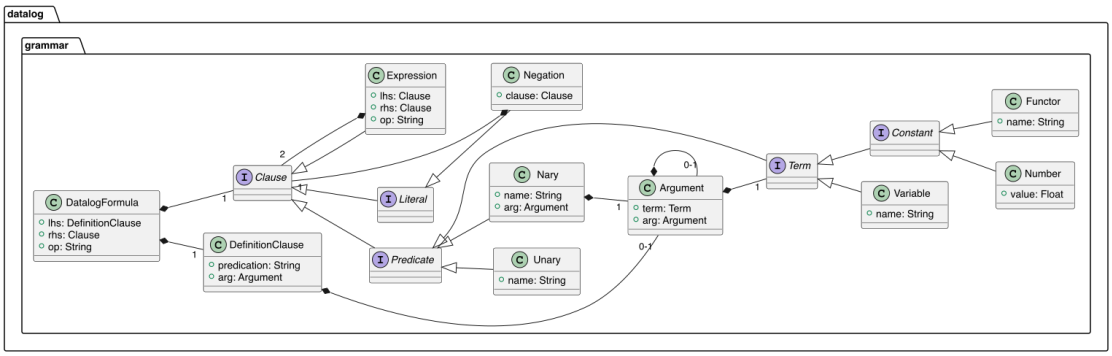


Figure 5.4: A supported grammar of PSyKI for logic formulas. The grammar is designed to represent Datalog formulas.

PSyKI supports the processing of symbolic knowledge represented via logic formulas. Based on the sort of logic, user can build an AST for each formula. The AST can be inspected through a fuzzifier via pattern visitor to encode the symbolic knowledge to 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 ζ Figure 5.2 – is injector specific; instead, the same parser Π can be used for logic formulas of the same sort independently of the injector.

The software is organized into well-separated packages to ensure easy extensibility towards new sort of logic and fuzzifiers—see Figure 5.3 An AST is a *formula* object, and it can have different language specific elements w.r.t. the logic form that is covered. Each formula implementation is self-contained inside a standalone package so that if a user wants to add a new logic form it is sufficient to add its implementation in a new package. Similarly, a fuzzifier object that targets a specific logic form can be added inside the same package of the logic, there can be any number of fuzzifiers for a given logic.

5.1.3 Implementation

Knowledge representation A crucial point in the SKI workflow is the embedding of knowledge from symbolic into sub-symbolic form. Ideally, there is no constraint on the formalism used to represent the prior knowledge (e.g., logic formulas, knowledge graph). The most common knowledge representation form that

SKI algorithms claim to support is FOL or one of its subsets. However, there are characteristics of FOL that are not ideal for some predictors. Recursion and function symbols – that allow recursive structures – cannot be easily integrated into a predictor that is acyclic – i.e., no recursive – by construction such as conventional NN (virtually all NN, with few exceptions like fibred NN [BGH05]). Conversely, in this work we consider one of the most general and expressive logic formalism that does not support recursion and function symbols: stratified Datalog with negation.

Stratified Datalog with negation has been already described in detail in Section 4.1.1. To support injection into a particular predictor, we further assume the input knowledge base defines at least one outer relation – say output or class – involving as many variables as the input and output features the predictor has been trained upon. Such a relation may be defined via one or more clauses, and each clause may leverage on other predicates in their bodies. In turn, each predicate may be defined through one or more clause. In that case, since we rely on stratified Datalog, we require input knowledge to not include any (directly or indirectly) recursive clause definition.

Once that the logic has been formalized, the implementation of a *Formula* – visitable data structure like an AST – is quite straightforward. Figure 5.4 depicts the general API for representing logic formulas, as currently supported by PSyKI. To make PSyKI able to parse bare text into actual logic formulas compliant to that API, we rely on well-established parser-generation facilities such as ANTLR [PQ95]. As further discussed below, the knowledge contained into a Formula object, can then be embedded in sub-symbolic form, via a fuzzifier, to be later injected into a predictor.

Later in the evolution of PSyKI (i.e., from v. 0.2.1), we considered switching to a more established logic language for representing the symbolic knowledge: Prolog [Kör+22]. In particular, we abandoned the manual parsing of logic formulas with ANTLR, and we rely on the tuProlog [DOR01] framework. We use the parser of tuProlog to parse the logic formulas, but PSyKI does not support the full Prolog language because of the limitations of injecting into NN mentioned in Section 3.1.5 (i.e., we still support only stratified Datalog with negation).

```python
import pandas as pd
from tensorflow.keras.models import Model
from psyki.logic import Theory
from psyki.ski import Injector


def create_uneducated_predictor() -> Model:
...

dataset = pd.read_csv("path_to_dataset.csv")  # load dataset
knowledge_file = "path_to_knowledge_file.pl"  # load knowledge
theory = Theory(knowledge_file, dataset)      # create a theory
uneducated = create_uneducated_predictor()    # create a NN
injector = Injector.kins(uneducated)          # create an injector
educated = injector.inject(theory)            # inject knowledge into the NN

# From now on you can use the educated predictor
# as you would use the uneducated one
```

Listing 5.1: General code snippet for the use of a SKI method in the current version of PSyKI (v. 0.4.3).

Fuzzification While logic formulas are commonly interpreted in a Boolean way – i.e., they can be assigned with one truth value among true and false –, sub-symbolic predictors are more flexible-hence supporting formulas to hold up to some extent. Switching from the former interpretation to the latter is, essentially, the purpose of fuzzifiers. In practice, this implies converting a logic formula into a function of real numbers. Along this line, fuzzification may be performed in (at least) two ways; real numbers may either represent *(i)* a *penalty* for the violation of a formula, or *(ii)* the *degree of truth* of that formula.

To serve this purpose, one may rely on a multivalued interpretation of logic inspired to Łukasiewicz's logic, where the value true is represented by 0 (resp. 1), while higher (resp. lower) values represent "falsity". In particular, this approach may be useful to constrain a predictor during its training. Examples of this fuzzification approach are KILL (Section 4.2.2) and KINS (Section 4.3.2). Tables 4.1 and 4.5 show the logic formulas encoding into real-valued functions for KILL and KINS, implementing *penalty* and *degree of truth* respectively.

Injection Knowledge injection is the central step in SKI. It is carried out by *injectors*, i.e., objects wrapping a particular injection strategy. Each injector is expected to accept one sub-symbolic predictors as input, other the logic formulas

that should be injected into that predictor. As part of its operation, the injector is in charge of producing a new predictor as output, of the same type as the one provided as input. The new predictor is altered in such a way that it is consistent w.r.t. the provided logic formulas.

While the main architecture of PSyKI has been maintained stable during the years, the implementation has been continuously improved. The current version of PSyKI (v. 0.4.3) is implemented in Python 3.9.13, with main dependencies being `tensorflow`, `2ppy`, `scikit-learn`, and `pandas`. Listing 5.1 shows a general code snippet for the use of a SKI algorithm in PSyKI (v. 0.4.3).

5.2 QoS metrics for SKI methods

In this section and in the next one, we present two works that study the QoS of SKI methods. QoS is a crucial aspect of any non-trivial systems, and it is particularly important for SKI because the vast majority of the works in this field do not provide any evaluation but the performance of the resulting model. Moreover, SKI methods and the resulting educated models have many other dimensions that can be evaluated a part from the straight prediction performance. The metrics introduced in these works have been implemented and integrated into PSyKI from version 0.2.14 and later.

The work "*Symbolic Knowledge Injection Meets Intelligent Agents*" [Agi+23], published in the *Autonomous Agents and Multi-Agent Systems* journal, presents a set of metrics for evaluating the performance of SKI methods in the context of intelligent agents. Here we present a summary of the work, including the motivations, the proposed metrics, and the evaluation of SKI methods in terms of QoS.

5.2.1 Motivations

The current practice of assessing SKI methods primarily focuses on measuring improvements in the predictive performance of an educated predictor $\hat{N}$ over its uneducated counterpart N. However, predictive performance is not the only relevant benefit of SKI that should be evaluated.

There are multiple aspects of MLs predictors that can be influenced by SKI, and for which specific metrics should be defined. For instance, SKI may impact the *memory footprint*, *latency*, *data efficiency*, and *energy consumption* of the predictors it is applied to. Collectively, these properties contribute to what we refer to as the *efficiency* of a predictor.

In this work, we consider the following efficiency properties:

P1 **Memory footprint**: the size of the predictor, typically measured in terms of the number of parameters or computational operations required.

P2 **Latency**: the time required to perform a single inference.

P3 **Data efficiency**: the amount of data required to train the predictor to achieve a certain level of performance.

P4 **Energy consumption**: the energy required to train and/or run the predictor.

P5 **Predictive performance**: standard metrics such as accuracy, F1-score, or mean squared error.

For brevity, we refer to any function that measures one of these properties as an *efficiency metric*. These metrics quantify the improvement in a given property P of the uneducated predictor N when transformed into its educated counterpart $\hat{N}$ through a SKI mechanism.

The resulting efficiency scores can be influenced by several factors:

F1 **Knowledge quality and coverage**: The educated predictor $\hat{N}$ is obtained by injecting symbolic knowledge K. Both N and $\hat{N}$ are trained on the same dataset D, which describes the learning task. Key questions include: *(i)* Are K and D coherent? *(ii)* Does K cover situations exemplified in D? *(iii)* Is K consistent, coherent, and correct? *(iv)* Can the same be said for D? The answers to these questions significantly affect the efficiency scores, as they depend on the interplay between K and D.

F2 **Baseline quality**: Both N and $\hat{N}$ target the same learning task. Questions to consider include: *(i)* Is N biased in a statistical sense? *(ii)* If so, can $\hat{N}$ improve any efficiency metric P? *(iii)* Even if N is unbiased, is the selected injection mechanism $\mathcal{I}$ suitable for N? These factors highlight the depen-

dency of efficiency measures on the nature of N and the injection mechanism $\mathcal{I}$.

F3 Task at hand: The learning task determines the training dataset D and the test dataset T. The choice of T impacts the evaluation of both N and $\hat{N}$, and consequently, the efficiency scores.

In summary, efficiency metrics evaluate a SKI method $\mathcal{I}$ in a specific context defined by: *(i)* the symbolic knowledge K, *(ii)* the predictor N, *(iii)* the training dataset D, and *(iv)* the test dataset T. Thus, any efficiency metric must be parametric with respect to K, N, D, and T.

5.2.2 Metrics

In this section, we propose the implementation and formalization of metrics to evaluate the efficiency of SKI methods. Specifically, we introduce *memory footprint*, *latency*, *energy consumption*, and *data efficiency* as key performance indicators for assessing SKI. These metrics aim to quantify the computational resource usage of SKI methods and provide insights into their design and optimization. Our focus is on understanding how these metrics can guide the development of SKI-based systems, particularly within the context of multi-agent systems (MASs). An in-depth analysis of the trade-offs between performance and efficiency is crucial for the effective implementation of AI predictors in MASs.

5.2.2.1 Memory footprint

In the context of MASs, the importance of sub-symbolic predictors is growing as the field moves towards more efficient and sustainable AI. The increasing demand for AI systems that can operate on resource-constrained devices, such as IoT and edge devices, has driven research into solutions requiring less memory and computational resources [Kör+22; ACO21; AO21]. Several metrics have been proposed to measure the memory footprint of AI predictors, particularly sub-symbolic ones [Kan+18; Wu+18; LDL21]. For instance, the memory footprint of NNs can be quantified by counting their parameters, floating point operations per seconds (FLOPs), or multiplication addition computations (MACs), which represent the operations required for a single inference [ARO22; Hua+18; Che+19].

These metrics are effective for analyzing predictor complexity and computational efficiency.

SKI mechanisms can reduce the memory footprint of sub-symbolic predictors by injecting prior knowledge, thereby reducing the need for data-driven learning of complex concepts. This reduction in learned concepts often translates to fewer parameters, FLOPs, and MACs, resulting in a smaller memory footprint. We define the memory footprint improvement score as the memory saved by the educated predictor $\hat{N}$ compared to its uneducated counterpart N:

$$\mu_{\Psi,K,N}(\mathcal{I}) = \Psi(N) - \Psi(\mathcal{I}(K,N)), \tag{5.1}$$

where $\mathcal{I}(K,N)$ (a.k.a., $\hat{N}$) represents the educated predictor obtained by injecting symbolic knowledge K into N.

The improvement score depends on the quality and coverage of the input knowledge (F1) and the memory footprint of the input predictor (F2). Higher-quality knowledge reduces the memory requirements of the educated predictor. Similarly, predictors with lower initial memory footprints exhibit smaller improvements. It is important to note that the memory footprint of a NN is task-independent, as it is a structural property of the network.

Finally, a negative score indicates that the educated predictor is more memory-intensive than the uneducated one, suggesting that the SKI approach is ineffective in reducing memory usage.

5.2.2.2 Energy consumption

The relationship between MASs and energy consumption is multifaceted and critical. To operate effectively in resource-constrained environments, MASs require AI systems that minimize energy usage. The distributed nature of MASs, combined with the increasing demand for scalable and power-efficient solutions, further emphasizes this need. However, the dynamic and real-time requirements of many MASs applications often lead to high energy consumption due to computational and resource demands. The complexity of MASs, with multiple agents and their interactions, adds another layer of difficulty, especially when processing large datasets or executing complex algorithms. Thus, energy consumption becomes a

crucial factor in the design and implementation of AI systems within MASs.

Several strategies can address this challenge. One approach involves using energy-efficient hardware, such as low-power processors, or adopting distributed and federated learning techniques to distribute computational loads across multiple devices [Sav+21]. Another approach focuses on optimizing algorithms and data structures to reduce the computational requirements of agents. The integration of sub-symbolic predictors, which typically require fewer computational resources than symbolic AI methods, can also significantly reduce energy consumption. Additionally, techniques such as compression and optimization of sub-symbolic predictors can further enhance energy efficiency.

SKI methods offer a promising opportunity to improve energy efficiency in MASs. By introducing injection mechanisms into the data-driven pipeline of sub-symbolic training, SKI reduces the computational complexity of training and running predictors. This is achieved by leveraging prior knowledge, which complements the training data and simplifies the learning process. As a result, it is essential to evaluate the extent to which SKI mechanisms contribute to reducing the energy consumption of sub-symbolic predictors throughout their lifecycle.

To analyze energy consumption, we first define the lifecycle of AI predictors, which includes the following stages:

1. **Model definition**: Data scientists analyze the task and select suitable sub-symbolic predictors and hyperparameters.
2. **Model training**: The predictor's parameters are tuned using training data, where the size and dimensionality of the dataset impact energy consumption.
3. **Model testing**: The predictor is evaluated on a limited test set to ensure satisfactory performance.
4. **Model deployment**: The predictor is used for inference, with energy consumption depending on usage frequency and application lifespan.

Among these stages, training and deployment are the most resource-intensive. Training involves repeated executions and updates of the predictor, while deployment energy consumption depends on the frequency and duration of usage, which are often unpredictable.

We propose two metrics to measure energy consumption:

- **Inference energy consumption**: The average energy consumed by a predictor N for a single inference is defined as:

$$\Upsilon^{\mathsf{i}}_{\upsilon}(N,T) = \frac{1}{|T|}\sum_{t\in T}\upsilon(N,t), \tag{5.2}$$

  where $\upsilon(N,t)$ measures the energy consumed by N during inference on a sample t from the test set T.
- **Training energy consumption**: The average energy consumed during training is defined as:

$$\Upsilon^{\mathsf{t}}_{\upsilon,\gamma}(e,N,T) = \frac{\gamma(e,N,T)}{e\cdot|T|} - \Upsilon^{\mathsf{i}}_{\upsilon}(N,T), \tag{5.3}$$

  where e is the number of epochs, $|T|$ is the size of the training set, and $\gamma(e,N,T)$ estimates the total energy consumed during training, including inference costs.

The energy consumption improvement of a SKI mechanism $\mathcal{I}$ is defined as the energy saved by the educated predictor $\hat{N} = \mathcal{I}(K,N)$ compared to the uneducated predictor N. We distinguish between improvements during inference and training:

$$\varepsilon^{\mathsf{i}}_{\upsilon,K,N,T}(\mathcal{I}) = \Upsilon^{\mathsf{i}}_{\upsilon}(N,T) - \Upsilon^{\mathsf{i}}_{\upsilon}(\mathcal{I}(K,N),T) \tag{5.4}$$

$$\varepsilon^{\mathsf{t}}_{\upsilon,\gamma,e,K,N,T}(\mathcal{I}) = \Upsilon^{\mathsf{t}}_{\upsilon,\gamma}(e,N,T) - \Upsilon^{\mathsf{t}}_{\upsilon,\gamma}(e,\mathcal{I}(K,N),T) \tag{5.5}$$

These metrics depend on several factors:

- **Input knowledge (F1)**: Complex knowledge may increase training energy consumption but reduce inference energy.
- **Baseline predictor (F2)**: Energy-hungry predictors are more likely to benefit from SKI.
- **Task at hand (F3)**: Task-specific datasets influence energy consumption improvements.

In summary, SKI mechanisms have the potential to significantly reduce energy consumption in MASs, particularly during inference, while potentially increasing

training energy costs. These trade-offs must be carefully evaluated to optimize the overall efficiency of AI systems.

5.2.2.3 Latency

Latency refers to the time required to generate a single prediction using a sub-symbolic predictor. Low latency is crucial in real-world applications where timely responses are essential, such as in intelligent transportation [Gri+20] and e-health [Est+21]. In MASs, latency plays a significant role as collaboration between agents requires minimal computational delays [Hou+17]. Additionally, tasks involving large datasets or complex algorithms, such as decision-making, can further increase latency, making it a critical efficiency measure.

SKI methods can help reduce latency by incorporating symbolic representations, which simplify computations and reduce the complexity of the system. This simplification can also aid in identifying the root causes of increased latency, making SKI a valuable approach for improving computational efficiency.

Latency is typically measured as the average time required to generate predictions over a test dataset T. Formally, the latency of a predictor N is defined as:

$$\Lambda(N, T) = \frac{1}{|T|} \sum_{t \in T} \Theta(N, t), \tag{5.6}$$

where $\Theta(N, t)$ represents the time taken by N to generate a prediction for a sample $t \in T$.

To evaluate the impact of SKI, we define the latency gain $\lambda_{K,N,T}(\mathcal{I})$ as the difference in latency between the educated predictor $\hat{N} = \mathcal{I}(K, N)$ and the uneducated predictor N:

$$\lambda_{K,N,T}(\mathcal{I}) = \frac{1}{|T|} \sum_{t \in T} \Big(\Theta(N, t) - \Theta(\hat{N}, t) \Big) = \Lambda(N, T) - \Lambda(\hat{N}, T), \tag{5.7}$$

where $\mathcal{I}(K, N)$ represents the SKI mechanism injecting symbolic knowledge K into N.

The latency gain depends on several factors:

- **Input knowledge (F1)**: Large or complex knowledge bases may increase

latency due to additional computations, but they can also reduce inference time by simplifying the predictor's operations.

- **Baseline predictor (F2)**: Predictors with higher initial latency are more likely to benefit from SKI.
- **Task at hand (F3)**: Task-specific datasets influence latency measurements, as different tasks may require varying levels of computational effort.

It is important to note that latency is not always directly proportional to the number of operations in the predictor. Sparse or inefficient operations at the hardware level, as well as variations in input data complexity, can significantly affect latency [Shu+21]. Thus, evaluating latency gain provides valuable insights into the computational efficiency of SKI methods.

5.2.2.4 Data efficiency

Data-efficiency is a critical aspect of MASs, as these systems often generate and process substantial amounts of data. Inefficient data management can lead to increased latency, reduced accuracy, and higher energy consumption, all of which negatively impact system performance.

Sub-symbolic predictors, which rely on data-driven training algorithms, offer significant flexibility and performance. However, they require large amounts of data for training, resulting in increased storage and processing demands. Moreover, the quality of the data, particularly its representativeness of the task, is crucial for effective learning. These requirements make data collection time-consuming and potentially prone to subjectivity or uncertainty, as seen in applications like emotion recognition [DR21].

Recent research has focused on developing data-frugal predictors [SS20]. Among these, SKI mechanisms play a significant role by leveraging *a priori* knowledge to reduce the computational burden of learning. Instead of learning all concepts from data, SKI allows injecting prior knowledge into the predictor, potentially reducing the amount of data required to achieve acceptable performance levels. In this sense, SKI can be considered a data-efficiency mechanism.

To quantify the data-efficiency gain of a given SKI mechanism $\mathcal{I}$, we first define the *data footprint* of a predictor N. The data footprint represents the amount of

data required to train N to achieve a specific performance level. Assuming N is trained on a dataset D with samples of varying dimensions, over e epochs, and achieves a performance score $\pi(N, T)$ on a test set T, the data footprint is defined as:

$$\Delta_\pi(e, N, D, T) = \frac{e}{\pi(N, T)} \sum_{d \in D} \beta(d), \tag{5.8}$$

where $\beta(d)$ is the memory size (in bytes) of a single training sample d.

The data-efficiency gain of a SKI mechanism $\mathcal{I}$ is then defined as the difference between the data footprints of the uneducated predictor N and the educated predictor $\hat{N} = \mathcal{I}(K, N)$, trained on datasets D and D', respectively:

$$\delta_{e,K,N,D,D',T}(\mathcal{I}) = \Delta_\pi(e, N, D, T) - \Delta_\pi(e, \mathcal{I}(K, N), D', T) \tag{5.9}$$

To improve data-efficiency, one can reduce the size of the training dataset D', either by decreasing the number of samples or by reducing their dimensionality through compression. Additionally, engineering the SKI mechanism and the educated predictor can further enhance data-efficiency. For instance, the input knowledge K can compensate for the reduced dataset size, as the gain depends on both the quality of K (F1) and the baseline predictor (F2). Predictors that are more data-hungry tend to exhibit higher data-efficiency gains.

5.2.3 Validation

The 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 its respective score. The `Metric` class provides a unified interface for all metrics, ensuring consistency and ease of use. Two main methods are available for computing metric values: *(i)* `compute_during_training`, which evaluates the metric during the training phase, and *(ii)* `compute_during_inference`, which evaluates the metric after the predictors are trained. Both methods accept predictors as input parameters and allow additional customization, such as specifying the training set or batch size.

The implemented metrics include:

1. **Memory**: evaluates memory consumption efficiency, as defined in Equation (5.1).
2. **Energy**: measures energy consumption efficiency, as defined in Equation (5.3).
3. **Latency**: assesses latency efficiency, as defined in Equation (5.7).
4. **Data Efficiency**: evaluates data efficiency, as defined in Equation (5.9).

These metrics are included in the `psyki.qos` package. Notably, the metrics can be computed for any pair of predictors, whether they are educated or uneducated, enabling flexible comparisons.

To validate the proposed QoS metrics, we conducted several experiments. The experimental design is as follows:

1. We selected three classification tasks from the literature, each associated with datasets of increasing cardinality.
2. For each task, we trained an uneducated neural predictor N on the dataset D, using a train/test split. Additionally, we selected a symbolic knowledge base K to inject into N.
3. We applied SKI using multiple injection techniques supported by PSyKI, namely KBANN, KINS, and KILL, resulting in educated predictors $\hat{N}$.
4. Finally, we computed the QoS metrics for each educated predictor $\hat{N}$ and compared them with the uneducated predictor N in terms of data efficiency, energy consumption, memory footprint, latency, and accuracy variation.

The goal of these experiments is to demonstrate the effectiveness of the QoS metrics in evaluating the efficiency of different SKI techniques.

It is important to note that these experiments are not intended as a comprehensive evaluation of SKI techniques. Instead, they aim to validate the proposed QoS metrics by highlighting their ability to reveal variations in efficiency metrics introduced by SKI. Negative values in the results may reflect limitations in the injection algorithms or their implementation in PSyKI. The primary objective of PSyKI is to provide correct, albeit not fully optimized, SKI techniques.

Datasets We selected three datasets from the UCI[3] repository: breast cancer Wisconsin (BCW), PSJGS, and census income (CI). These datasets were chosen due to their varying cardinalities, ranging from 10^2 to 10^4, allowing us to evaluate the scalability and robustness of our predictors and metrics.

The **breast cancer Wisconsin (BCW)** dataset [Wol90] contains 699 instances of breast cancer biopsy results. Each instance includes 9 features summarizing biological characteristics and one class label. Feature values are integers in the range $[1, 10]$, and the `BareNuclei` feature has 16 missing values, which we replaced with 0. The target variable is binary, indicating whether a biopsy is benign or malignant, with class distributions of 458 and 241, respectively. The dataset is used to develop predictors for accurate breast cancer diagnosis based on biopsy features.

The **primate splice-junction gene sequence (PSJGS)** dataset [91] consists of 3,190 instances, each representing a sequence of 60 DNA nucleotides. This dataset has been already introduced in this work in Section 4.3.3.

The **census income (CI)** dataset [Koh96] contains 48,842 instances, each representing an individual from the 1994 United States Census. Features include demographic and occupational information, and the target variable is binary, indicating whether an individual's annual income exceeds $\$50,000$. Class distributions are 37,155 for incomes less or equal to $\$50,000$ and 11,687 for incomes greater than $\$50,000$. We converted the target variable to binary and removed irrelevant or biased features, such as `Fnlwgt` (similarity metric computed over the other features), `Education` (duplicated because of `EducationNumeric`), and `Race` (bias). Remaining features were discretised, with `CapitalGain` and `CapitalLoss` binarised, and nominal categorical features one-hot encoded.

For all datasets, we divided the data into training and testing sets with a 2:3 ratio. The knowledge bases for injection were obtained in a task-specific manner. For the PSJGS dataset, we used a knowledge base described in [TSN90], converted into Prolog form. For the BCW and CI datasets, we generated knowledge bases using a SKE method [Sab+22][4].

[3] `https://archive.ics.uci.edu/`

[4] more details about the knowledge extraction process is explained in the appendix of the original paper [Agi+23]

Dataset	Model	Data efficiency (KB)	Energy (mWh)	Memory (FLOPs)	Latency (ms)	Accuracy (%)
BCW	uneducated	–	–	–	–	94.53
	KBANN	35.89	-1.47 / -0.10	3933	-1.70	95.45
	KILL	4.09	-0.99 / 0	0	0.35	94.63
	KINS	-9.97	-1.22 / -0.09	-559	-1.41	94.29
PSJGS	uneducated	–	–	–	–	93.91
	KBANN	-4946.81	-4.67 / -0.22	-66944	-2.56	92.84
	KILL	553.89	-3 / 0	0	0.04	94.02
	KINS	-954.80	-6.53 / -0.51	-161779	-4.75	93.70
CI	uneducated	–	–	–	–	84.63
	KBANN	1653.79	-1.41 / -0.02	-2468	-0.43	84.78
	KILL	4016.90	-0.70 / 0	4200	0	84.81
	KINS	4263.50	-1.41 / -0.02	-6220	-0.44	84.77

Table 5.1: Comparison of the performance of different models after SKI w.r.t. the uneducated one on the three datasets in terms data efficiency, energy consumption, memory usage, latency, and accuracy. The values reported are the average of 30 runs for each model.

Methodology For each dataset, we define and train several NNs, including one uneducated network and multiple educated counterparts. The educated networks are obtained by applying SKI using the KINS, KILL, and KBANN algorithms, each employing a distinct approach to knowledge injection. This setup allows us to compare and evaluate the performance and efficiency metrics of the predictors.

For the uneducated predictors, we tune the structural hyperparameters, such as the number of layers and neurons per layer, using grid search with cross-validation. The same process is applied to the educated predictors, except for those generated by KBANN, where the architecture is determined by the input knowledge. Specifically, we vary the number of layers (1 to 3) and the number of neurons per layer (10, 50, and 100) to ensure optimal hyperparameter selection while maintaining reasonable computation time.

To ensure statistical significance, we repeat the training process 30 times for each predictor, using different initial conditions and random seeds. This approach reduces variability and provides a more accurate estimate of the predictors' average accuracy. After computing the average accuracy, we evaluate the efficiency metrics for each predictor, including data-efficiency, energy consumption, memory footprint, and latency. The results of this procedure are presented in Table 5.1.

5.2.4 Results and discussion

Each column of Table 5.1 is examined to provide insights into the performance of the predictors. The *data-efficiency* scores vary significantly across predictors and datasets. A positive score indicates that the educated predictor is more efficient than its uneducated counterpart, while a negative score suggests the opposite. This variation highlights the importance of selecting the most suitable predictor for a given task. For instance, the KINS-based solution shows lower data-efficiency scores on the BCW dataset compared to other predictors, suggesting it may not be the best choice for this task. Conversely, all three predictors exhibit positive data-efficiency scores on the CI dataset, indicating improvements across all SKI algorithms.

The *energy consumption* metrics, shown in the second column, are mostly negative for both training and testing phases. This indicates that educated predictors often consume more energy than uneducated ones. For example, the KBANN-based solution generally requires more energy, while the KILL-based solution is more energy-efficient. The complexity of the input knowledge can significantly impact energy consumption, particularly during training. Complex knowledge may improve data efficiency but at the cost of higher energy requirements.

The *memory footprint* results, presented in the third column, reveal mixed outcomes. A positive value indicates reduced memory usage by the educated predictor, while a negative value suggests increased memory consumption. For instance, the KBANN-based solution reduces memory usage on the BCW dataset but increases it on the PSJGS dataset. The KILL-based solution often shows no difference in memory usage, with a metric value of zero.

The *latency* results, shown in the fourth column, compare the time required for inference between educated and uneducated predictors. The KILL-based solution exhibits similar latency to its uneducated counterpart, with metrics close to zero. However, the KBANN and KINS-based solutions show slightly higher latency across all datasets. This increase is likely due to the complexity of the injected knowledge.

Finally, the *accuracy* scores, presented in the last column, indicate that both educated and uneducated predictors perform similarly across all datasets. This

suggests that the predictors are capable of achieving consistent and accurate results regardless of the injection mechanism.

In summary, educated predictors generally require fewer data to achieve similar accuracy, making them advantageous in resource-constrained settings. However, they may incur higher energy and latency costs, depending on the complexity of the injected knowledge. The memory usage varies, with some predictors showing improvements and others not. These findings emphasize the importance of using specific metrics beyond accuracy to evaluate SKI methods.

In this work, we propose a set of QoS metrics to evaluate SKI mechanisms, focusing on efficiency gains. The metrics include *memory footprint efficiency*, *energy efficiency*, *latency efficiency*, and *data efficiency*. To support their practical application, we extend the PSyKI library with a general-purpose implementation of these metrics.

Our experiments demonstrate that these metrics provide valuable insights into the efficiency of SKI mechanisms. For instance, they reveal whether a given mechanism improves a predictor's efficiency according to specific criteria. Additionally, the results highlight areas for improvement in the current PSyKI injection mechanisms.

In the context of MASs, these metrics address challenges such as energy consumption, latency, memory, and data efficiency. By reducing computational requirements and improving data quality, SKI approaches can enhance the performance and efficiency of intelligent systems. Measuring efficiency gains enables the automation of decision-making processes, allowing agents to dynamically optimize their sub-symbolic components to achieve their goals.

5.3 About robustness

Here we present the work *"An Empirical Study on the Robustness of Knowledge Injection Techniques Against Data Degradation"* [Raf+24], presented at the 25th Workshop "From Objects to Agents" (WOA 2024)[5]. With this work, we extend the QoS metrics introduced in Section 5.2 to evaluate the robustness of SKI methods.

[5] `https://www.univda.it/woa2024/`

5.3.1 Motivations

Robustness is a critical challenge for NNs [Liu+18], referring to their ability to maintain performance despite input perturbations. In general, a more robust neural network produces more reliable and accurate predictions.

Several techniques have been proposed to enhance the robustness of NNs, including regularization [MOM12], data augmentation [SK19], and adversarial training [Shr+17]. The interest in robustness stems from two main factors:

1. The need to address limitations of NNs, such as their sensitivity to random perturbations [FFF18; Sze+14].
2. The growing focus on *trustworthy AI* (TAI), which emphasizes transparency, responsibility, ethics, and safety in AI systems.

TAI has gained significant attention due to the increasing adoption of AI and the interest of policymakers, as reflected in guidelines such as those from the European Union [Eur19]. Robustness is a key component of TAI, ensuring that AI systems function consistently and accurately across diverse scenarios. In this context, research on NN trustworthiness has advanced significantly, with neuro-symbolic methods emerging as a promising approach. This work focuses on SKI as a neuro-symbolic integration approach and explores its robustness. The effectiveness of SKI is typically measured by the performance gain of the educated predictor $\hat{N}$ compared to its uneducated counterpart N, using metrics such as accuracy, F1-score, or mean squared error. However, these metrics primarily assess predictive performance and do not capture the robustness of the injection mechanism. To address this gap, this work introduces a metric to evaluate the robustness of SKI mechanisms relative to their uneducated counterparts.

5.3.2 Robustness of SKI methods

In counterfactual analysis [VDH20], controlled perturbations of training data are commonly used to evaluate the robustness of predictors. Inspired by this approach, we define the *statistical robustness* of a SKI procedure. An injection mechanism is considered robust if the predictive performance of the educated predictor is minimally affected by small perturbations in the training data. This section provides an

overview of robustness, detailing the types of perturbations applicable to training data and how their magnitude can be quantified. We define a data perturbation as a modification of a training dataset D by adding, removing, or editing its entries. The perturbed dataset is denoted as $D' = D \circ \Delta D$, where $\circ$ represents the perturbation operator. The magnitude of the perturbation, $\|\Delta D\|$, is a non-negative scalar quantifying the extent of changes introduced. Let $\mathcal{D} = \{\Delta D_1, \ldots, \Delta D_n\}$ be a set of perturbations applied to D. Let N be a predictor trained on D, and $N_{\Delta D}$ be the predictor trained on the perturbed dataset $D \circ \Delta D$, with the same hyperparameters as N. The robustness score of N with respect to $\mathcal{D}$ is defined as:

$$\rho_{N,D}(\mathcal{D}) = \frac{1}{n} \sum_{\Delta D \in \mathcal{D}} \|\Delta D\| \cdot \frac{\pi(N_{\Delta D}, D \circ \Delta D)}{\pi(N, D)}, \tag{5.10}$$

where π is a performance metric, such as accuracy, computed on the respective dataset. Robustness is proportional to the magnitude of perturbations and the ratio of the performance of the perturbed predictor to the unperturbed one. Although all components in Equation (5.1) are positive, not all perturbations improve performance. The performance ratio can exceed 1 when the perturbed model performs better, indicating a positive perturbation. Conversely, it falls below 1 when the perturbed model performs worse, indicating a negative perturbation. A robust predictor exhibits minimal performance degradation under significant perturbations, implying that its robustness increases as the impact of perturbations decreases. The robustness of an injection mechanism can be evaluated by applying Equation (5.1) to an educated predictor $\hat{N} = \mathcal{I}(N, K, D)$, where $\mathcal{I}$ is the injection mechanism and K is the symbolic knowledge. The robustness gain score is defined as:

$$R_{N,D}(\mathcal{I}) = \frac{\rho_{\hat{N},D}(\mathcal{D})}{\rho_{N,D}(\mathcal{D})}. \tag{5.11}$$

A robustness gain $R_{N,D}(\mathcal{I}) > 1$ indicates that the injection mechanism improves robustness compared to the uneducated predictor. Conversely, $R_{N,D}(\mathcal{I}) < 1$ suggests that the educated predictor is less robust under data perturbations. This metric provides an intuitive measure for analyzing the robustness of SKI mechanisms.

5.3.2.1 Perturbation strategies

This section discusses three primary strategies for perturbing datasets: *sample drop*, *noise addition*, and *label flipping*. These strategies simulate common issues that degrade dataset quality, such as missing data, sensor noise, and mislabeling.

Sample drop The *sample drop* strategy models scenarios where an intelligent agent has limited access to training data. This approach is often used to evaluate the effectiveness of neuro-symbolic methods in handling data scarcity [Xu+18]. The process involves randomly removing samples from a dataset D to produce a perturbed dataset D'. The removal is controlled by a *dropping probability* $p \in (0, 1)$, which is shared across all data entries in D. Formally, $p = \mathbb{E}[X_d]$, where $X_d \sim \mathcal{B}(p)$ is a Bernoulli random variable indicating whether a data entry $d \in D$ is included ($X_d = 1$) or excluded ($X_d = 0$) in D'. Under these assumptions, the perturbed dataset is defined as:

$$D' = \{d \mid d \in D \text{ and } X_d = 1\}.$$

To construct a set of perturbations $\mathcal{D} = \{\Delta D_1, \ldots, \Delta D_n\}$, the sample drop process is applied n times with increasing dropping probabilities $0 < p_1 < \ldots < p_n < 1$.

Noise addition The *noise addition* strategy simulates errors in the data acquisition process, such as sensor noise, to evaluate the robustness of neuro-symbolic methods against unreliable data. This involves adding random noise to the entries of D to produce a perturbed dataset D'. The process is controlled by a *noise intensity* $v \geq 0$, which determines the covariance matrix of the noise. Specifically, the noise is modeled as a zero-mean, multivariate normal random variable $V_d \sim \mathcal{N}(0, v \cdot I)$, where I is the identity matrix. The perturbed dataset is then defined as:

$$D' = \{d + V_d \mid d \in D\}.$$

The number of samples remains unchanged, i.e., $|D| = |D'|$. To construct a set of perturbations $\mathcal{D} = \{\Delta D_1, \ldots, \Delta D_n\}$, the noise addition process is applied n times with increasing noise intensities $0 < v_1 < \ldots < v_n$.

Handling discrete features For datasets with ordinal or categorical features, the noise addition process must account for discontinuities. In such cases, the normal noise distribution $\mathcal{N}(0, v \cdot I)$ is replaced with its discrete counterpart [KLS21]. For ordinal features, the noise is more likely to select values close to the original. For categorical features, the noise selects values uniformly, similar to the label flipping strategy described below.

Label flipping The *label flipping* strategy models errors in the labeling process, such as human mislabeling, and is primarily applicable to classification tasks. This involves randomly altering the output labels of data entries in D to produce a perturbed dataset D'. Assume the dataset has a single output feature with K possible classes. For a data entry $d = (x_{d,1}, \ldots, x_{d,m-1}, y_d) \in D$, where $x_{d,j}$ are input features and $y_d \in \{1, \ldots, K\}$ is the output label, the flipping process is controlled by a *flipping probability* $f \in (0, 1)$. The new label Y_d is defined as:

$$P(Y_d = k) = \begin{cases} f/(K-1), & \text{if } k \neq y_d, \\ 1 - f, & \text{if } k = y_d. \end{cases}$$

Thus, the perturbed dataset is:

$$D' = \{(x_{d,1}, \ldots, x_{d,m-1}, y'_d) \mid d \in D \text{ and } Y_d = y'_d\}.$$

To construct a set of perturbations $\mathcal{D} = \{\Delta D_1, \ldots, \Delta D_n\}$, the label flipping process is applied n times with increasing flipping probabilities $0 < f_1 < \ldots < f_n < 1$.

Applicability to regression problems Label flipping is not directly applicable to regression tasks, as the output features are continuous. In such cases, label flipping can be modeled as a specific form of noise addition applied only to the output features. However, this extension is beyond the scope of this work.

5.3.3 Validation

In this section, we describe the experimental setup and results used to evaluate the proposed robustness metric. For reproducibility, the source code of the experiments

is publicly available.[6] We use the same three datasets as in Section 5.2.3: These datasets were selected for their varying characteristics, allowing a comprehensive evaluation of the robustness metric.

For each dataset, we use symbolic knowledge in Prolog syntax to define classification rules. The PSJGS dataset uses the same knowledge base described in Table 4.7. For the BCW and CI datasets, we define custom logic rules due to the lack of predefined knowledge bases like we did in Section 5.2.3.

We use the PSyKI framework to apply three SKI algorithms: KINS [MCO22c], KILL [MCO22a], and KBANN [TSN90]. For each dataset, we train an uneducated model and apply the three injectors using the corresponding knowledge base. The uneducated model consists of two hidden layers, one input layer, and one output layer, with `ReLU` activation functions and a `softmax` output layer. The number of neurons per hidden layer is dataset-specific: [16, 8] for BCW, [32, 16] for CI, and [64, 32] for PSJGS. Training uses the categorical cross-entropy loss function, a batch size of 32, and a maximum of 100 epochs.

We apply three perturbation strategies: *sample drop*, *noise addition*, and *label flipping*. The *drop probability* d varies from 0.0 to 0.95 in steps of 0.05. The *flipping probability* f varies from 0.0 to 0.90 in steps of 0.08. The *noise intensity* v varies from 0.0 to 1.0 in steps of 0.1. Each experiment is repeated 30 times to ensure statistical significance.

Our analysis focuses exclusively on SKI approaches, as no existing metrics directly evaluate training robustness. The only related metric [RGB23] measures generalization capability. Thus, we aim to validate the proposed robustness metric by assessing its ability to identify robust SKI mechanisms.

5.3.4 Results and discussion

This section presents the results of our experiments, focusing on the robustness of educated and uneducated predictors under different perturbation strategies. To compare their performance, we use the *Mann-Whitney U Test* [MN10], a non-parametric statistical test. A p-value ≥ 0.05 indicates no significant difference between the average accuracy distributions, while a p-value < 0.05 suggests oth-

[6] `https://github.com/pikalab-unibo/ski-data-robustness-experiments`

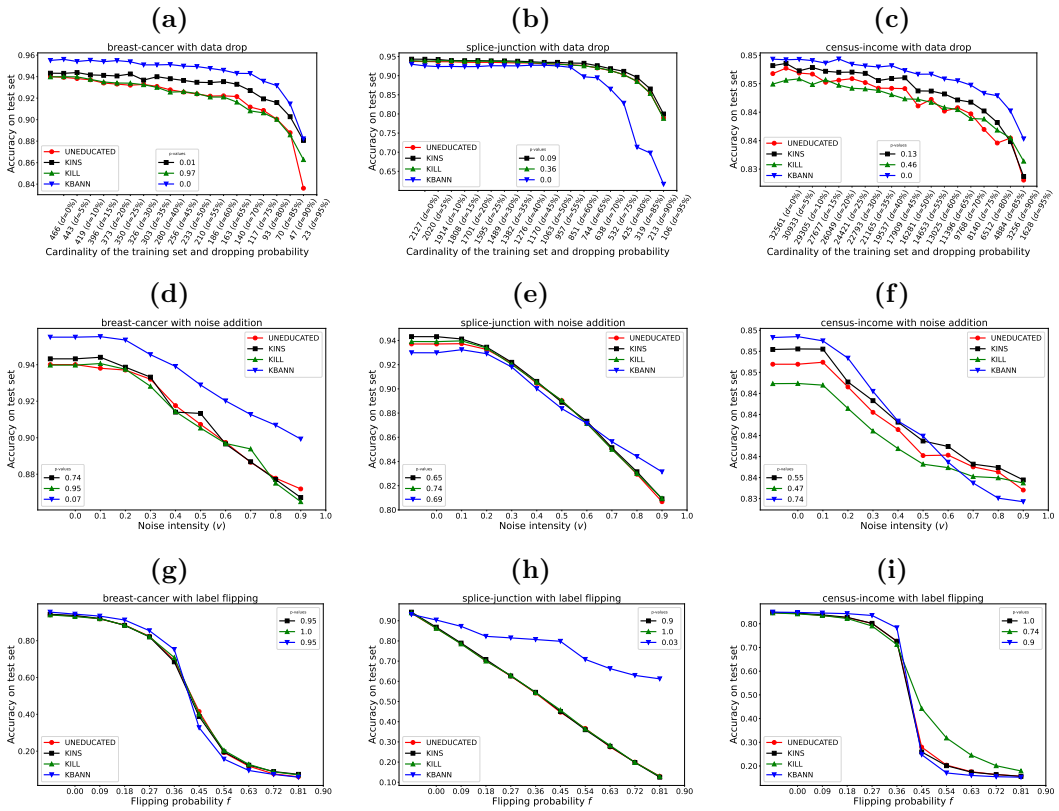


Figure 5.5: Average accuracy over the datasets with different perturbation strategies

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

Table 5.2: Robustness relative scores $R_{N,D}(\mathcal{I})$ for the three perturbation strategies: drop, noise and label flip. Bold numbers are the ones greater than 1 (i.e., the educated model is more robust than the uneducated one).

erwise.

Data drop In the *data drop* experiments (Figures 5.5a to 5.5c), KBANN is the only predictor showing significant improvements compared to the uneducated model. It performs better on the BCW (Figure 5.5a) and CI (Figure 5.5c) datasets but exhibits a sharp decline on the PSJGS dataset (Figure 5.5b) when 60% of the data is removed ($d = 0.6$). KINS shows slightly better performance than the uneducated model, with a p-value of 0.01 on the BCW dataset, indicating significant differences. Overall, educated predictors improve robustness in 6 out of 9 experiments, as shown in Table 5.2.

Noise addition The *noise addition* experiments (Figures 5.5d to 5.5f) reveal that SKI mechanisms are more sensitive to noise than to missing data. Predictors trained on noisy data show a rapid decline in accuracy, starting early in the perturbation process. KBANN outperforms other methods and the uneducated model on the BCW dataset (Figure 5.5d) but performs poorly on the CI (Figure 5.5f) and PSJGS (Figure 5.5e) datasets. Interestingly, KINS performs worse than KBANN, likely due to its trainable modules being more prone to overfitting noisy data. KILL shows similar or worse performance compared to the uneducated model, suggesting that its penalty-based approach does not effectively handle noise perturbations.

Label flipping In the *label flipping* experiments (Figures 5.5g to 5.5i), predictors exhibit similar behavior on the BCW (Figure 5.5g) and CI (Figure 5.5i) datasets, with no significant differences (p-values close to 1). Performance degrades rapidly when more than 54% of labels are flipped ($f = 0.54$). On the PSJGS dataset (Figure 5.5h), KBANN retains nearly 70% accuracy even at the highest flipping probability ($f = 0.9$), outperforming other models, which drop to 20%. This highlights KBANN's strong adherence to injected knowledge, which mitigates the impact of flipped labels. KILL demonstrates good robustness across all datasets, likely due to its loss manipulation strategy, which de-emphasizes corrupted labels during training.

Conclusions The experiments show that SKI methods are most effective in handling missing data, leveraging integrated knowledge to compensate for data scarcity. However, their robustness to noise perturbations is limited, with no significant improvements observed. For label flipping, KILL performs well due to its penalty-based approach, while KBANN excels on the PSJGS dataset due to its strong reliance on injected knowledge. In summary:

1. SKI methods enhance robustness under data scarcity by utilizing integrated knowledge.
2. Loss-manipulating techniques like KILL better tolerate label corruption by reducing the impact of flawed labels.
3. Structuring-based methods like KBANN are more robust than trainable approaches like KINS, as they preserve the integrity of injected knowledge.

These findings emphasize the importance of selecting appropriate SKI methods based on the type of perturbation and dataset characteristics.

Chapter 6

Fairness through SKI

In which other contexts can SKI methods be used effectively?
How AI and society can affect each other?
– **RQ0 and RQ2 to RQ4**

Contents

AI has transformed various aspects of modern society. With recent groundbreaking advancements, its applications are expected to grow exponentially. However, the issue of fairness has become a significant concern. When AI systems are deployed without adequate safeguards, they risk perpetuating or amplifying existing social biases. For example, a recruitment system trained on historical

data dominated by male hires may discriminate against female applicants, reinforcing gender bias [KW20]. To address such challenges, numerous techniques have been developed to mitigate bias in AI. Among these, regularization-based fairness techniques have gained prominence for balancing fairness and predictive performance. Regularization, initially introduced to prevent overfitting, involves adding a penalty term to the loss function during training. Recently, it has been extended to promote fairness by incorporating penalties derived from fairness constraints [KAS11]. These methods aim to reduce the dependence of predictions on sensitive attributes, such as gender, making them effective for bias mitigation during optimization steps like stocastic gradient descent (SGD) in ML algorithms.

6.1 Background

We provide a brief but comprehensive overview of fairness in ML to understand our contributions without delving into the extensive literature on the topic. The section is organised as follows. In Section 2.3.2, we discussed the motivations for fairness in ML. In Section 6.1.1, we introduce common group fairness metrics used in the literature, along with some of their limitations that motivate the development novel metrics. We conclude the background section with a presentation of related works in Section 6.1.2. Finally, in Section 6.2, we present our contributions to fairness in ML.

6.1.1 Fairness metrics

We introduce three of the most popular *group fairness* metrics used in ML to evaluate the fairness of models. Despite being used in many works, these metrics suffer certain limitations, such as being applicable only to binary classification tasks or requiring specific types of sensitive attributes. In particular, all three metrics – as defined below in their first formulation – can support only binary or categorical sensitive attributes, i.e., $A \in \{0, 1\}$ or $A \in \{a_1, a_2, \ldots, a_n\}$ for some $n \in \mathbb{N}$. Moreover, the metrics do not take into account the option to weight groups differently. This possibility could be useful in scenarios of highly unbalanced groups. These limitations are the driver that will lead to the development of novel

fairness metrics in Section 6.2.

Demographic parity (DP) is a fairness metric, also referred to as *statistical parity*, that evaluates whether the predictions of a ML model are independent of a given protected attribute. This implies that the values of the sensitive feature do not influence the model's output [Dwo+12]. Demographic parity (DP) compares the distribution of the model's predictions with the distribution of predictions conditioned on the values of the sensitive attribute. For a binary classifier h and a discrete sensitive attribute A, DP is mathematically defined as:

$$\mathrm{DP}_{h,A}(X) = \sum_{a \in A} \left| \mathbb{E}[h(X) \mid A = a] - \mathbb{E}[h(X)] \right|, \tag{6.1}$$

where X represents the test data, A denotes the sensitive attribute, a is a specific value of A, $\mathbb{E}$ is the expectation operator, and $|\cdot|$ is the absolute value. A model h satisfies DP if the computed value is below a predefined bias threshold ϵ, commonly set to 0.01. DP is particularly useful in applications such as loan approvals, where it ensures that approval rates are consistent across demographic groups, thereby mitigating discrimination.

Disparate impact (DI) quantifies the disproportionate effect of a classifier on individuals based on a sensitive attribute [Fel+15]. For binary classification tasks and binary sensitive attributes, disparate impact (DI) is initially defined as the ratio:

$$\mathrm{di}_{h,A}(X) = \frac{\mathbb{E}[h(X) \mid A = 1]}{\mathbb{E}[h(X) \mid A = 0]}. \tag{6.2}$$

To ensure bounded values within $[0, 1]$, DI is commonly standardized using the function $\eta(x) = \min\{x, x^{-1}\}$, resulting in:

$$\mathrm{DI}_{h,A}(X) = \eta(\mathrm{di}_{h,A}(X)). \tag{6.3}$$

Values of DI above 0.8 are generally considered acceptable, with lower values indicating higher fairness violations. In scenarios requiring positive scores, $1 - \mathrm{DI}$ may be reported, such as during the training of neural networks. DI is applicable in contexts like loan approvals, where it helps identify disproportionate denial rates affecting specific demographic groups.

Equalized odds (EO) measures the extent to which a classifier predicts a given class equally across all values of a sensitive attribute [HPS16]. For binary classification ($Y \in \{0, 1\}$) and a categorical sensitive attribute A, equalized odds (EO) is defined as:

$$\mathrm{EO}_{h,A}(X) = \sum_{(a,y)\in A\times Y} \mathrm{eo}_{h,A}(X, a, y), \tag{6.4}$$

where Y represents the ground truth, and $\mathrm{eo}_{h,A}(X, a, y)$ is given by:

$$\mathrm{eo}_{h,A}(X, a, y) = \left|\mathbb{E}[h(X) \mid A = a, Y = y] - \mathbb{E}[h(X) \mid Y = y]\right|. \tag{6.5}$$

Similar to DP, a classifier is considered fair if its EO value is below a predefined bias threshold ϵ. EO is valuable in applications like loan approvals, where it identifies disparities in approval rates that may favor specific demographic groups.

6.1.2 Fairness and SKI

How promoting fairness in ML models can be achieved through SKI methods? It is definitively possible if we keep our attention on *in-processing* methods, which enforce fairness during model training, and on *group fairness* metrics, which measure the extent to which a model is biased against certain groups. Indeed, the process of including fairness constraints in the training loss can be seen as a form of regularization, similar to how SKI methods incorporate constraints into the optimization process. The knowledge that usually SKI methods leverage is about the ML task at hand (e.g., classification, regression) and the relationships between features and target variables, most commonly in order to improve generalization and interpretability. In the context of fairness, this knowledge includes fairness-related constraints – still consisting in relationships between features and target variables – that are used to ensure that the model does not discriminate against certain groups based on *sensitive attributes*. Instead of using logical operators between expressions involving features and constants (see for example Section 4.2.3), fairness metrics are directly involved. Still, a knowledge of this kind can be expressed with a symbolic formalism, and therefore the whole process can be seen as a form of SKI according Definition 3.1.

6.1.2.1 Related Works

This section highlights key contributions in the literature on *in-processing* fairness methods applied to *group fairness* metrics. Among these, regularization-based approaches are widely used to mitigate bias in AI systems. These methods incorporate fairness constraints directly into the training process by modifying the loss function. The regularization term penalizes unfair predictions, with variations in its definition leading to differences in effectiveness, efficiency, and scalability.

Kernel density estimation (KDE) KDE [CHS20] introduces a non-parametric fairness regularization approach using kernel density estimation. The method estimates probabilities such as $\mathbb{P}[h(X) = y^+ \mid S = s]$ and $\mathbb{P}[h(X) = y^+ \mid S = s, Y = y]$ as differentiable functions with respect to model parameters. This enables direct optimization through gradient-based methods for fairness metrics like DP and EO. Instead of using the final model output $h(X)$, the method relies on an internal value $\tilde{Y} \in [0, 1]$, representing the probability of an instance belonging to the positive class. The final output $h(X)$ is assigned to the positive class if $\tilde{y}_i \geq \tau$, where τ is a predefined threshold. For DP, the probability density function is estimated using a Gaussian kernel:

$$f_k(\tilde{y}) = \frac{1}{\sqrt{2\pi}} \exp\left(-\frac{\tilde{y}^2}{2}\right). \tag{6.6}$$

The probability estimate is obtained by integrating over the region $\tilde{y} \geq \tau$:

$$\mathbb{P}[h(X) = y^+ \mid S = s] = \int_{\tau}^{\infty} \frac{1}{|I_s| b} \sum_{i \in I_s} f_k\left(\frac{\tilde{y} - \tilde{y}_i}{b}\right) d\tilde{y}, \tag{6.7}$$

where I_s is the set of instances with sensitive attribute $S = s$, $\tilde{y}_i$ is the internal model value for instance i, and b is the bandwidth parameter. To achieve DP, the difference in probabilities across sensitive groups is minimized:

$$R_{\text{KDE}}(h, D_{\text{train}}, b) \approx \sum_{s \in S} H_\delta\left(\mathbb{P}[h(X) = y^+ \mid S = s] - \mathbb{P}[h(X) = y^+]\right), \tag{6.8}$$

where $H_\delta(\cdot)$ is the Huber function, a smooth approximation of the absolute value operator [Hub64].

Generalized demographic parity (GDP) GDP [Jia+22] extends DP to continuous sensitive attributes while ensuring differentiability. It measures the deviation between the conditional probability $\mathbb{P}[h(X) = y^+ \mid S = s]$ and the marginal probability $\mathbb{P}[h(X) = y^+]$, weighted by the probability density function of S. Two estimation methods are proposed: histogram-based and kernel-based. The histogram-based method partitions the range of S into non-overlapping bins $B = \{B_1, \ldots, B_{|B|}\}$ of equal width b. Within each bin B_i, the local and global prediction averages are computed as:

$$\text{avg}_{\text{local}}(B_i, h) = \sum_{j \in I_{B_i}} \tilde{y}_j, \quad \text{avg}_{\text{global}}(h, D_{\text{train}}) = \frac{1}{|D_{\text{train}}|} \sum_{n=1}^{|D_{\text{train}}|} \tilde{y}_n, \tag{6.9}$$

where I_{B_i} contains the indices of instances in bin B_i. The fairness regulariser is defined as:

$$R_{\text{hist-GDP}}(h, D_{\text{train}}, b) = \sum_{i=1}^{b} \left(\text{avg}_{\text{local}}(B_i, h) - \text{avg}_{\text{global}}(h, D_{\text{train}})\right) \cdot \mathbb{P}[S \in B_i]. \tag{6.10}$$

Kernel-based estimation provides smoother results by assigning soft group memberships using a kernel function $K(\cdot)$, avoiding discontinuities at bin boundaries.

Fairness through neural networks (FNNC) FNNC [MG20] formulates fairness as a constrained optimization problem. The goal is to minimize predictive performance loss subject to a fairness constraint. For DP, the problem is defined as:

$$h^* = \arg\min_{h \in H} L(h, D_{\text{train}}) \quad \text{s.t.} \quad \mathbb{P}[h(X) = y^+ \mid S = s] - \mathbb{P}[h(X) = y^+ \mid S \neq s] \leq \epsilon, \tag{6.11}$$

where ϵ is a predefined fairness threshold. The fairness constraint is integrated into the loss function using the Lagrangian multiplier method:

$$R_{\text{FNNC}}(h, D_{\text{train}}) = \mathbb{P}[h(X) = y^+ \mid S = s] - \mathbb{P}[h(X) = y^+ \mid S \neq s] - \epsilon. \quad (6.12)$$

Optimization is performed using a two-step SGD procedure, alternating between minimizing model parameters and maximizing Lagrangian multipliers.

Neuro-symbolic integration for fairness (NSIF) NSIF [WG21] leverages the LTN framework to enforce fairness constraints. Fairness constraints are expressed as formal logic rules, which influence the training of NNs. The method validates its effectiveness using DP and DI metrics for categorical sensitive attributes. Although not regularization-based, NSIF provides a unique approach to fairness through symbolic reasoning.

6.2 Fairness under constraints injection

In this section, we present the contribution of the work "Enforcing Fairness via Constraint Injection with FaUCI" [Mag+24], presented at the 2nd AEQUITAS workshop on fairness and bias in AI 2024. The work introduces two distinct contributions: the introduction of novel fairness metrics – more precisely the extensions of DP, DI, and EO to categorical and continuous sensitive attributes – and the development of a novel fairness regularization method, called fairness under constraints injection (FaUCI), which injects fairness constraints into the training loss of a model.

6.2.1 Novel fairness metrics

This section introduces weighted and generalized variants of fairness metrics to support both categorical and continuous sensitive attributes. These extensions aim to generalize fairness metrics beyond binary data, enabling their application to real-world datasets. For instance, they allow prioritizing fairness for underrepresented groups, such as specific ethnicities, when more than two groups exist.

We focus on three metrics: DP, DI, and EO, and propose their weighted and generalized formulations.

Weighted and Generalized Demographic Parity DP evaluates whether the predictions of a model are independent of a sensitive attribute. For binary classification, the values $\mathbb{E}[h(X) \mid A = a]$ and $\mathbb{E}[h(X)]$ from Equation (6.1) are bounded within $[0, 1]$. The maximum theoretical value of DP depends on the number of possible values of the sensitive attribute A, i.e., $0 \leq \text{DP} \leq |A|$. This variability makes comparisons across datasets challenging. To address this, we propose two variants: weighted demographic parity (WDP) and generalized demographic parity (GDP).

Weighted demographic parity (WDP) is defined as:

$$\text{WDP}_{h,A}(X) = \sum_{a \in A} |\mathbb{E}[h(X) \mid A = a] - \mathbb{E}[h(X)]| \cdot w_a, \tag{6.13}$$

where w_a represents the weight of the a-th value of A, and $\sum_{a \in A} w_a = 1$. Weights can be chosen based on the distribution of A or set equally to avoid bias towards frequent values.

GDP extends DP to continuous sensitive attributes:

$$\text{GDP}_{h,A}(X) = \int_l^u |\mathbb{E}[h(X) \mid A = a] - \mathbb{E}[h(X)]| \cdot w_a \, da, \tag{6.14}$$

where l and u are the minimum and maximum values of A, and w_a is a user-defined weight function satisfying $\int_l^u w_a \, da = 1$. The definition of the GDP has been already introduced in a recent work [Jia+22]. It is the only metric among the ones presented in this section that was already existing in the literature.

Weighted and generalized disparate impact DI quantifies the disproportionate effect of a classifier on groups defined by a sensitive attribute. For categorical attributes, we define weighted disparate impact (WDI) as:

$$\text{WDI}_{h,A}(X) = \sum_{a \in A} \eta \left(\frac{\mathbb{E}[h(X) \mid A = a]}{\mathbb{E}[h(X) \mid A \neq a]} \right) \cdot w_a, \tag{6.15}$$

where $\eta(x) = \min\{x, x^{-1}\}$ ensures bounded values within $[0, 1]$.

For continuous attributes, generalized disparate impact (GDI) is defined as:

$$\mathrm{GDI}_{h,A}(X) = \int_l^u \eta\left(\frac{\mathbb{E}[h(X) \mid A = a]}{\mathbb{E}[h(X) \mid A \neq a]}\right) \cdot w_a \, da. \tag{6.16}$$

Weighted and generalized equalized odds EO measures whether a classifier predicts equally across sensitive attribute values and ground truth classes. To address the lack of bounded values, we define weighted equalized odds (WEO) as:

$$\mathrm{WEO}_{h,A}(X) = \sum_{(a,y)\in A\times Y} \mathrm{eo}_{h,A}(X, a, y) \cdot w_a, \tag{6.17}$$

where $\mathrm{eo}_{h,A}(X, a, y)$ is defined in Equation (6.4).

Generalized equalized odds (GEO) extends EO to continuous attributes:

$$\mathrm{GEO}_{h,A}(X) = \int_l^u (\mathrm{eo}_{h,A}(X, a, 0) + \mathrm{eo}_{h,A}(X, a, 1)) \cdot w_a \, da. \tag{6.18}$$

6.2.2 Novel SKI fairness method

The proposed method, FaUCI, is designed for ML models trained using SGD, particularly NNs. It introduces a fairness-specific cost factor into the loss function, which depends on the fairness metric to be minimized. The metrics considered include WDP, GDP, weighted disparate impact (WDI), generalized disparate impact (GDI), weighted equalized odds (WEO), and generalized equalized odds (GEO), as defined in Section 6.2.1. Unlike other methods, FaUCI ensures that the fairness metric is bounded within $[0, 1]$, supports binary, categorical, and continuous sensitive attributes, and can address intersectional fairness.

The loss function is originally formulated as:

$$\mathcal{L}_{h,A}(X, Y) = E(h(X), Y) + \lambda F_{h,A}(X, Y), \tag{6.19}$$

where E represents the error function, $F_{h,A}$ is the fairness regularization term, and $\lambda \in \mathbb{R}_{>0}$ is a hyperparameter controlling the weight of the fairness term. The choice of E depends on the task, such as accuracy or cross-entropy for classification,

while F is selected from the fairness metrics introduced earlier. In a later work – currently under peer review – Equation (6.19) has been modified as follows:

$$\mathcal{L}_{h,A}(X,Y) = (1-\lambda)E(h(X),Y) + \lambda F_{h,A}(X,Y), \tag{6.20}$$

with $0 \leq \lambda \leq 1$. In this way the comparison between FaUCI and other fairness methods is simplified since the range of the hyperparameter λ is well-defined.

The fairness metric $F_{h,A}$ is computed on the input data X, which is divided into batches during training. The batch size, a hyperparameter of SGD, affects the accuracy of fairness estimation. Larger batch sizes improve fairness estimation but slow down the training process, creating a trade-off.

FaUCI offers several advantages over existing methods. First, it supports sensitive attributes of any type and is agnostic to the choice of fairness metric, making it applicable to diverse scenarios. Second, it avoids the need for additional hyperparameter tuning, such as those required by KDE-based methods [CHS20]. Third, the weighted fairness metrics introduced in Section 6.2.1 mitigate biases in unbalanced datasets, ensuring fairness across underrepresented groups. Finally, FaUCI is extensible to other fairness metrics and can be adapted for intersectional fairness, as discussed in Section 6.2.2.2.

6.2.2.1 Theoretical analysis

During the training of a NN, the training set is divided into batches. For each epoch, all batches are processed once for inference (forward propagation) and gradient descent (backpropagation). Between these steps, the loss function is computed to update the model parameters. This section outlines algorithms for computing the fairness cost factor for WDP, WDI, and WEO metrics. Continuous variants are omitted as their computation follows identical steps. For simplicity, the weights are chosen based on the frequencies of the sensitive attribute values, which does not affect the computational complexity.

Algorithm 1 computes the WDP metric for a single batch. The cost of estimating probabilities in line 2 is $O(B)$, where B is the batch size. Lines 4 and 5 also have a cost of $O(B)$, while line 6 has a constant cost. These steps are repeated for each value of A, resulting in a total computational cost of

Algorithm 1 Weighted Demographic Parity (returns wdp)

Require: h predictive model
Require: A sensitive attribute
Require: $batch$ dataset batch
1: $wdp \leftarrow 0$
2: $estimated \leftarrow \mathbb{E}[h(batch)]$
3: **for all** a in A **do**
4: $\quad estimated\_a \leftarrow \mathbb{E}[h(batch) \mid A = a]$
5: $\quad p\_a \leftarrow \mathbb{P}[A = a]$
6: $\quad wdp \leftarrow wdp + (\|estimated - estimated\_a\| \cdot p\_a)$
7: **end for**

$O(B) + 2O(BN) + O(N) = O(BN)$, where N is the number of distinct values of A.

Algorithm 2 Weighted Disparate Impact (returns wdi)

Require: h, A, $batch$ as in Algorithm 1
1: $wdi \leftarrow 0$
2: **for all** a in A **do**
3: $\quad est\_a \leftarrow \mathbb{E}[h(batch) \mid A = a]$
4: $\quad est\_not\_a \leftarrow \mathbb{E}[h(batch) \mid A \neq a]$
5: $\quad p\_a \leftarrow \mathbb{P}[A = a]$
6: $\quad wdi \leftarrow wdi + \left(\min\left\{\frac{est\_a}{est\_not\_a}, \frac{est\_not\_a}{est\_a}\right\} \cdot p\_a\right)$
7: **end for**

Algorithm 2 computes the WDI metric. Lines 3, 4, and 5 have a cost of $O(B)$, while line 6 has a constant cost. These steps are repeated N times, resulting in a total computational cost of $3O(BN) + O(N) = O(BN)$.

Algorithm 3 computes the WEO metric for binary classification tasks. Lines 2 and 3 have a cost of $O(B)$, while lines 5-7 inside the loop also cost $O(B)$. Lines 8-10 have a constant cost. The total computational cost is $2O(B)+3O(BN)+3O(N) = O(BN)$.

All three algorithms have a computational cost linear in both the batch size (B) and the number of values of A (N). For continuous sensitive attributes, the computation involves integrals instead of discrete values. The implementation used in experiments relies on histogram-based methods, dividing the range of A into non-overlapping intervals of equal width. In this case, the computational cost

Algorithm 3 Weighted Equalized Odds (returns *weo*)

Require: h, A, *batch* as in Algorithm 1, Y: ground truth

$weo \leftarrow 0$
$est\_y\_i \leftarrow \mathbb{E}[h(batch) \mid Y = i]$ $\forall i \in \{0, 1\}$
for all a in A **do**
    $est\_a\_y\_i \leftarrow \mathbb{E}[h(batch) \mid A = a, Y = i]$ $\forall i \in \{0, 1\}$
    $p\_a \leftarrow \mathbb{P}[A = a]$
    $weo\_i \leftarrow \|est\_a\_y\_i - est\_y\_i\|$ $\forall i \in \{0, 1\}$
    $weo \leftarrow weo + ((weo\_0 + weo\_1) \cdot p\_a)$
end for

depends on the number of intervals rather than the number of distinct values of A.

6.2.2.2 Intersectional fairness extension

FaUCI is extended to optimize fairness in intersectional settings, where subgroups formed by the intersections of protected attributes are considered. This extension supports fairness metrics involving multiple sensitive attributes through two approaches. The first approach creates a dummy protected attribute by combining the original attributes, enabling FaUCI to optimize fairness for the intersection of these attributes. However, this method may become computationally intractable due to the increased complexity of the fairness metric. To address this, an alternative approach is proposed. The second approach combines multiple loss factors, each penalizing fairness violations for a single protected attribute. The modified loss function is defined as:

$$\mathcal{L}_{h,\bar{A}}(X, Y) = E(h(X), Y) + \lambda_1 F_{h,A_1}(X) + \cdots + \lambda_n F_{h,A_n}(X), \tag{6.21}$$

where $\bar{A}$ represents a set of protected attributes $\{A_1, \ldots, A_n\}$, E is the error function, and F_{h,A_i} is the fairness regularization term for attribute A_i. This formulation ensures that the computational cost of the fairness metric remains linear in the number of attributes, rather than exponential. It provides a trade-off between computational efficiency and the complexity of the fairness metric.

6.2.3 Validation

We validate FaUCI using the UCI Adult dataset, a widely-used benchmark for fairness algorithms [Koh96]. The dataset has been already described in Section 5.2.3 under the name *census income*. Despite its limitations [Din+21], the dataset is a standard in the fairness community due to its inherent biases, making it suitable for our experiments.

Experimental setup To ensure consistency with prior works [MG20; WG21], we employ 5-fold cross-validation for all experiments. The original dataset is combined into a single set of 48,842 records, standardized, and split into training (80%) and test (20%) sets. The training set is further divided into 5 folds for validation. We train a feed-forward NN with two hidden layers containing 100 and 50 neurons, respectively. Training is conducted for up to 5,000 epochs with a batch size of 500. Two early stopping conditions are applied: one when both accuracy and fairness metrics reach predefined thresholds, and another when no improvement is observed for 100 epochs. The target accuracy is set to 0.9, and fairness thresholds are 0.01, 0.99, and 0.01 for DP, DI, and EO, respectively. For DI, the final metric value is expressed as $1 - \text{DI}$ to align with the literature. The baseline model, referred to as the "*uneducated*", is trained without fairness constraints.

Experiments are conducted using WDP, GDP, WDI, GDI, WEO, and GEO metrics across three sensitive attributes: *sex* (binary), *ethnicity* (categorical), and *age* (continuous). For WDP, WDI, and WEO, equal weights are assigned to each value of the sensitive attribute, i.e., $w_a = \frac{1}{N}$, where N is the number of unique values of A. For GDP, GDI, and GEO, weights are based on the density of the continuous attribute to account for its balanced distribution. Users can define custom weights, provided they satisfy the properties outlined in Section 6.2.1. We successfully reproduced the code for two state-of-the-art methods [CHS20; Jia+22], but results for other approaches [MG20; WG21] are reported as provided by the original authors due to replication challenges [1].

[1] for [WG21] we contacted the authors asking explanations about the missing source code that was mentioned in the paper. We received a reply from the authors stating that the code was not public anymore, and it was never shared with us. In the case of [MG20], the source code was not even mentioned in the paper, and we were not able to find it online.

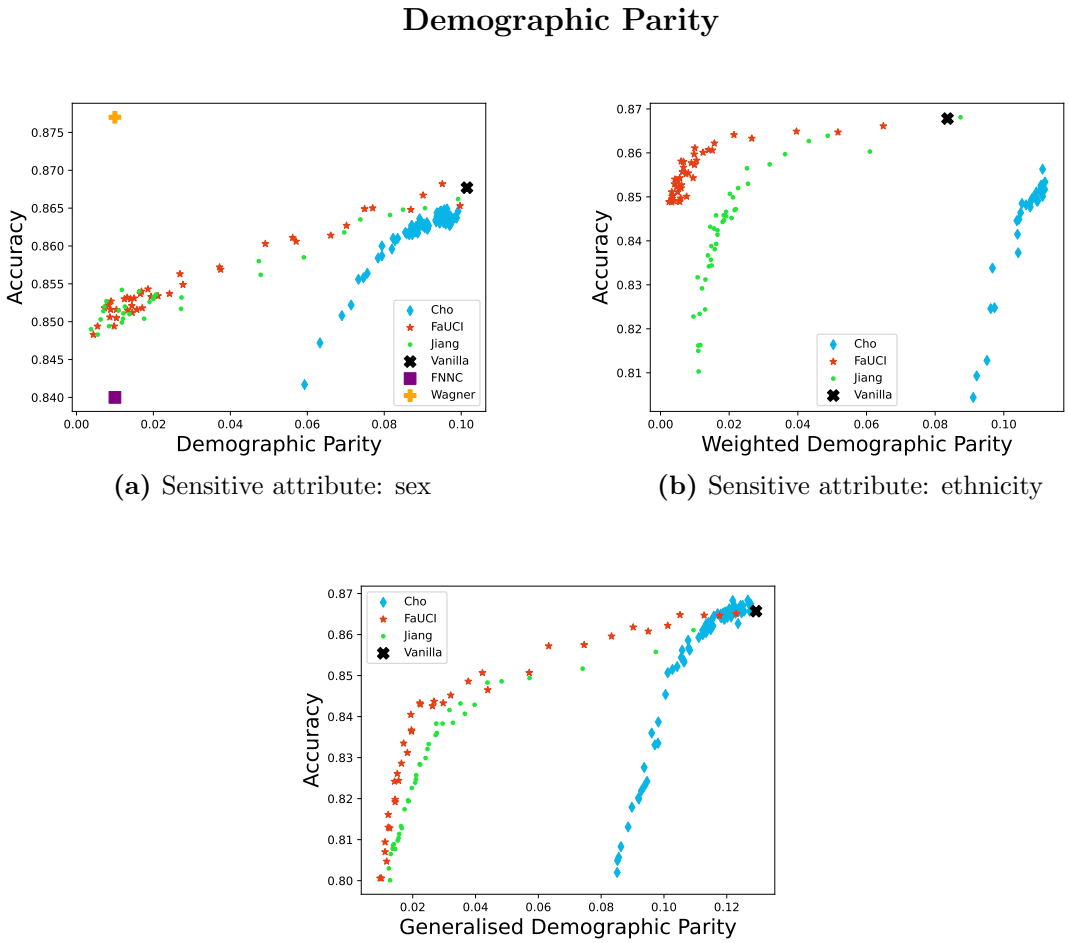
6.2.4 Results and discussion

Demographic parity results Figures 6.1a to 6.1c illustrates the trade-off between accuracy and DP for FaUCI, KDE, GDP, FNNC, and NSIF methods. For *sex* as a binary sensitive attribute, FaUCI achieves an accuracy of 85.27% with DP ≤ 0.01, outperforming KDE and matching GDP's performance. FNNC and NSIF results are represented by single points, with accuracies of 84% and 87.7%, respectively. For *ethnicity*, a categorical sensitive attribute, FaUCI outperforms all methods, particularly when DP ≤ 0.04. For *age*, a continuous sensitive attribute, FaUCI and GDP perform similarly, but FaUCI generally achieves better results. Overall, FaUCI demonstrates superior performance across all sensitive attributes.

Disparate impact results Figures 6.1d to 6.1f shows the trade-off between accuracy and DI. For *sex*, FaUCI achieves DI ≥ 0.8 with an accuracy of 85.6%, surpassing FNNC (82.2%) and matching NSIF (87.7%). For *ethnicity*, FaUCI maintains the accuracy of the uneducated model (86.8%) while achieving DI ≥ 0.8. For *age*, FaUCI improves DI to 0.58 with an accuracy of 85.5%, though it does not reach the desired threshold of 0.8. These results highlight FaUCI's ability to enhance DI while preserving high accuracy.

Equalized odds results Figures 6.1g to 6.1i presents the trade-off between accuracy and EO. For *sex*, FaUCI achieves EO ≤ 0.02 with an accuracy of $\geq 85.9\%$, outperforming KDE and FNNC (83.9%). For *ethnicity*, FaUCI maintains high accuracy while reducing EO, whereas Cho's accuracy drops significantly. For *age*, FaUCI outperforms KDE in both accuracy and EO. Overall, FaUCI demonstrates consistent improvements in EO while preserving a favorable accuracy-fairness trade-off.

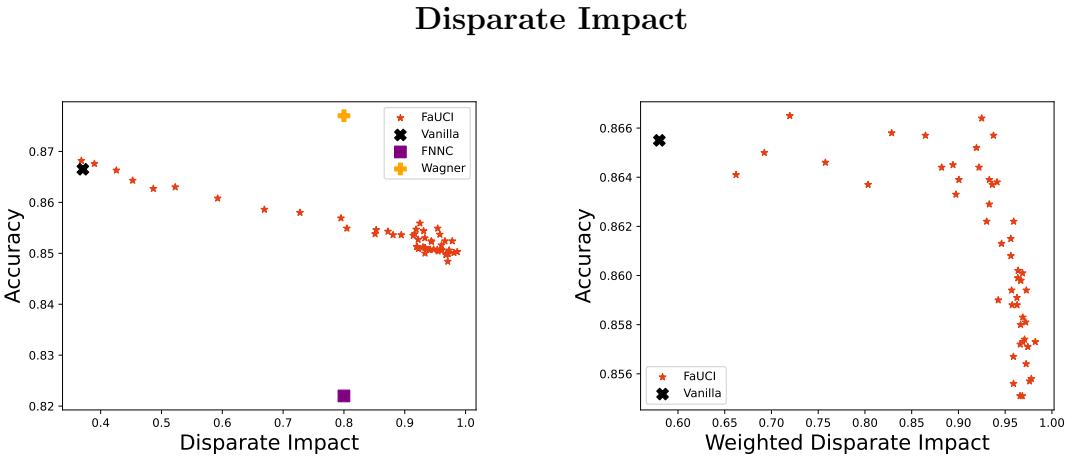
Figure 6.1: Experiments comparing FaUCI to related works, using different (variants of) the DP, DI, and EO fairness metrics. Each block corresponds to a different fairness metric. Each dot represents the average of 5 runs (5-fold cross-validation). Red stars represent FaUCI, blue diamonds Cho's method, green dots Jiang's method, and the black cross the vanilla neural network (i.e., the NN without fairness constraints). For FNNC and LTN we report the results provided by the respective authors.



(a) Sensitive attribute: sex

(b) Sensitive attribute: ethnicity

(c) Sensitive attribute: age

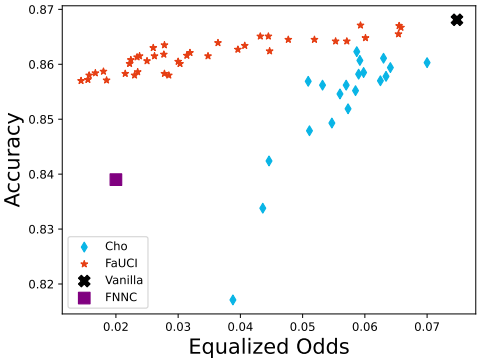


(d) Sensitive attribute: sex

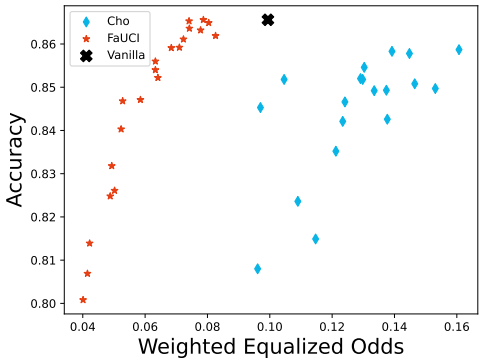
(e) Sensitive attribute: ethnicity

(f) Sensitive attribute: age

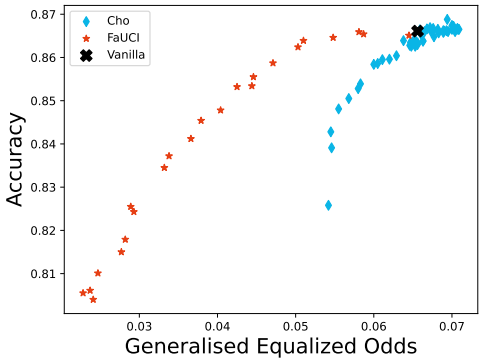
Equalized Odds



(g) Sensitive attribute: sex



(h) Sensitive attribute: ethnicity



(i) Sensitive attribute: age

Part III

Engineering of intelligent systems

Chapter 7

NeSy AI for real world applications

How to apply SKI and SKE methods to design real-world AI systems?
How can LLMs be exploited in the context of SKI and SKE?
– **RQ0, RQ3 and RQ4**

Contents

In this chapter we present a selection of contributions about NeSy application in real-world scenarios. These works involve the use of both SKE and SKI methods, as well as traditional ML models and the recent LLMs.

7.1 SKE for explainable nutritional recommenders

"Symbolic knowledge extraction for explainable nutritional recommenders" [Mag+23], published on the Journal of Computer Methods and Programs in Biomedicine, presents a novel nutritional recommender system that leverages SKE.

7.1.1 Nutritional recommender systems

Eating habits significantly impact the well-being of individuals across all age groups, highlighting the importance of developing nutritional recommender systems (RSs) [EHN16; Cio+18; SSW22]. These systems address diverse user needs, including diet programs, chronic disease management, treatment for critically ill patients, allergies, lifestyle choices (e.g., sporty, vegetarian, organic, halal), and physical activity levels [OKM21; Hez+19; Aga+18; FT90; Tra+18]. User preferences are represented either by expert knowledge, such as daily nutritional intake

limits based on physical activity levels, or by user feedback, such as reviews on recipes [Wan+21b]. Recommendation items, such as food, recipes, and meals, are represented in terms of their attributes, including cuisine, category, cooking style, preparation time, and nutritional levels. The complexity of nutritional RSs arises from the combination of multiple ingredients to form recipes and the diverse attributes influencing user preferences.

Classical approaches to nutritional recommendation rely on content based and collaborative filtering methods, which derive user profiles from past activities, such as ratings, clicks, reviews, and browsing history [MJJ20]. Recent advancements leverage ML techniques, including question answering over knowledge bases, recipe retrieval from visual records of ingredients, and learning recipe representations from multi-modal data [FZ11; Bia+17; FB10; Che+21; Tia+22]. Despite the availability of recipes online, many are suboptimal in terms of health [TE17]. Recent studies aim to promote healthy eating by incorporating healthiness indicators into recommendations or enhancing the visual appeal of food [OKM21; Wan+21b]. Providing explanations for recommendations improves user trust and acceptance, as transparent systems are preferred over black-box models [Anj+19]. Explainable approaches, such as explanation ontologies and multi-agent architectures, have been proposed to enhance both personalisation and explainability in nutritional RSs [Pad+21; YAM22; Cal+21].

7.1.2 Methodology

We propose a general architecture for nutritional RSs, designed to provide personalised dietary recommendations aligned with specific user goals. Figure 7.1 illustrates the main components and data flow within the system. The proposed architecture personalises dietary recommendations by identifying recipes that lie at the intersection of user preferences and expert prescriptions. This process relies on three distinct sources of information: *(i)* the user, *(ii)* the nutrition expert, and *(iii)* the recipe database. Users provide their preferences, which are learned adaptively by a sub-symbolic ML predictor. Nutrition experts translate dietary goals into structured prescriptions, which define patterns of recipes suitable for achieving these goals. The recipe database stores information about recipes, their

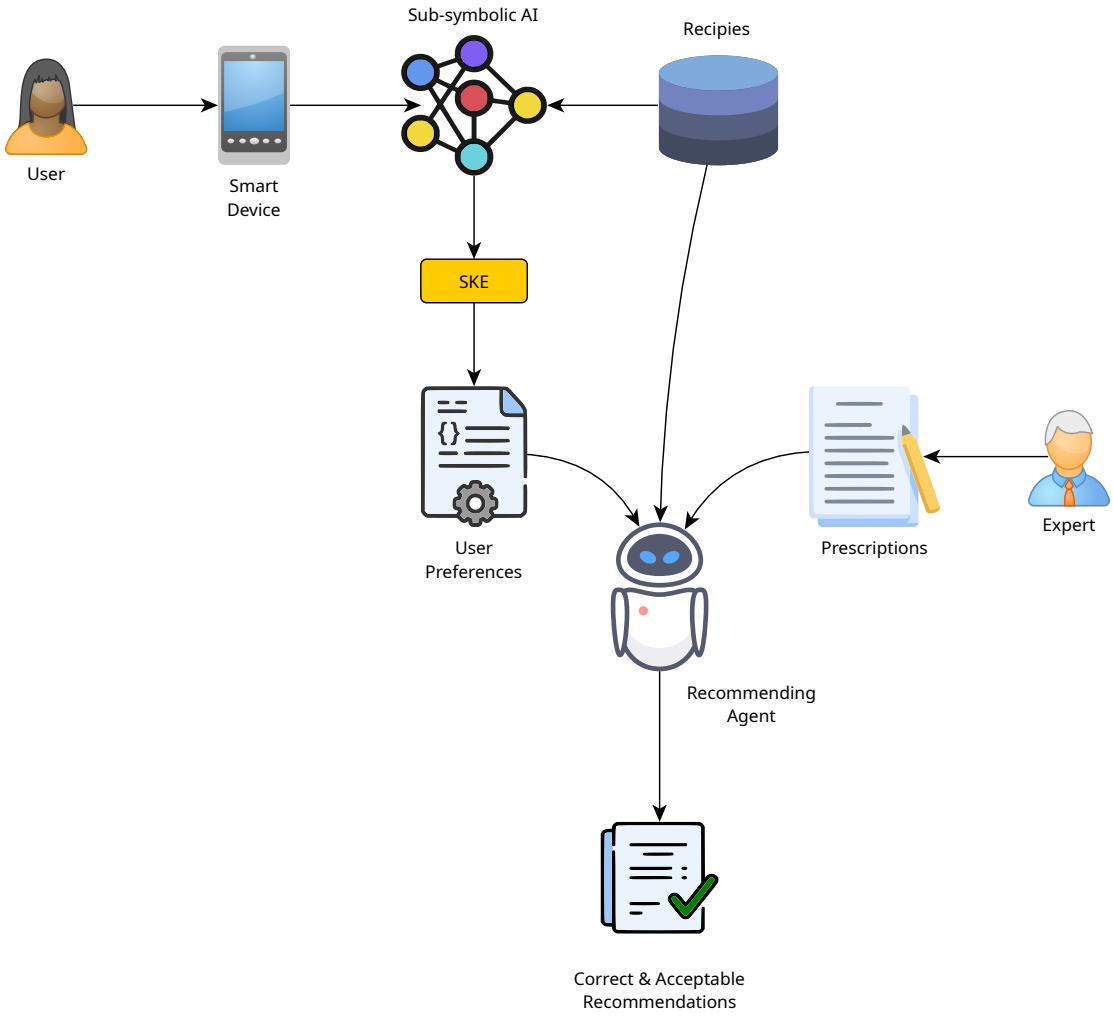


Figure 7.1: Data-flow perspective of a nutritional RS. The user interacts – via some smart/wearable device – with a sub-symbolic AI predictor, continuously trained to predict whether the user likes a given recipe or not. A knowledge base is then extracted from the predictor, representing the user's preferences in a human-interpretable logic form. Dietary prescriptions – provided by human experts – are transformed into the same logic form. Conversely, databases providing information about recipies – there including ingredients and their nutrients – are assumed to be remotely available via the Internet. Finally, the recommending agent exploits a logic engine, combining all such information into recommendations which are simultaneously correct (w.r.t. experts prescriptions), acceptable (w.r.t. users' preferences), and explainable.

ingredients, and corresponding nutrients, enabling the agent to propose admissible options. Recommendations are generated by combining these sources using a logic-based approach, ensuring alignment with both user preferences and expert prescriptions. In the following sections, we detail how preferences, prescriptions, and recipes are represented and manipulated within the system. We also describe the algorithmic process for producing recommendations and discuss its underlying assumptions.

Recipes Our architecture identifies three key data types: recipes, ingredients, and nutrients. Recipes are composed of ingredients, which in turn contain various nutrients. Each data type is named, meaning recipes, ingredients, and nutrients are uniquely identified by their names. The architecture does not track the preparation process of meals but focuses on the nominal composition of ingredients and their quantities. By analyzing the composition of ingredients in terms of nutrients, the system computes the overall nutritional values of recipes. The recipes' database is responsible for storing information about recipes, ingredients, and nutrients. It supports several types of queries, such as:

- Selecting recipes based on their ingredients or nutrients.
- Filtering recipes by specific quantities of ingredients or nutrients.
- Retrieving the ingredients or nutrients associated with a recipe.
- Clustering recipes with similar nutritional profiles.

These queries are essential for the recommender agent's functionality.

Users' preferences User preferences are represented sub-symbolically using a ML predictor. This predictor learns user tastes adaptively from data, which includes information about recipes the user likes or dislikes. Preferences are modeled as a function:

$$\text{appreciation} : R \to \mathbb{R} \tag{7.1}$$

where R is the set of admissible recipes, and the output is an appreciation score. Positive values indicate liking, negative values indicate dislike, and zero represents neutrality. The appreciation score encapsulates user opinions, including factors

such as taste, allergies, and ethical beliefs. Data is collected during user interactions, often via smart or wearable devices, and used to train the ML predictor. The learning process is continual, ensuring the system adapts to evolving user preferences. The sub-symbolic approach enables generalization of user data while leveraging information from the recipes' database.

Dietary prescriptions Dietary prescriptions are structured representations of what a user should eat and when, to achieve specific goals. They are typically created by nutrition experts based on the user's physiological characteristics and expert knowledge. Prescriptions consist of two components:

- The *what* component specifies ingredients or nutrients and their quantities for each meal.
- The *when* component indicates the time of consumption (e.g., breakfast, lunch, dinner).

The goal reflects the expected long-term effect on the user's health, though it may not be explicitly stated. Prescriptions are often provided in quasi-natural language or tabular formats. In tabular form, each cell corresponds to a specific time and meal, listing the recommended nutrients or ingredients. Users may construct recipes based on these suggestions unless a nutritional RS is available. For simplicity, we focus on a single prescription for a given time t, expressed as logic formulas. These formulas define the properties a meal should have to align with dietary goals. Details on the use of logic formulas for dietary prescriptions are provided in the next subsection.

The role of logic formulas The proposed architecture utilizes Horn clauses (see Section 2.2.1.2) to represent both user preferences and expert prescriptions. Horn logic provides a clear and computationally tractable framework for expressing dietary prescriptions. These formulas can be manually written by humans or algorithmically generated, enabling their use in RSs for dietary recommendations.

Expert prescriptions at time t are defined as a set of Horn clauses describing the `should_eat/1` predicate. This predicate specifies the recipes that the user should consume, based on admissible or forbidden ingredients and nutrients. Two

auxiliary predicates, `has/2` and `has_no/2`, are used to assert the presence or absence of specific ingredients or nutrients. Groups of ingredients or nutrients can also be defined using unary predicates, such as `vegetable/1`.

For example, a dietary prescription for Monday lunch can be expressed as:

$$\begin{aligned} \texttt{should\_eat}(R) \leftarrow\ & \texttt{has}(R, \texttt{rice}) \wedge \texttt{has}(R, \texttt{chicken}) \\ \wedge\ & \texttt{has\_no}(R, \texttt{salt}) \\ \wedge\ & \texttt{has}(R, X) \wedge \texttt{vegetable}(X) \end{aligned}$$

indicating that the user should consume recipes containing rice, chicken, and any vegetable, but excluding salt.

To ensure self-containment, additional rules must define the semantics of predicates such as `has/2`, `has_no/2`, and `vegetable/1`. For brevity, these definitions are omitted here.

Alternative encodings, such as including a predicate `quantity(X, Q)` to specify the quantity Q of ingredient X, are possible but do not alter the core contribution. Thus, simpler syntactic choices are preferred.

User preferences can also be represented as Horn clauses, defining the `likes/1` predicate. These clauses may use the same auxiliary predicates (`has/2`, `has_no/2`) and custom predicates for grouping ingredients or nutrients.

Consider the following example:

$$\begin{aligned} \texttt{likes}(R) \leftarrow\ & \texttt{has}(R, \texttt{rice}) \wedge \texttt{has}(R, \texttt{chicken}) \\ \wedge\ & \texttt{has\_no}(R, \texttt{broccoli}) \wedge \texttt{has}(R, \texttt{peas}) \\ \texttt{vegetable}(\texttt{peas}) \leftarrow & \\ \texttt{vegetable}(\texttt{broccoli}) \leftarrow & \\ \texttt{has}(\texttt{paella}, \texttt{rice}) \leftarrow & \\ \texttt{has}(\texttt{paella}, \texttt{chicken}) \leftarrow & \\ \texttt{has}(\texttt{paella}, \texttt{peas}) \leftarrow & \\ \texttt{has}(\texttt{paella}, \texttt{seafood}) \leftarrow & \end{aligned}$$

This example states that the user likes recipes containing rice, chicken, and peas (e.g., paella), but dislikes broccoli. It also defines peas and broccoli as vegetables and specifies the composition of paella.

When both prescriptions and preferences are expressed as Horn clauses, their intersection can be computed using logic resolution. This involves proving the query:

$$\texttt{likes}(R) \wedge \texttt{should\_eat}(R) \tag{7.2}$$

against the merged clause set of prescriptions and preferences. The logic solver identifies recipes R that satisfy both conditions or determines that no such recipes exist.

For instance, testing the query above against the merged clauses from the examples provided yields $R = \texttt{paella}$. This result is denoted as:

$$(3) \cup (4) \models \texttt{likes}(\texttt{paella}) \wedge \texttt{should\_eat}(\texttt{paella}) \tag{7.3}$$

The role of SKE Our proposed architecture requires both user preferences and expert prescriptions to be expressed as sets of Horn clauses. This representation enables the use of logic resolution to construct recommendations. Expert prescriptions are typically provided in formats, such as timetables of suggested recipes, that can be directly expressed or automatically converted into Horn clauses [Mak87]. This assumption aligns with current practices in dietary planning. In contrast, user preferences are modeled using sub-symbolic predictors, such as trained NNs, which adaptively learn preferences from data. While sub-symbolic representations are effective for capturing dynamic user preferences, they are incompatible with direct logic resolution. To bridge this gap, the architecture incorporates a SKE step. This step extracts symbolic knowledge in the form of Horn clauses from the sub-symbolic predictor trained to model user preferences. The choice of SKE algorithm is left to the implementer, allowing flexibility to select the most suitable method for their application. However, the extraction process must produce Horn clauses to ensure compatibility with the logic-based recommendation framework. This approach combines the adaptability of sub-symbolic models with the interpretability and computational efficiency of symbolic reasoning.

7.1.3 Validation

To validate the proposed architecture, we conducted a series of experiments to assess its effectiveness in nutritional RSs. The source code and instructions for reproducing these experiments are public available[1] The experiments involved four main steps:

1. Generating synthetic datasets to simulate a single user's food preferences.
2. Training a ML predictor, specifically a neural network, to predict whether a recipe would be liked by the user.
3. Applying a SKE algorithm to extract symbolic knowledge that represents the decision-making behavior of the predictor.
4. Evaluating the system's ability to recommend recipes that align with both user preferences and expert nutritional prescriptions.

Details regarding data selection, synthesis, and experimental procedures are provided in the following subsections.

Datasets We utilized a public dataset of recipes[2] and generated 12 synthetic datasets to represent the preferences of imaginary users. The recipe dataset consists of four files:

- **Recipe details:** Contains recipe IDs, titles, sources, and cuisines.
- **Ingredients:** Lists basic ingredients with aliases, synonyms, entity IDs, and categories.
- **Compound ingredients:** Includes compound ingredients with their constituent components and categories.
- **Recipe-ingredients aliases:** Maps recipes to their ingredients using aliases and entity IDs.

The dataset includes 929 unique basic ingredients and 103 compound ingredients, categorized into 21 groups (e.g., additive, bakery, beverage). Recipes with at least one ingredient total 45,749, while recipes without ingredients are excluded.

[1] `https://github.com/pikalab-unibo/mccao-cmpb-experiments-2022`
[2] available at `https://cosylab.iiitd.edu.in/culinarydb`

Synthetic datasets were created in two steps. First, unconditional preferences for individual ingredients were generated based on predefined user profiles. Each profile specifies ranges of values for ingredients or categories (e.g., vegetables, meat), which are used to compute likelihoods via uniform distribution. Second, recipe likability labels (like/dislike) were generated based on the likelihood values of the recipe's ingredients. The synthesis process ensures no real personal data is used, avoiding ethical or privacy concerns.

7.1.3.1 Learning user preferences via sub-symbolic predictors

User preferences were modeled using fully-connected neural networks. Each network consists of one input layer, two hidden layers, and one output layer, with 1032, 16, 8, and 1 neurons, respectively. The activation functions for the input and hidden layers are ReLU, while the output layer uses a sigmoid function. The input to the network is a tensor representing the presence of 1032 ingredients in a recipe, and the output is a scalar indicating the likelihood of user appreciation.

A separate NN was trained for each of the 12 synthetic users. Training was performed on half of the dataset (22,874 records), while the remaining half was used for testing. The training process lasted 20 epochs with a batch size of 32. The average accuracy achieved on the test set was 85.6%, with precision computed as the ratio of true positives to total positive predictions. Precision is critical for systems where identifying true positives (liked recipes) is more important than minimizing false positives.

7.1.3.2 Extracting user preferences via SKE

Symbolic knowledge was extracted from the trained ML predictors using the CART algorithm [Bre+84]. This algorithm generates decision trees, which are converted into logic rules. Each path from the root to a leaf in the decision tree corresponds to a rule, where nodes represent logical conditions (e.g., presence or absence of ingredients) and leaves denote the predicted class.

The maximum number of leaves was set to $R = 50$, and the maximum depth of the decision tree was limited to $D = 10$. These constraints balance computational efficiency and interpretability, ensuring the rules are concise and comprehensible.

Table 7.1: Datasets statistics for each users on test sets. The second column shows the *like* class ratio. Third and fourth columns describe the accuracy and precision score of trained neural networks. Fifth and sixth columns describe the accuracy and precision score of the extracted rules. The last column shows the fidelity of the extracted rules w.r.t. the NN.

users	**liked ratio**	**net acc.**	**net prec.**	**r. acc.**	**r. prec.**	**r. fid.**
user 1	0.336	0.9233	0.8855	0.864	0.8228	0.873
user 2	0.2842	0.9714	0.9564	0.794	0.7688	0.7972
user 3	0.3941	0.8503	0.8221	0.7726	0.7136	0.7955
user 4	0.4502	0.8364	0.8165	0.7562	0.7145	0.7813
user 5	0.3022	0.9621	0.9478	0.8381	0.8301	0.842
user 6	0.2719	0.9722	0.9514	0.7997	0.7109	0.8034
user 7	0.3504	0.8916	0.8369	0.7782	0.7311	0.7901
user 8	0.449	0.7782	0.7594	0.6868	0.674	0.7283
user 9	0.393	0.8085	0.7783	0.7493	0.7524	0.803
user 10	0.4734	0.7381	0.7582	0.6862	0.7452	0.7616
user 11	0.4359	0.7708	0.7863	0.7098	0.7426	0.7804
user 12	0.4353	0.7772	0.7979	0.7047	0.7467	0.7814
average	0.3813	0.8567	0.8414	0.7616	0.7461	0.7948

For example, extracted rules for a user might include:

$$\text{likes}(R) \leftarrow \text{has\_no}(R, \text{egg}) \wedge \text{has\_no}(R, \text{pepper}) \wedge \text{has}(R, \text{almond})$$
$$\neg\text{likes}(R) \leftarrow \text{has}(R, \text{egg}) \wedge \text{has}(R, \text{parsley})$$

These rules approximate the decision-making process of the neural network, enabling explainability.

We constrain the output rules for two main reasons. First, limiting the growth of the DT reduces computational complexity. Given N binary features, such as ingredients, the maximum depth of the DT is $N+1$, and the maximum number of leaves is 2^N. For $N = 1032$, this would be computationally infeasible. Second, we aim to ensure the extracted rules remain interpretable. Rules with excessively long right-hand sides or an overwhelming number of conditions are difficult for humans to read and understand. This trade-off prioritizes interpretability over performance metrics, such as accuracy and precision, which is essential for explaining why a prescription may or may not be suitable for a user.

Table 7.2: Precision values of the algorithm per user and prescription. Precision values denote actually liked recipes (i.e., true positive) over the all proposed recipes (i.e., true positive plus false positive) obtained by prescriptions and extracted rules.

users	p. 1	p. 2	p. 3	p. 4	p. 5	p. 6	average
user 1	0.831	0.8	0.8621	0.6667	0.6875	0.7857	0.7722
user 2	0.6338	0.2979	0.8	0.7083	0.9268	0.7727	0.6899
user 3	0.7625	0.4333	0.7297	0.75	0.6111	0.6604	0.6578
user 4	0.75	0.6	0.8387	0.65	0.5435	0.7755	0.693
user 5	0.8571	0.7667	0.95	0.7083	0.8182	0.8095	0.8183
user 6	0.6	0.5116	0.8636	0.8261	0.55	0.619	0.6617
user 7	0.7625	0.4667	0.1852	0.8276	0.7391	0.7826	0.6273
user 8	0.85	0.75	0.9048	0.5758	0.6667	0.7872	0.7558
user 9	0.6316	0.9	0.8929	0.8085	0.8936	0.6809	0.8012
user 10	0.875	0.66	0.7692	0.7879	0.717	0.7692	0.763
user 11	0.8625	0.84	0.963	0.7391	0.64	0.8113	0.8093
user 12	0.7875	0.8333	0.9024	0.8108	0.7037	0.8776	0.8192

Table 7.1 summarizes the accuracy of the extracted rules on the test sets. It also reports the fidelity of the rules with respect to the sub-symbolic predictors. Fidelity is computed similarly to accuracy but compares the predictions of the extracted rules against the class values predicted by the ML predictor, rather than the ground truth. In other words, fidelity measures how closely the extracted rules replicate the behavior of the neural networks.

7.1.3.3 Proposed recipes

The experiments aim to evaluate how user preferences and expert prescriptions are combined to recommend recipes. We rely on sets of logic rules to express domain-expert prescriptions, ensuring consistency with the formalism used for user preferences. In total, six prescriptions are defined, corresponding to three days with two meals per day. For each meal, multiple rules (ranging from two to four) are specified, one for each dish.

p.1 First day, lunch: "Rice with vegetables." Ingredients include 80 grams of raw rice, 35 grams of raw lentils, 120 grams of raw chicken, 120 grams of

Table 7.3: Precision values of the algorithm per user and prescription. Note that precision values denote actually liked recipes (i.e., true positive) over the all proposed recipes (i.e., true positive plus false positive) obtained by prescriptions and NN.

users	**p. 1**	**p. 2**	**p. 3**	**p. 4**	**p. 5**	**p. 6**	**average**
user 1	0.8267	0.8333	1.0	0.9474	0.8696	0.8913	0.8947
user 2	0.95	0.9474	0.931	1.0	0.95	0.94	0.9531
user 3	0.7875	0.7797	0.7273	0.8478	0.88	0.8163	0.8064
user 4	0.775	0.7833	0.871	0.8158	0.7442	0.7447	0.789
user 5	0.9265	0.9333	0.9565	0.9474	0.9524	0.9375	0.9423
user 6	0.9375	0.9833	1.0	0.9615	0.9592	0.9583	0.9666
user 7	0.8	0.8667	0.9	0.8571	0.8723	0.8913	0.8646
user 8	0.8625	0.7833	0.8837	0.7	0.8077	0.8065	0.8073
user 9	0.7875	0.7833	0.9118	0.8095	0.6786	0.6842	0.7758
user 10	0.8125	0.5667	0.8049	0.7632	0.7959	0.84	0.7639
user 11	0.8875	0.7667	0.7674	0.8723	0.7917	0.7931	0.8131
user 12	0.825	0.8667	0.8049	0.68	0.8246	0.8364	0.8063

mixed vegetables, garlic, herbs, 2 teaspoons of olive oil, and 1 orange.

$$\begin{aligned}
\text{should_eat}(R) &\leftarrow \text{has}(R, \text{chicken}) \wedge \text{has}(R, \text{rice}) \\
\text{should_eat}(R) &\leftarrow \text{has}(R, \text{lentils}) \\
\text{should_eat}(R) &\leftarrow \text{has}(R, \text{orange}) \\
\text{should_eat}(R) &\leftarrow \text{has}(R, \text{garlic}) \\
&\wedge \ \text{has}(R, X) \wedge \text{has}(R, Y) \wedge \text{has}(R, Z) \\
&\wedge \ \text{vegetable}(X) \wedge \text{herb}(Y) \wedge \text{essential_oil}(Z)
\end{aligned}$$

p.2 First day, dinner: "Burger and grilled vegetables." Ingredients include 90 grams of beef, 80 grams of bread, 120 grams of vegetables, 1 teaspoon of oil, and 1 cup of strawberries.

$$\begin{aligned}
\text{should_eat}(R) &\leftarrow \text{has}(R, \text{beef}) \wedge \text{has}(R, \text{bread}) \\
\text{should_eat}(R) &\leftarrow \text{has}(R, \text{strawberry}) \\
\text{should_eat}(R) &\leftarrow \text{has}(R, X) \wedge \text{has}(R, Y) \\
&\wedge \ \text{vegetable}(X) \wedge \text{essential_oil}(Y)
\end{aligned}$$

p.3 Second day, lunch: "Tuna salad." Ingredients include 120 grams of tuna,

120 grams of vegetables, 1 teaspoon of olive oil, 80 grams of bread, 35 grams of raw beans, and 1 cup of blueberries.

$$\begin{aligned}
\text{should_eat}(R) &\leftarrow \text{has}(R, \text{bread}) \wedge \text{has}(R, \text{beans}) \\
\text{should_eat}(R) &\leftarrow \text{has}(R, \text{tuna}) \\
&\wedge \text{has}(R, X) \wedge \text{has}(R, Y) \\
&\wedge \text{vegetable}(X) \wedge \text{herb}(X) \wedge \text{essential_oil}(Y) \\
\text{should_eat}(R) &\leftarrow \text{has}(R, \text{blueberry})
\end{aligned}$$

p.4 Second day, dinner: "Chicken with mustard and lemon juice." Ingredients include 90 grams of chicken, 120 grams of vegetables, 80 grams of raw pasta, 1 teaspoon of olive oil, mustard, lemon juice, and 1 cup of clementines.

$$\begin{aligned}
\text{should_eat}(R) &\leftarrow \text{has}(R, \text{chicken}) \wedge \text{has}(R, \text{mustard}) \wedge \text{has}(R, \text{lemon_juice}) \\
\text{should_eat}(R) &\leftarrow \text{has}(R, \text{pasta}) \\
&\wedge \text{has}(R, X) \wedge \text{essential_oil}(X) \\
\text{should_eat}(R) &\leftarrow \text{has}(R, X) \wedge \text{vegetable}(X) \\
\text{should_eat}(R) &\leftarrow \text{has}(R, \text{citrus_fruits})
\end{aligned}$$

p.5 Third day, lunch: "Salmon with potatoes." Ingredients include 120 grams of salmon, 240 grams of cooked potatoes, 120 grams of vegetables, 1 teaspoon of butter, and 1 pear.

$$\begin{aligned}
\text{should_eat}(R) &\leftarrow \text{has}(R, \text{compound_salmon}) \wedge \text{has}(R, \text{potato}) \\
\text{should_eat}(R) &\leftarrow \text{has}(R, \text{pear}) \\
\text{should_eat}(R) &\leftarrow \text{has}(R, \text{butter}) \\
&\wedge \text{has}(R, X) \wedge \text{vegetable}(X)
\end{aligned}$$

p.6 Third day, dinner: "Turkey in papillote." Ingredients include 90 grams of turkey, 1 teaspoon of olive oil, 120 grams of vegetables, 35 grams of raw gram

beans, 80 grams of raw wholegrain rice, and 1 orange.

$$
\begin{aligned}
\text{should\_eat}(R) &\leftarrow \text{has}(R, \text{gram\_bean}, \\
\text{should\_eat}(R) &\leftarrow \text{has}(R, \text{rice}) \\
\text{should\_eat}(R) &\leftarrow \text{has}(R, \text{orange}) \\
\text{should\_eat}(R) &\leftarrow \text{has}(R, \text{turkey}) \\
&\wedge \text{has}(R, X) \wedge \text{has}(R, Y) \\
&\wedge \text{vegetable}(X) \wedge \text{essential\_oil}(Y)
\end{aligned}
$$

For each user and prescription, recipes are computed based on the logic rules. Precision is calculated as the ratio of recipes liked by users to the total number of proposed recipes. Corner cases, where no recipes are recommended due to conflicting preferences and prescriptions, are resolved by adjusting prescriptions to better align with user preferences.

7.1.4 Results and discussion

To ensure realistic experiments, we adopted criteria to generate synthetic datasets representing user preferences. We avoided synthesizing users with trivial rules, such as always liking a specific ingredient, which would result in predictable recommendations. To address this, we introduced noise into the dataset synthesis process, assigning preference values within distributions and stochastically labeling classes. This approach discourages oversimplified logic rules and mimics real-life scenarios where diverse ingredient combinations influence user preferences in complex ways. Table 7.1 reports the accuracy of the neural networks trained to predict user preferences, alongside the accuracy and fidelity of the extracted logic rules. The accuracy of individual networks ranges from 0.74 to 0.97, with a mean value of approximately 0.86. This variability reflects differences in user profiles, as some preferences are inherently more predictable than others. The accuracy of the extracted rules ranges from 0.68 to 0.86, with a mean value of 0.76. The mean difference of 0.095 between network accuracy and rule accuracy is expected, given the constraints imposed on decision tree depth during the extraction process. Similar observations apply to precision measures. It is important to note that extracted rules approximate the decision-making process of the neural networks and cannot

outperform the original models.

Table 7.3 summarizes the precision obtained during the recommendation phase using prescriptions and extracted rules. The mean precision value across all experiments is approximately 0.74, which is close to the average precision of the rules reported in Table 7.1. Applying the Student's t-test to the precision values in Table 7.1 ("r. prec." column) and Table 7.2 ("average" column) yields a p-value of 0.386. This indicates no statistical difference between the two distributions. In other words, recommending recipes liked by users from the entire dataset is as effective as recommending recipes prescribed by human experts.

Table 7.3 compares the precision obtained using prescriptions and sub-symbolic predictors. Similar statistical analysis yields a p-value of 0.403, leading to the same conclusion: the recommendation process is equally effective for prescribed liked recipes and liked recipes predicted by neural networks.

In summary, experiments demonstrate that sub-symbolic predictors outperform symbolic predictors in terms of overall precision for personalized food recommendations. However, the primary goal of the framework is not to achieve higher performance compared to sub-symbolic predictors or existing systems. Instead, the focus is on enhancing explainability, enabling users and experts to understand why certain recipes are recommended or rejected. Extracted rules provide insights that allow experts to adjust prescriptions to better align with user preferences. Despite lower performance compared to neural networks, the rules remain acceptable in real-world scenarios. For instance, consider the recipe "Shakkara (Sweet) Pongal" (recipe ID $4,055$), which is liked by User 1. This recipe contains ingredients such as basmati rice, butter, camphor, cardamom, cashew nuts, lentils, milk, raisins, and sugar. The recommendation is justified by the presence of milk and sugar, as indicated by the extracted rules for User 1. Conversely, the recipe "Lasagna Spinach Roll-Ups" (recipe ID $10,815$) is disliked due to the presence of eggs and pepper. The symbolic approach adds value by providing explainability, allowing motivations for recommendations to be derived systematically.

7.2 A general-purpose protocol for multi-agent based explanations

In section we present the work "A General-Purpose Protocol for Multi-agent Based Explanations" [Cia+23], presented at the EXTRAAMAS international workshop, 5th edition, 2023. This contribution is quite different from the previous one, and in general from the other contributions in this thesis. While the majority of the works presented in this thesis vertically focus on SKI or SKE methods, or on the design of systems that leverage such methods, this work is more horizontal, in the sense that it proposes a general-purpose protocol for multi-agent based explanations. Nonetheless, this work still falls within the scope of this chapter because its contribution is relevant to the design of explainable MASs that can potentially leverage SKE methods.

7.2.1 Motivation

The current focus of XAI research is on developing techniques to "open up" black-box models and provide insights into how an intelligent system reaches specific decisions or predictions [Gui+19]. These techniques include methods for visualizing the internal workings of the system, such as feature importance scores, attention maps, and decision trees. The primary goal of these methods is to assist AI experts in understanding the system's behavior, rather than addressing the needs of non-expert users who seek to understand *why* the system behaves in a particular way.

However, the expectations of the XAI community extend beyond merely interpreting black-box models. Ideally, XAI systems should autonomously provide explanations that go beyond describing how the system works [Cia+19]. These explanations should offer insights into *why* the system behaves – or fails to behave – in a specific manner, potentially through autonomous interaction with the explainee.

To achieve this, recent research emphasizes the automation and interactivity of the explanation process [Cia+20a]. This involves designing AI systems capable of generating explanations dynamically and tailoring them to the explainee's knowl-

edge level and needs. In this context, MASs emerge as a suitable paradigm for building intelligent explainable systems, as they inherently support interaction and autonomy. Explanations can thus be modeled as multi-agent interactions, where the explainee and the explainer agent (either human or software) collaborate to achieve the shared goal of providing clear and effective explanations.

This work addresses the general problem of enabling interaction between explainee and explainer agents. To this end, it proposes a general-purpose protocol for multi-agent-based recommendations and explanations. The protocol defines the roles and responsibilities of the explainee and explainer agents, as well as the types of information exchanged to ensure clarity and effectiveness. Notably, the protocol builds upon prior efforts to model explanations as multi-agent interactions [Buz+22]. Its key features include: *(i)* the separation of recommendations from explanations, and *(ii)* support for contrastive explanations.

As a secondary contribution, the paper introduces a `Spade`-based Python library, `PyXMas`, which implements the proposed protocol. This library allows the integration of various explanation strategies and representations. It serves as a foundation for developing intelligent explainable systems where recommendation and explanation behaviors are delegated to individual agents. Overall, this contribution represents a significant step toward building XAI systems capable of providing automatic and interactive explanations.

7.2.2 Background

Interactive recommendation systems Interactive RSs have gained significant attention due to their ability to dynamically provide personalised recommendations based on user feedback and interactions [KWH10]. The key aspect of interactivity lies in collecting user feedback during the recommendation session to refine subsequent recommendations. For instance, some systems employ a one-shot recommendation approach, where questions learned offline from past interactions are posed to users before generating recommendations [CRH16]. The answers to these questions enable the system to personalise and improve future suggestions.

XAI has been increasingly integrated into RSs to enhance transparency [Buz+22; ZC20]. This is achieved through iterative recommendation

sessions, where the system not only provides recommendations but also explanations, leveraging the positive impact of transparency on user trust [ODo+08]. For example, visual explanations, such as grouped bar charts, have been used to compare user preferences with recommendation attributes [Mil+19]. Similarly, conversational explanations have been employed to mimic human salesmanship, persuading users to consider alternative options [Shi02].

Prior work on explanation protocols This work builds upon the protocol introduced in [Buz+22], which focuses on food recommendations and explanations. In this protocol, users specify constraints, such as allergies, preferred or disliked ingredients, and desired cuisine types. The system responds with a recipe suggestion and an explanation. Users can accept, reject, or provide feedback on the recommendation or explanation, initiating a turn-based interaction until the session concludes. This framework promotes transparency by combining recommendations and explanations, which can increase user acceptance. However, unrequested explanations may impose cognitive load, highlighting the importance of parsimony [Mua+22]. Parsimonious explanations are defined as the simplest descriptions that adequately convey the situation.

To address this, the revised protocol proposed in this work allows users to request explanations dynamically. It also supports "zooming" explanations, where additional details are provided only upon user request. This approach enables users to control the level of detail, ensuring explanations are tailored to their needs.

Spade: multi-agent programming in python SPADE is an open-source MAS platform implemented in Python[3]. It provides a modular and extensible library for developing intelligent agents capable of interacting with each other and their environment. SPADE systems are distributed, with agents operating on the same or different network nodes. Agent activities are governed by concurrent behaviours, implemented as Python classes that developers can extend.

Unlike JADE [BCG07], which is Java-based, SPADE leverages Python's ecosystem, facilitating integration with ML and AI frameworks. Agent communication in SPADE is mediated by the extensible messaging and presence protocol (XMPP)

[3] `https://spade-mas.readthedocs.io`

protocol, ensuring robust, interoperable, and scalable interactions. This design supports blended applications where agents interact with both humans and software agents.

SPADE includes features such as communication protocols, message passing, and event handling. It also supports the implementation of interaction protocols using finite-state machine behaviours. These capabilities make SPADE a powerful tool for developing intelligent agents, widely adopted in MAS research and development.

7.2.3 Explanation-based recommendation protocol

The term "explanation" originates from the Latin word *explicare*, meaning "to unfold." In this context, an explanation is understood as the process of clarifying the meaning of a concept. This process is inherently interactive, involving an explainee and an explainer. Explanations are thus considered a social protocol.

In human interactions, explanation protocols are typically informal and unstructured. They involve an explainee seeking clarification from an explainer, who is assumed to possess greater knowledge. Explainers adapt their strategies and level of detail to the explainee's needs and knowledge as the interaction progresses. Explanations are generally provided upon request and may respond to prior information shared by either party.

Modern intelligent systems, such as XAI systems, aim to support decision-making by providing recommendations. In these systems, the user typically acts as the explainee, while the software system assumes the roles of both the recommender and the explainer. By adopting a MAS perspective, recommendation and explanation can be modeled as a single interaction protocol between two agents. One agent, often the explainee, is a human user, while the other, the explainer, is a software agent.

The proposed protocol assumes that the user initiates the interaction by submitting a query. Upon receiving the query, the agent generates a recommendation. This recommendation is computed using available information, such as the user's profile, interaction history, and possibly aggregated data from other users. The agent may utilize both symbolic reasoning and ML predictors to generate the

recommendation.

After receiving the recommendation, the user may either accept or reject it, or request an explanation. The explanation phase may involve multiple rounds of interaction, where the user can ask for additional details or comparisons. The agent provides the requested information, aiming to clarify the recommendation. Ultimately, the user decides to accept or reject the recommendation based on the explanation provided.

Feedback from the user, including the acceptance or rejection of recommendations and the amount of explanatory information required, is used to improve future interactions. In cases of rejection, the agent may also seek the reason for the rejection to refine its recommendation and explanation strategies.

Explanations in this protocol are always: *(i)* provided upon request, *(ii)* related to the recommendation, and *(iii)* tailored to the user. They can be categorized into two types:

- **Ordinary explanations:** These address the question, "Why did you recommend this?"
- **Contrastive explanations:** These address the question, "Why did you not recommend that instead?"

The protocol supports both types of explanations, allowing the user to choose the type they prefer. The content and structure of the exchanged messages depend on the type of explanation requested.

7.2.4 Abstract formulation of the protocol

Figure 7.2 presents an abstract formulation of the proposed protocol, which is independent of the specific representation or computation of recommendations and explanations. The focus is on the exchange of messages between the explainer and the explainee, the information these messages carry, and the sequence in which they are exchanged. The protocol defines two roles:

- The explainee, who initiates the interaction.
- The explainer, who responds to the explainee's queries.

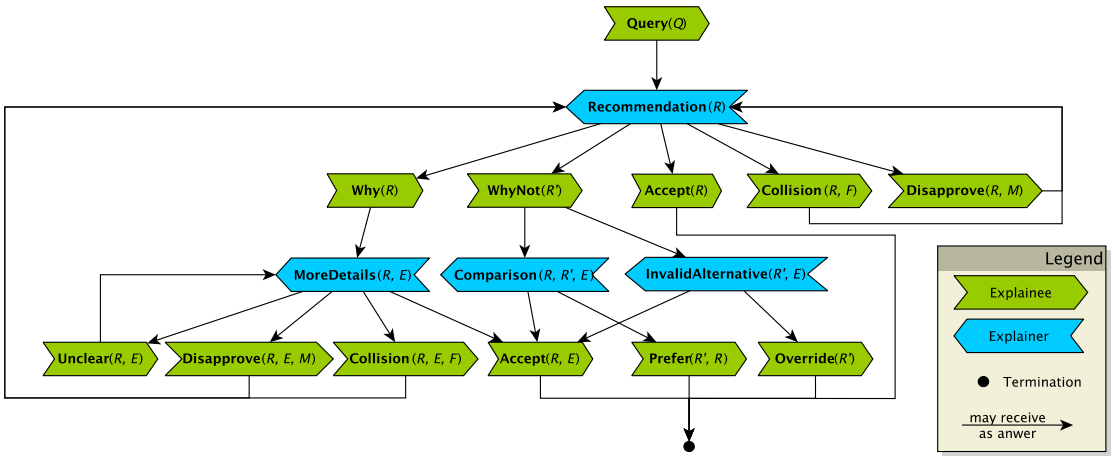


Figure 7.2: Message communication diagram between an explainer agent (blue boxes) and an explainee (green boxes). Each box represents a message. Arrows indicate the possible replies to each message.

Five data types are identified as the potential payloads exchanged during the protocol:

- **Queries** (Q): Requests for recommendations submitted by the explainee.
- **Recommendations** (R, R'): Responses to queries, provided by the explainer.
- **Explanations** (E, E'): Information provided by the explainer to clarify recommendations.
- **Features** (F): User-specific aspects that justify the rejection of a recommendation, which the explainer should consider in future interactions.
- **Motivations** (M): Reasons for rejecting a recommendation, which may influence the explainer's behavior.

Thirteen message types are defined, each represented as a named record of the form `Name(Payload)`, where `Payload` consists of instances of the aforementioned data types. Optional fields in the payload are denoted with a question mark. The message types are summarized as follows:

1. `Query(`Q`)`: Sent by the explainee to initiate the protocol, carrying a recommendation request.
2. `Recommendation(`Q, R`)`: Sent by the explainer in response to a query, carrying the query and the computed recommendation.

3. `Why`(Q, R): Sent by the explainee to request an explanation for a recommendation.
4. `WhyNot`(Q, R, R'): Sent by the explainee to request a contrastive explanation, comparing the recommendation with an alternative.
5. `Accept`$(Q, R, E?)$: Sent by the explainee to accept a recommendation, optionally including the explanation.
6. `Collision`$(Q, R, F, E?)$: Sent by the explainee to notify the explainer of a conflict between the recommendation and a personal feature.
7. `Disapprove`$(Q, R, M, E?)$: Sent by the explainee to reject a recommendation, providing a reason and optionally the explanation.
8. `Details`(Q, R, E): Sent by the explainer to provide additional details about a recommendation.
9. `Comparison`(Q, R, R', E): Sent by the explainer to provide a contrastive explanation when the alternative recommendation is also valid.
10. `Invalid`(Q, R', E): Sent by the explainer to indicate that the alternative recommendation is invalid, with an explanation.
11. `Unclear`(Q, R, E): Sent by the explainee to indicate that the provided explanation is unclear.
12. `Prefer`(Q, R, R'): Sent by the explainee to express a preference for an alternative recommendation.
13. `Override`(Q, R, R'): Sent by the explainee to enforce an alternative recommendation, even if deemed invalid by the explainer.

This abstract formulation ensures flexibility, allowing implementers to tailor the protocol to specific application domains.

Notably, messages are designed by keeping the representational state transfer (ReST) [9] architectural style into account. Hence, each message type is designed to carry all the information necessary for any involved party to decide which action to take next. This is the reason why all/most messages carry the original query Q and the recommendation R (or R') which they are referring to. The message communication diagram from Figure 7.2 depicts not only the messages exchanged by the explainee and explainer, but also the admissbile request-response patterns which the protocol allows. There, a more detailed view of the message flow is

provided, which we briefly summarise in the following.

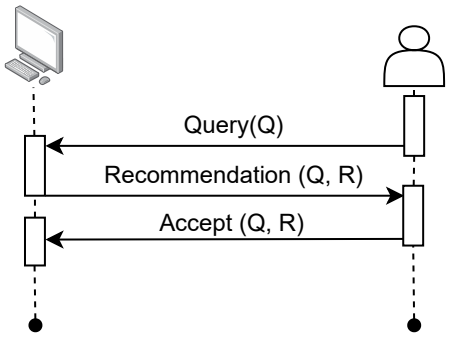
7.2.4.1 Relevant scenarios and protocol analysis

The proposed protocol is versatile and accommodates various user needs and desires. These include: *(i)* requesting a recommendation, *(ii)* seeking an explanation for the recommendation, *(iii)* asking for additional details about the explanation, *(iv)* simulating alternative recommendations, and *(v)* providing feedback on recommendations or explanations. Figure 7.3 illustrates these scenarios, which are detailed below.

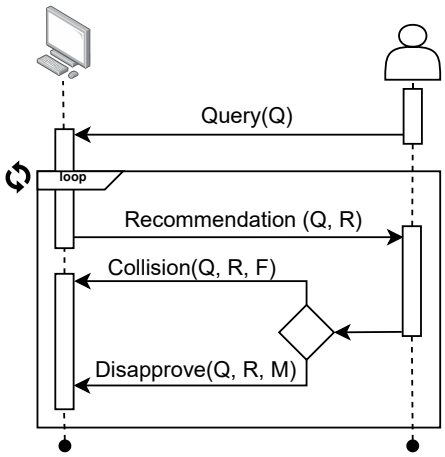
Quick accept In this scenario, depicted in Figure 7.3a, the user accepts the recommendation without requiring an explanation. For example, a user requests a restaurant recommendation, and the agent suggests a restaurant that the user finds satisfactory.

Quick retry In Figure 7.3b the user rejects the recommendation without asking for an explanation. The rejection may occur because the recommendation conflicts with the user's preferences or is unsuitable for the current context. For instance, if the agent recommends a steakhouse to a vegetarian user, the user may signal a conflict with their preferences. Alternatively, the user may simply disapprove of the recommendation without providing specific feedback. In both cases, the agent generates a new recommendation. The agent is expected to learn from conflicts but not from generic disapprovals.

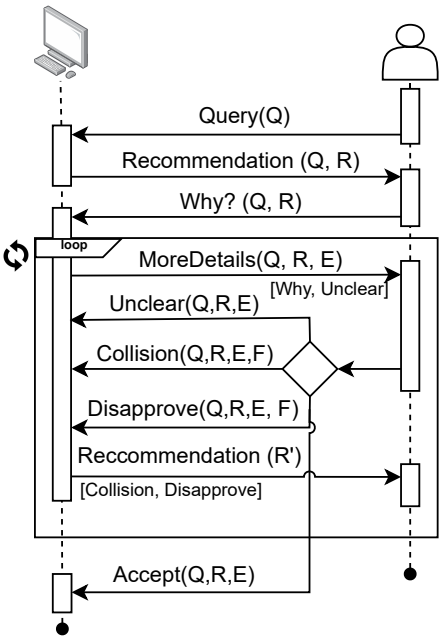
Ordinary explanation loop In the case shown in Figure 7.3c, the user requests an explanation for the recommendation. If the explanation is unsatisfactory, the user may ask for further details, initiating an iterative process. This loop continues until the user either accepts the recommendation or requests a new one. The protocol supports "zooming" explanations, where the agent adjusts the granularity of the explanation. For example, the agent may first provide local explanations, describing how the specific recommendation was generated. Subsequently, it may offer global explanations, detailing the general logic behind its recommendations. The agent can also switch between textual and visual explanations to enhance



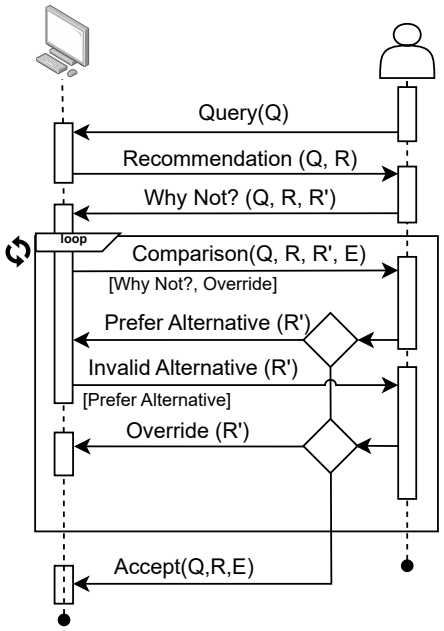
(a) Quick accept: the user accepts the recommendation without asking for explanations.



(b) Quick retry: the user rejects the recommendation without asking for explanations. Another recommendation is proposed, accordingly.



(c) Ordinary explanation loop: the user asks 'why' after a recommendation, and then agent answers with further details. The request for details may be repeated several times.



(d) Contrastive explanation loop: the user asks 'why not' another recommendation. The agent may then explain why the other recommendation is acceptable or invalid. The user may either accept the original recommendation or prefer their own.

Figure 7.3: Sequence diagrams describing most common scenarios of the protocol.

clarity. Consider a user who requests a restaurant recommendation. The agent suggests an Asian restaurant with a high rating and proximity to the user. If the user asks for an explanation, the agent may state that the restaurant matches the user's taste for sushi and is within 1 km. If further details are requested, the agent may explain its general recommendation strategy, such as prioritizing highly rated restaurants within a certain distance.

Contrastive explanation loop Lastly, in Figure 7.3d, the user requests a contrastive explanation, comparing the given recommendation R with an alternative R'. If R' is valid, the agent provides a comparison, highlighting why one recommendation is preferable. If R' is invalid, the agent explains why it cannot be recommended. The user may then accept the original recommendation, prefer the alternative, or override the agent's decision. For example, if the agent recommends an Asian restaurant, but the user prefers a steakhouse, the agent may compare the two options. If the steakhouse aligns with the user's dietary goals, the agent may note that the Asian restaurant is closer. If the steakhouse violates dietary goals, the agent explains this conflict. In either case, the user decides whether to accept the original recommendation or override it. The agent learns from overrides to refine future recommendations.

7.2.4.2 Which sorts of explanations and recommendations?

The proposed explanation protocol is agnostic regarding the specific representation of explanations and recommendations. It is the responsibility of the implementer to define how explanations and recommendations are represented and computed. The protocol only specifies *when* these elements should be computed.

RSs typically rely on one or more ML predictors trained on user data. Whether the training of these predictors is performed by the recommender agent or if the agent is equipped with pre-trained predictors at deployment is an implementation detail. In either case, the recommender agent must have access to user profile information. This information can be obtained during an initial configuration phase or inferred from user interactions, such as accepted or rejected recommendations. To support dynamic learning, the agent should include a learning algorithm capable of updating the predictors when new user data becomes available. From this

perspective, the explainer agent acts as a proxy for the ML predictor(s).

Explanations, however, are not necessarily derived from ML predictors. The XAI literature offers a wide range of approaches for generating explanations, including visual, textual, and numerical methods [Arr+20; Gui+19; Cia+24]. The explainer agent must not only wrap the ML predictor(s) but also encapsulate the logic for computing and representing explanations.

A critical challenge arises when recommendations and explanations use different representation formats, such as textual and visual. In such cases, the explainer agent must bridge the gap by providing a unified representation of both the recommendation and its explanation. To address this, designers may consider adopting computational logic as a unifying framework for recommendations and explanations. In computational logic, both knowledge bases and queries are represented as logic formulas. These formulas can be used to represent recommendation requests, solutions, and explanations.

For example, a recommendation query can be expressed as the logic goal:

$$\texttt{should\_eat(Food, lunch)},$$

where `Food` is a logic variable representing an unknown value. Recommendations, such as R, R', R'', correspond to logic solutions, e.g., $\texttt{Food} = \texttt{paella}$. Explanations E, E', E'' can take various forms:

- **Local explanations:** The path in the proof tree computed by the explainer agent to derive the recommendation.
- **Global explanations:** The logic program used by the explainer agent to generate the recommendation.
- **Contrastive explanations:** Metrics comparing multiple recommendations or identifying constraints that make certain recommendations invalid.
- **Combinations:** Any combination of the above types.

User features, such as F, F', F'', may include raw facts describing the user, e.g., `age(31)`, `goal(lose_weight)`, or `category(vegetarian)`. Motivations for disapproval can include predefined facts, such as:

- `dislike`: The user dislikes the recommendation, prompting the agent to

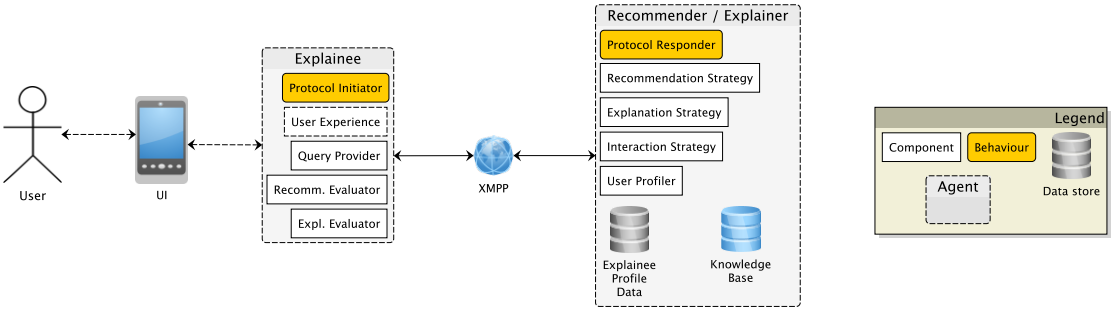


Figure 7.4: Modular architecture of PyXMas. The library provides parametric behaviours that can be customized by implementing specific components. If the explainee agent is human, a UX component may be required to facilitate interaction via a device.

learn from this feedback.

- `not_now`: The user does not want the recommendation at the moment, but may accept it in the future. In this case, the agent should not memorize the rejection.

7.2.5 From theory to practice with PyXMas

This section describes how the proposed protocol can be implemented as agent-oriented software. We introduce PyXMas[4], a Python library for explainable multi-agent systems (MASs). The library is built on Spade and provides reusable implementations of the protocol described in Section 7.2.4. This allows researchers and developers to focus on designing recommender and explainer agents, as well as defining the representation of recommendations and explanations, without re-implementing the protocol.

7.2.5.1 PyXMas Architecture

Figure 7.4 illustrates the modular architecture of PyXMas. The library defines two main behaviours: *(i)* the *initiator*, responsible for sending recommendation queries and processing responses, and *(ii)* the *responder*, responsible for computing and returning recommendations and explanations. The initiator behaviour is designed for the explainee agent, while the responder behaviour is intended for

[4] `https://github.com/pikalab-unibo/pyxmas`

the explainer agent. Both behaviours are parametric, allowing users to customize their functionality by implementing specific components.

Explainer agent The explainer agent requires the following components:

- **Recommendation strategy:** Computes recommendations based on user queries, preferences, and goals (e.g., "vegetarian" or "weight loss"). The strategy can adapt over time by learning from user feedback.
- **Explanation strategy:** Generates explanations for recommendations, leveraging user profiles and domain knowledge (e.g., "ingredient X is plant-based").
- **User profiler:** Learns user preferences from feedback using heuristic or ML-based methods. This enables the agent to refine recommendations and explanations dynamically.
- **Interaction strategy:** Manages how recommendations and explanations are presented to the explainee. For example, a humanoid robot may use gestures or facial expressions to enhance interaction.

The explainer agent stores two types of data: *(i)* user profile data, and *(ii)* domain knowledge. These are maintained in dedicated data stores and updated as needed.

Explainee agent The explainee agent requires the following components:

- **Query provider:** Generates queries based on the explainee's goals.
- **Recommendation evaluator:** Assesses recommendations and decides whether to accept or reject them.
- **Explanation evaluator:** Evaluates explanations and influences the recommendation evaluator accordingly.

If the explainee is a human user, the agent acts as a proxy, mediating interactions via a user interface (UI) on a device (e.g., a smartphone). In this case, a **user experience (UX)** component is required to manage the UI, process user inputs, and present recommendations and explanations.

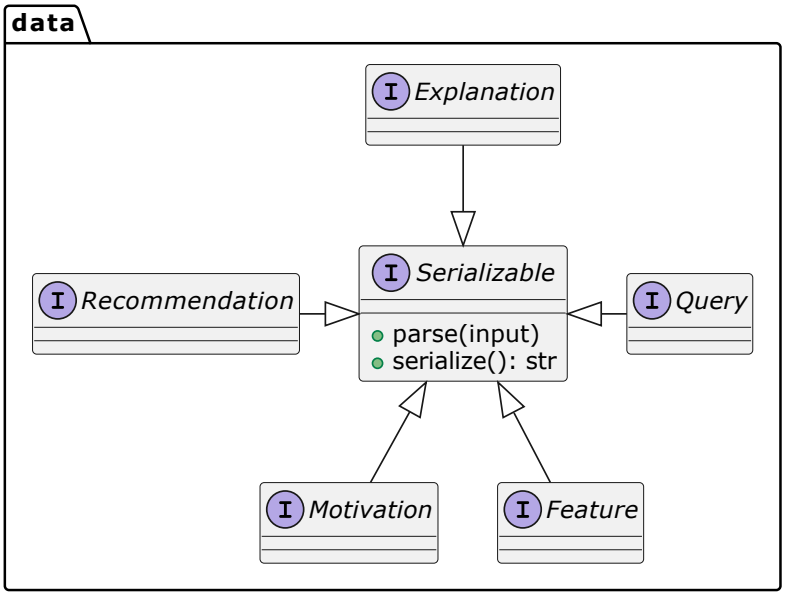


Figure 7.5: Abstract classes for message payloads in PyXMas.

7.2.5.2 PyXMas design

PyXMas is a Python library designed for developing explainable MASs. It provides:

- Abstract classes for the (de)serialization of message payloads exchanged between the explainer and explainee agents (see Figure 7.5).
- Abstract classes defining the initiator and responder behaviors.

These abstract classes allow developers to extend and customize their functionality by overriding specific methods. This flexibility enables the creation of tailored explainable MASs.

Data types for message payloads As illustrated in Figure 7.5, PyXMas provides five abstract classes corresponding to the data types defined in Section 7.2.4. These classes enforce serialization, ensuring that data can be converted to and from strings. This is essential for enabling communication between the explainer and explainee agents over the network. Developers can define custom representations for queries, recommendations, explanations, and other data types by extending

these abstract classes. The only requirement is that the serialized data must be both machine- and human-readable.

Predefined behaviors PyXMas provides two abstract classes for the protocol roles:

- The *initiator*, which represents the explainee agent.
- The *responder*, which represents the explainer agent.

These behaviors are implemented as finite-state machines (FSMs) using Spade. Each state corresponds to a specific action, such as waiting for a message or processing a response. Figure 7.6 illustrates the state diagrams for the initiator and responder behaviors. On the initiator side, developers can override callbacks to control:

- Query generation.
- Evaluation of recommendations and decisions to accept or reject them.
- Evaluation of explanations and their influence on recommendation acceptance.

On the responder side, developers can override callbacks to control:

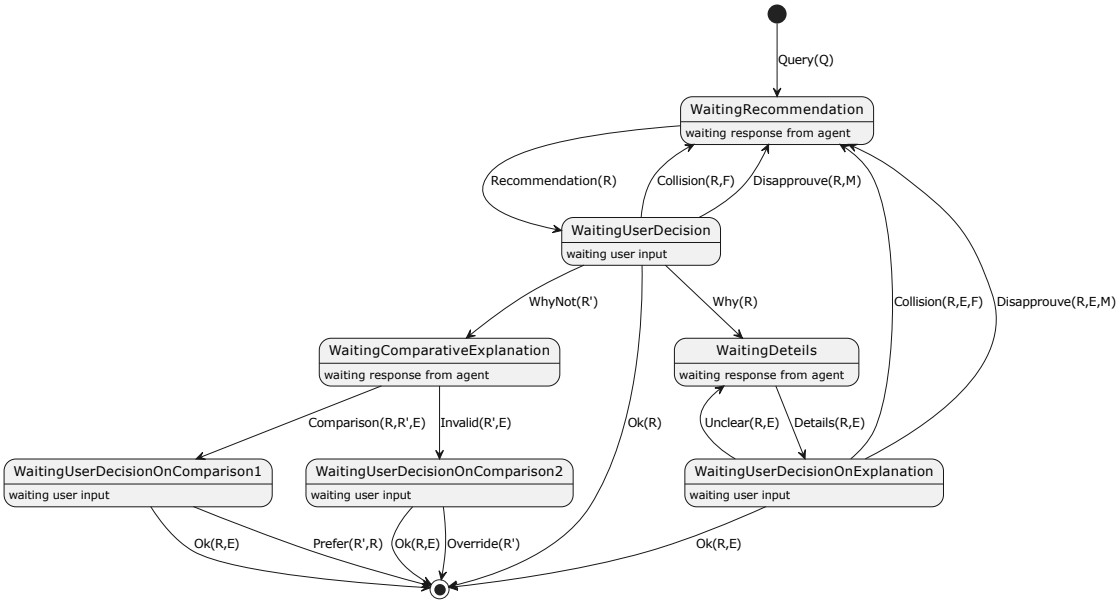
- Recommendation generation.
- Explanation generation.
- Handling of accepted or rejected recommendations.

This design ensures that PyXMas can be adapted to various application domains while maintaining a clear and modular structure.

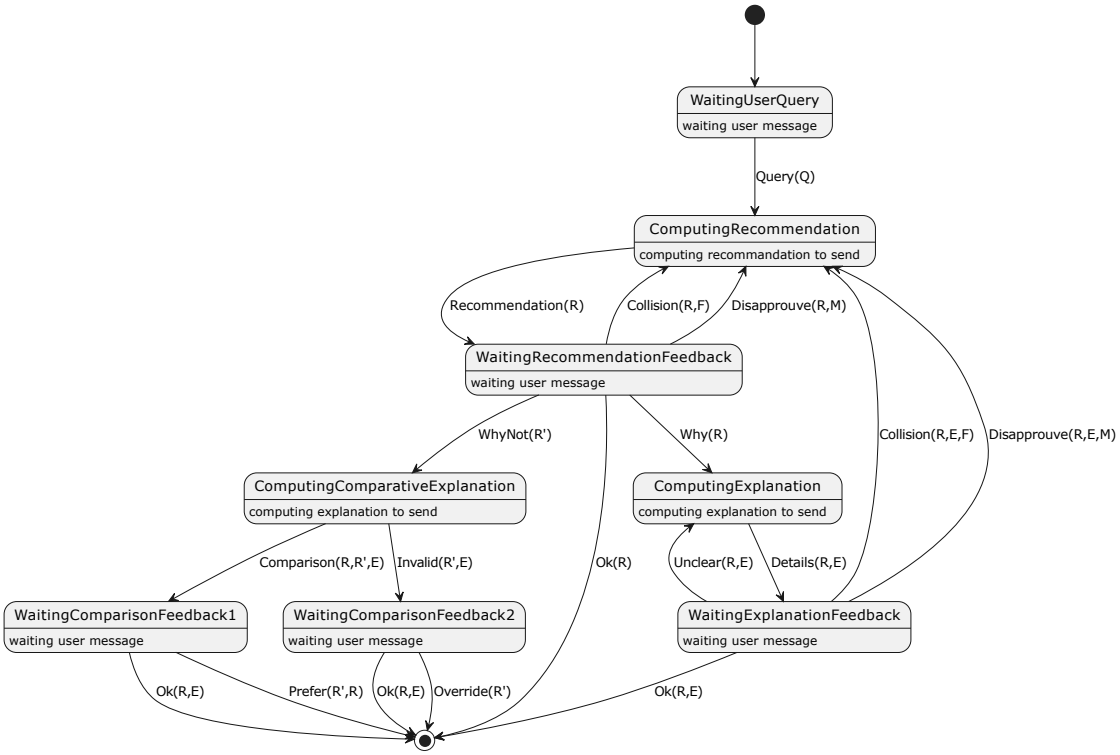
7.3 NeSy AI for supporting Chronic disease diagnosis and monitoring

7.3.1 Motivations and background

Telemedicine platforms are increasingly used for personal assistance, enabling patients to share their symptoms, medical history, and diagnostic data, such as medical images, lab results, or vital signs. These platforms often leverage AI data-driven



(a) Initiator-side state diagram.



(b) Responder-side state diagram.

Figure 7.6: State diagrams describing the initiator and responder behaviors in PyX-Mas.

models [HH23; Raj+22]. While promising, accurate disease monitoring and diagnosis require more than identifying patterns in data. The integration of AI into clinical practice remains limited due to several challenges. First, the performance of AI models is often suboptimal, primarily due to insufficient dataset sizes for effectively training ML algorithms [BDM20]. Second, even high-performing AI models frequently operate as black boxes, making their outputs difficult to interpret. This lack of transparency hinders adoption in clinical settings, where healthcare professionals require interpretable outputs to make informed decisions [Ali+23]. Traditional clinical protocols, while validated and robust, also exhibit limitations. They often fail to classify all clinical cases, leaving certain scenarios unresolved. Additionally, these protocols do not leverage data analytics, which could uncover hidden patterns and provide valuable insights for decision-making.

Combining data-driven and knowledge-based models has been identified as a promising solution to enhance ML performance and improve explainability [Rüd+23]. This work focuses on NeSy AI, specifically SKI [Cia+24]. Among the various SKI methods, we evaluate the LTN approach [Bad+22]. The potential of this approach is demonstrated using the Pima Indians diabetes (PID) dataset for diabetes prediction [Smi+88]. We compare its performance against classical ML models, including k-nearest neighbor (KNN), logistic regression (LR), DT, RF, and an uneducated MLP. Performance is evaluated based on criteria such as classification accuracy and robustness.

Results indicate that classical ML models struggle to achieve consistent performance. In contrast, injecting domain knowledge into neural networks via the LTN method significantly improves clinically relevant metrics, such as recall, balanced accuracy, F1 score, and Matthews Correlation Coefficient [Mat75]. Moreover, LTN enhances adherence to clinical knowledge, a critical requirement in the medical domain. It also demonstrates high robustness against data manipulation, including noise injection, data dropping, and label flipping.

7.3.2 Knowledge integration in medicine

The integration of medical knowledge into ML pipelines can occur at various stages [Rüd+23; KKK23].

- **Data pre-processing:** Errors in datasets can be mitigated by removing anomalous samples based on clinical norms. Virtual samples adhering to medical knowledge can be generated to address insufficient or missing data.
- **Feature engineering:** Novel features can be derived using mathematical or logical models informed by medical knowledge. Feature selection can also be guided by prior knowledge.
- **Model learning:** Rules can be incorporated into the model's loss function or architecture.
- **Output evaluation:** ML outputs can be integrated with clinical guidelines to filter predictions or verify consistency with domain knowledge.

While these strategies are promising, they often do not reference the SKI framework or its complementary SKE paradigm [Cia+24]. These paradigms have proven effective in other domains and hold potential for healthcare applications. Preliminary experiments [SM24] suggest that further evaluation of SKI methods, such as LTN, is warranted.

7.3.3 Materials and methods

In this study, we utilize the PID dataset [5], a widely recognized benchmark in medical-related ML research [KMJ11; Cha+23]. We focus on the LTN approach [Bad+22], a SKI method that combines structuring and constraining injection strategies. This section details the dataset, the LTN methodology, and the metrics used to evaluate the impact of SKI on predictive performance and QoS aspects.

7.3.3.1 Dataset and knowledge

The PID dataset consists of 768 medical profiles of women aged 21 and above who underwent an oral glucose tolerance test. The dataset includes features such as glucose and insulin concentrations, blood pressure (BP), skin thickness (ST), body mass index (BMI), diabetes pedigree function, number of pregnancies, and age.

[5] `https://www.kaggle.com/datasets/uciml/pima-indians-diabetes-database`

The target variable is binary, indicating whether diabetes was diagnosed within five years.

Missing values are present in several attributes, including insulin (48.70%), ST (29.56%), BP (4.55%), BMI (1.43%), and glucose (0.65%). These missing values were imputed using the median value of the respective attribute. The dataset is imbalanced, with 65.1% of instances labeled as negative and 34.9% as positive.

Public health guidelines on type-2 diabetes risks indicate that individuals with a high BMI (≥ 30) and high blood glucose levels (≥ 126) are at severe risk for diabetes. Conversely, individuals with a normal BMI (≤ 25) and low blood glucose levels (≤ 100) are less likely to develop diabetes. These guidelines are encoded into logic formulas as follows:

$$\forall x.\,(\text{glucose}(x) \geq 125 \wedge \text{bmi}(x) \geq 30 \rightarrow \text{Diabetic}(x)), \tag{7.4}$$

$$\forall x.\,(\text{glucose}(x) \leq 100 \wedge \text{bmi}(x) \leq 25 \rightarrow \neg\text{Diabetic}(x)). \tag{7.5}$$

7.3.3.2 Logic tensor network for SKI

The LTN framework [Bad+22] extends FOL with fuzzy semantics, where formulas are interpreted as continuous truth values in the interval $[0, 1]$. This allows the definition of logic rules that are not strictly true or false but partially true, with predicates computed by neural networks.

In this study, we use LTN to inject the knowledge encoded in Equations (7.4) and (7.5). The predicate Diabetic(x) is modeled as a neural predicate backed by a feed-forward NN with three hidden layers. The NN outputs a real number in $[0, 1]$, representing the truth value of Diabetic(x) for an input $x \in \mathbb{R}^8$.

The training process maximizes the truth value of the logic formulas while considering the available data. This is achieved by minimizing a loss function that penalizes violations of the injected knowledge. The predicates glucose(x) and bmi(x) are grounded to the actual values in the dataset, while the following dummy formulas are added to supervise the training:

$$\forall x.\,(\text{outcome}(x) = \text{diabetes} \rightarrow \text{Diabetic}(x)), \tag{7.6}$$

$$\forall x.\,(\text{outcome}(x) = \text{healthy} \rightarrow \neg\text{Diabetic}(x)). \tag{7.7}$$

7.3.3.3 Evaluation Metrics

To assess the impact of SKI, we evaluate both predictive performance and QoS metrics.

Performance metrics Predictive performance is evaluated using standard metrics for binary classification, including accuracy, precision, recall, F1-score, balanced accuracy (BA), and the Matthews correlation coefficient (MCC).

Logic adherence Inspired by fidelity metrics in the SKE framework [Cia+24], we propose a metric to assess adherence to clinical rules. For each rule, we create a test set of instances satisfying the rule, assign labels based on the rule, and compute the model's accuracy on this subset.

Quality of service QoS metrics evaluate non-functional aspects such as robustness and complexity. This study focuses on robustness to data degradation, which occurs when input data is noisy, incomplete, or biased. Robustness is quantified using a reproducible procedure where training data is perturbed, and the model's performance is evaluated on the perturbed data [Agi+23]. The robustness score $\rho_{N,D}(D)$ of a predictor N trained on D is defined in Equation (5.10). The robustness gain $R_{N,D}(I)$ of an injection mechanism I is computed as shown in Equation (5.11). A value $R_{N,D}(I) > 1$ indicates improved robustness, while $R_{N,D}(I) < 1$ indicates reduced robustness.

7.3.3.4 Experimental design

The experiments are designed to evaluate the impact of the LTN approach on the performance of ML models in a clinical context. The implementation of LTN used in this study is publicly available[6], along with the code for replicating the experiments. The evaluation focuses on three main aspects:

1. The predictive performance of ML models, with particular attention to recall and F1-score, given the medical domain's sensitivity to false negatives.

[6] `https://github.com/pikalab-unibo/experiments-ltn-2024`

2. The adherence of ML models to established medical protocols, assessing whether the educated model aligns more closely with the injected knowledge compared to the uneducated one.
3. The robustness of ML models to data degradation, analyzing whether the performance of the educated model degrades less than that of the uneducated model under perturbed data.

To ensure a fair comparison, a grid search is performed to optimize the hyperparameters of all models. The best hyperparameters for the baseline models are as follows:

- KNN: $k = 7$, L1 distance.
- DT: Entropy as the split criterion, maximum depth of 10.
- RF: 50 trees, maximum depth of 20, entropy as the criterion, and a minimum of 5 samples per leaf.
- LR: Regularization parameter of 10, maximum iterations of 1,000, and `liblinear` as the solver.

The uneducated MLP and the educated LTN models share the same feedforward NN architecture:

- An input layer with 8 neurons, one for each feature in the dataset, using the leaky rectified linear unit (LeakyReLU) [MHN+13] activation function and batch normalization.
- Three hidden layers with 256, 128, and 64 neurons, respectively, each using LeakyReLU and batch normalization.
- An output layer with a single neuron and a sigmoid activation function.

The training of the LTN model consists of two phases. First, the uneducated model is trained for 50 epochs using the Adam optimizer with a learning rate of 10^{-4}, weight decay of 10^{-4}, a batch size of 32, and a binary cross-entropy loss function. An early stopping criterion is applied, with a patience of 5 epochs. Second, the knowledge is injected, and the model is fine-tuned for 200 epochs using the same early stopping criterion but with a reduced learning rate of 10^{-6}. All models are evaluated using multiple runs of a 50–50 train-test split, with performance metrics averaged over 50 runs. Statistical significance is assessed using the

Model	Accuracy	Precision	Recall	F1 Score	Balanced Accuracy	MCC
KNN	0.738 ± 0.02	0.636 ± 0.03	0.583 ± 0.05	0.607 ± 0.03	0.702 ± 0.02	0.413 ± 0.05
DT	0.697 ± 0.02	0.565 ± 0.03	0.580 ± 0.05	0.571 ± 0.03	0.670 ± 0.02	0.339 ± 0.05
RF	0.758 ± 0.02	0.677 ± 0.03	0.592 ± 0.05	0.630 ± 0.03	0.720 ± 0.02	0.456 ± 0.04
LR	**0.763 ± 0.02**	**0.702 ± 0.04**	0.560 ± 0.04	0.622 ± 0.03	0.716 ± 0.02	0.460 ± 0.04
MLP	0.754 ± 0.02	0.644 ± 0.03	0.665 ± 0.05	0.653 ± 0.03	0.733 ± 0.02	0.464 ± 0.04
LTN	0.747 ± 0.02	0.620 ± 0.03	**0.725 ± 0.05**	**0.666 ± 0.02**	**0.742 ± 0.02**	**0.471 ± 0.03**

Table 7.4: Performance metrics (mean ± std) for ML models. The best scores are highlighted in bold. According to the Wilcoxon signed-rank test with p-value=0.05, the LTN model outperforms all other models in terms of Recall, F1 Score, and Balanced Accuracy.

Wilcoxon signed-rank test with a significance level of 0.05. To evaluate robustness, the experiments include multiple runs with different types and magnitudes of data perturbations. The perturbations include noise addition, sample drop, and label flipping, following the methodology in [Agi+23]. The noise magnitude μ ranges from 0.1 to 1.0 in steps of 0.1, while the percentages of sample drop and label flipping vary from 0 to 0.9 in steps of 0.1.

7.3.4 Results and discussion

We now present and discuss the results obtained for predictive performance metrics, logical adherence, and robustness.

7.3.4.1 Performance metrics

The performance metrics are summarized in Table 7.4. While LR achieves the highest accuracy, this metric is less reliable for imbalanced datasets. In this study, the minority class, representing diabetic instances, is the primary focus. The LTN model achieves the highest scores in balanced accuracy, F1-score, and MCC. These metrics are more suitable for evaluating clinical models as they account for class imbalance. The LTN also achieves the highest recall value of 0.725, which is critical in clinical settings. Recall measures the proportion of diabetic patients correctly identified, minimizing false negatives. False negatives can lead to missed diagnoses and untreated conditions, which are particularly problematic in healthcare. In contrast, KNN and DT achieve recall values of 0.583 and 0.580, respectively, indicating their limited ability to detect positive cases. LR achieves a

Model	Rule 1	Rule 2
KNN	70.8 ± 4.8	100.0 ± 0.0
DT	66.5 ± 6.3	97.0 ± 4.8
RF	76.6 ± 5.1	100.0 ± 0.0
LR	75.0 ± 4.2	100.0 ± 0.0
MLP	82.0 ± 5.8	100.0 ± 0.0
LTN	**87.4 ± 5.2**	100.0 ± 0.0

Table 7.5: Logic adherence results for Equation (7.4) and Equation (7.5) (mean ± std) across various models.

recall of 0.560, while RF slightly improves with a recall of 0.592. Although the MLP shows some improvement with a recall of 0.66, the injection of domain knowledge via the LTN yields a significant performance boost. Precision, while highest for LR, is less emphasized in clinical contexts, as false positives typically lead to follow-up testing with minimal consequences. These results highlight the limitations of classical ML models in achieving sufficient recall for clinical deployment. The superior recall of the LTN can be attributed to its incorporation of logic formulas, which effectively utilize background knowledge. By grounding predictions in logical structures, the LTN identifies patterns that simpler models may overlook.

7.3.4.2 Logical adherence

In clinical applications, models must not only make accurate predictions but also align with established medical protocols. Adherence to clinical guidelines enhances trust and reliability, increasing the likelihood of adoption in practice. The logical adherence results are presented in Table 7.5. All models, except for DT, fully comply with the second rule, which classifies healthy patients. However, the primary focus is on adherence to the first rule, which identifies diabetic patients. The MLP achieves 82% adherence to the first rule, outperforming classical ML models such as RF, which achieves 76%. The LTN further improves adherence, achieving 87%. These results confirm that injecting domain knowledge through the LTN modifies the model's structure, aligning it more closely with medical knowledge and clinical protocols.

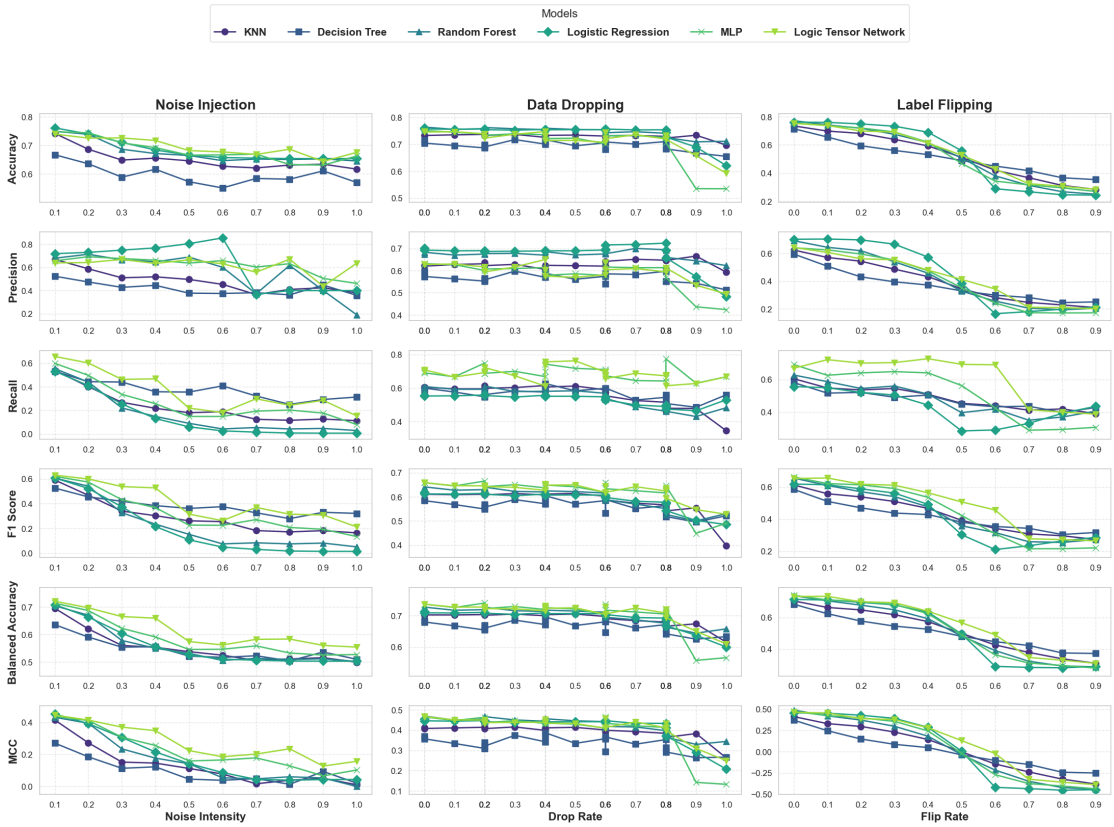


Figure 7.7: Robustness analysis under noise injection, label flipping, and data dropping. The LTN demonstrates superior robustness across all metrics compared to other models.

7.3.4.3 Robustness analysis

Robustness refers to the model's ability to maintain performance under data perturbations, such as noise, missing data, or label flipping. This evaluation is critical in clinical settings, where data collection is often noisy or incomplete. The robustness results are illustrated in Figure 7.7. Under noise injection, the LTN demonstrates the most stable performance across MCC, balanced accuracy, F1-score, and recall. Performance declines significantly only at noise levels above 0.5, which are uncommon in clinical environments. For label flipping, the LTN shows resilience up to a flipping rate of 0.6, with recall remaining virtually unchanged. A notable drop is observed only at a flipping rate of 0.7, which is rare in practice. In the case of data dropping, all models maintain stable performance up to a drop rate of

0.8. However, the LTN uniquely preserves recall, highlighting its ability to recover positive cases even with missing data. These findings underscore the advantages of knowledge injection via the LTN, which enhances robustness in complex clinical data scenarios.

7.4 Applying RAG on open LLMs for a medical chatbot supporting hypertensive patients

In this section, we report the contribution of the paper "Applying Retrieval-Augmented Generation on Open LLMs for a Medical Chatbot Supporting Hypertensive Patients", presented at the 3rd Workshop on Artificial Intelligence for Healthcare (HC@AIxIA 2024), co-located with the 23rd International Conference of the Italian Association for Artificial Intelligence (AIxIA 2024) [Agu+24]. The paper explores the use of retrieval-augmented generation (RAG) with open-source large language models to develop a medical chatbot for supporting hypertensive patients. By integrating external, domain-specific medical knowledge into the generation process, the proposed approach aims to improve factual accuracy and reduce hallucinations. The study provides empirical evidence on the feasibility of RAG-based architectures for safe and reliable patient-oriented digital health applications.

7.4.1 Motivations and background

Managing chronic diseases, such as hypertension, poses significant challenges for both healthcare systems and patients. These conditions require continuous monitoring, lifestyle adjustments, and often lead to substantial healthcare costs. Frequent interactions with healthcare professionals can result in long wait times and limited accessibility for patients. To address these issues, we propose the development of a chatbot to support patients in the self-management of chronic conditions, with a specific focus on hypertension. The chatbot aims to empower hypertensive patients by providing timely, accurate, and empathetic guidance, particularly for acquiring vital signs and maintaining a healthy lifestyle [Mon+24; Mon+23]. Two critical requirements emerge for such a system. First, the interaction must

be empathetic to ensure patient engagement and motivation. Second, the chatbot must provide highly accurate information, as no healthcare professional mediates the conversation. Additionally, the system must comply with data privacy regulations, precluding the use of third-party systems for NLP and natural language generation (NLG).

Given these requirements, we consider LLMs as the core technology for the chatbot. LLMs have demonstrated the ability to generate trustworthy, reliable, and empathetic text, making them suitable for this application. For instance, studies have shown that patients often prefer responses generated by LLMs over those from physicians in online forums, rating the chatbot's quality and empathy higher [Aye+23]. To avoid reliance on proprietary third-party services, we focus on open LLMs, including both domain-specific and general-purpose models. To enhance their performance, we integrate RAG techniques [Lew+20]. RAG enriches the LLM with symbolic knowledge, which aligns with the definition of SKI provided by [Cia+24]. Unlike traditional SKI methods that rely on logic rules, RAG incorporates symbolic knowledge in the form of free text, enabling the LLM to generate contextually relevant and precise responses.

The RAG approach involves constructing a knowledge base – usually stored in a vectorial data base – from data provided by medical professionals and enriching it with retrieval techniques. This allows for a comprehensive comparison of retrieval strategies and LLMs, both specialized and general-purpose. Our findings demonstrate that RAG significantly improves model performance, often surpassing specialized models in most tested cases.

7.4.1.1 Applications of LLMs in healthcare

The adoption of LLMs in healthcare has grown exponentially, with applications spanning patient care, research, and education. In patient care, LLMs-based chatbots can assist healthcare professionals by abstracting key results from literature, detecting medical errors, and supporting clinical decisions. For patients, these chatbots can provide trustworthy and empathetic answers, resembling a dialogue with a physician, and proactively suggest actions based on tracked activities and vital signs. In research, LLMs can automate tasks such as data analysis, summa-

rization, and literature search. In education, they can serve as interactive tutors, providing teaching material and demonstrating strong performance in medical examinations.

7.4.1.2 Challenges and techniques

Three key requirements must be addressed for deploying LLMs in healthcare. First, ethical concerns, including privacy and security risks, must be mitigated [Li+23]. Second, the system must be reliable, avoiding hallucinations and ensuring accuracy [HR24]. Third, the chatbot must communicate empathetically, motivating patients and providing real-time support [Aye+23]. To meet these requirements, open-source LLMs are preferred, as they allow local deployment and direct access to model weights. However, open models may generate irrelevant or incomplete content, undermining trust and compliance. To address these limitations, two primary techniques are recommended: RAG and fine-tuning. While fine-tuning aligns models with domain-specific datasets, RAG dynamically updates the knowledge base, ensuring relevance to recent information.

7.4.1.3 RAG in the medical domain

RAG represents an innovative integration of information retrieval and generative models, enabling access to a medical knowledge base for generating precise and contextually relevant responses. This approach enhances the safety and effectiveness of LLMs in healthcare, where accuracy and specificity directly impact patient care quality. For example, RAG has been used to improve LLMs in digestive diseases [Giu+24] and Hepatitis C management [Kre+24], outperforming baseline models in delivering guideline-specific recommendations. However, implementing RAG in healthcare requires tailoring solutions to local needs, considering factors such as demographics, resources, and cultural practices. This flexibility necessitates a modular and configurable architecture.

7.4.1.4 Fine-tuning in the medical domain

Fine-tuning has demonstrated impressive results in specialized medical domains [Mah+24]. For instance, Wang et al. [Wan+24] fine-tuned the Llama 2

model using clinical concepts from standardized vocabularies, such as the Human Phenotype Ontology (HPO), to address underdiagnosis and misdiagnosis. However, due to the limited dataset size in this study, we focus on RAG, which allows dynamic updates to the knowledge base, ensuring the model remains relevant with recent information.

7.4.2 Materials and methods

7.4.2.1 Fine-tuning

LLMs are initially pre-trained on large, general-purpose text corpora to predict the next token in a sequence. This process equips the model with a broad understanding of language and serves as a foundation for fine-tuning. Fine-tuning involves further training the model on smaller, domain-specific datasets to adapt it to specific tasks or fields. This approach is computationally efficient as it leverages the model's pre-existing knowledge while refining it for specialized applications.

Full fine-tuning updates all the model's parameters to optimize its performance for a specific task. While effective, this method is resource-intensive, requiring significant computational power and memory.

Parameter-efficient fine-tuning (PEFT) updates only a subset of the model's parameters, freezing the rest. This reduces memory requirements while retaining the model's general linguistic knowledge. Techniques such as low-rank adaptation (LoRA) and quantized LoRA (QLoRA) further minimize resource usage by refining smaller weight matrices and lowering the precision of adapter weights [Hu+22; Det+23]. By selecting the appropriate fine-tuning method, models can be optimized for specific use cases while minimizing resource consumption.

7.4.2.2 RAG

Traditionally, LLMs generate responses based solely on patterns and information learned during pre-training. This limitation often results in responses lacking depth or specific knowledge. RAG addresses this by integrating external data during the response generation process.

The RAG framework consists of two main phases:

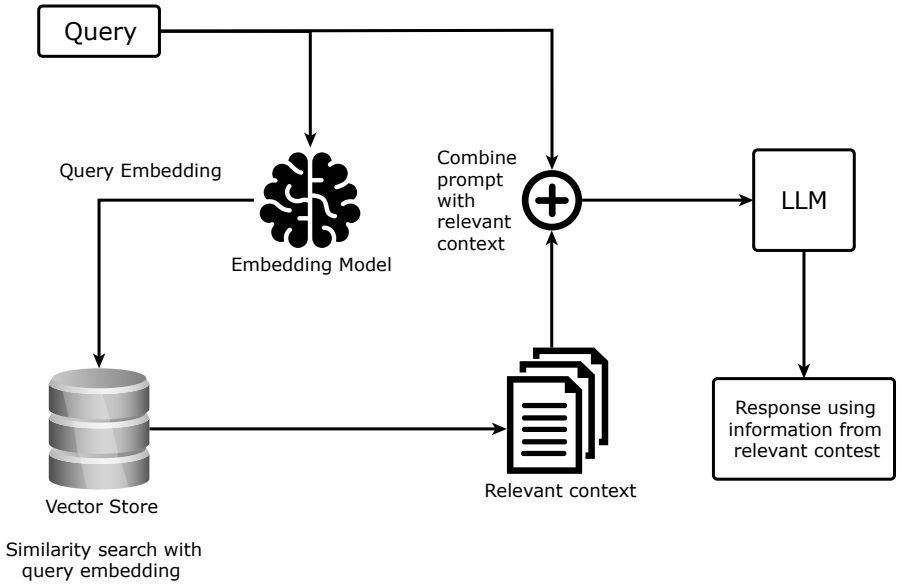


Figure 7.8: RAG architecture. The retriever identifies relevant information from a knowledge base, which the generator uses to produce a contextually appropriate response.

1. Retrieving relevant information from a large dataset or knowledge base in response to a query.
2. Using the retrieved information to guide the generation of the response.

This enables software agents, such as chatbots, to provide accurate and context-specific answers by supplementing the model's internal knowledge with external information, such as private documentation or databases.

The retriever component identifies relevant information by splitting documents into smaller fragments (chunks) and converting these chunks into embedding vectors. These vectors are stored in an indexed knowledge base for efficient retrieval. When a query is processed, the system generates a query vector and matches it with stored document vectors using vector similarity techniques. Sparse embeddings, such as those based on TF-IDF or BM25 [LRU14; RZ09], rely on keyword matches but may struggle with synonyms and semantic meaning. Dense embeddings, generated by models like bidirectional encoder representations from transformers (BERT), capture deeper semantic relationships, enabling retrieval based on meaning rather than exact words [Dev+19]. Hybrid approaches combine both methods to balance speed and semantic depth. The generator component, a LLM,

produces the final text response. Figure 7.8 illustrates the RAG architecture, highlighting the interaction between the retriever and generator components.

7.4.2.3 RAG vs. fine-tuning

RAG and fine-tuning represent distinct approaches to enhancing LLMs, each with unique advantages.

- **Knowledge integration vs. task specialization:** RAG integrates external knowledge dynamically, enhancing versatility and ensuring up-to-date information. Fine-tuning specializes the model for specific tasks, improving task-specific accuracy.
- **Dynamic vs. static learning:** RAG allows dynamic access to external data, while fine-tuning updates the model based on a static training cycle.
- **Generalization vs. customization:** RAG preserves the model's generality, making it adaptable across tasks, whereas fine-tuning tailors the model for specific use cases.
- **Resource demands:** RAG requires runtime retrieval mechanisms, while fine-tuning is resource-intensive during training but efficient during deployment.

7.4.2.4 Chosen approach

This study employs RAG techniques to enhance the performance of open LLMs for developing a chatbot supporting hypertensive patients. RAG was chosen over fine-tuning for several reasons:

- It dynamically integrates external data, allowing the system to adapt to diverse contexts and provide accurate information.
- Fine-tuning requires extensive computational resources, which are currently unavailable.
- RAG mitigates issues such as model drift, hallucinations, and biases that can arise during fine-tuning.

Retrieval strategies Three retrieval strategies were tested:

Model	Description
Llama3.1	A general-purpose LLM based on the Llama3 architecture with 8B parameters, trained on a diverse range of text data [Gra+24].
Llama3.1-Medical	A medical-domain-specific version of Llama3.1, fine-tuned on medical data to enhance its performance in healthcare applications: `https://ollama.com/qordmlwls/llama3.1-medical`.
Qwen2	A general-purpose LLM with 7B parameters, trained on 29 different languages to improve cross-lingual performance [Yan+24].
Qwen2-Medical	A medical-domain-specific version of Qwen2, fine-tuned on medical data to improve its performance in healthcare tasks: `https://ollama.com/echelonify/med-qwen2`.
Mistral-Nemo	A 12B parameter model with a large context window (128K tokens) developed by Nvidia: `https://ollama.com/library/mistral-nemo`.
Phi3	A relatively small LLM with 3B parameters, trained by Microsoft on filtered high-quality data [Abd+24].
Gemma2	A 9B parameter model based on Deepmind Gemini developed by Google [Ta24].

Table 7.6: Evaluated LLM for our study.

- **Base retriever:** Retrieves information based on vector similarity using maximum marginal relevance [CG98].
- **Multi-query retriever:** Enhances diversity and relevance by generating multiple queries with a LLM and retrieving relevant chunks for each query.
- **Ensemble retriever:** Combines outputs from multiple retrievers, such as the base retriever and BM25, using reciprocal rank fusion [CCB09].

The dataset used for RAG consists of 1,473 question-answer pairs extracted from medical consultations. The questions cover topics related to hypertension, including symptoms, causes, treatments, and lifestyle recommendations. All answers, written in Italian, were reviewed by medical professionals to ensure accuracy and relevance.

The evaluated LLMs include both general-purpose models (e.g., Llama 3.1, Qwen2, Mistral Nemo) and medical-domain-specific models (e.g., Llama 3.1-Medical, Qwen2-Medical). Further details are provided in Table 7.6.

7.4.3 Evaluation

7.4.3.1 Experimental setup

We evaluated the effectiveness of various RAG techniques using the RAGAS framework[7]. This framework provides a comprehensive suite of metrics for assessing retrieval and generation aspects of LLM-based systems. Instead of partitioning the dataset into training and test sets, we leveraged an external LLM (GPT-4o) to generate a test set of 20 question-context-answer triplets. These triplets were designed to maintain statistical relevance to the original dataset. We evaluated a range of state-of-the-art open-source LLMs, both general-purpose and domain-specific, with and without RAG. The evaluation employed the following prompt:

Medical system prompt

You are an AI medical assistant specializing in hypertension. Provide detailed and evidence-based answers, using clear and accessible language. Always respect patient privacy, and if you are unsure of the answer, state "I am not sure of the answer." Base your response on the provided context to answer accurately. Include current recommendations and explain medical concepts in an understandable way.
Context: context

[7] `https://docs.ragas.io/en/stable/`

```
**Question:** question
```

In the case of RAG, the context was the retrieved information from the knowledge base. Otherwise, the context was left empty.

7.4.3.2 Metrics

To evaluate the performance of the RAG systems, we employed three key metrics: *Answer Relevancy*, *Answer Correctness*, and *Faithfulness*. These metrics were computed using a reference LLM (GPT-4o) to analyze the quality of the generated responses.

Answer relevancy This metric measures the alignment between the generated response and the original question. It is computed as the cosine similarity between the embedding of the original question, E_o, and the embeddings of the generated responses, E_{g_i}, for $i \in \{1, \ldots, N\}$:

$$\text{Answer Relevancy} = \frac{1}{N} \sum_{i=1}^{N} \cos(E_{g_i}, E_o) = \frac{1}{N} \sum_{i=1}^{N} \frac{E_{g_i} \cdot E_o}{\|E_{g_i}\| \|E_o\|}. \tag{7.8}$$

Higher values indicate greater relevance.

Answer correctness This metric, denoted as AC, evaluates the factual accuracy of a generated answer A with respect to a ground truth answer G. It combines *Factual Correctness* (FC) and *Semantic Similarity* (SS), both ranging from 0 to 1:

$$AC = w_1 \cdot FC + w_2 \cdot SS, \quad w_1 + w_2 = 1. \tag{7.9}$$

The FC is computed using the F_1-score:

$$FC = \frac{|TP|}{|TP| + 0.5 \cdot (|FP| + |FN|)}, \tag{7.10}$$

where TP, FP, and FN represent true positives, false positives, and false negatives, respectively. The SS is calculated as the cosine similarity between the embeddings of A and G.

Faithfulness This metric assesses the consistency of the generated answer A with the provided context C. Let $C_A = \{c_1, c_2, \ldots, c_n\}$ be the set of claims in A. The faithfulness score F is defined as:

$$F(A, C) = \frac{|\{c_i \in C_A \mid c_i \text{ can be inferred from } C\}|}{n}. \tag{7.11}$$

Higher scores indicate greater factual consistency.

7.4.3.3 Results

Our experiments revealed that RAG-based systems consistently outperformed plain LLMs in terms of *Answer Relevancy* and *Answer Correctness* as shown in Figure 7.9. The performance boost was particularly significant for models trained on medical data, with improvements of up to 20% in correctness and 40% in relevancy.

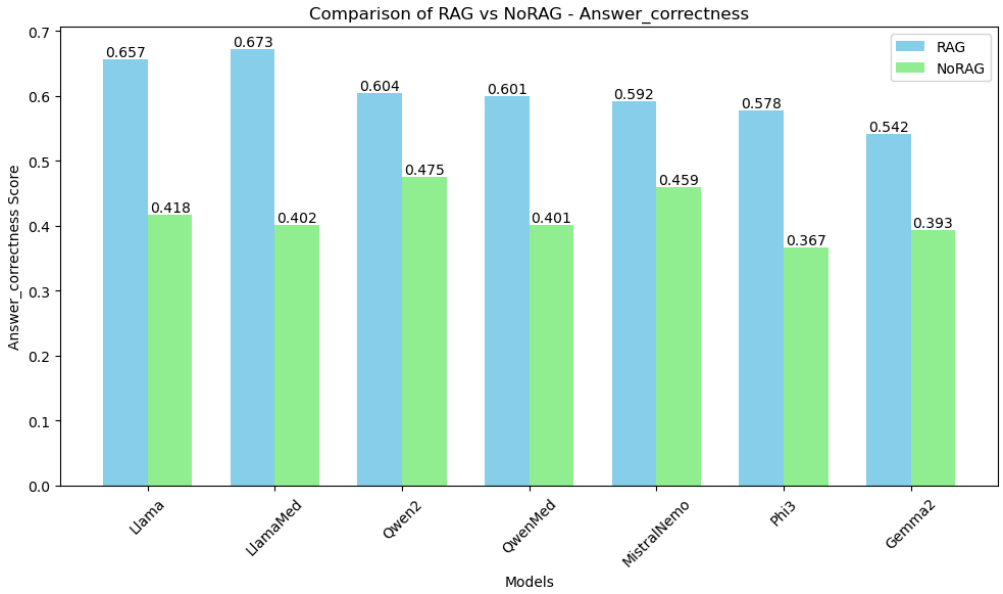
Impact of retrieval strategies The choice of retrieval strategy significantly influenced performance. As illustrated in Figure 7.10, each strategy exhibited distinct strengths and weaknesses across the evaluation metrics. The *Base retriever* excelled in correctness and faithfulness, while the *Ensemble Retriever* achieved superior relevancy. The *Multi-query retriever* showed promise in relevancy but struggled with correctness due to the inclusion of irrelevant information.

Domain-specific fine-tuning Specialized models, such as LlamaMed and Qwen2-Med, outperformed their general-purpose counterparts in relevancy (up to 5%) and correctness (up to 3%). However, RAG-augmented base models consistently surpassed even specialized models without RAG, highlighting the advantages of retrieval augmentation.

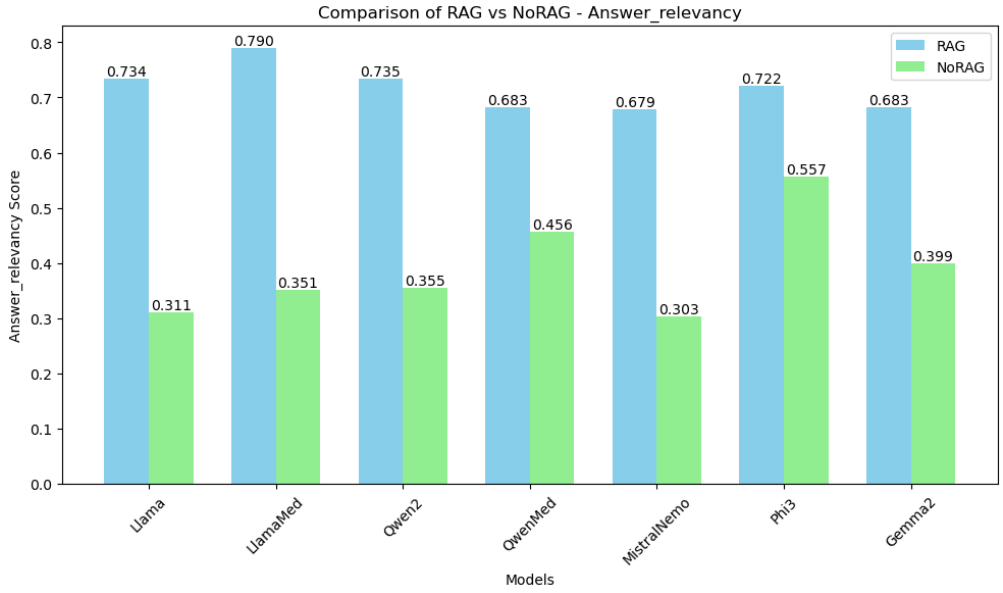
7.4.3.4 Discussion

The findings demonstrate the effectiveness of RAG as a robust tool for enhancing open-source LLMs in the medical domain. By incorporating a verified knowledge base, RAG significantly improves the accuracy and relevance of chatbot responses, making it an optimal solution for supporting hypertensive patients. Even without

large datasets for fine-tuning, RAG provides an efficient, out-of-the-box solution for creating high-performance chatbots in specialized domains. Future work should include human evaluation by medical professionals to assess clinical accuracy and explore metrics for evaluating empathetic responses.

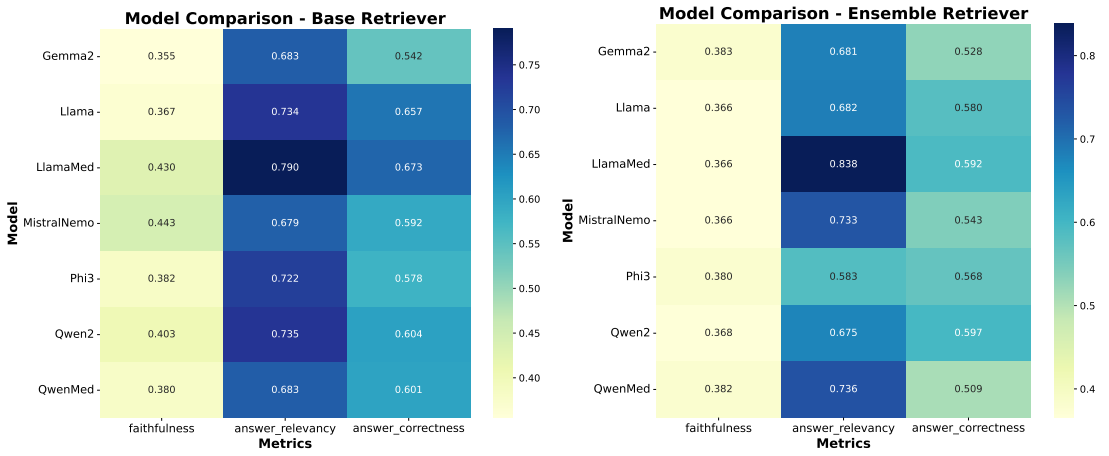


(a) Answer Correctness

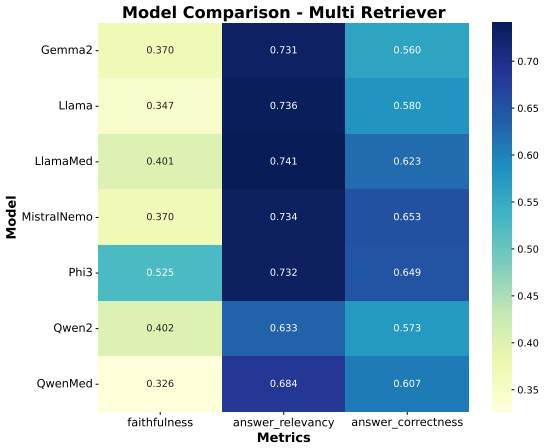


(b) Answer Relevancy

Figure 7.9: Performance comparison of RAG-based systems against plain LLM systems. (a) Answer Correctness, (b) Answer Relevancy. The results demonstrate that RAG significantly enhances both answer correctness and relevancy across all tested models.



(a) Base Retriever **(b)** Ensemble Retriever



(c) MultiQuery Retriever

Figure 7.10: Performance comparison of different retrieval strategies: (a) Base Retriever, (b) Ensemble Retriever, (c) MultiQuery Retriever. Each strategy demonstrates strengths and weaknesses across various evaluation metrics, highlighting the need for careful selection based on the specific task and model.

Chapter 8

Autonomous learning systems

What if SKI and SKE methods are paired together?
How to design autonomous learning systems exploiting SKI and SKE?
– **RQ0, RQ3 and RQ4**

Contents

This is arguably the most important chapter of the thesis, the crucial moment when all the bricks come together. We choose to place this chapter at towards the end of the thesis mainly for two reasons. First of all, the works described in this chapter build upon the concepts and previous contributions on NeSy AI, presented in the rest of the thesis. Second, from a chronological perspective, these works were carried out in the last part of the PhD and this thesis tries to follow a logical and temporal order.

Here we present key contributions of the PhD towards the development of autonomous learning systems exploiting SKI and SKE. In particular, we make use of SKE to systematically extract symbolic knowledge from a trained sub-symbolic model, and conversely, we exploit SKI to provide the sub-symbolic model with relevant context information. This cycle between SKI and SKE is an early idea of my PhD, and it is presented in Section 8.1.

The goal of autonomous learning systems is straightforward: to create systems that can learn – and possibly adapt – independently, without requiring constant human intervention. The knowledge that is learnt can be presented in many forms. We discussed in details about the differences between symbolic and sub-symbolic representations in Section 2.2 and also the many forms of symbolic knowledge that can be used to represent knowledge, such as rules, ontologies, and knowledge graphs Section 2.2.1. We believe that the knowledge learnt by autonomous learning systems should be represented symbolically. The advantages of symbolic representations are many, as we discussed in Section 2.2.1. Here, we recall the most important ones: *(i)* interpretability: symbolic knowledge is human-readable, which makes it easier to understand and verify; *(ii)* reasoning: symbolic knowledge can

be used to perform logical reasoning; *(iii)* transferability: symbolic knowledge can be easily transferred between different systems and domains, which makes it more versatile; *(iv)* model compression: symbolic knowledge can be more compact than sub-symbolic representations, which makes it more efficient; *(v)* model checking: symbolic knowledge can be formally verified, which makes it more reliable.

On the other hand, sub-symbolic models have recently reached impressive performance in many tasks, especially in NLP and NLG, thanks to the advent of LLMs. Indeed, LLMs are de-facto the new drivers of AI. Their presence is already ubiquitous, and they are used daily by millions of people. Many LLMs are so good at understanding and generating natural language that they can be considered as domain experts in many fields. Still, limitations exist, and the possibility to generate hallucinations is always present.

In the following sections, we will present first the conceptual SKI-SKE loop, and then two main contributions towards the development of autonomous learning systems. Both contributions exploit LLMs to generate or finetune symbolic knowledge about specific domains.

8.1 Bridging symbolic and sub-symbolic AI

In the discussion paper "Bridging Symbolic and Sub-symbolic AI: Towards Cooperative Transfer Learning in Multi-Agent Systems" [MCO22b] presented at the 22nd International Conference of the Italian Association for Artificial Intelligence (AIxIA 2022), we introduced the ideas of a loop between SKI and SKE processes in the context of MASs. For computational agents, algorithms exist to mimic basic cognitive capabilities such as induction, communication, knowledge representation, and reasoning. These capabilities are developed under the domains of symbolic AI and ML. However, unlike humans, symbolic AI and ML algorithms are typically designed to solve specific tasks and do not inherently support interaction, cooperation, or knowledge exchange.

This work proposes that ML-based intelligent systems can benefit from symbolic knowledge exchange to enhance their learning capabilities [Cia+21; Omi20; Cal+21]. We argue that symbolic knowledge exchange can play a crucial role in enabling software agents to achieve the ability to *learn to learn* new tasks.

We introduce two types of intelligent systems: cooperative learning (CoL) and cooperative transfer learning (CoTL). CoL systems are MASs in which agents can retrieve or provide knowledge about specific tasks to or from other agents. This knowledge can be used during learning or inference. CoTL systems extend CoL systems by enabling agents to acquire, exploit, and combine knowledge about related tasks to learn novel tasks they were not explicitly designed for. Both systems aim to mimic the learning processes of human societies, albeit to varying extents. Agents in these systems collaborate by sharing symbolic and, potentially, sub-symbolic knowledge about their tasks, similar to human interactions.

This chapter outlines a roadmap for leveraging knowledge representation and sharing to achieve higher levels of artificial intelligence through CoL and CoTL systems. We analyze the requirements of these systems in relation to the state of the art and discuss their potential implementation.

8.1.1 Background

We assume the understanding of symbolic and sub-symbolic representations as given, as these concepts have been extensively discussed in Sections 2.2 and 2.2.1 and throughout the thesis. Symbolic representations, characterized by their interpretability and reasoning capabilities, contrast with sub-symbolic models, which excel in pattern recognition and scalability. This duality forms the foundation for the integration of these paradigms, which is central to the development of autonomous learning systems.

8.1.1.1 Transfer vs. multi-task learning

A *task* in ML refers to any supervised learning problem. Transfer learning (TL) enables a predictor P trained on a task T' to transfer knowledge K to another predictor P' targeting a related task T. The primary goals of TL are to: *(i)* reduce the data required to train P', *(ii)* accelerate training, and *(iii)* improve predictive performance.

TL methods vary in *what*, *how*, and *when* to transfer knowledge [PY10]. Most TL techniques focus on sub-symbolic knowledge transfer, such as reusing shallow layers of a NN [Shi+16]. To date, no TL methods explicitly leverage symbolic

knowledge transfer.

multi-task learning (MTL) extends TL by training multiple predictors $P_1, \ldots, P_m$ on a set of related tasks $\{T_1, \ldots, T_m\}$ simultaneously [Car97]. This approach allows each predictor to benefit from the knowledge shared across tasks [ZY22]. Unlike TL, where knowledge flows in one direction, MTL facilitates bidirectional knowledge transfer.

MTL techniques are classified based on whether they target *homogeneous* or *heterogeneous* tasks [DK17]. Homogeneous tasks share the same input and output attributes, differing only in data distribution. Heterogeneous tasks, on the other hand, have distinct attributes with little or no overlap.

In MTL, determining task similarity remains an open challenge, particularly for heterogeneous tasks. Empirical methods, such as testing performance improvements with MTL, are often used to assess task relatedness.

8.1.2 Contributions

8.1.2.1 CoL

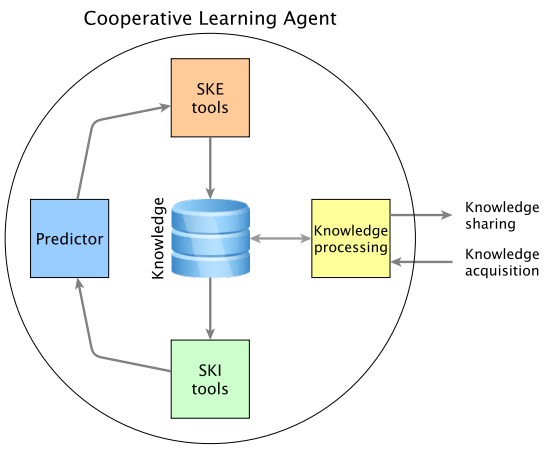
A CoL system is a MAS where agents can exchange knowledge about specific tasks. This exchange includes both retrieving knowledge from other agents and providing knowledge when requested. Symbolic knowledge sharing is particularly crucial, as it enables effective agent-to-agent communication and transfer [Omi20].

To enable CoL, agents must possess the following capabilities:

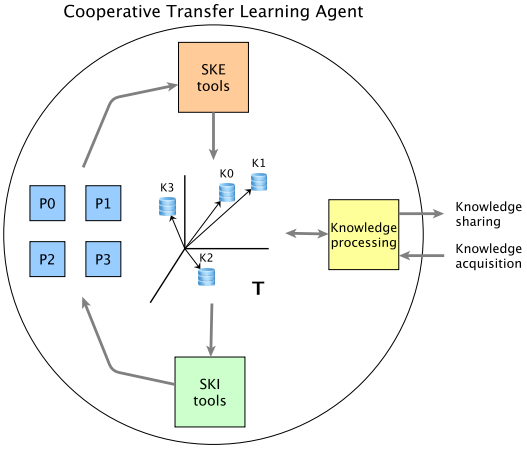
1. learning from experience and adapting their behavior accordingly,
2. representing their internal behavior in a symbolic format,
3. updating their behavior based on symbolic specifications,
4. interacting with other agents to exchange symbolic knowledge.

Capabilities 2 and 3 are complementary. When combined with 4, they facilitate cooperative learning through interaction. Capability 1 ensures that agents can independently acquire new behaviors.

In practice, CoL systems often rely on sub-symbolic ML to support capability 1. Each agent is typically equipped with an ML predictor to learn from local data.



(a) Agent's architecture for CoL. The agent is equipped with a set of SKE and SKI methods to manipulate knowledge, plus some functions to pre/post-process knowledge.



(b) Agent's architecture for CoTL. The agent is equipped with a set of SKE and SKI methods to manipulate knowledge, functions to pre/post-process knowledge, plus a task space and relatedness functions to evaluate if two different tasks are related. Each task in the task space T is paired with its specific knowledge. For each task the agent has at least one predictor. In the example are represented the knowledge of four tasks but they can come in arbitrary number.

Figure 8.1: Architectures for CoL and CoTL agents.

Since MASs often consist of heterogeneous agents with diverse purposes, multiple types of predictors may coexist within the same system.

To support capability 4, agents must agree on a shared symbolic representation for describing and exchanging behavioral specifications. SKE and SKI techniques are instrumental in enabling capabilities 2 and 3, respectively.

Figure 8.1a illustrates the architecture of a CoL agent. Each agent is equipped with one or more SKE and SKI algorithms to handle symbolic knowledge input and output. When queried, an agent extracts symbolic knowledge from its internal ML predictor using SKE and shares it with the requesting agent. The recipient agent can then update its predictor by injecting the received knowledge using SKI. Pre- and post-processing steps, such as pruning or merging formulas, may be applied to regulate the transferred knowledge.

Key design considerations for CoL systems include: *(i)* selecting appropriate formalisms for knowledge representation, and *(ii)* choosing suitable SKI/SKE

techniques based on the predictors' knowledge representation. For *(i)*, while FOL offers high expressiveness, it may limit the availability of compatible techniques. Less expressive logics, such as propositional logic, may also be considered for practical reasons.

Agents can be trained without prior knowledge. During training, they can extract knowledge, combine it with knowledge from other agents, inject it back, and continue training. This iterative process, known as the train-extract-fix-inject cycle, improves the agent's performance over time. With each cycle, the extracted knowledge becomes increasingly accurate, as the predictor itself improves.

8.1.2.2 CoTL

A CoTL system extends CoL by enabling agents to leverage knowledge from related tasks to learn *novel* tasks. The ultimate goal of CoTL systems is to enable agents to "learn to learn."

The concept of learning to learn builds on the definition of ML [Mit97] and was introduced in [TP98]. Formally, given:

- a set of tasks $\mathcal{T} = \{T_1, \ldots, T_n\}$,
- trainable experience for each task $\mathcal{E} = \{E_1, \ldots, E_n\}$, such as ML predictors or symbolic knowledge bases, and
- a performance measure $f_i : \mathcal{T} \times \mathcal{E} \to \mathbb{R}$ for each task,

an agent is capable of learning to learn if f_i improves as a function of all items in $\mathcal{E}$ and the number of tasks n.

Practical aspects of CoTL include determining *when* and *how* to transfer experience. For the 'when,' agents must autonomously compute task similarity, as manual intervention is infeasible. Similarity measures, such as distance-based or learning-based methods, can be employed [Car97].

For the 'how,' the approach depends on whether tasks are *homogeneous* or *heterogeneous*. Homogeneous tasks share input and output spaces but differ in data distribution, allowing straightforward knowledge transfer. Heterogeneous tasks, however, differ in both input and output features, requiring more complex strategies.

Consider two heterogeneous classification tasks T_1 and T_2 with knowledge bases K_1 and K_2 composed of Horn clauses. Knowledge transfer can proceed as follows:

1. If a rule $r \in K_1 \cup K_2$ refers only to shared input features and classes, it can be used as-is.
2. If r's body refers to shared features but its head targets task-specific classes, SKI techniques or class mappings may be required.
3. If r involves both shared and task-specific features:
   (a) Relax the rule to include only shared features, then apply step 1.
   (b) Map task-specific features between tasks [DK17], then apply step 1.
   (c) If neither is possible, discard the rule.

This procedure can be adapted for other tasks, such as regression, and other knowledge representations.

Figure 8.1b depicts the architecture of a CoTL agent. In addition to SKE and SKI algorithms, CoTL agents require a task similarity function $\sigma : \mathcal{T}^2 \to \mathbb{R}_{\geq 0}$ to evaluate task relatedness.

Another design consideration is the criterion for selecting knowledge from related tasks. For example, designers may use a threshold-based approach, selecting knowledge from tasks with a similarity score above a certain threshold. Alternatively, the k most similar tasks can be chosen.

8.1.3 Discussion and conclusions

The integration of symbolic and sub-symbolic knowledge representations, along with knowledge manipulation tools, represents a promising direction for advancing MASs. This approach aims to mimic the knowledge-sharing capabilities of human societies. It has the potential to address several limitations in the current state of the art. For instance, TL focuses exclusively on sub-symbolic knowledge, which lacks human interpretability. Similarly, MTL requires simultaneous training on related tasks, posing scalability challenges. Moreover, MTL is currently limited to sub-symbolic knowledge alone.

CoL and CoTL systems offer significant opportunities for the development of intelligent systems. Some of their key advantages include: *(i)* providing more

accurate and human-interpretable explanations for tasks; *(ii)* improving the performance of agents or predictors in solving individual tasks; *(iii)* enabling improvements in one agent to positively influence others; *(iv)* facilitating knowledge transfer between tasks, leading to improvements across multiple domains; *(v)* supporting continuous and automated learning without human intervention.

Despite these advantages, several challenges remain. First, while many SKE and SKI algorithms have been proposed, their practical implementations are still rare. Second, determining task similarity in CoTL systems is non-trivial and can significantly impact system performance. Third, trust mechanisms must be considered. For example, how reliable is the knowledge received by an agent? What is the reputation of the knowledge source? Finally, there is a need for datasets – likely smaller than those required for systems without CoL and CoTL – to train predictors effectively on new tasks. Humans, for instance, can perform new tasks with minimal training, often relying solely on explanations (e.g., learning to play a new game). Achieving such capabilities would mark a significant step toward artificial general intelligence.

In summary, this work introduces the novel concepts of CoL and CoTL within the scope of MASs. These systems integrate symbolic and sub-symbolic knowledge representations and manipulation tools to emulate the learning processes of human societies. We propose a general agent architecture for both CoL and CoTL, highlighting their advantages and limitations.

8.2 Ontology instantiation with LLMs

In this section we present the paper "Large language models as oracles for instantiating ontologies with domain-specific knowledge" [Cia+25], which is published in the international journal *Knowledge-Based Systems*.

The demand for intelligent systems capable of understanding, reasoning, and interacting with complex information is rapidly increasing. The development of such systems relies on the integration of machine-interpretable knowledge representations that can encapsulate human domain-specific knowledge across diverse fields.

The proliferation of data and its varied representation formats raises the ques-

tion: *How can domain-specific information be precisely represented in a machine-understandable manner?* The answer lies in the concept of ontology [Gri10]. Ontologies are formal, extensional representations of knowledge based on the principles of DLs. They define concepts, relationships, and properties in a structured, machine-readable format, enabling computational systems to reason about the world in a manner akin to human cognition.

Unlike sub-symbolic approaches, which rely on distributed or loosely organized data [Cia+24], ontologies provide a structured framework. This framework bridges the *semantic gap*, allowing AI systems to move beyond superficial pattern recognition and grasp the underlying meaning of data. From a human perspective, ontologies offer a shared vocabulary and an unambiguous understanding of domain-specific knowledge. However, creating ontologies is a meticulous and time-consuming process, requiring adherence to the real-world domain they represent.

Currently, ontology population is performed either manually by domain experts or communities, or through semi-automatic extraction from data [Pet+11]. Manual population, while time-intensive and prone to human error, can yield high-quality results if the process is thorough and inclusive. In contrast, data-driven approaches offer speed and scalability but may result in lower-quality ontologies due to biases in the data or limitations in the extraction process.

An ideal ontology population method would combine human- and data-driven approaches. The former would refine and validate the ontology, while the latter would provide the necessary volume of instances. To this end, we introduce KGFiller, a novel approach that amalgamates the strengths of both methods.

Leveraging the observation that LLMs are trained on diverse data from the web, we hypothesize that these models encapsulate substantial domain-specific knowledge. KGFiller is an automated procedure designed to extract this knowledge from LLMs and use it to populate ontologies. Starting with an initial schema of interrelated classes and properties, and a set of query templates, the method queries the LLM multiple times. It generates instances for classes, relationships, and properties based on the responses. Further queries refine the ontology and balance the class hierarchy, ensuring compliance with the initial schema. The result is an enriched ontology that experts can review, adjust, or complement as needed.

Our method has several advantages over state-of-the-art approaches. First, it is not tied to any specific dataset, as it uses LLMs as oracles to generate data. Second, it supports both ontology population and refinement, making it incremental and applicable to pre-existing ontologies. Finally, it is general-purpose, suitable for various domains and different LLMs.

To validate our approach, we implemented KGFiller in Python and used it to populate a custom web ontology language (OWL) ontology. We evaluated the quality of the populated ontology through a detailed inspection of the generated instances, assessing their meaningfulness and placement within the ontology structure. The experiment was repeated with different LLMs, and the results were compared.

In summary, our method contributes to the ongoing discourse on ontology population by presenting a hybrid approach that leverages the capabilities of LLMs. To further analyze the strengths, weaknesses, opportunities, and threats of our approach, we dedicate a section to a detailed strengths, weaknesses, opportunities and threats (SWOT) analysis.

8.2.1 Related work

Ontology population The task of populating ontologies with instances, commonly referred to as *ontology population*, is well-established in the semantic web community [LNM19]. This process involves expanding an ontology with additional assertions, also known as *ABox axioms*, while ensuring compliance with the concepts and properties defined in the *TBox axioms*. Some authors use the term *ontology learning* to emphasize the (semi-)automatic inference of the entire ontology, including TBox axioms. However, in this work, we adhere to the term *ontology population*.

The need for (semi-)automatic ontology population arises from the limitations of manual methods, which are time-consuming, error-prone, and inherently costly [KAG21]. To address these challenges, various methods for structured data extraction from text have been proposed. These methods can be broadly categorized into *linguistics-based* and ML approaches.

Linguistics-based approaches [FM99; Mor99; Har+03] include techniques such

as syntactic analysis, pattern-based extraction, part-of-speech tagging, and the use of dictionaries. While effective, these methods require significant human effort for tasks like corpus selection, supervision, and parameter fine-tuning. Additionally, they are often domain-specific, as different corpora may yield varying statistical information, necessitating tailored thresholds.

ML-based approaches can be further divided into *shallow ML* and *deep learning* techniques. Shallow ML methods rely on classical NLP techniques, such as term frequency-inverse document frequency (TF-IDF), to convert text into numerical data, which is then processed by algorithms like decision trees [TM06; Yoo+07; MLP08; CV04; Etz+05; Jia+11; SCR15]. However, these methods often struggle to capture contextual information. In contrast, deep learning approaches automatically learn representations from data, enabling them to better capture contextual nuances [Liu+13; Zen+14; Aya+19].

Despite their advancements, all these methods require a large corpus of textual documents for extracting concepts, instances, and properties. This reliance on external data introduces challenges such as bias, incompleteness, and domain non-representativeness. Moreover, state-of-the-art methods are highly domain-sensitive and lack incrementality. Domain sensitivity refers to the difficulty of finding sufficient relevant documents for narrow domains or avoiding irrelevant information in broad domains. Poor incrementality implies that once an ontology is populated, adding new instances often requires re-populating it from scratch.

LLMs and knowledge graphs In recent years, LLMs have been applied to (semi-)automatic manipulation of KGs [Pan+24; Zhu+24; Pan+23]. KGs represent knowledge as structured triplets of the form (s, p, o), where s is the subject, p the predicate, and o the object. Unlike ontologies, KGs do not impose strict schema constraints, making them more flexible but less amenable to automated reasoning [EW16]. Conversely, ontologies use a well-defined set of axioms, ensuring compliance with the original schema. Applications of LLMs to KGs can be categorized into *KG completion* and *KG construction*.

KG completion involves predicting missing facts, where LLMs act as either *encoders* or *generators*. As encoders, LLMs encode textual information to assist another model in predicting missing facts [CJK21; Wan+21a; She+22]. As gen-

erators, they directly extract missing facts [SKG22; Che+22; Xie+22; Zhu+24]. However, these methods often require input text or ad-hoc training, which our approach does not.

KG construction involves building a KG from scratch, including tasks like entity discovery, end-to-end construction, and KG distillation. Entity discovery identifies entities in text data and links them to form a KG [Ayo+22; Cao+21]. End-to-end methods generate KGs directly but rely on domain-specific textual data [Kum+20; MDD21; Han+23]. KG distillation uses LLMs as oracles to generate triplets. Notable examples include `Comet` [Bos+19] and `Harvest` [Hao+23], which generate triplets based on prompts. While similar to our method, these approaches are not tailored for ontologies and may produce inconsistent KGs. Additionally, they often require prompt engineering or specific training phases, which our method avoids.

8.2.2 Filling ontologies with KGFiller

The KGFiller framework is designed for semi-automatic ontology population, leveraging LLMs as oracles. An overview of the KGFiller framework is depicted in Figure 8.2. This section provides a concise yet general description of the framework.

Core functionality At its core, KGFiller operates on a partially instantiated ontology. The ontology must include class and property definitions, organized hierarchically as a directed acyclic graph (DAG). Classes may or may not have instances.

Class names should be meaningful, concise, and unambiguous to avoid issues during query generation for the LLM. The same applies to property names.

Given these assumptions, KGFiller generates individuals for all classes and associates them using the ontology's properties. Generated individuals are assigned meaningful names and placed in the most specific class available.

The framework supports two primary use cases: *(i)* ontology designers have defined the ontology's structure but lack instances, and *(ii)* designers want to add more instances to an already populated ontology. These use cases can be combined, allowing iterative rounds of population and manual refinement.

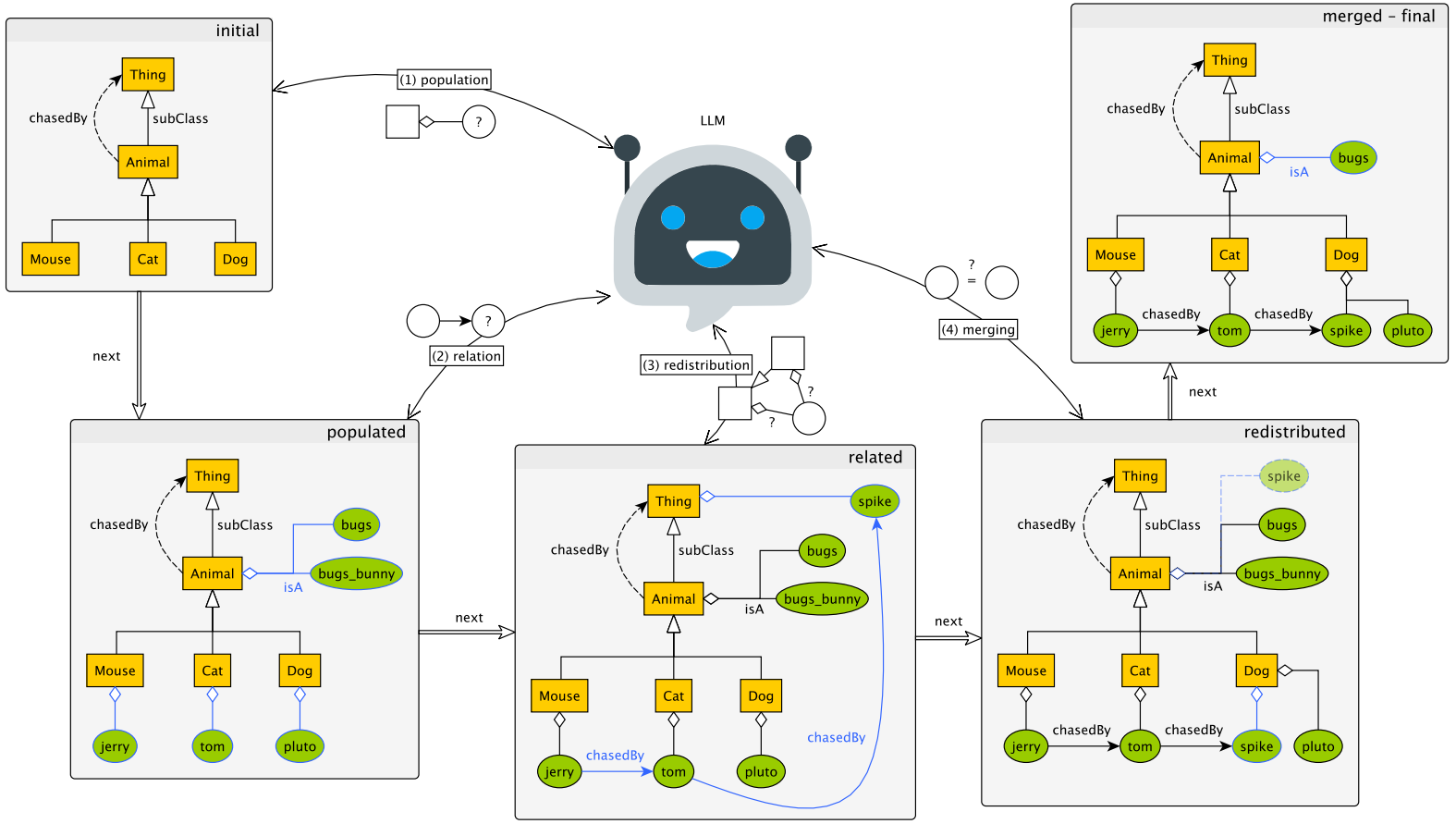


Figure 8.2: Overview of KGFiller, illustrated with a running example. The example assumes an ontology about animals, including classes such as *Cat*, *Dog*, and *Mouse*, which are subclasses of *Animal*, itself a subclass of *Thing* ($\top$). Initially, the ontology contains no instances. A property chasedBy : $Animal \times \top$ is defined, indicating that animals can be chased by other entities. The figure outlines the four phases of the KGFiller algorithm, with changes highlighted in blue.

Query templates KGFiller relies on query templates to generate questions for the LLM. Templates are text strings containing placeholders, which are replaced with actual values during execution. For example, the template $t =$"What is the capital of $\langle c \rangle$?" can be instantiated with the substitution $\sigma = \{\langle c \rangle \mapsto$ "Italy"$\}$, resulting in $t/\sigma =$ "What is the capital of Italy?".

We define four types of query templates:

- **Individual-seeking templates:** Ask for instances of a class, e.g., "Examples of ⟨class⟩?".
- **Relation-seeking templates:** Ask for individuals related to a given individual via a property, e.g., "Examples of ⟨property⟩ for ⟨individual⟩?".
- **Best-match templates:** Ask for the best class for an individual among candidates, e.g., "What class is best for ⟨individual⟩ among ⟨classes⟩?".
- **Individuals-merging templates:** Ask if two instances in a class are identical, e.g., "In ⟨class⟩, are ⟨ind1⟩ and ⟨ind2⟩ the same?".

LLM oracle We do not impose constraints on the nature of the LLM oracle or the underlying model. `KGFiller` is agnostic to the specific LLM used, as long as it supports generating textual responses from textual prompts. Both prompts and responses are assumed to be strings of arbitrary length, written in a natural language. The natural language must match the one used for naming classes and properties in the ontology. Without loss of generality, we assume this language is English. A key assumption is that the LLM oracle is pre-trained on a large corpus of textual data, which includes domain-specific knowledge relevant to the ontology.

Problem statement We formalize the ontology population problem addressed by `KGFiller` as follows:

1. A partially instantiated ontology $\mathcal{O} = \mathcal{C} \cup \mathcal{P} \cup \mathcal{I}$, where:
   - $\mathcal{C} \neq \emptyset$ is a non-empty set of class definitions,
   - $\mathcal{P} \neq \emptyset$ is a non-empty set of property definitions, and
   - $\mathcal{I}$ is a possibly empty set of individuals and their relationships.
2. A subsumption relation $\sqsubseteq$ among the classes in $\mathcal{C}$.
3. A set of query templates $\mathcal{T} = \mathcal{T}_I \cup \mathcal{T}_R \cup \mathcal{T}_B \cup \mathcal{T}_M$, where:

- $\mathcal{T}_I$ contains individual-seeking templates,
- $\mathcal{T}_R$ contains relation-seeking templates,
- $\mathcal{T}_B$ contains best-match templates, and
- $\mathcal{T}_M$ contains individuals-merging templates.

4. A trained LLM oracle $\mathcal{L}$, encapsulating domain-specific knowledge.

The goal of `KGFiller` is to generate a set of individuals and relationships $\mathcal{I}'$ such that $\mathcal{I} \subseteq \mathcal{I}'$. All new individuals and relationships in $\mathcal{I}'$ must be consistent with the class and property definitions in $\mathcal{C}$ and $\mathcal{P}$, respectively. Specifically:

- Every individual is associated with the most specific concept in $\mathcal{C}$ with respect to $\sqsubseteq$.
- For every property in $\mathcal{P}$, every individual in the property's domain is associated with one or more individuals in the property's range.

Phases To compute $\mathcal{I}'$, `KGFiller` executes four sequential phases:

1. **Population Phase:** Novel individuals are identified for each class in $\mathcal{C}$.
2. **Relation Phase:** New relationships are identified for each property in $\mathcal{P}$.
3. **Redistribution Phase:** Individuals are redistributed among the classes in $\mathcal{C}$ to ensure each individual is placed in the most specific class available.
4. **Merge Phase:** Semantic duplicates within each class in $\mathcal{C}$ are detected and merged.

The order of these phases is critical. The population phase generates individuals used in the relation phase, which may, in turn, generate additional individuals. The redistribution phase ensures all individuals are placed in the most specific class, while the merge phase reduces semantic duplication.

Ancillary functions The next section introduces several auxiliary functions used throughout the framework. The function `GetRange` (resp., `GetDomain`) retrieves the range (resp., domain) of a given property. The function `AskOracle` models queries to the LLM oracle, accepting and returning arbitrary strings. The function `ExtractBinary` (resp., `ExtractNames`) extracts binary values (resp., relevant names of individuals or concepts) from the textual responses of the LLM.

It returns a Boolean value (resp., a list of names) and accepts a string as input. Finally, the function `AddToClass` assigns an individual to a class. If the individual is already part of the class or any of its subclasses, the function performs no action.

8.2.2.1 Population phase

Algorithm 4 Populates the given ontology with novel individuals queried from an LLM oracle

Require: $\mathcal{O} = \mathcal{C} \cup \mathcal{P} \cup \mathcal{X}$: partially populated ontology
Require: $\mathcal{T}_I$: individual seeking query templates
Require: $\mathcal{L}$: LLM oracle
Require: $R \in \mathcal{C}$: root concept to be populated
Ensure: $\mathcal{X}'$ contains novel individuals, assigned to the classes in $\mathcal{C}$

function POPULATE($\mathcal{O}, \mathcal{T}_I, \mathcal{L}, R$)
  $\mathcal{X}' \leftarrow \mathcal{X}$
  for all $C \in \mathcal{C}$ s.t. $C \sqsubset R \wedge C \neq \bot$ **do**
    $\mathcal{O}' \leftarrow \mathcal{C} \cup \mathcal{P} \cup \mathcal{X}'$
    $\mathcal{X}' \leftarrow$ POPULATE($\mathcal{O}', \mathcal{T}_I, \mathcal{L}, C$)
  end for
  for all $t \in \mathcal{T}_I$ **do**
    $q \leftarrow t/\{class \mapsto R\}$
    $text \leftarrow$ ASKORACLE($\mathcal{L}, q$)
    for all i $\in$ EXTRACTNAMES($text$) **do**
      $\mathcal{O}' \leftarrow \mathcal{C} \cup \mathcal{P} \cup \mathcal{X}'$
      $\mathcal{X}' \leftarrow$ ADDTOCLASS($\mathcal{O}'$, i, R)
    end for
  end for
  return $\mathcal{X}'$
end function

The population phase is based on the POPULATE function, as defined in Algorithm 4. This function is designed to populate a partially instantiated ontology, denoted as $\mathcal{O}$, with new instances. It achieves this by querying a LLM oracle, represented as $\mathcal{L}$, using a set of instance-seeking query templates, $\mathcal{T}_I$.

The process begins with a root class $R \in \mathcal{O}$. The function then traverses the class hierarchy defined by the subsumption relation $\sqsubseteq$, following a depth-first post-order strategy. This means that all direct and indirect subclasses of R are visited before R itself, ensuring that the most specific classes are populated first. Such a strategy is crucial to guarantee that any individual generated during this phase is assigned to the most specific class available.

For each subclass $C \sqsubseteq R$, the function generates queries by replacing the placeholder *class* in each template from $\mathcal{T}_I$ with the name of C. These queries are then submitted to the LLM oracle $\mathcal{L}$, and the responses are processed to extract

the names of the individuals. The number of individuals generated per query is unbounded and depends on the capabilities of the LLM oracle $\mathcal{L}$ and other implementation-specific factors.

It is possible for the same individual to be generated by multiple queries or for multiple subclasses of R. However, this does not pose any issues, as the AddToClass function ensures that individuals are not duplicated and are always assigned to the most specific class. This behavior aligns with the post-order exploration strategy, which prioritizes the population of specific concepts over more general ones.

To populate the entire ontology $\mathcal{O}$, the function can be invoked with the top class $\top$ as the root: $\textsc{Populate}(\mathcal{O}, \mathcal{T}_I, \mathcal{L}, \top)$. This ensures that all classes in $\mathcal{O}$ are visited and populated systematically.

8.2.2.2 Relation phase

Algorithm 5 Populates the given ontology with novel relationships queried from an LLM oracle

Require: $\mathcal{O} = \mathcal{C} \cup \mathcal{P} \cup \mathcal{X}$: partially populated ontology
Require: $\mathcal{T}_R$: relation seeking query templates
Require: $\mathcal{L}$: LLM oracle
Require: $\mathtt{p} \in \mathcal{P}$: property to be populated
Ensure: $\mathcal{X}'$ contains novel relationships, involving individuals in $\mathcal{X}$

function Relate($\mathcal{O}, \mathcal{T}_R, \mathcal{L}, \mathtt{p}$)
  $\mathcal{X}' \leftarrow \mathcal{X}$
  $D \leftarrow$ GetDomain($\mathtt{p}$)
  $R \leftarrow$ GetRange($\mathtt{p}$)
  for all $(\mathtt{i} : D) \in \mathcal{X}$ **do**
    for all $t \in \mathcal{T}_R$ **do**
      $q \leftarrow t/\{individual \mapsto \mathtt{i}, property \mapsto \mathtt{p}\}$
      $text \leftarrow$ AskOracle($\mathcal{L}, q$)
      for all $\mathtt{i}' \in$ ExtractNames($text$) **do**
        $\mathcal{O}' \leftarrow \mathcal{C} \cup \mathcal{P} \cup \mathcal{X}'$
        $\mathcal{X}' \leftarrow$ AddToClass($\mathcal{O}', \mathtt{i}', R$)
        $\mathcal{X}' \leftarrow \mathcal{X}' \cup \{\mathtt{p}(\mathtt{i}, \mathtt{i}')\}$
      end for
    end for
  end for
  return $\mathcal{X}'$
end function

The relation phase utilizes the Relate function to populate a partially instantiated ontology, $\mathcal{O}$, with new relationships. These relationships link existing individuals and may introduce novel individuals, based on queries to a LLM oracle, $\mathcal{L}$, using a set of relation-seeking query templates, $\mathcal{T}_R$.

Consider a property $\mathtt{p} \in \mathcal{O}$, defined as $\mathtt{p} : D \times R$, where D and R represent the domain and range of $\mathtt{p}$, respectively. The function queries the LLM oracle, $\mathcal{L}$, to identify relationships between each individual $\mathtt{i} \in D$ and potential individuals $\mathtt{i}' \in R$ through $\mathtt{p}$.

For each individual $\mathtt{i}$ and each query template $t \in \mathcal{T}_R$, a query q is generated by substituting placeholders *property* and *individual* in t with the names of $\mathtt{p}$ and $\mathtt{i}$, respectively. The query q is submitted to the LLM oracle, and the response is processed to extract the names of individuals to be added to the ontology. Each new individual $\mathtt{i}'$ is added to the range R of $\mathtt{p}$, along with the relationship $\mathtt{p}(\mathtt{i}, \mathtt{i}')$.

To populate relationships for all properties $\mathtt{p} \in \mathcal{O}$, the function is invoked for each property as follows: $\textsc{Relate}(\mathcal{O}, \mathcal{T}_R, \mathcal{L}, \mathtt{p})$.

It is important to note that the RELATE function may generate new individuals during its execution. In some cases, the range R of the property $\mathtt{p}$ may not be the most specific class for these individuals. For example, a subclass $C \sqsubset R$ might better represent some of the generated individuals. This necessitates the redistribution phase, which ensures that all individuals are assigned to the most specific class available.

8.2.2.3 Redistribution phase

The redistribution phase utilizes the REDISTRIBUTE function, as defined in Algorithm 6. This function redistributes individuals among the classes of an instantiated ontology, $\mathcal{O}$, ensuring that each individual is assigned to the most specific class available. To achieve this, the function queries the LLM oracle, $\mathcal{L}$, using a set of best-match query templates, $\mathcal{T}_B$.

Starting from a root class $R \in \mathcal{O}$, the function recursively explores the class hierarchy defined by the subsumption relation $\sqsubseteq$. It follows a depth-first pre-order traversal strategy, visiting each class before its subclasses. This approach ensures that individuals can be reassigned to more specific classes as the traversal progresses.

For each visited class $C \sqsubseteq R$, the function evaluates whether C is the most appropriate class for all individuals currently assigned to it. If a more specific subclass of C is better suited for an individual, the function reassigns the individual

Algorithm 6 Redistributes individuals from the given ontology's classes, in such a way that each individual is assigned to the most specific class available

Require: $\mathcal{O} = \mathcal{C} \cup \mathcal{P} \cup \mathcal{X}$: partially populated ontology
Require: $\mathcal{T}_B$: relation seeking query templates
Require: $\mathcal{L}$: LLM oracle
Require: $R \in \mathcal{C}$: root concept within which redistribution should occur
Ensure: $\mathcal{X}'$ contains the different assignments of individuals to classes

1: **function** REDISTRIBUTE($\mathcal{O}, \mathcal{T}_B, \mathcal{L}, R$)
2: $\mathcal{X}' \leftarrow \mathcal{X}$
3: $\mathcal{S} \leftarrow \{S \in \mathcal{C} \mid S \sqsubset R\}$
4: **for all** $(\mathtt{i} : R) \in \mathcal{X}'$ **do**
5: $B \leftarrow R$
6: **for all** $t \in \mathcal{T}_B$ **do**
7: $q \leftarrow t/\{individual \mapsto \mathtt{i}, classes \mapsto \mathcal{S}\}$
8: $text \leftarrow$ ASKORACLE($\mathcal{L}, q$)
9: **for all** $C \in$ EXTRACTNAMES($text$) **do**
10: $B \leftarrow C$
11: **break** going to line 14
12: **end for**
13: **end for**
14: $\mathcal{O}' \leftarrow \mathcal{C} \cup \mathcal{P} \cup \mathcal{X}'$
15: $\mathcal{X}' \leftarrow$ ADDTOCLASS($\mathcal{O}', \mathtt{i}, B$)
16: **end for**
17: **for all** $C \in \mathcal{C}$ s.t. $C \sqsubset R \wedge C \neq \bot$ **do**
18: $\mathcal{O}' \leftarrow \mathcal{C} \cup \mathcal{P} \cup \mathcal{X}'$
19: $\mathcal{X}' \leftarrow$ REDISTRIBUTE($\mathcal{O}', \mathcal{T}_B, \mathcal{L}, C$)
20: **end for**
21: **return** $\mathcal{X}'$
22: **end function**

accordingly. For example, if $Cat \sqsubset Animal$ and an individual `tom` is initially assigned to *Animal*, the function reassigns `tom` to *Cat*.

To determine the best class for an individual $\texttt{i} : C$, the function generates a query for each template $t \in \mathcal{T}_B$. The placeholders *individual* and *classes* in t are replaced with the name of `i` and the concatenated names of all direct subclasses of C, respectively. The resulting query is submitted to the LLM oracle, and the response is processed to extract candidate class names. The first class in the response is selected as the best match, and `i` is reassigned to it.

Since the function is recursive, an individual may be reassigned multiple times until it reaches the most specific class available in the ontology. The depth-first pre-order strategy is crucial for this process, as it allows individuals to "move down" the class hierarchy efficiently while keeping query sizes manageable.

To redistribute all individuals in an ontology, the function can be invoked with the top class $\top$ as the root: $\textsc{Redistribute}(\mathcal{O}, \mathcal{T}_B, \mathcal{L}, \top)$. This ensures that every individual in $\mathcal{O}$ is assessed and reassigned to the most specific class, if necessary.

8.2.2.4 Merge phase

The Merge function is responsible for identifying and merging duplicated individuals in an instantiated ontology, denoted as $\mathcal{O}$. Duplicated individuals are defined as those with slightly different names but representing the same semantic entity. Such duplicates often arise as a by-product of earlier phases in the ontology population process.

For example, consider the individuals `maincoone_cat` and `maincoone`, both belonging to the class $Cat \sqsubset Animal$. Although their names differ syntactically, they refer to the same entity. To address this, the Merge function queries the LLM oracle, denoted as $\mathcal{L}$, to identify semantically similar individuals and merge them into a single entity.

The Merge function operates by scanning the ontology $\mathcal{O}$ for pairs of syntactically similar individuals. For each pair, it uses a set of merging query templates, $\mathcal{T}_M$, to query the LLM oracle. The oracle's response determines whether the individuals should be merged.

The process begins with a root class $R \in \mathcal{O}$ and recursively explores the class

Algorithm 7 Merges syntactically and semantically similar individuals from the given ontology's classes

Require: $\mathcal{O} = \mathcal{C} \cup \mathcal{P} \cup \mathcal{X}$: partially populated ontology
Require: $\mathcal{T}_M$: indivuals merging query templates
Require: $\mathcal{L}$: LLM oracle
Require: $R \in \mathcal{C}$: root concept within which merge should occur
Ensure: $\mathcal{X}'$ contains no duplicated individuals

```
1:  function MERGE(O, T_M, L, R)
2:      X' ← X
3:      for all C ∈ C s.t. C ⊏ R do
4:          D ← ∅
5:          for all i, j : C s.t. (i, j) ∈ X' × X' do
6:              if SYNTACTICALLYSIMILAR(i, j) then
7:                  D ← D ∪ {{i, j}}
8:              end if
9:          end for
10:         for all {i, j} ∈ D do
11:             for all t ∈ T_M do
12:                 q ← t/{ind_1 ↦ i, ind_2 ↦ j, class ↦ C}
13:                 text ← ASKORACLE(L, q)
14:                 if EXTRACTBINARY(text) then
15:                     X' ← MERGEINDIVIDUALS(i, j)
16:                     break going to line 11
17:                 end if
18:             end for
19:         end for
20:     end for
21: end function
```

hierarchy defined by the subsumption relation $\sqsubseteq$. For each class $C \sqsubseteq R$, the function identifies a set $\mathcal{D}$ of candidate duplicate pairs using the SyntacticallySimilar function. Each pair is then evaluated using the ExtractBinary function, which interprets the oracle's response as a Boolean value. If the response is positive, the MergeIndividuals function is invoked to merge the individuals. This involves transferring all information from one individual to the other and removing the redundant individual, resulting in an updated ontology $\mathcal{X}'$.

The merging process relies on templates in $\mathcal{T}_M$. For each candidate pair, a query is generated by replacing the placeholders *ind_1*, *ind_2*, and *class* in the template with the names of the individuals and the class C, respectively. The first template yielding a positive response is considered definitive, and further templates are not queried. To ensure deterministic behavior and avoid issues arising from the oracle's creativity, the temperature parameter is set to 0.0 during these queries.

To detect and merge all duplicated individuals in an ontology $\mathcal{O}$, the function can be invoked as follows: $\text{MERGE}(\mathcal{O}, \mathcal{T}_M, \mathcal{L}, \top)$. This ensures that all classes in $\mathcal{O}$ are systematically visited, and any semantic duplicates are resolved.

Remarks Algorithm 7 employs a post-order exploration strategy. However, the choice of exploration strategy is *not* critical for the merging phase. Alternative strategies can be used, provided they visit all direct and indirect subclasses of R.

The SyntacticallySimilar function plays a crucial role in determining whether two instances, `i` and `j`, are sufficiently similar to be considered for merging. Its implementation significantly impacts the overall performance of the Merge function.

Regarding the merging phase, one might question why entity alignment approaches [Zha+22] are not used. While effective for identifying similar instances across different KGs, these methods typically require additional domain-specific data [Che+18; Xu+19]. This is because entity alignment systems rely on ad-hoc ML models trained for the alignment task, introducing extra complexity.

In contrast, our approach leverages the same LLM oracle for both ontology population and duplicate identification. Although not entirely error-free, this method avoids additional computational overhead during the population phase. It also eliminates the need for users to provide extra domain-specific data or train additional models, simplifying the overall process.

8.2.3 Case study and experiments

This section presents a case study designed to empirically validate KGFiller.

8.2.3.1 Experimental setup

The experimental setup involves designing a non-trivial ontology, as detailed in Section 8.2.3.3. The ontology includes a class hierarchy and a set of properties, which are populated using KGFiller. We fine-tune specific query templates (Section 8.2.3.4) and utilize various LLMs from different families and technologies (Section 8.2.3.5). This process generates a set of populated ontologies, all sharing the same structure but differing in their content. The populated ontologies are analyzed and compared to evaluate the performance of KGFiller and the impact of the chosen LLM on the quality of the population process. To achieve this, we define a taxonomy of errors and a set of metrics (Section 8.2.4) to measure the quality of the generated ontologies. The evaluation is performed through a manual

inspection of the populated ontologies, examining each individual and relationship (Section 8.2.5).

Testing KGFiller on a custom ontology, rather than a publicly available one, offers several advantages. First, it allows precise control over the experiment's complexity and content. Second, it avoids potential biases where LLMs perform well due to prior exposure to the ontology during training. However, this approach lacks ground truth data, making automatic validation infeasible. Thus, manual inspection is necessary to identify errors and ensure the correctness of the algorithm's output.

8.2.3.2 Code and data

The experiments are conducted using a Python implementation of KGFiller, which is publicly available on GitHub[1]. The experimental results are also available on a dedicated repository[2], where each Git branch corresponds to a specific experiment. The `main` branch contains the initial ontology, which includes only class and property definitions without any individuals.

For reproducibility, each experiment branch includes the populated ontology and the cache of query–response pairs from the LLM. This allows others to inspect the exact queries and responses and reproduce the experiments deterministically. Additionally, the experiments are automated, with each commit in the branch corresponding to a single operation performed by KGFiller (e.g., populating a class or property). This commit history provides a detailed sequence of operations for each experiment.

8.2.3.3 Ontology

The ontology is designed in the *nutritional* domain, as required by the Expectation project [Cal+21]. In Expectation, the ontology serves as the foundation for a nutritional recommender system, which suggests recipes based on users' dietary needs and preferences [Mag+23]. Although the recommender system is outside the scope of this work, this context helps clarify the ontology design choices.

[1] `https://github.com/Chistera4-Expectation/kg-filler`
[2] `https://github.com/Chistera4-Expectation/knowledge-graphs/branches`

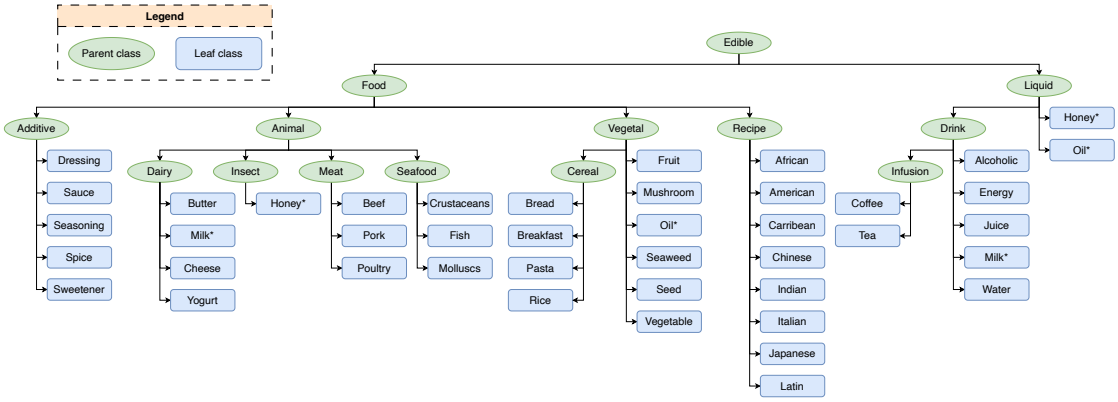


Figure 8.3: Class hierarchy of the case study ontology. Notice that the hierarchy is not really a tree, but rather a DAG. The asterisk (*) denotes classes having multiple super-classes (they are depicted once per super-class for the sake of readability).

The ontology focuses on representing *recipes* and their *ingredients*. It categorizes *edible* items into several subclasses to accommodate diverse dietary requirements and preferences. Figure 8.3 illustrates the class hierarchy, which is structured as a directed acyclic graph (DAG). The root class is *Edible*, with subclasses such as *Recipe*, which further branches into various cuisines (e.g., *Italian*, *Chinese*).

Some classes include an annotation property, fancyName, to provide a more descriptive label for the class. Another key property is ingredientOf : $Edible \times Recipe$, which links recipes to their ingredients. Initially, the ontology is a skeleton containing no individuals.

8.2.3.4 Query templates

To query the LLM oracle, we define and fine-tune specific query templates.

The set of individual-seeking templates, $\mathcal{T}_I$, is defined as: "(`instances`|`examples`) `list for class` *class*(`, names only`)?". Here, $(A|B)$ denotes two template variants, one with A and one with B, while $(A)?$ indicates optional inclusion of A.

The set of relation-seeking templates, $\mathcal{T}_R$, includes: "`ingredient list for` *individual*`, names only`". This template uses a hard-coded property name for natural phrasing.

The set of best-match templates, $\mathcal{T}_B$, is defined as: `"most adequate class for` *individual* `among:` *classes*`. concise"`.

Finally, the set of individual-merging templates, $\mathcal{T}_M$, is: `"in the` *class* `class, should instances` ind_1 `and` ind_2 `be merged together as semantic and ontologic duplicates? yes or no answer only"`.

8.2.3.5 LLM oracles

To evaluate the impact of LLM quality on KGFiller, we integrate several state-of-the-art LLMs. These include both open-source models (e.g., OpenChat, Llama) and closed-source models (e.g., GPT), as well as mixture of experts (MoE) models like Mixtral.

Open-source models are queried using the Hugging Chat API[3], a third-party library for the HuggingFace platform[4]. This API supports multiple simultaneous conversations with different LLMs. The selected open-source models include:

Llama 2: A family of open-source LLMs, with the 70 billion parameters model used in our experiments [Tou+23a].

OpenChat 3.5: A 13 billion parameters model fine-tuned from the Llama family, optimized for mixed-quality data [Wan+23].

Mistral 7B: A 7 billion parameters model, designed for high performance relative to its size [Jia+23].

Gemma: A lightweight model from Google, based on the Gemini research [Ani+23].

For MoE models, we include:

Mixtral: A sparse MoE aggregating eight 7 billion parameters models [Jia+24].

Nous Hermes: A flagship model trained on high-quality data, including GPT-4 outputs.

Closed-source models are queried using the OpenAI API[5]. The selected models include:

[3] `https://github.com/Soulter/hugging-chat-api`
[4] `https://huggingface.co`
[5] `https://openai.com/blog/openai-api`

Table 8.1: Size (number of parameters) and experiments setup of the different LLMs used in KGFiller. Values reported with * are an educated estimate – not confirmed –, as the corresponding models have not be fully disclosed to the public.

LLM	size [B]	max tokens	max retries	back-off time [s]	temperature
GPT 3.5 Turbo [Bro+20]	375*	1000	2	30	0.7
GPT 4 Turbo [Ope23]	1500*	1000	2	30	0.7
Openchat [Wan+23]	13	1000	2	30	0.1
Llama2 [Tou+23a]	70	1000	2	30	0.1
Mistral [Jia+23]	7	1000	2	30	0.1
Gemma [Ani+23]	7	1000	2	30	0.1
Mixtral [Jia+24]	56	1000	2	30	0.1
Nous Hermes	56	1000	2	30	0.1

GPT 3.5 Turbo: A widely used model with significant performance improvements [Bro+20].

GPT 4 Turbo: The latest version, reported to be a MoE [Ope23].

Table 8.1 summarizes the LLMs used, their sizes, and relevant hyperparameters. The size of the LLM significantly impacts its performance, with GPT-4 being nearly 200 times larger than the largest open-source model (Llama 2). This size difference is expected to influence the quality of the constructed ontology, as discussed in Section 8.2.5.

8.2.4 Performance metrics

Evaluating the quality of ontology construction and population processes is challenging. This is due to the interplay of various factors, such as the quality of the generated instances and the validity of the relationships. In this section, we outline the potential issues in the KGFiller framework and define corresponding quantitative performance metrics.

8.2.4.1 Types of errors

When using sub-symbolic black-box models, such as LLMs, for ontology population, several types of errors may arise. These errors depend on the quality of the oracle's output and the robustness of the post-processing procedure. LLMs are

prone to hallucinations, which can lead to unreliable suggestions. Additionally, the structure of their output is often unpredictable, making post-processing complex and error-prone. We categorize the errors as follows:

Misplacement error ($E_{\mathbf{mis}}$) This error occurs when an individual is added to the ontology but assigned to the wrong class. It is common in large ontologies with many classes. We distinguish between two cases: *(i)* the individual is assigned to a class that is too general, where a more specific subclass exists, and *(ii)* the individual is assigned to a semantically incorrect class. While both cases are treated equally in our analysis, the first represents a lack of precision, whereas the second is a more severe semantic issue.

Incorrect individual ($E_{\mathbf{ii}}$) This error arises when an individual is added to the ontology but is irrelevant to its context, despite having a meaningful name. Such errors may result from: *(i)* hallucinated responses from the LLM, suggesting out-of-scope instances, or *(ii)* incorrect post-processing of valid LLM outputs. For example, the LLM might suggest that "*dog*" is an instance of the class "*Edible meat*".

Meaningless individual ($E_{\mathbf{mi}}$) This error occurs when the name of an individual is nonsensical and should not be part of the ontology. It often results from the failure to interpret negative responses or to correctly extract valid instances from the LLM output. For instance, failing to recognize a response like "*I do not have access to such data*" as negative could lead to meaningless entries.

Class-Like individuals ($E_{\mathbf{ci}}$) These errors occur when an individual has the same or a very similar semantic meaning as the class it belongs to. This often happens due to overly broad responses from the LLM. For example, querying for "*meat*" instances might result in the LLM suggesting "*meat*" as an individual of the class "*Meat*".

Duplicate individuals ($E_{\mathbf{di}}$) Duplicate individuals are semantically identical or very similar entries in the ontology. They may arise from: *(i)* repeated suggestions

by the LLM, or *(ii)* errors in post-processing. The iterative nature of the ontology population process can also lead to duplicates. The *merge phase* of KGFiller aims to address this issue by identifying and merging duplicates, but errors may still persist.

Wrong relation ($E_{\mathbf{wr}}$) This error occurs when a relationship between two individuals is invalid. During the relation phase, KGFiller queries the LLM to populate relationships. However, hallucinated responses can lead to incorrect relationships. For example, the LLM might suggest that "*onions*" are an ingredient of "*carbonara spaghetti*," which is inconsistent with the ontology.

On subjectivity The identification of these errors involves a degree of subjectivity, particularly for E_{ii} and E_{wr}. To mitigate this, multiple evaluators independently assess the ontology, and disagreements are resolved through majority voting.

8.2.4.2 Measures

To assess the quality of the ontology population process, we define the following metrics:

1. The total number of generated individuals, TI, which evaluates the ability of KGFiller to generate diverse instances.
2. The minimum and maximum class weights, $minCW$ and $maxCW$, representing the smallest and largest number of individuals per class, respectively.
3. The total number of individuals in leaf classes, TL, which measures the specificity of the generated instances.
4. The total number of individuals affected by errors, TE, where each individual is counted once, regardless of the number of errors.
5. The relative individual error, RIE, defined as:

$$RIE = \frac{TE}{TI}. \tag{8.1}$$

6. The total number of errors for each error type, as defined in Section 8.2.4.1.

7. The total number of generated relationships, TR.
8. The relative relation error, RRE, defined as:

$$RRE = \frac{E_{\text{wr}}}{TR}. \tag{8.2}$$

Finally, we define an overall quality metric, Q, which summarizes the correctness of the ontology:

$$Q = \frac{TI - TE + TR - E_{\text{wr}}}{TI + TR}. \tag{8.3}$$

This metric ranges from 0 (all data is invalid) to 1 (all data is valid).

8.2.4.3 Quality of service

To evaluate the QoS of the LLM oracle, we consider the following metrics:

- The time required to populate the ontology, Δt, measured as the difference between the first and last query timestamps.
- The total number of queries, N, submitted to the LLM, including inconclusive ones.
- The total cost, \$, in USD, of a single KGFiller run, considering cached queries.

These metrics are influenced by factors such as network latency, response length, and the pricing scheme of the LLM provider. At the time of writing, Hugging Face models are free but rate-limited, while OpenAI models have the following costs:

GPT 3.5 Turbo: \$0.5 (input) and \$1.5 (output) per million tokens.
GPT 4 Turbo: \$10 (input) and \$30 (output) per million tokens.

8.2.5 Results

Table 8.2 summarizes the results obtained from different runs of KGFiller, supported by the various LLMs discussed in Section 8.2.3.5. Overall, the results demonstrate the feasibility of constructing ontologies by leveraging the implicit knowledge embedded in LLMs during their training. Focusing on the quality

metric Q, at least 75% of the instances and relations added to the initial empty ontology are valid for most of the evaluated LLMs. For instance, GPT 3.5 Turbo achieves 92.4% valid instances and relations, while its successor, GPT 4 Turbo, achieves a comparable performance with $Q = 89.6\%$. These findings highlight the reliability of the proposed approach.

The analysis of error types during the ontology population phase provides additional insights. Most errors are related to the relative positioning of added instances rather than the inclusion of nonsensical entries. This observation reinforces the potential of LLMs to extract structured and precise knowledge for specific domains, such as the food domain in this case study.

Different models excel in different performance metrics. For example, the *Mistral* model constructs one of the largest ontologies, adding 1176 instances and 1211 relations. However, the large size of the ontology introduces quality issues, resulting in a high relative error rate. In contrast, GPT-based models achieve lower relative error rates, with GPT 3.5 Turbo achieving a relative individual error of 0.0861 and a relative relation error of 0.0693. Despite this, GPT-based models tend to produce smaller ontologies, with the GPT 3.5 Turbo version containing nearly half the instances of other models.

When minimizing erroneous instances and relations, GPT-based solutions, particularly GPT 3.5, emerge as the most effective. The significant size difference between GPT-based models and their counterparts provides an advantage in handling complex tasks, such as classifying instances into semantic classes or listing recipe ingredients without hallucinations or inaccuracies. Ontologies constructed using the GPT family exhibit the fewest critical mistakes or hallucinations (see Section 8.2.5.1).

Nevertheless, the results also indicate that closed-source models are not the only viable option. Some open-source models achieve comparable performance levels. For instance, the *Llama 2* model constructs a smaller but effective ontology, maintaining low counts of incorrect (E_{ii}) and meaningless (E_{mi}) individuals, thereby demonstrating its potential.

Additionally, the *Nous Hermes* model, leveraging the MoE process, constructs a large ontology—nearly twice the size of the GPT 3.5 version. It achieves nearly 90% correctness for added individuals, surpassing GPT 4 Turbo, with an overall

quality metric of $Q = 85.1\%$. These results confirm the correlation between the dimensionality of LLMs and the quality of the constructed ontology, as *Llama 2* and *Nous Hermes* are among the largest open-source models (see Table 8.1).

A strong proportionality is observed between the relative individual error (RIE) and the relative relation error (RRE). Models achieving low RIE also tend to achieve low RRE. This suggests that the ability of LLMs to suggest instances for specific classes or classify random instances correctly is proportional to their ability to establish valid relationships between instances without proposing meaningless associations. This finding underscores the general capability of LLMs to handle both instances and relationships in ontology construction.

QoS-related remarks Table 8.3 summarizes the QoS measurements from the experiments. GPT-based models generally exhibit faster execution times (Δt), primarily due to the looser rate limitations applied by OpenAI for paying users. However, this comes at the cost of moderate expenses in USD ($). In contrast, models accessible via the Hugging Face API are free to use but subject to stricter rate limitations, resulting in longer execution times for KGFiller.

The total number of queries (N) varies significantly across runs, depending on the LLM model. Given that all models were subject to the same length limitations, this variability likely stems from differences in the verbosity of the LLMs's responses, which depend on their training data and architecture.

8.2.5.1 Mistakes and hallucinations examples

This section presents and analyzes notable examples of mistakes and hallucinations encountered during the investigation. The examples are ranked from least to most significant to provide insights into the shortcomings of KGFiller.

Minor mistakes Most models, particularly smaller open-source ones, struggle with identifying the correct spices for recipes. These errors are not considered hallucinations, as they remain within the recipe context and are not blatantly incorrect. For instance, the concept of a recipe varies based on personal preferences. Such mistakes, including misclassifying sauces as toppings or vice versa, are minor

Table 8.2: Performance of KGFiller over different state-of-the-art LLMs. For each performance metric (row) we denote with † and ‡ the best and second best performing model respectively. For presentation purposes, the columns with GPT models omit the Turbo suffix and Nous-Hermes is abbreviated as NH.

Metric	GPT 3.5	GPT 4	Openchat	Llama2	Mistral	Gemma	Mixtral	NH
TI	511	735	997	665	1176†	433	841	1121‡
minCW	4	5	9	6	11†	1	3	11†
maxCW	40	56	105	55	110	40	111‡	121†
TL	495	727	970	609	1113†	407	819	1082‡
TE	44†	91‡	244	119	255	107	228	123
RIE	0.0861†	0.1238	0.2447	0.1789	0.2168	0.2471	0.2711	0.1097‡
E_{mis}	37†	40‡	105	84	133	70	94	59
E_{ii}	1†	5	65	8	14	2‡	22	12
E_{mi}	0†	18	11	4‡	60	21	78	11
E_{ci}	13†	19	23	21	32	16‡	22	28
E_{di}	3†	8	32	7	72	3†	15	11
TR	736	1061	1329†	1062	1211	989	960	1222‡
E_{wr}	51†	96‡	440	197	284	290	174	227
RRE	0.0693†	0.0905‡	0.3311	0.1855	0.2345	0.2932	0.1813	0.1858
Q	0.924†	0.896‡	0.706	0.817	0.774	0.721	0.777	0.851

Table 8.3: QoS measurements (duration, number of queries, cost) of KGFiller over different state-of-the-art LLMs. Each line describes the QoS of the experiment corresponding to the same line in Table 8.2.

LLM	Δt	N	$
GPT 3.5 Turbo [Bro+20]	15m 39s	1236	0.06
GPT 4 Turbo [Ope23]	58m 43s	2414	2.16
Openchat [Wan+23]	11h 50m 25s	4710	n.a.
Llama2 [Tou+23a]	5h 8m28s	1990	n.a.
Mistral [Jia+23]	2h 19m 40s	907	n.a.
Gemma [Ani+23]	4h 55m 2s	1141	n.a.
Mixtral [Jia+24]	12h 51m 2s	2620	n.a.
Nous Hermes	14h 40m 3s	6083	n.a.

and can be easily corrected after the ontology is populated. An example of a fuzzy classification is whether *peanut butter* qualifies as *butter*.

Mistakes due to ontology structure The completeness of the initial empty ontology skeleton significantly impacts the classification of fuzzy instances. Uncommon suggestions or peculiar items may not fit neatly into predefined classes. For example, the *Nous Hermes* model classified *crocodile* and *alligator* as instances of the *poultry-derived food* class. While these are valid food items in some contexts, their classification highlights the need for a rigorous and complete ontology structure. The boundary between reasonable and unreasonable suggestions is often tight, requiring careful consideration during ontology population.

LLM hallucinations LLMs are prone to hallucinations, which can lead to the inclusion of irrelevant or factually incorrect instances in the ontology. For example, the *Mistral* model suggested *chemotherapy infusion* as an instance of the *infusion* class in the food domain. Although grammatically and semantically valid, this suggestion is contextually incorrect. Similarly, the *Mixtral* model confidently classified *pizza* as an American recipe, despite its Italian origin. Addressing such issues is challenging, as verifying the validity of all suggestions would require multiple queries, significantly increasing inefficiency.

Worrying hallucinations In some cases, hallucinations result in factually incorrect but contextually plausible suggestions, posing a significant challenge. For instance, the *Gemma* model suggested *Amanita muscaria*, a poisonous mushroom, as an instance of the *edible mushrooms* class. While the suggestion aligns with the context, it introduces dangerous misinformation. Interestingly, the *Gemma* model correctly identified *Amanita muscaria* as poisonous when queried separately. Such cases highlight the complexity of detecting and addressing hallucinations in ontology construction.

8.2.6 Comparison with related works

This section compares KGFiller with other methods for LLM-augmented KG construction, as surveyed in Section 8.2.1. The comparison is conducted in two

Table 8.4: Feature-based comparison between KGFiller and state-of-the-art KG construction methods. The arrow (→) denotes the best-featured method from the literature, namely Harvest, which we compare with KGFiller under a performance-based perspective.

Method	Document Free *(F1)*	Training Free *(F2)*	Construction *(F3)*	Prompt Templating *(F4)*	Consistency *(F5)*
[Lua+18]	✗	✗	✓	✗	✗
[Bos+19]	✗	✗	✓	✗	✗
[Kum+20]	✗	✗	✓	✗	✗
[CJK21]	✗	✗	✗	✗	✗
[Wan+21a]	✗	✗	✗	✗	✗
[Cao+21]	✗	✗	✓	✗	✗
[MDD21]	✗	✗	✓	✗	✗
[SKG22]	✗	✗	✗	✗	✗
[Che+22]	✗	✗	✗	✗	✗
[Xie+22]	✗	✗	✗	✗	✗
[Ayo+22]	✗	✓	✓	✗	✗
[She+22]	✗	✗	✗	✗	✗
[Han+23]	✗	✓	✓	✓	✗
→[Hao+23]	✓	✓	✓	✓	✗
[Li+24]	✓	✗	✓	✓	✗
[FC24]	✓	✗	✓	✓	✗
Ours (KGFiller)	✓	✓	✓	✓	✓

phases. First, we perform a feature-based analysis to identify key characteristics of an ideal ontology population method. Next, we filter out methods that differ significantly in their approach, retaining only those suitable for a performance-based comparison with KGFiller.

8.2.6.1 Feature-based comparison

An ideal ontology population method should satisfy the following features: *(F1)* It should adopt a *document-free* approach, relying solely on the knowledge embedded in the LLM oracle during training, without requiring textual input for instance or relation extraction. *(F2)* It should be *training-free*, avoiding the need for additional training or fine-tuning of the LLM oracle. *(F3)* It should support the *construction* of the entire ontology, encompassing both entities and relationships.

(F4) It should allow user-defined *prompt templates* to customize LLM queries for the specific domain. *(F5)* It must ensure *consistency* in the generated ontology, preserving its structural integrity throughout the population process. Table 8.4 summarizes the feature-based comparison of KGFiller and other methods discussed in Section 8.2.1. Most surveyed approaches either depend on user-provided textual data or require costly model training. Additionally, they often fail to guarantee the structural integrity of the generated ontology, as they primarily target KGs without considering ontological constraints. In contrast, KGFiller satisfies all the above features, offering a document- and training-free approach that ensures consistency through prompt templating and the use of the LLM as an oracle. Harvest [Hao+23] emerges as the second-best approach in terms of feature satisfaction.

Given this, Harvest is the only method suitable for a fair performance-based comparison with KGFiller. The next section evaluates their empirical performance.

8.2.6.2 Performance-based comparison

The performance comparison focuses on populating the ontology schema described in Section 8.2.3.3. We evaluate Harvest using the same metrics defined in Section 7.3.4.1, replicating the experiments on the LLM families tested in the original paper [Hao+23]. The selected models include:

RoBERTa base: A 12-layer, 768-hidden, 12-heads model with 125M parameters [Liu+19].

RoBERTa large: A larger version with 24 layers, 1024 hidden dimensions, 16 heads, and 355M parameters.

BERT large cased: A 24-layer, 1024-hidden, 16-heads model with 340M parameters, trained on cased English text [Dev+19].

The terms *heads* and *hidden* refer to the number of attention heads and the hidden dimension of each transformer block, respectively[6].

[6] `https://huggingface.co/transformers/v3.3.1/pretrained_models.html`

Table 8.5: Performance of Harvest [Hao+23] over different state-of-the-art pre-trained models. Each row is a metric, and each column is a model.

Metric	RoBERTa base	RoBERTa large	BERT large cased
TI	812	580	1086
$minCW$	2	0	22
$maxCW$	36	36	61
TL	682	867	942
TE	617	440	909
RIE	0.7599	0.7586	0.8370
E_{mis}	391	479	734
E_{ii}	118	126	153
E_{mi}	84	54	45
E_{ci}	70	30	97
E_{di}	20	25	125
TR	40	55	51
E_{wr}	25	27	37
RRE	0.6250	0.4909	0.7255
Q	0.2465	0.2646	0.1680

The results of these experiments are presented in Table 8.5. The source code for the comparison is publicly available[7]. To facilitate comparison, Figure 8.4 highlights the best-performing runs of both KGFiller and Harvest. As shown, KGFiller outperforms Harvest across all metrics. The quality metric of the generated ontology is up to five times higher with KGFiller, while the relative individual error (RIE) and relative relation error (RRE) are up to ten times lower.

8.2.7 Discussion and SWOT analysis

To the best of our knowledge, this work introduces a novel methodology for ontology population, enabling the generation of structured knowledge tailored to specific domains or schemas. This section provides a SWOT analysis, highlighting the *Strengths*, *Weaknesses*, *Opportunities*, and *Threats* of KGFiller. The discussion transitions from technical aspects to strategic considerations, including potential applications, limitations, and future directions.

[7] `https://github.com/Chistera4-Expectation/experiments-knowledge-harvest-from-llm`

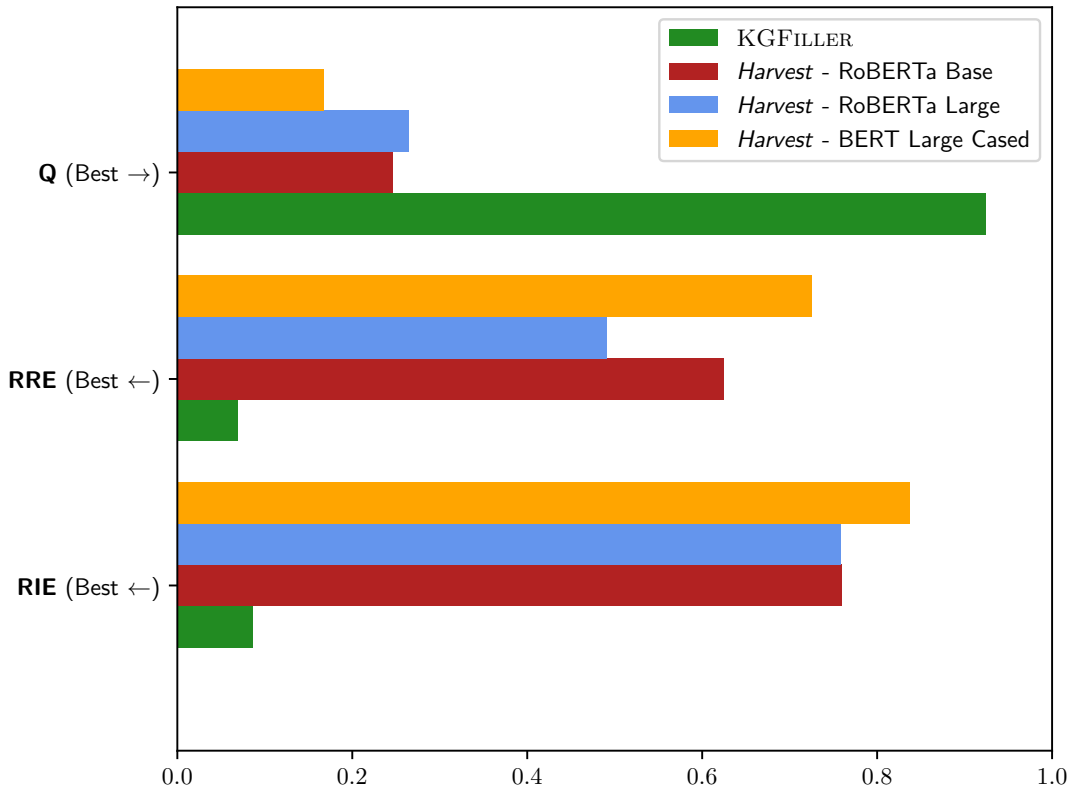


Figure 8.4: Performance comparison of Harvest [Hao+23] and KGFiller for populating the food ontology (Section 8.2.3.3). The best-performing closed-source and open-source LLM models are considered: GPT 3.5 Turbo and Nous Hermes, respectively.

8.2.7.1 Strengths

Automation KGFiller leverages LLMs trained on extensive general-knowledge datasets. This eliminates the need for large *corpora* of textual documents and makes it the first fully automatic ontology population method. No prior domain knowledge representation is required from the user.

Generality LLMs enable KGFiller to operate across virtually any domain, irrespective of the availability of domain-specific data. Unlike traditional methods, which rely on potentially incomplete or biased document collections, KGFiller benefits from the broad and diverse knowledge embedded in LLMs.

Incrementality KGFiller supports both ontology population and refinement. It can be applied iteratively to enrich existing ontologies, allowing for incremental improvements over multiple runs. This flexibility contrasts with traditional methods, which are often constrained by predefined input datasets.

Future potential As LLMs continue to evolve, their emergent capabilities are expected to enhance the quality of ontology population. Future advancements may enable KGFiller to extract knowledge beyond the explicit content of training data.

8.2.7.2 Weaknesses

Completeness KGFiller does not guarantee completeness in the populated ontology. Instances or relations may be missing, as the method is not designed to extract all knowledge from the LLM. This limitation is inherent to any (semi-)automatic data-driven approach.

Sampling bias The reliance on LLMs trained on web data introduces potential sampling biases. Underrepresented topics on the web may result in unbalanced or biased ontologies. Further research is needed to mitigate these biases in LLMs.

Correctness KGFiller cannot ensure the correctness of extracted information. LLMs are prone to hallucinations, which may lead to the inclusion of false or misleading data in the ontology [Zha+23]. Careful validation of the populated ontology is necessary to address this limitation.

8.2.7.3 Opportunities

Cold Start KGFiller addresses the cold start problem in knowledge-based systems by providing a quick and cost-effective way to generate domain-specific data. This capability is particularly valuable in early-stage projects where data availability is limited.

Automatic Hierarchy Construction Extensions of KGFiller could enable the automatic construction of ontology schemas, a task often referred to as *hierarchy construction* [Fun+23]. Such advancements could automate the entire ontology creation process, combining schema construction with population.

Zero-shot ontology creation Building on the idea of hierarchy construction, KGFiller could be adapted to create ontologies from scratch. This would involve generating both the schema and instances based on a natural-language description of the domain.

Procedural knowledge extraction KGFiller could be extended to extract procedural knowledge, representing structured steps to achieve specific goals. This application has potential in fields like automated planning and MASs, where agents require executable plans to perform tasks.

8.2.7.4 Threats

Dependence on LLMs The effectiveness of KGFiller is tied to the quality and capabilities of the underlying LLMs. Limitations in LLMs, such as biases or inaccuracies, directly impact the quality of the populated ontology.

Cost and efficiency The computational and financial costs of querying LLMs may pose challenges, especially for large-scale ontology population tasks. Optimizing the process to balance cost and quality remains an open research question.

Ethical concerns The use of LLMs trained on web data raises ethical concerns, including data privacy and the propagation of biases. Addressing these issues is critical to ensure responsible deployment of KGFiller.

8.2.8 Conclusion

Ontologies provide a structured framework to define concepts, relationships, and properties within a specific domain. This ensures an unambiguous and comprehensible representation of domain-specific knowledge. However, the process of ontology population is often meticulous, time-consuming, and prone to human errors and biases.

In this work, we hypothesize that LLMs[8] embed a significant amount of domain-specific knowledge. This knowledge is acquired during their domain-agnostic training on vast datasets collected from the Web. Based on this hypothesis, we propose KGFiller, a novel approach for automatic ontology population leveraging LLMs.

The proposed method starts with: *(i)* an initial schema consisting of interrelated classes and properties, and *(ii)* a set of query templates. KGFiller queries the LLM multiple times to generate instances for classes, relationships, and properties based on its responses. Additional queries are performed to refine the ontology, ensuring balanced class hierarchies and avoiding duplicate entries.

The validity of KGFiller is demonstrated through a case study in the food domain. This domain requires the population of an ontology with food ingredients, categorized across multiple classes, and recipes. We evaluate KGFiller using various state-of-the-art LLMs, including open-source, closed-source, and MoEs. The evaluation employs a range of metrics to assess the quality of the constructed ontology.

Our results show that KGFiller reliably populates ontologies across different LLMs. For instance, the GPT 3.5 model achieves accuracies of up to 0.91 for

[8]Defined in `my_acronyms.sty`

instances and 0.93 for relations. As expected, larger LLMs generally perform better. We also observe a correlation between erroneous instances and misplaced relations, highlighting areas for improvement.

Finally, we analyze the severity of errors, emphasizing how hallucinations from LLMs can negatively impact the ontology population process. Despite these challenges, the results are promising. They demonstrate that the complex task of ontology construction can be partially automated, requiring only minor post-processing efforts.

8.3 Actively learning $\mathcal{EL}$ terminologies from LLMs

We now present the paper "Actively Learning $\mathcal{EL}$ Terminologies from Large Language Models", accepted at the 28th European Conference on Artificial Intelligence (ECAI 2025)[9]. In active learning, a *learner* attempts to learn from a *teacher* by posing questions. The questions made by the learner are called *membership queries* and are answered with 'yes' or 'no'. This kind of query is often studied as part of a communication protocol that also includes *equivalence queries*. Intuitively, equivalence queries ask whether the idea of the learner about the knowledge of the teacher is correct or not. If not, then the teacher should provide a counterexample showing the difference (or equivalently, with 'true' or 'false'). In this work, we consider the teacher as a LLM and study the case in which knowledge is expressed as an $\mathcal{EL}$ terminology. Membership queries ask whether concept inclusions are true or not. Equivalence queries are simulated by a sample with concept inclusions labelled as positive or negative. We present a non-trivial extension of the ExactLearner tool to extract $\mathcal{EL}$ terminologies from LLMs. The ExactLearner was originally designed to run with a synthetic teacher that answers membership queries together with equivalence queries. Given the relevant symbols as input (e.g., algae, plant, etc.), the tool tries to find how these symbols should be logically connected by posing questions to LLMs. To evaluate the approach, we present performance results of the ExactLearner in the task of reconstructing existing $\mathcal{EL}$ terminologies.

[9] `https://ecai2025.org/`

8.3.1 Introduction

LLMs have accumulated vast amounts of information and significantly improved their question-answering capabilities. This has led to the development of methods to interact with and extract knowledge from these models. Prompts to LLMs range from general knowledge questions, such as definitions and historical events, to domain-specific queries, such as scientific facts in health and medicine. Recent efforts have focused on automating the extraction of knowledge from LLMs, often representing this knowledge in the form of an *ontology* [Cau+24; Cia+25; Dur+24; Fun+23; SW24]. Ontologies, a cornerstone of KR, provide a structured representation of the conceptual knowledge within a domain. The most widely used formalism for specifying ontologies is *description logic* [Baa+17]. In life sciences, ontologies are extensively used for tasks such as data integration, information retrieval, and automated reasoning [Noy+08].

A key question arises: *What can we learn from LLMs?* Moreover, given that LLMs can produce incorrect or inconsistent information, is there an automated way to identify such inaccuracies? To address these questions, this work explores an active learning approach to construct *description logic ontologies* from LLMs. Our approach is grounded in Angluin's exact learning framework [Ang87], where a *learner* acquires knowledge from a *teacher* by posing queries. This method, known as *active learning* [Ang92], involves two types of queries: *membership queries* and *equivalence queries*. Here, we consider the LLM as the teacher and aim to learn ontologies formulated in the $\mathcal{EL}$ description logic [BBL05].

Exact learning of lightweight description logic ontologies has been studied extensively for theoretical purposes [Fun+19; FJL21; KOW16; Kon+17; OPM20]. However, a major challenge in applying these algorithms is finding a suitable teacher capable of answering the required queries. The only known implementation for exact learning of $\mathcal{EL}$ ontologies is the *ExactLearner* tool [DKO18]. This tool learns $\mathcal{EL}$ terminologies[10] from a synthetic teacher that can answer membership and equivalence queries. Membership queries verify whether a given concept inclusion is true or false, while equivalence queries check whether the learner's hypothesis matches the teacher's knowledge. If the hypothesis is incorrect, the

[10] An $\mathcal{EL}$ terminology is an ontology with specific syntactic restrictions (see Section 8.3.2)

teacher provides a counterexample.

The synthetic teacher in *ExactLearner* was designed to test the tool using ontologies from the Oxford Ontology Repository[11]. Given a set of relevant symbols, referred to as the *vocabulary* or *signature* (e.g., algae, plant), the tool determines how these symbols are logically connected by posing queries to the teacher.

In this work, we extend the *ExactLearner* tool to interact with LLMs as teachers, naming the extended version *ExactLearner+LLM*. We use the Manchester OWL Syntax [Hor+06] and a custom parsing function to translate concept inclusions from description logic into natural language. While membership queries can be directly mapped to a question-answering setting with LLMs, equivalence queries are more challenging [WGY24; Blu+24]. To address this, we simulate equivalence queries using a sample of randomly generated concept inclusions classified as true or false by the LLM. This approach aligns with the theoretical analysis of Angluin's framework within the probably approximately correct (PAC) learning paradigm [Ang87].

To evaluate our approach, we conducted experiments using *ExactLearner+LLM* with open-source LLMs. The tool was provided with a signature and tasked with reconstructing existing ontologies. We tested it on small ontologies from the original *ExactLearner* benchmark and modules extracted from the GALEN ontology [Ala96], which contains medical terms. Our results also examine whether there is statistical evidence of correlation between the concept inclusions extracted by *ExactLearner+LLM* and those in the original ontologies created by human engineers.

8.3.2 Background and related work

In this section, we define the concept of an $\mathcal{EL}$ terminology, elaborate on the exact and PAC learning frameworks, and describe the algorithm implemented in the *ExactLearner* tool [DKO18], which is designed to learn $\mathcal{EL}$ terminologies from a teacher capable of answering membership and equivalence queries.

[11] `https://www.cs.ox.ac.uk/isg/ontologies/`

The $\mathcal{EL}$ Description Logic Let $\mathsf{N_C}$ and $\mathsf{N_R}$ be countably infinite and disjoint sets of *concept names* and *role names*, respectively. In some cases, we may consider $\mathsf{N_C}$ and $\mathsf{N_R}$ as finite sets, depending on the context. An $\mathcal{EL}$ *concept* is defined as $C, D \coloneqq C \sqcap D \mid \exists r.C \mid A \mid \top$, where $A \in \mathsf{N_C}$ and $r \in \mathsf{N_R}$. An $\mathcal{EL}$ *concept inclusion* is an expression of the form $C \sqsubseteq D$, where C and D are $\mathcal{EL}$ concepts. An $\mathcal{EL}$ *equivalence* is expressed as $C \equiv D$, which is equivalent to $C \sqsubseteq D$ and $D \sqsubseteq C$. An $\mathcal{EL}$ *ontology* is a finite set of $\mathcal{EL}$ concept inclusions $C \sqsubseteq D$. For a given ontology $\mathcal{O}$, we denote the sets of concept names and role names occurring in $\mathcal{O}$ as $\mathsf{N_C}(\mathcal{O})$ and $\mathsf{N_R}(\mathcal{O})$, respectively. These sets are also referred to as the *signature* of $\mathcal{O}$. If, for all concept inclusions $C \sqsubseteq D$ in $\mathcal{O}$, at least one of C or D belongs to $\mathsf{N_C}$, and there is at most one inclusion of the form $A \sqsubseteq C$ for every $A \in \mathsf{N_C}$, then $\mathcal{O}$ is called a *terminology*[12]. The semantics of $\mathcal{EL}$ is defined using interpretations, as described in [Baa+17]. Given an ontology $\mathcal{O}$ and a concept inclusion $C \sqsubseteq D$, we say that $\mathcal{O}$ *entails* $C \sqsubseteq D$, denoted as $\mathcal{O} \models C \sqsubseteq D$, if every interpretation satisfying $\mathcal{O}$ also satisfies $C \sqsubseteq D$. Two ontologies $\mathcal{O}$ and $\mathcal{H}$ are *equivalent*, denoted as $\mathcal{O} \equiv \mathcal{H}$, if they are satisfied by the same interpretations. This means that, for all concept inclusions $C \sqsubseteq D$, $\mathcal{O} \models C \sqsubseteq D \iff \mathcal{H} \models C \sqsubseteq D$.

Exact learning The exact learning framework, introduced by Angluin [Ang87], involves a *learner* attempting to acquire a target concept, which in this work is an $\mathcal{EL}$ terminology, from a *teacher*. The communication protocol between the learner and the teacher includes two types of queries:

- *Membership queries*, where the learner asks whether a specific statement is true or false.
- *Equivalence queries*, where the learner proposes a hypothesis and receives either confirmation that the hypothesis matches the target or a counterexample highlighting the difference [Ang87].

In our context, membership queries verify whether an $\mathcal{EL}$ concept inclusion $C \sqsubseteq D$ is true or false. For example, the learner may ask whether $\mathsf{Algae} \sqsubseteq \mathsf{Plant}$ holds true.

[12]This definition is slightly more flexible than the standard one in the description logic literature [Baa+17].

Simulating equivalence queries To simulate equivalence queries, we adopt a *sampling* strategy. Instead of directly asking the LLM to evaluate the hypothesis and provide a counterexample, we generate a random set of $\mathcal{EL}$ concept inclusions and query the LLM to classify them as true or false. We then verify whether the learner's hypothesis is consistent with the LLM's classification. Consistency is achieved if all true concept inclusions are logical consequences of the hypothesis and all false inclusions are not. If the hypothesis is consistent, the learning process terminates. Otherwise, a counterexample is identified, and the process continues as if the LLM had returned a counterexample in response to an equivalence query. This approach has been used in prior work to simulate equivalence queries [WGY24; Blu+24]. Under certain conditions, this strategy provides theoretical guarantees within the PAC learning framework [Ang87; Val84]. In PAC learning, the required sample size is determined by the hypothesis space $\mathbf{H}$ (in this case, $\mathcal{EL}$ terminologies of a specific size and structure defined by a given signature) and two parameters, ϵ and δ, representing the error tolerance and confidence level, respectively. According to PAC theory [SB14, Cor. 2.3], the sample size must be at least $\ln(|\mathbf{H}|/\delta)/\epsilon$ for finite $\mathbf{H}$.

Algorithm 8 The learning algorithm for $\mathcal{EL}$ [DKO18]

Input: $\mathsf{N_C}(\mathcal{O}) \cup \mathsf{N_R}(\mathcal{O})$ and access to a teacher that can answer membership and equivalence queries (or just membership queries if equivalence queries are simulated in a PAC setting).
Output: An $\mathcal{EL}$ terminology $\mathcal{H}$ such that $\mathcal{O} \equiv \mathcal{H}$.

Initialize $\mathcal{H} \leftarrow \{A \sqsubseteq B \mid \mathcal{O} \models A \sqsubseteq B, A, B \in \mathsf{N_C}(\mathcal{O})\}$
while $\mathcal{H} \not\equiv \mathcal{O}$ **do**
    Let $C \sqsubseteq D$ be a positive counterexample for $\mathcal{O}$ relative to $\mathcal{H}$
    Compute $C' \sqsubseteq D'$ with C' or D' in $\mathsf{N_C}(\mathcal{O})$
    if $C' \in \mathsf{N_C}(\mathcal{O})$ **then**
        Compute a right $\mathcal{O}$-essential α from $C' \sqsubseteq D' \sqcap \bigsqcap_{C' \sqsubseteq F' \in \mathcal{H}} F'$
    else
        Compute a left $\mathcal{O}$-essential α from $C' \sqsubseteq D'$
    end if
    Add α to $\mathcal{H}$
end while
return $\mathcal{H}$

The algorithm for $\mathcal{EL}$ terminologies The *ExactLearner* tool [DKO18] is designed to learn $\mathcal{EL}$ terminologies by interacting with a synthetic teacher that answers membership and equivalence queries. The teacher's responses are logically consistent with a target $\mathcal{EL}$ terminology, and the communication is conducted in

OWL rather than natural language. The main steps of the learning procedure are outlined in Algorithm 8. The algorithm performs equivalence queries in line 4 and membership queries in lines 6, 8, and 10. In line 6, a counterexample $C' \sqsubseteq D'$ is extracted from $C \sqsubseteq D$, ensuring that one of the sides belongs to $\mathsf{N_C}(\mathcal{O})$. This step is guaranteed to find a counterexample because $\mathcal{O}$ is a terminology [DKO18]. The algorithm applies various operations using membership queries, including *concept saturation for* $\mathcal{O}$, *sibling merging*, *decomposition on the right*, and *decomposition on the left*. Additionally, two heuristics, *desaturation* and *branching*, are employed. These operations aim to maximize the informativeness of concept inclusions while minimizing their size. The resulting concept inclusions, referred to as $\mathcal{O}$-essential, are added to the hypothesis in line 12.

8.3.3 LLMs as teachers

LLMs present significant opportunities for acquiring knowledge in the form of ontologies. However, several challenges must be addressed to effectively employ them as teachers in an active learning framework. The main challenges are as follows:

- Membership queries in the *ExactLearner* tool are expressed in OWL, whereas LLMs are designed to process natural language inputs.
- Responses from LLMs may deviate from the expected format, even when explicitly instructed to answer with 'yes' or 'no'.
- Queries generated by the *ExactLearner* can be lengthy and complex, involving multiple logical operators, which may hinder the ability of LLMs to provide accurate answers.
- Even when responses are concise and well-formatted, they may be logically inconsistent with previous answers or incorrect, and they may vary if the same query is repeated.
- The expressivity of the ontology language used ($\mathcal{EL}$) limits the scope of the knowledge that can be extracted.

Below, we discuss how these challenges are addressed.

Input format The format of the queries plays a crucial role in ensuring effective communication with LLMs. We use both the Manchester OWL syntax [Hor+06] and controlled natural language to standardize the queries. Queries in Manchester OWL syntax take the form `[A] SubClassOf [B]`. In natural language, these are translated into questions such as:

"Can `[A]` be considered a subcategory of `[B]`? Answer only with yes or no."

Here, `[A]` and `[B]` are placeholders for the left-hand and right-hand sides of the concept inclusion. This approach aligns with the third formulation proposed by Funk et al. [Fun+23], which is well-suited to our setting. The translation of OWL concept expressions into natural language is applied recursively:

1. Expressions of the form `[A] and [B]` are replaced with "`[A]` that is also `[B]`".
2. Expressions of the form `[r] some [A]`, where `[r]` is a role name, are replaced with "something that `[r] [A]`".

Example 1 The concept inclusion `Person SubClassOf Human` is translated into: "Can Person be considered a subcategory of Human?" Similarly, `Carnivore SubClassOf Animal and eats some Meat` becomes: "Can Carnivore be considered a subcategory of Animal that is also something that eats Meat?" Finally, `(Bird and Fish) SubClassOf Animal` is translated to: "Can Bird that is also Fish be considered a subcategory of Animal?"

Unexpected responses LLMs may return arbitrary responses, even when the expected answer is a single word such as 'true' or 'false'. To mitigate this, we explicitly instruct the LLM to answer with 'true' or 'false' by appending this requirement to the query. Alternatively, we use system prompts or adjust hyperparameters, such as temperature, to guide the model's responses. Despite these precautions, unexpected responses may still occur:

1. The response may include additional text beyond 'true' or 'false'.
2. Both 'true' and 'false' may appear in the response.
3. Neither 'true' nor 'false' may be present in the response.

To address these issues, we limit the maximum number of tokens in the response. If the response does not contain 'true' or 'false', we treat it as 'false'.

Length of queries Long and complex queries can lead to biased responses from LLMs, such as always answering 'true'. To address this, we modify the *ExactLearner* tool as follows:

- *Simplification:* Concept expressions are simplified by removing redundant concept names. For example, if $A \sqcap B$ is a concept expression and $A \sqsubseteq B$ is entailed by the hypothesis, B is omitted.
- *Splitting:* Long queries with multiple conjuncts are split into smaller queries. For instance, instead of asking whether $C \sqsubseteq D_1 \sqcap \ldots \sqcap D_n$, we ask whether $C \sqsubseteq D_1$, $C \sqsubseteq D_2$, and so on.

These modifications reduce the likelihood of biased responses and improve the accuracy of the LLM.

Correctness and logical consistency Responses from LLMs may not always reflect the truth or be logically consistent with an $\mathcal{EL}$ ontology. Errors can occur in the following cases:

1. A false inclusion $C \sqsubseteq D$ is incorrectly answered as 'true'.
2. A true inclusion $C \sqsubseteq D$ is incorrectly answered as 'false'.
3. The LLM answers 'true' for all inclusions in an ontology $\mathcal{O}$, but answers 'false' for a logical consequence of $\mathcal{O}$.

To handle logical inconsistencies, we compute the *closure* under logical consequence [Blu+24]. Additionally, we use a caching mechanism to store the first response to each query, ensuring consistency across repeated queries.

Expressivity The algorithm focuses on extracting $\mathcal{EL}$ terminologies, which may not fully capture knowledge better expressed in more expressive languages. However, $\mathcal{EL}$ is widely used in real-world ontologies, particularly in the medical domain [Ala96; Noy+08]. Its polynomial-time reasoning complexity [BBL05] and the availability of efficient reasoning tools [KKS14] make it a practical choice for ontology construction.

8.3.4 Experiments

The code for the experiments is publicly available on GitHub[13]. All experiments were conducted on a workstation equipped with two NVIDIA Tesla V100S-PCIE-32GB GPUs and dual Intel© Xeon© Gold 6226R CPUs, providing a total of 64 threads. The system was configured with CUDA 12.5 and NVIDIA driver version 535.183.01.

Evaluation approach Algorithm 8 takes as input the signature, which consists of finite sets of concept and role names, and constructs an $\mathcal{EL}$ terminology by posing membership and equivalence queries. Equivalence queries are simulated using a *PAC sample*, which is a set of concept inclusions randomly generated and answered by the LLM. For an ontology $\mathcal{O}$ with n logical axioms, the PAC sample size is computed as:

$$\frac{\ln(A^n/\delta)}{\epsilon},$$

where $\delta = 0.1$, $\epsilon = 0.2$, and A is the number of all possible axioms of the form:

$$A \sqsubseteq B, \quad A \sqcap B \sqsubseteq C, \quad B \sqsubseteq \exists r.A, \quad \text{and} \quad \exists r.A \sqsubseteq B,$$

with $A, B, C \in \mathsf{N_C}$ and $r \in \mathsf{N_R}$, formulated using the signature of $\mathcal{O}$. We refer to these axioms as *normalized*. The ability of ExactLearner+LLM to reconstruct existing ontologies is tested by posing queries to the LLM.

Hypotheses We test the following hypotheses in our experiments:

1. Even though LLMs may make mistakes and the simulation strategy does not guarantee exact learnability, we expect a correlation between the ontologies built by ExactLearner+LLM and the original ontologies.
2. Queries expressed in controlled natural language using our parsing function will yield better results than those using the OWL Manchester syntax.
3. Larger LLMs will perform better than smaller ones.
4. Ontologies expressing common knowledge, such as family relations, will be

[13] `https://github.com/MatteoMagnini/ExactLearner-LLM/`

Table 8.6: Ontology statistics and PAC sample sizes with $\epsilon = 0.2$ and $\gamma = 0.1$. $\mathsf{N_C}$ and $\mathsf{N_R}$ are the number of concept and role names occurring in the ontologies.

Ontology	$\mathsf{N_C}$	$\mathsf{N_R}$	**Log. Ax.**	**PAC Sample**	**Poss. Ax.**
Animals	17	4	12	542	6,936
Cell	22	0	24	1,119	10,164
Football	10	3	9	341	1,500
Generations	20	4	18	847	10,800
University	7	3	4	139	588

easier to learn than those with specialized knowledge, such as medical ontologies.

5. Simplifications introduced to reduce query length (see Section 8.3.3) will improve the performance of LLMs and, consequently, the ontologies constructed by ExactLearner+LLM.

Datasets We use two datasets:

1. Five small ontologies from the ExactLearner benchmark [DKO18].
2. Ten modules extracted from the GALEN ontology [Ala96], which contains medical information.

The small ontologies include:

- *Animals*, which describes the animal realm, including subphyla, classes, and orders.
- *Cell*, which provides information about cells based on type, development stage, and organism.
- *Football*, a minimal ontology describing relations between football games, teams, players, and managers.
- *Generations*, which describes family members and their relations.
- *University*, focusing on the professor role.

Table 8.6 summarizes the size of the signature, logical axioms, PAC sample size, and possible normalized axioms for these ontologies.

Table 8.7: Ontology statistics and PAC sample sizes with $\epsilon = 0.2$ and $\gamma = 0.1$ for medical ontologies.

Ontology	$\mathsf{N_C}$	$\mathsf{N_R}$	**Log. Ax.**	**PAC Sample**	**Pos. Ax.**
Ab. Elb. J. C.	27	14	43	2,286	39,366
BNF Sec.	36	24	80	4,646	107,568
Chlorhexidine	23	14	38	1,946	26,450
Cone of Tissue	42	42	100	6,163	220,500
Kalli Krein	18	10	27	1,279	11,988
Neon	16	10	25	1,149	8,960
Pin	43	40	99	6,113	225,578
Pros. Drug	29	14	47	2,540	47,096
Zopiclone	32	36	77	4,465	105,472
Zuccini	33	22	58	3,295	82,764

The GALEN ontology is a large medical ontology with $22,286$ concept names, 950 role names, and $46,558$ logical axioms. Since not all axioms are connected, we modularize GALEN using the bottom locality strategy [Gra+08][14]. This method generates $22,286$ modules, one for each concept name. We randomly select 10 modules, ensuring each has at least one concept and one role name, and no more than 50 concept or role names. Table 8.7 lists the selected modules.

LLMs, query format, and prompts We use four open LLMs: Mistral [Jia+23] (7B parameters), Mixtral [Jia+24] (47B parameters), Llama2 (13B parameters), and Llama3 (8B parameters) [Tou+23a], accessed via Ollama's API[15]. Queries are expressed in either Manchester OWL syntax or controlled natural language (see Section 8.3.3). We use two system prompts:

Simple system prompt

Answer with only True or False.

[14] `https://github.com/ernestojimenezruiz/locality-module-extractor`
[15] `https://github.com/ollama/ollama`

Advanced system prompt

You need to classify the following statements as True or False. The statement will be provided in either Manchester OWL syntax or natural language. Strictly follow these guidelines:

1. Answer with only True or False.
2. Entities with `has part` relations are not in a subclass relation.
3. Statements containing numerous entities are most probably False.
4. Take a deep breath before answering.
5. If unsure, answer False.

The advanced prompt provides additional contextual information and guidelines to improve LLM performance. For instance, it emphasizes that entities with `has part` relations should not be in a subclass relation [Fun+23]. It also mitigates the tendency of LLMs to answer "True" to long queries, which may arise during the computation of $\mathcal{O}$-essential concept inclusions (Section 8.3.2).

Experimental details To constrain LLM responses, we limit the maximum number of tokens to 2, ensuring answers are either "True" or "False"[16]. We also employ a caching mechanism to store LLM responses, reducing computation time and ensuring consistency when the same query is posed multiple times.

8.3.5 Evaluation and results

We evaluate the experimental results by computing the following metrics: accuracy, precision, recall, and F1-score. These metrics are calculated considering all possible axioms of the form:

$$A \sqsubseteq B, \quad A \sqcap B \sqsubseteq C, \quad B \sqsubseteq \exists r.A, \quad \text{and} \quad \exists r.A \sqsubseteq B,$$

where $A, B, C \in \mathsf{N_C}$ and $r \in \mathsf{N_R}$, formulated using a finite signature. Tautologies, such as $A \sqsubseteq A$, $A \sqcap B \sqsubseteq B$, and $A \sqcap B \sqsubseteq A$, are excluded to avoid artificially inflating the number of true positives. Our language does not allow for the expression of contradictions.

Axioms entailed by both the original and the learned ontology are labeled as true positives. Those not entailed by either are labeled as true negatives. Axioms

[16] A token is a unit of text, typically corresponding to a few characters. For example, "False" requires 2 tokens. See `https://help.openai.com/en/articles/4936856`.

entailed by the original ontology but not by the learned one are labeled as false negatives, and vice versa for false positives.

8.3.5.1 Learning performance

Table 8.8: Results of ExactLearner+LLM grouped by ontologies.

Ontology	Accuracy	Recall	Precision	F1-Score
Animals	0.737	0.858	0.381	0.428
Cell	0.391	0.733	0.206	0.284
Football	0.553	0.890	0.422	0.477
Generations	0.691	0.658	0.564	0.476
University	0.622	0.629	0.313	0.302

Table 8.9: Results of ExactLearner+LLM grouped by models.

Model	Accuracy	Recall	Precision	F1-Score
Llama2 (13b)	0.521	0.710	0.294	0.314
Llama3 (8b)	0.430	0.947	0.218	0.333
Mistral (7b)	0.741	0.747	0.450	0.490
Mixtral (47b)	0.705	0.611	0.547	0.436

Table 8.10: Results of ExactLearner+LLM grouped by prompt types.

Prompt Type	Accuracy	Recall	Precision	F1-Score
M. OWL Syntax	0.340	0.930	0.165	0.262
Natural Language	0.751	0.811	0.414	0.511
A. M. OWL Syntax	0.537	0.767	0.326	0.347
A. Natural Language	0.767	0.506	0.603	0.454

For the Animals ontology learned using the Mistral model, the learner correctly inferred that "Carnivore is an animal that eats some meat," consistent with the original ontology. However, it also inferred that "Carnivore is a mammal," which is incorrect as some carnivores are not mammals. Additionally, it inferred that "Carnivore is an animal that eats some animal," which aligns with real-world knowledge but is not entailed by the dataset ontology. To provide a comprehensive analysis, we present aggregated results in Tables 8.8 to 8.10.

Table 8.11: Individual results of ExactLearner+LLM on the Animals ontology. All results have been analysed with the Chi-Squared test (p-value < 0.05). The null hypothesis is that there is no correlation between the ontology constructed by ExactLearner+LLM and the Animals ontology. They all have p-value lower than 0.05 (i.e., refused null hypothesis).

Model	Prompt & Query	Accuracy	Recall	Precision	F1-Score
Llama2 (13b)	M. OWL Syntax	0.276	0.982	0.074	0.138
	Natural Language	0.859	0.930	0.286	0.437
	E. M. OWL Syntax	0.380	0.945	0.083	0.152
	E. Natural Language	0.970	0.651	0.801	0.718
Llama3 (8b)	M. OWL Syntax	0.309	1.000	0.078	0.145
	Natural Language	0.920	0.890	0.414	0.565
	E. M. OWL Syntax	0.314	0.985	0.078	0.145
	E. Natural Language	0.894	0.842	0.339	0.483
Mistral (7b)	M. OWL Syntax	0.552	0.996	0.116	0.207
	Natural Language	0.955	0.912	0.575	0.706
	E. M. OWL Syntax	0.830	0.945	0.250	0.396
	E. Natural Language	0.943	0.890	0.511	0.649
Mixtral (47b)	M. OWL Syntax	0.712	0.904	0.158	0.269
	Natural Language	0.970	0.893	0.690	0.779
	E. M. OWL Syntax	0.960	0.739	0.636	0.684
	E. Natural Language	0.955	0.228	1.000	0.371

Table 8.8 summarizes the results grouped by ontology. Ontologies in technical domains, such as the Cell ontology, exhibit lower average F1-scores (e.g., 0.284), while non-technical domains, such as Animals (0.428), Football (0.477), and Generations (0.476), perform better. This supports **hypothesis (4)**.

Table 8.9 shows the aggregated results by LLM. The Mistral model achieves the highest average F1-score (0.49), outperforming larger models like Mixtral. This contradicts **hypothesis (3)**.

Table 8.10 compares query formats and prompt types. Natural language prompts yield significantly higher F1-scores (0.511 and 0.454) compared to Manchester OWL syntax (0.262 and 0.347), confirming **hypothesis (2)**.

Results on GALEN modules For experiments on GALEN ontology modules, we used the best-performing LLM from Table 8.9 with the advanced natural language prompt. Due to the larger size of these modules, we limited experiments to

Table 8.12: Individual results of ExactLearner+LLM on the Cell ontology. All results have been analysed with the Chi-Squared test. A yellow line means p-value ≥ 0.05 (null hypothesis cannot be refused). A red line means no axiom has been learnt (empty ontology).

Model	Prompt & Query	Accuracy	Recall	Precision	F1-Score
Llama2 (13b)	M. OWL Syntax	0.163	1.0	0.163	0.280
	Natural Language	0.586	0.715	0.241	0.360
	E. M. OWL Syntax	0.163	1.0	0.163	0.280
	E. Natural Language	0.830	0.002	0.049	0.005
Llama3 (8b)	M. OWL Syntax	0.163	1.0	0.163	0.280
	Natural Language	0.208	1.0	0.171	0.292
	E. M. OWL Syntax	0.208	1.0	0.171	0.292
	E. Natural Language	0.254	1.0	0.179	0.304
Mistral (7b)	M. OWL Syntax	0.254	1.0	0.179	0.304
	Natural Language	0.779	0.432	0.354	0.390
	E. M. OWL Syntax	0.224	0.946	0.167	0.284
	E. Natural Language	0.769	0.337	0.308	0.322
Mixtral (47b)	M. OWL Syntax	0.163	1.0	0.163	0.280
	Natural Language	0.654	0.970	0.317	0.477
	E. M. OWL Syntax	0.840	0.320	0.514	0.394
	E. Natural Language	-	-	-	-

one LLM. Table 8.13 shows heterogeneous F1-scores, ranging from 0.253 (Neon) to 0.434 (Prostaglandin Drug). This variability is attributed to the lack of fine-tuning for medical domains. Nevertheless, all experiments passed a Chi-Square correlation test, confirming **hypothesis (1)**.

Synthetic teacher evaluation To validate our implementation, we tested ExactLearner+LLM with a synthetic teacher capable of perfectly answering membership queries. Table 8.14 reports the results. Precision is always 1, as the synthetic teacher has full knowledge of the target ontology. Other metrics may not reach 1 due to the PAC stopping criterion with $\epsilon = 0.2$ and $\delta = 0.1$.

Learner behavior analysis Figure 8.5 illustrates the operations performed by the learner, grouped by teacher type. For LLMs, saturation is the most frequent operation, followed by right decomposition, left decomposition, and desaturation. In contrast, the synthetic teacher primarily involves saturation and right decom-

Table 8.13: Results of the Exact Learner using Mistral with the advanced system prompt and the natural language query format on the modules of Galen. The orange colour line means that we ran out of memory.

Ontology	**Accuracy**	**Recall**	**Precision**	**F1-Score**
Above Elbow Jacket Cast	0.86	0.185	0.441	0.261
BNF Section 13.11	0.834	0.394	0.341	0.365
Chlorhexidine	0.891	0.199	0.522	0.288
Cone of Tissue	-	-	-	-
Kallikrein	0.877	0.283	0.771	0.414
Neon	0.868	0.163	0.564	0.253
Pin	0.87	0.482	0.204	0.287
Prostaglandin Drug	0.887	0.36	0.546	0.434
Zopiclone	0.923	0.233	0.303	0.263
Zuccini	0.915	0.182	0.588	0.278

Table 8.14: Results of ExactLearner+LLM with the synthetic teacher on small ontologies and modules extracted from Galen.

Ontology	**Accuracy**	**Recall**	**Precision**	**F1-Score**
Animals	0.997	0.941	1.0	0.97
Cell	1.0	1.0	1.0	1.0
Football	0.995	0.969	1.0	0.984
Generations	0.977	0.859	1.0	0.924
University	0.984	0.794	1.0	0.885
Above Elbow Jacket Cast	0.991	0.932	1.0	0.965
BNF Section 13.11	0.992	0.935	1.0	0.967
Chlorhexidine	0.993	0.936	1.0	0.967
Cone of Tissue	0.996	0.931	1.0	0.964
Kallikrein	0.994	0.962	1.0	0.981
Neon	1.0	1.0	1.0	1.0
Pin	0.997	0.94	1.0	0.969
Prostaglandin Drug	0.993	0.941	1.0	0.97
Zopiclone	0.995	0.916	1.0	0.956
Zuccini	0.993	0.918	1.0	0.957

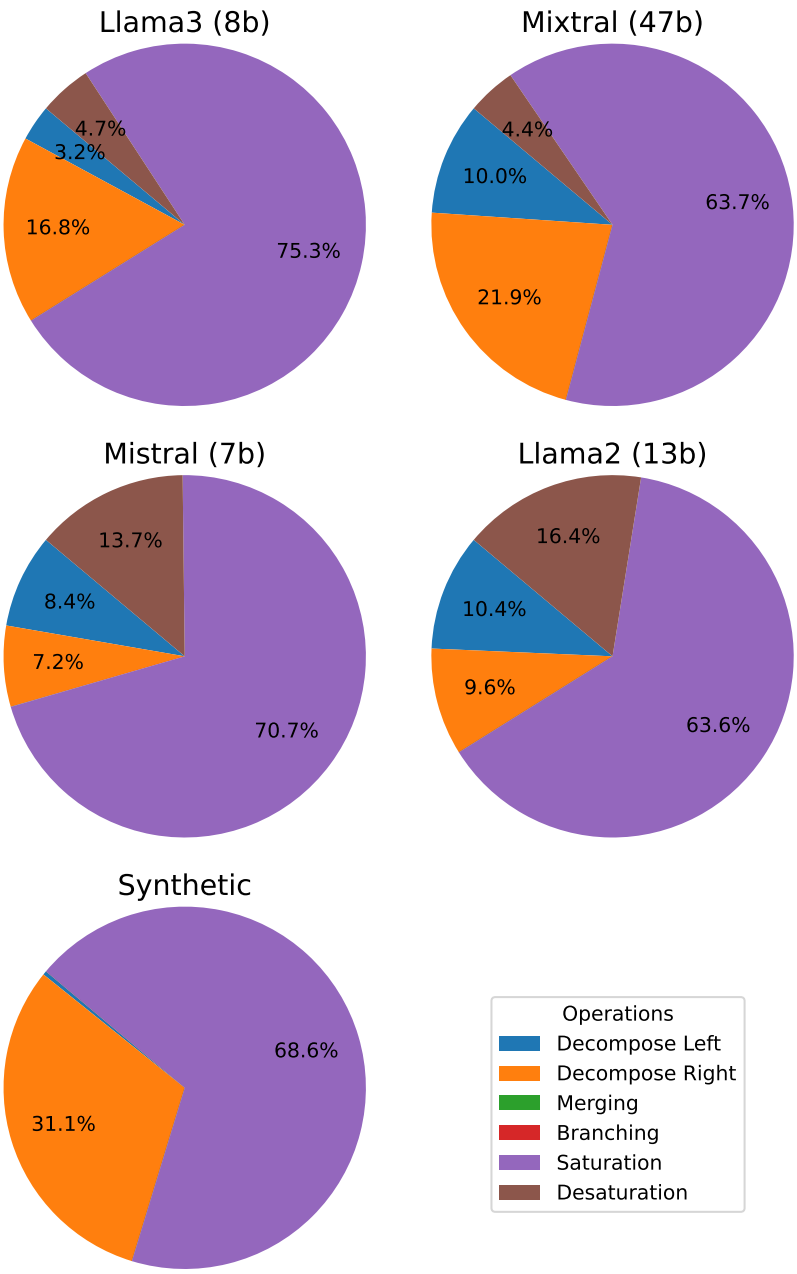


Figure 8.5: Aggregated results of the operations performed by the learner during the PAC learning of all the ontologies grouped by teacher type (LLMs and synthetic).

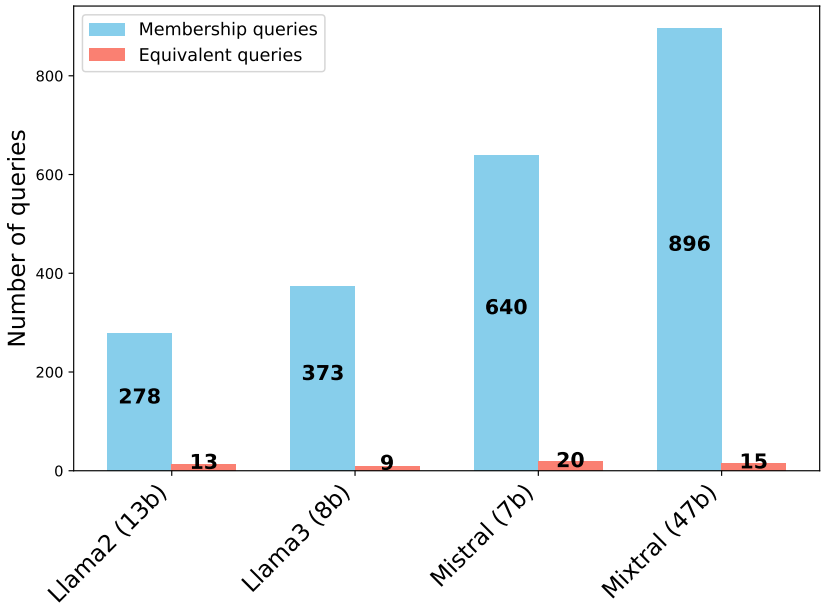


Figure 8.6: Average number of membership and (simulated) equivalence queries grouped by LLM. Each equivalence query is simulated by a batch of randomly generated queries.

position.

Figure 8.6 shows the average number of membership and simulated equivalence queries per LLM. The choice of LLM significantly impacts the number of queries posed during learning.

Simplification and splitting operations improved F1-scores across most LLM and prompt combinations, confirming **hypothesis (5)**. For instance, the Mistral model with natural language prompts saw an F1-score increase from approximately 0.5 to 0.7.

8.3.6 Conclusion and discussion

We introduced *ExactLearner+LLM*, an extension of the ExactLearner tool that leverages LLMs as teachers to learn $\mathcal{EL}$ ontologies. The tool addresses challenges such as query input format, unexpected responses, and logical consistency. Mechanisms like query simplification and splitting were implemented to adapt to LLM behavior.

Experimental results demonstrate statistical evidence of correlation between learned and original ontologies. Key findings include:

1. Concept inclusions of the form $A \sqsubseteq B$ are easier to learn than $A \sqsubseteq \exists r.B$ or $\exists r.B \sqsubseteq A$, leading to worse results for ontologies with more of the latter types.
2. Operations such as saturation, desaturation, and right decomposition were successfully applied.
3. The Mistral model outperformed larger models like Mixtral.
4. Natural language prompts significantly improved performance compared to Manchester OWL syntax.

Our approach guarantees the construction of $\mathcal{EL}$ terminologies with a specified vocabulary, even in a zero-shot setting. Future work includes exploring efficient counterexample generation, enhancing ontology language expressivity, and fine-tuning LLMs for technical domains like medicine.

Chapter 9

Conclusions

Contents

This thesis situates itself at the intersection of symbolic and sub-symbolic AI, focusing on the integration of both to enhance the capabilities of intelligent systems. The foundation of this work lies in classical AI techniques and KR such as computational logic and ontologies (Chapter 2). An SLR was conducted to explore the current landscape of SKI and SKE techniques (Chapter 3) defining a taxonomy to categorize existing approaches and identifying gaps in the literature. A considerable amount of the work carried out in this thesis contributed to the design and development of SKI methods, a novel SKI framework (PSyKI), and evaluation metrics (Chapters 4 and 5). Also, studies about AI and society, in particular related to the fairness of AI systems through SKI methods, were conducted (Chapter 6). Real-world applications of SKI techniques were explored, demonstrating their effectiveness in various domains (Chapter 7). Finally, because the ultimate goal of this work is to design and implement intelligent systems that can autonomously learn knowledge about a given domain, solutions based on SKI and SKE techniques were proposed and evaluated (Chapter 8).

9.1 Discussion

About the relevance of SKI and SKE techniques

Regarding the research question RQ0, introduced in Chapter 1, we conducted an SLR to investigate the current state of SKI and SKE techniques (Chapter 3). The SLR investigated *249* relevant papers, identifying *117* distinct SKI works and *132* SKE methods. These high numbers, along with the increasing trend of publications in recent years, indicate that SKI and SKE are active research areas within the AI community. In the course of other chapters – e.g., when presenting the possibility to use SKI techniques to enhance the fairness of AI systems in Chapter 6 and when showing real-world applications in Chapter 7 – we further demonstrated the relevance of SKI and SKE techniques in addressing contemporary challenges in AI. In Chapter 4 we also presented several novel SKI methods and in Chapter 5 a new framework (PSyKI) to facilitate the application of SKI techniques.

RQ0: *are SKI and SKE relevant in the context of modern AI?*
Answer: not only SKI and SKE are relevant, but they are arguably one of the leading areas of research in AI today, especially considering the recent advancements and integration with LLM technologies. In the near future, we expect many contributions in this area, both from a theoretical and practical standpoint, further solidifying the importance of SKI and SKE in the broader AI landscape.

About the characteristics of SKI and SKE techniques

The SLR also provided insights into the characteristics of existing SKI and SKE techniques posed by the research question RQ1. A comprehensive taxonomy was developed to categorize the techniques based on various dimensions.

RQ1: *what are the characteristics of SKI and SKE techniques?*
Answer: for SKI, the dimensions that characterise most the methods are *input knowledge type*, *sub-symbolic model target*, and *injection strategy*. We

observed that SKI techniques predominantly utilize structured knowledge representations, such as ontologies and knowledge graphs, to inject knowledge into sub-symbolic models. Also logic rules are widely used. Regarding the target sub-symbolic models, SKI methods mainly focus on NNs of any kind, with a predominance of feed-forward architectures. Finally, the injection strategies are quite balanced among the three main categories: *structuring*, *embedding*, and *guided learning*.
For SKE, the most characterising dimensions are *output knowledge shape*, *output knowledge expressiveness*, and *translucency*. SKE techniques predominantly generate knowledge in the form of logic rules, with DT being the second most common shape. The expressiveness of the extracted knowledge is often limited to propositional logic, with fewer methods producing first-order logic or more complex representations. Regarding translucency, SKE methods are fairly evenly distributed among *pedagogical* and *decompositional* approaches.
We believe that future contributions in the field of SKI and SKE will further diversify the characteristics of the techniques. In particular, we expect more methods to leverage unstructured knowledge sources (e.g., text documents) for SKI, and more expressive knowledge representations (e.g., ontologies with rich semantics, natural language) for SKE.

Measuring the effectiveness of SKI and SKE techniques

Accuracy, precision, recall, and F1-score are the most commonly used metrics to evaluate ML models, but these metrics do not capture the full spectrum of qualities of SKI and SKE techniques. To answer the research question RQ2, we proposed QoS metrics specifically designed to assess other aspects beyond traditional performance metrics (Chapter 4). In particular, in Section 5.2 four different QoS metrics were proposed: *memory footprint*, *energy consumption*, *latency*, and *data efficiency*. Additionally, in Section 5.3, we introduced a *robustness* metric to evaluate the resilience of SKI methods against different types of data degradation. All the metrics were rigorously defined and empirically validated through experiments. Finally, the fairness dimension was explored in Chapter 6, where we demonstrated

how SKI techniques can mitigate the bias of ML models, quantifiable through established fairness metrics.

RQ2: *how can the effects of SKI and SKE be measured?*
Answer: classical ML metrics are still relevant to measure the performance of SKI and SKE techniques, but they are not sufficient to capture all the relevant aspects involved. They should be complemented with additional metrics that capture other important aspects such as resource efficiency, robustness, and fairness depending on the context of application. We acknowledge that the research on this topic is still in its early stages, and we believe that future researches will further refine and expand the set of metrics available to evaluate SKI and SKE techniques.

When and where to use SKI and SKE techniques

In the context of real-world applications, we addressed the research question RQ3 by exploring various domains and purposes where SKI and SKE techniques have been successfully applied (Chapter 7). Specifically: *(i)* we mitigated bias in AI systems (Section 6.2) using SKI techniques to enhance fairness, *(ii)* we examined applications in healthcare (Section 7.1) using SKE to enhance explainability and customization of nutritional recommendations, *(iii)* we introduced a protocol for multi-agent based explanations in AI systems (Section 7.2) to improve transparency and user trust, *(iv)* again in the healthcare domain (Section 7.3), we used SKI to provide established domain knowledge to ML predictors in the context of chronic diseases, *(v)* finally, we explored the use of RAG – that falls into our SKI definition – by providing contextual prior knowledge to LLMs.

RQ3: *when and where should SKI and SKE be used?*
Answer: SKI techniques are particularly beneficial in scenarios where well established domain knowledge exists and can be leveraged to enhance the performance or other qualities of sub-symbolic models, including fairness and robustness. On the other hand, SKE techniques should be employed when there is a need to extract human-understandable knowledge, especially in domains

where explainability and transparency are critical, such as healthcare.

SKI and SKE techniques for designing NeSy AI systems

The last research question RQ4 was addressed in both Chapter 7 (already summarised in the previous paragraph) and Chapter 8. Because the ultimate goal of this thesis is to design and implement intelligent systems that can autonomously learn, in Chapter 8 we focused entirely on this aspect. In Section 8.1 we proposed a conceptual architecture for autonomous learning systems that leverage SKI and SKE techniques to continuously acquire, integrate, and refine knowledge. The idea consists in cycle of SKI and SKE steps, where knowledge is injected into sub-symbolic models, which are then used to extract new knowledge that can be reintegrated. Concrete systems that automatically learn knowledge are presented in Sections 8.2 and 8.3. The first work (Section 8.2) uses LLMs as oracles to populate ontologies starting from a predefined schema (the schema itself can be asked to the LLM). One can perform multiple iterations to refine the ontology at will. The second work (Section 8.3) instead consists in an active-learning framework that exploits LLMs as a teacher and the student learns the target ontology by posing queries to the teacher. Membership queries are directly answered by the LLM, while equivalence queries are simulated through the PAC framework. These contributions demonstrate the feasibility of designing autonomous learning systems.

RQ4: *how to design and develop NeSy AI systems that leverage SKI and SKE?*
Answer: SKI and SKE are fundamental components in the design and development of NeSy AI systems. There are various ways to integrate these techniques, depending on the specific requirements and constraints of the application domain. Generally speaking, these techniques should be employed whenever the AI system deals with both symbolic and sub-symbolic knowledge representations or requirements (e.g., explainability, transparency, fairness). For more advanced AI systems, such as autonomous learning agents, SKI and SKE can be combined in many ways. We suggest to use them in a

cyclic manner to continuously acquire, integrate, and refine knowledge about the environment and the tasks at hand. Also, the conjoint use of LLMs and SKI/SKE techniques appears to be the promising direction for future research and applications.

9.2 Future work

Universal SKI language

The vast majority of the existing SKI methods in the literature suffer from a lack of implementations and standardization. The techniques that actually have an implementation often rely on hard-coded rules and procedures that make them difficult to adapt to different contexts. Moreover, the use of logic programming formalism is often required, which can be a barrier for practitioners not familiar with it. PSyKI (Section 5.1) partially addresses these issues by providing a simpler interface to apply SKI techniques, but it still requires the user to know logic programming to define the input knowledge. To address these issues and make SKI accessible to a broader audience of users, a simplified and universal SKI language could be developed.

Practical implementation of the SKI-SKE cycle

The conceptual architecture for extracting and refining knowledge through a cycle of SKI and SKE steps was proposed in Section 8.1. However, a practical implementation of this architecture is still missing in the literature to the best of our knowledge. Developing a concrete system that embodies this cycle, possibly involving multiple intelligent agents sharing knowledge between each others, would be of great interest.

Social implication of SKI and SKE

Both SKI and SKE can affect the social implications of AI systems. Concerning SKI, we demonstrated in Chapter 6 how these techniques can be used to mitigate

bias in AI systems. Moreover, they can be used to enhance the transparency and explainability – by design – of AI models, which are crucial for user trust and acceptance (Chapter 4). SKE techniques can also contribute to explainability by providing human-understandable representations of the knowledge learned by sub-symbolic models (Sections 7.2 and 7.3).

Despite having these positive aspects, a measurement of the social impact of SKI and SKE in real-world applications among the users is still missing. Moreover, SKI methods and in particular the most recent techniques that involve the use of LLMs raise concerns about data privacy and AI safety. Indeed, if the knowledge used to inject into a sub-symbolic model contains sensitive information, and this knowledge is sent to a third party, there is a risk that this information could be inadvertently exposed or misused. Finally, whenever LLMs are used, there is always a chance of having hallucinated or biased information that could compromise the integrity of the knowledge being injected. We believe that these aspects should be further investigated in future studies.

Autonomous learning in the real world

Knowledge and data are just a representation of the real world. The best way to learn about the real world is to interact with it, like animals and humans did in millions of years of evolution (billions for the life in general). We believe that the next step for autonomous learning systems – and ultimately AGI – is to be embodied in the real world, through robots and other physical agents that can perceive and act in the environment. When this will take place, it will be of paramount importance to guarantee human and environmental safety, as well as ethical behavior of the agents. This will require a multidisciplinary approach – involving computer science, robotics, ethics, lawmakers, and social sciences – and a strong collaboration between academia, industry, and the society at large.

References

[91] *Molecular Biology (Splice-junction Gene Sequences)*. UCI Machine Learning Repository. 1991. DOI: 10.24432/C5M888.

[Abd+24] Marah Abdin et al. *Phi-3 Technical Report: A Highly Capable Language Model Locally on Your Phone*. 2024. arXiv: 2404.14219 [cs.CL]. URL: https://arxiv.org/abs/2404.14219.

[ACO21] Andrea Agiollo, Giovanni Ciatto, and Andrea Omicini. "*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*. Ed. by Davide Calvaresi et al. Vol. 12688. Lecture Notes in Computer Science. Cham, Switzerland: Springer, 2021, pp. 63–82. ISBN: 978-3-030-82016-9. DOI: 10.1007/978-3-030-82017-6_5.

[AG94] Miklós Ajtai and Yuri Gurevich. "Datalog vs First-Order Logic". In: *J. Comput. Syst. Sci.* 49.3 (1994), pp. 562–588. DOI: 10.1016/S0022-0000(05)80071-6.

[Aga+18] Giuseppe Agapito et al. "DIETOS: A dietary recommender system for chronic diseases monitoring and management". In: *Computer Methods and Programs in Biomedicine* 153 (2018), pp. 93–104. ISSN: 0169-2607. DOI: 10.1016/j.cmpb.2017.10.014.

[Agi+23] Andrea Agiollo et al. "Symbolic knowledge injection meets intelligent agents: QoS metrics and experiments". In: *Auton. Agents Multi Agent Syst.* 37.2 (2023), p. 27. DOI: 10.1007/S10458-023-09609-6.

[Agu+24] Gianluca Aguzzi et al. "Applying Retrieval-Augmented Generation on Open LLMs for a Medical Chatbot Supporting Hypertensive Patients". In: *Proceedings of the 3rd AIxIA Workshop on Artificial Intelligence For Healthcare (HC@AIxIA 2024) co-located with the 23rd International Conference of the Italian Association for Artificial Intelligence (AIxIA 2024), Bolzano, Italy, 27-28 November 2024*. Ed. by Francesco Calimeri, Mauro Dragoni, and Fabio Stella. Vol. 3880. CEUR Workshop Proceedings. CEUR-WS.org, 2024, pp. 189–201. URL: https://ceur-ws.org/Vol-3880/paper17.pdf.

[Ala96] P. Pole Alan L. Rector J.E. Rogers. "The GALEN High Level Ontology". In: *Medical Informatics Europe*. IOS Press, 1996, pp. 174–178. DOI: 10.3233/978-1-60750-878-6-174.

[Ali+23] Subhan Ali et al. "The enlightening role of explainable artificial intelligence in medical & healthcare domains: A systematic literature review". In: *Computers in Biology and Medicine* 166 (2023), p. 107555. ISSN: 0010-4825. DOI: https://doi.org/10.1016/j.compbiomed.2023.107555.

[Ang87] Dana Angluin. "Queries and Concept Learning". In: *Mach. Learn.* 2.4 (1987), pp. 319–342. DOI: 10.1007/BF00116828.

[Ang92] Dana Angluin. "Computational Learning Theory: Survey and Selected Bibliography". In: *Proceedings of the 24th Annual ACM Symposium on Theory of Computing*. Ed. by S. Rao Kosaraju et al. ACM, 1992, pp. 351–369. DOI: 10.1145/129712.129746.

[Ani+23] Rohan Anil et al. "Gemini: A Family of Highly Capable Multimodal Models". In: *CoRR* abs/2312.11805 (2023). DOI: 10.48550/ARXIV.2312.11805.

[Anj+19] Sule Anjomshoae et al. "Explainable Agents and Robots: Results from a Systematic Literature Review". In: *Proceedings of the 18th International Conference on Autonomous Agents and MultiAgent Systems*. AAMAS '19. International Foundation for Autonomous Agents

and Multiagent Systems, 2019, pp. 1078–1088. ISBN: 9781450363099. URL: https://dl.acm.org/doi/10.5555/3306127.3331806.

[AO21] Andrea Agiollo and Andrea Omicini. "Load Classification: A Case Study for Applying Neural Networks in Hyper-Constrained Embedded Devices". In: *Applied Sciences* 11.24 (Dec. 2021). Special Issue "Artificial Intelligence and Data Engineering in Engineering Applications". ISSN: 2076-3417. DOI: 10.3390/app112411957.

[ARO22] Andrea Agiollo, Andrea Rafanelli, and Andrea Omicini. "Towards Quality-of-Service Metrics for Symbolic Knowledge Injection". In: *WOA 2022 – 23rd Workshop "From Objects to Agents"*. Ed. by Angelo Ferrando and Viviana Mascardi. Vol. 3261. CEUR Workshop Proceedings. Nov. 2022, pp. 30–47. URL: http://ceur-ws.org/Vol-3261/paper3.pdf.

[Arr+20] Alejandro Barredo Arrieta et al. "Explainable Artificial Intelligence (XAI): Concepts, taxonomies, opportunities and challenges toward responsible AI". In: *Inf. Fusion* 58 (2020), pp. 82–115. DOI: 10.1016/J.INFFUS.2019.12.012.

[Aya+19] Ali Ayadi et al. "Ontology population with deep learning-based NLP: a case study on the Biomolecular Network Ontology". In: *Knowledge-Based and Intelligent Information & Engineering Systems: Proceedings of the 23rd International Conference KES-2019, Budapest, Hungary, 4-6 September 2019*. Ed. by Imre J. Rudas et al. Vol. 159. Procedia Computer Science. Elsevier, 2019, pp. 572–581. DOI: 10.1016/j.procs.2019.09.212.

[Aye+23] John W Ayers et al. "Comparing Physician and Artificial Intelligence Chatbot Responses to Patient Questions Posted to a Public Social Media Forum." In: *JAMA Internal Medicine* (2023). ISSN: 2168-6114 (Electronic); 2168-6106 (Print); 2168-6106 (Linking). DOI: 10.1001/jamainternmed.2023.1838.

[Ayo+22] Tom Ayoola et al. "ReFinED: An Efficient Zero-shot-capable Approach to End-to-End Entity Linking". In: *Proceedings of the 2022*

Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies: Industry Track, NAACL 2022, Hybrid: Seattle, Washington, USA + Online, July 10-15, 2022. Ed. by Anastassia Loukina, Rashmi Gangadharaiah, and Bonan Min. Association for Computational Linguistics, 2022, pp. 209–220. DOI: 10.18653/V1/2022.NAACL-INDUSTRY.24.

[Baa+03] Franz Baader et al., eds. *The Description Logic Handbook: Theory, Implementation, and Applications*. Cambridge University Press, 2003. ISBN: 0-521-78176-0.

[Baa+17] Franz Baader et al. *An Introduction to Description Logic*. Cambridge University Press, 2017. ISBN: 978-0-521-69542-8. URL: http://www.cambridge.org/de/academic/subjects/computer-science/knowledge-management-databases-and-data-mining/introduction-description-logic?format=PB%5C#17zVGeWD2TZUeu6s.97.

[Bad+22] Samy Badreddine et al. "Logic Tensor Networks". In: *Artif. Intell.* 303 (2022), p. 103649. DOI: 10.1016/J.ARTINT.2021.103649.

[BBL05] Franz Baader, Sebastian Brandt, and Carsten Lutz. "Pushing the EL Envelope". In: *IJCAI-05, Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence, Edinburgh, Scotland, UK, July 30 - August 5, 2005*. Ed. by Leslie Pack Kaelbling and Alessandro Saffiotti. Professional Book Center, 2005, pp. 364–369. URL: http://ijcai.org/Proceedings/05/Papers/0372.pdf.

[BCG07] Fabio Luigi Bellifemine, Giovanni Caire, and Dominic Greenwood. *Developing Multi-Agent Systems with JADE*. Wiley, Feb. 2007. ISBN: 978-0-470-05747-6. URL: http://eu.wiley.com/WileyCDA/WileyTitle/productCd-0470057475.html.

[BDM20] Stan Benjamens, Pranavsingh Dhunnoo, and Bertalan Meskó. "The state of artificial intelligence-based FDA-approved medical devices and algorithms: an online database". In: *npj Digit. Medicine* 3 (2020). DOI: 10.1038/S41746-020-00324-0.

[Ben+21] Emily M. Bender et al. “On the Dangers of Stochastic Parrots: Can Language Models Be Too Big?” In: *FAccT ’21: 2021 ACM Conference on Fairness, Accountability, and Transparency, Virtual Event / Toronto, Canada, March 3-10, 2021*. Ed. by Madeleine Clare Elish, William Isaac, and Richard S. Zemel. ACM, 2021, pp. 610–623. DOI: 10.1145/3442188.3445922.

[BGH05] Sebastian Bader, Artur S. d’Avila Garcez, and Pascal Hitzler. “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*. Ed. by Ingrid Russell and Zdravko Markov. AAAI Press, 2005, pp. 314–319. URL: http://www.aaai.org/Library/FLAIRS/2005/flairs05-052.php.

[Bhu+24] Bikram Pratim Bhuyan et al. “Neuro-symbolic artificial intelligence: a survey”. In: *Neural Comput. Appl.* 36.21 (2024), pp. 12809–12844. DOI: 10.1007/S00521-024-09960-Z.

[Bia+17] Devis Bianchini et al. “PREFer: A prescription-based food recommender system”. In: *Computer Standards & Interfaces* 54 (2017). SI: New modeling in Big Data, pp. 64–75. ISSN: 0920-5489. DOI: 10.1016/j.csi.2016.10.010.

[BL04] Ronald J. Brachman and Hector J. Levesque. “Chapter 16 - The Tradeoff between Expressiveness and Tractability”. In: *Knowledge Representation and Reasoning*. Ed. by Ronald J. Brachman and Hector J. Levesque. The Morgan Kaufmann Series in Artificial Intelligence. San Francisco: Morgan Kaufmann, 2004, pp. 327–348. ISBN: 978-1-55860-932-7. DOI: https://doi.org/10.1016/B978-155860932-7/50101-1.

[Blu+24] Sophie Blum et al. “Learning Horn envelopes via queries from language models”. In: *Int. J. Approx. Reason.* 171 (2024), p. 109026. DOI: 10.1016/J.IJAR.2023.109026.

[Bos+19] Antoine Bosselut et al. "COMET: Commonsense Transformers for Automatic Knowledge Graph Construction". In: *Proceedings of the 57th Conference of the Association for Computational Linguistics, ACL 2019, Florence, Italy, July 28- August 2, 2019, Volume 1: Long Papers*. Ed. by Anna Korhonen, David R. Traum, and Lluís Màrquez. Association for Computational Linguistics, 2019, pp. 4762–4779. DOI: 10.18653/V1/P19-1470.

[Bre+84] Leo Breiman et al. *Classification and Regression Trees*. Wadsworth, 1984. ISBN: 0-534-98053-8. DOI: 10.1201/9781315139470.

[Bro+20] Tom B. Brown et al. "Language Models are Few-Shot Learners". In: *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*. Ed. by Hugo Larochelle et al. 2020. URL: https://proceedings.neurips.cc/paper/2020/hash/1457c0d6bfcb4967418bfb8ac142f64a-Abstract.html.

[BSF94] Yoshua Bengio, Patrice Y. Simard, and Paolo Frasconi. "Learning long-term dependencies with gradient descent is difficult". In: *IEEE Trans. Neural Networks* 5.2 (1994), pp. 157–166. DOI: 10.1109/72.279181.

[Bur16] Jenna Burrell. "How the machine 'thinks': Understanding opacity in machine learning algorithms". In: *Big Data Soc.* 3.1 (2016). DOI: 10.1177/2053951715622512.

[Buz+22] Berk Buzcu et al. "Explanation-Based Negotiation Protocol for Nutrition Virtual Coaching". In: *PRIMA 2022: Principles and Practice of Multi-Agent Systems – 24th International Conference, Valencia, Spain, November 16-18, 2022, Proceedings*. Ed. by Reyhan Aydogan et al. Vol. 13753. Lecture Notes in Computer Science. Springer, 2022, pp. 20–36. DOI: 10.1007/978-3-031-21203-1_2.

[Cal+21] Davide Calvaresi et al. "EXPECTATION: Personalized Explainable Artificial Intelligence for Decentralized Agents with Heterogeneous Knowledge". In: *Explainable and Transparent AI and Multi-Agent*

Systems. Third International Workshop, EXTRAAMAS 2021, Virtual Event, May 3–7, 2021, Revised Selected Papers. Ed. by Davide Calvaresi et al. Vol. 12688. Lecture Notes in Computer Science. Basel, Switzerland: Springer Nature, 2021, pp. 331–343. ISBN: 978-3-030-82016-9. DOI: 10.1007/978-3-030-82017-6_20.

[Cao+21] Nicola De Cao et al. "Autoregressive Entity Retrieval". In: *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net, 2021. URL: https://openreview.net/forum?id=5k8F6UU39V.

[Car+21] Nicholas Carlini et al. "Extracting Training Data from Large Language Models". In: *30th USENIX Security Symposium, USENIX Security 2021, August 11-13, 2021*. Ed. by Michael Bailey and Rachel Greenstadt. USENIX Association, 2021, pp. 2633–2650. URL: https://www.usenix.org/conference/usenixsecurity21/presentation/carlini-extracting.

[Car97] Rich Caruana. "Multitask Learning". In: *Mach. Learn.* 28.1 (1997), pp. 41–75. DOI: 10.1023/A:1007379606734.

[Cau+24] J Harry Caufield et al. "Structured Prompt Interrogation and Recursive Extraction of Semantics (SPIRES): a method for populating knowledge bases using zero-shot learning". In: *Bioinformatics* 40.3 (2024), btae104. ISSN: 1367-4811. DOI: 10.1093/bioinformatics/btae104.

[CCB09] Gordon V. Cormack, Charles L. A. Clarke, and Stefan Büttcher. "Reciprocal rank fusion outperforms condorcet and individual rank learning methods". In: *Proceedings of the 32nd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR 2009, Boston, MA, USA, July 19-23, 2009*. Ed. by James Allan et al. ACM, 2009, pp. 758–759. DOI: 10.1145/1571941.1572114.

[CG98] Jaime G. Carbonell and Jade Goldstein. "The Use of MMR, Diversity-Based Reranking for Reordering Documents and Produc-

ing Summaries". In: *SIGIR '98: Proceedings of the 21st Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, August 24-28 1998, Melbourne, Australia*. Ed. by W. Bruce Croft et al. ACM, 1998, pp. 335–336. DOI: 10.1145/290941.291025.

[Cha+23] Victor Chang et al. "Pima Indians diabetes mellitus classification based on machine learning (ML) algorithms". In: *Neural Comput. Appl.* 35.22 (2023), pp. 16157–16173. DOI: 10.1007/S00521-022-07049-Z.

[Che+18] Muhao Chen et al. "Co-training Embeddings of Knowledge Graphs and Entity Descriptions for Cross-lingual Entity Alignment". In: *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI 2018, July 13-19, 2018, Stockholm, Sweden*. ijcai.org, 2018, pp. 3998–4004. DOI: 10.24963/IJCAI.2018/556.

[Che+19] Hsin-Pai Cheng et al. "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*. 2019, pp. 2033–2036. DOI: 10.1109/ICCVW.2019.00254.

[Che+21] Yu Chen et al. "Personalized Food Recommendation as Constrained Question Answering over a Large-Scale Food Knowledge Graph". In: *Proceedings of the 14th ACM International Conference on Web Search and Data Mining*. WSDM '21. Virtual Event, Israel: Association for Computing Machinery, 2021, pp. 544–552. ISBN: 9781450382977. DOI: 10.1145/3437963.3441816.

[Che+22] Chen Chen et al. "Knowledge Is Flat: A Seq2Seq Generative Framework for Various Knowledge Graph Completion". In: *Proceedings of the 29th International Conference on Computational Linguistics, COLING 2022, Gyeongju, Republic of Korea, October 12-17, 2022*. Ed. by Nicoletta Calzolari et al. International Committee on

Computational Linguistics, 2022, pp. 4005–4017. URL: https://aclanthology.org/2022.coling-1.352.

[Cho+14] Kyunghyun Cho et al. "Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation". In: *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing, EMNLP 2014, October 25-29, 2014, Doha, Qatar, A meeting of SIGDAT, a Special Interest Group of the ACL*. Ed. by Alessandro Moschitti, Bo Pang, and Walter Daelemans. ACL, 2014, pp. 1724–1734. DOI: 10.3115/V1/D14-1179.

[CHS20] Jaewoong Cho, Gyeongjo Hwang, and Changho Suh. "A Fair Classifier Using Kernel Density Estimation". In: *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*. Ed. by Hugo Larochelle et al. 2020. URL: https://proceedings.neurips.cc/paper/2020/hash/ac3870fcad1cfc367825cda0101eee62-Abstract.html.

[Cia+19] Giovanni Ciatto et al. "Towards XMAS: eXplainability through Multi-Agent Systems". In: *Proceedings of the 1st Workshop on Artificial Intelligence and Internet of Things co-located with the 18th International Conference of the Italian Association for Artificial Intelligence (AI*IA 2019), Rende (CS), Italy, November 22, 2019*. Ed. by Claudio Savaglio et al. Vol. 2502. CEUR Workshop Proceedings. CEUR-WS.org, 2019, pp. 40–53. URL: https://ceur-ws.org/Vol-2502/paper3.pdf.

[Cia+20a] Giovanni Ciatto et al. "Agent-Based Explanations in AI: Towards an Abstract Framework". In: *Explainable, Transparent Autonomous Agents and Multi-Agent Systems - Second International Workshop, EXTRAAMAS 2020, Auckland, New Zealand, May 9-13, 2020, Revised Selected Papers*. Ed. by Davide Calvaresi et al. Vol. 12175. Lecture Notes in Computer Science. Springer, 2020, pp. 3–20. DOI: 10.1007/978-3-030-51924-7_1.

[Cia+20b] Giovanni Ciatto et al. “An Abstract Framework for Agent-Based Explanations in AI”. In: *Proceedings of the 19th International Conference on Autonomous Agents and Multiagent Systems, AAMAS '20, Auckland, New Zealand, May 9-13, 2020*. Ed. by Amal El Fallah Seghrouchni et al. International Foundation for Autonomous Agents and Multiagent Systems, 2020, pp. 1816–1818. DOI: 10.5555/3398761.3398992.

[Cia+21] Giovanni Ciatto et al. “Towards Explainable Visionary Agents: License to Dare and Imagine”. In: *Explainable and Transparent AI and Multi-Agent Systems. Third International Workshop, EXTRAAMAS 2021, Virtual Event, May 3–7, 2021, Revised Selected Papers*. Ed. by Davide Calvaresi et al. Vol. 12688. Lecture Notes in Computer Science. Basel, Switzerland: Springer Nature, 2021, pp. 139–157. ISBN: 978-3-030-82016-9. DOI: 10.1007/978-3-030-82017-6_9.

[Cia+23] Giovanni Ciatto et al. “A General-Purpose Protocol for Multi-agent Based Explanations”. In: *Explainable and Transparent AI and Multi-Agent Systems - 5th International Workshop, EXTRAAMAS 2023, London, UK, May 29, 2023, Revised Selected Papers*. Ed. by Davide Calvaresi et al. Vol. 14127. Lecture Notes in Computer Science. Springer, 2023, pp. 38–58. DOI: 10.1007/978-3-031-40878-6_3.

[Cia+24] Giovanni Ciatto et al. “Symbolic Knowledge Extraction and Injection with Sub-symbolic Predictors: A Systematic Literature Review”. In: *ACM Comput. Surv.* 56.6 (2024), 161:1–161:35. DOI: 10.1145/3645103.

[Cia+25] Giovanni Ciatto et al. “Large language models as oracles for instantiating ontologies with domain-specific knowledge”. In: *Knowl. Based Syst.* 310 (2025), p. 112940. DOI: 10.1016/J.KNOSYS.2024.112940.

[Cim06] Philipp Cimiano. *Ontology Learning and Population from Text*. Springer US, 2006. DOI: 10.1007/978-0-387-39252-3.

[Cio+18] Tudor Cioara et al. “Expert system for nutrition care process of older adults”. In: *Future Generation Computer Systems* 80 (Mar. 2018), pp. 368–383. DOI: 10.1016/j.future.2017.05.037.

[Cip21] Carlo M. Cipolla. *The Basic Laws of Human Stupidity*. New York: Knopf Doubleday Publishing Group, 2021. ISBN: 9780385546485. URL: https://books.google.com/books/about/The_Basic_Laws_of_Human_Stupidity.html?id=9E3CDwAAQBAJ.

[CJK21] Bonggeun Choi, Daesik Jang, and Youngjoong Ko. “MEM-KGC: Masked Entity Model for Knowledge Graph Completion With Pre-Trained Language Model”. In: *IEEE Access* 9 (2021), pp. 132025–132032. DOI: 10.1109/ACCESS.2021.3113329.

[CO02] Robert Cattral and Franz Oppacher. *Poker Hand*. UCI Machine Learning Repository. 2002. DOI: 10.24432/C5KW38.

[CO94] Davida Charney and T.C. Ormerod. “Review of Communication at a Distance: The Influence of Print on Sociocultural Organization and Change, by David S. Kaufer and Kathleen M. Carley and Human Reasoning: the Psychology of Deduction, by J.St.B.T. Evans, S.E. Newstead and R.M.J. Byrne”. In: *International Journal of Human-Computer Studies* 40.6 (1994), pp. 1067–1073. DOI: 10.1006/ijhc.1994.1048.

[CRH16] Konstantina Christakopoulou, Filip Radlinski, and Katja Hofmann. “Towards Conversational Recommender Systems”. In: *KDD ’16: Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. Aug. 2016, pp. 815–824. ISBN: 9781450342322. DOI: 10.1145/2939672.2939746.

[CS93a] Marco Cadoli and Marco Schaerf. “A Survey of Complexity Results for Nonmonotonic Logics”. In: *J. Log. Program.* 17.2/3&4 (1993), pp. 127–160. DOI: 10.1016/0743-1066(93)90029-G.

[CS93b] R. Scott Cost and Steven Salzberg. “A Weighted Nearest Neighbor Algorithm for Learning with Symbolic Features”. In: *Mach. Learn.* 10 (1993), pp. 57–78. DOI: 10.1023/A:1022664626993.

[CV04] David Celjuska and Maria Vargas-Vera. "Ontosophie: A semi-automatic system for ontology population from text". In: *International Conference on Natural Language Processing (ICON)*. Vol. 60. 2004.

[Det+23] Tim Dettmers et al. "QLoRA: Efficient Finetuning of Quantized LLMs". In: *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*. Ed. by Alice Oh et al. 2023. URL: http://papers.nips.cc/paper%5C_files/paper/2023/hash/1feb87871436031bdc0f2beaa62a049b-Abstract-Conference.html.

[Dev+19] Jacob Devlin et al. "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, NAACL-HLT 2019, Minneapolis, MN, USA, June 2-7, 2019, Volume 1 (Long and Short Papers)*. Ed. by Jill Burstein, Christy Doran, and Thamar Solorio. Association for Computational Linguistics, 2019, pp. 4171–4186. DOI: 10.18653/V1/N19-1423.

[Din+21] Frances Ding et al. "Retiring Adult: New Datasets for Fair Machine Learning". In: *Advances in Neural Information Processing Systems 34: Annual Conference on Neural Information Processing Systems 2021, NeurIPS 2021, December 6-14, 2021, virtual*. Ed. by Marc'Aurelio Ranzato et al. 2021, pp. 6478–6490. URL: https://proceedings.neurips.cc/paper/2021/hash/32e54441e6382a7fbacbbbaf3c450059-Abstract.html.

[DK17] Oscar Day and Taghi M. Khoshgoftaar. "A survey on heterogeneous transfer learning". In: *Journal of Big Data* 4 (2017), p. 29. DOI: 10.1186/s40537-017-0089-0.

[DKO18] Mario Ricardo Cruz Duarte, Boris Konev, and Ana Ozaki. "ExactLearner: A Tool for Exact Learning of EL Ontologies". In: *KR*. Ed. by Michael Thielscher, Francesca Toni, and Frank Wolter. AAAI

Press, 2018, pp. 409–414. URL: https://aaai.org/ocs/index.php/KR/KR18/paper/view/18006.

[DOR01] Enrico Denti, Andrea Omicini, and Alessandro Ricci. "tu Prolog: A Light-Weight Prolog for Internet Applications and Infrastructures". In: *Practical Aspects of Declarative Languages, Third International Symposium, PADL 2001, Las Vegas, Nevada, USA, March 11-12, 2001, Proceedings*. Ed. by I. V. Ramakrishnan. Vol. 1990. Lecture Notes in Computer Science. Springer, 2001, pp. 184–198. DOI: 10.1007/3-540-45241-9_13.

[DR21] Jiawen Deng and Fuji Ren. "A Survey of Textual Emotion Recognition and Its Challenges". In: *IEEE Transactions on Affective Computing* (2021). DOI: 10.1109/TAFFC.2021.3053275.

[Dur+24] Damiano Duranti et al. "LLM-Driven Knowledge Extraction in Temporal and Description Logics". In: *EKAW*. Ed. by Mehwish Alam et al. Vol. 15370. Lecture Notes in Computer Science. Springer, 2024, pp. 190–208. DOI: 10.1007/978-3-031-77792-9_12.

[Dwo+12] Cynthia Dwork et al. "Fairness through awareness". In: *Innovations in Theoretical Computer Science 2012, Cambridge, MA, USA, January 8-10, 2012*. Ed. by Shafi Goldwasser. ACM, 2012, pp. 214–226. DOI: 10.1145/2090236.2090255.

[EHN16] Vanesa Espín, María V. Hurtado, and Manuel Noguera. "Nutrition for Elder Care: a nutritional semantic recommender system for the elderly". In: *Expert Systems* 33.2 (2016), pp. 201–210. DOI: 10.1111/exsy.12143.

[End72] Herbert B. Enderton. *A mathematical introduction to logic*. Academic Press, 1972. ISBN: 978-0-12-238450-9.

[Est+21] Andre Esteva et al. "Deep learning-enabled medical computer vision". In: *npj Digital Medicine* 4.1 (2021), pp. 1–9. DOI: 10.1038/s41746-020-00376-2.

[Etz+05] Oren Etzioni et al. “Unsupervised named-entity extraction from the Web: An experimental study”. In: *Artificial Intelligence* 165.1 (2005), pp. 91–134. DOI: 10.1016/j.artint.2005.03.001.

[Eur19] European Commission. *Ethics Guidelines for Trustworthy AI*. https://ec.europa.eu/digital-single-market/en/news/ethics-guidelines-trustworthy-ai. 2019.

[EW16] Lisa Ehrlinger and Wolfram Wöß. “Towards a Definition of Knowledge Graphs”. In: *Joint Proceedings of the Posters and Demos Track of the 12th International Conference on Semantic Systems - SEMANTiCS2016 and the 1st International Workshop on Semantic Change & Evolving Semantics (SuCCESS'16) co-located with the 12th International Conference on Semantic Systems (SEMANTiCS 2016), Leipzig, Germany, September 12-15, 2016*. Ed. by Michael Martin, Martí Cuquet, and Erwin Folmer. Vol. 1695. CEUR Workshop Proceedings. CEUR-WS.org, 2016. URL: https://ceur-ws.org/Vol-1695/paper4.pdf.

[FB10] Jill Freyne and Shlomo Berkovsky. “Intelligent Food Planning: Personalized Recipe Recommendation”. In: *IUI '10: Proceedings of the 15th international conference on Intelligent user interfaces*. IUI '10. ACM, 2010, pp. 321–324. ISBN: 9781605585154. DOI: 10.1145/1719970.1720021.

[FC24] Zhanling Fan and Chongcheng Chen. “CuPe-KG: Cultural perspective-based knowledge graph construction of tourism resources via pretrained language models”. In: *Inf. Process. Manag.* 61.2 (2024), p. 103646. DOI: 10.1016/J.IPM.2024.103646.

[Fel+15] Michael Feldman et al. “Certifying and Removing Disparate Impact”. In: *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Sydney, NSW, Australia, August 10-13, 2015*. Ed. by Longbing Cao et al. ACM, 2015, pp. 259–268. DOI: 10.1145/2783258.2783311.

[FFF18] Jean-Yves Franceschi, Alhussein Fawzi, and Omar Fawzi. "Robustness of classifiers to uniform l_p and Gaussian noise". In: *International Conference on Artificial Intelligence and Statistics (AISTATS 2018)*. Ed. by Amos Storkey and Fernando Perez-Cruz. Vol. 84. Proceedings of Machine Learning Research. Playa Blanca, Lanzarote, Spain: ML Research Press, Sept. 2018, pp. 1280–1288. URL: http://proceedings.mlr.press/v84/franceschi18a/franceschi18a.pdf.

[FJL21] Maurice Funk, Jean Christoph Jung, and Carsten Lutz. "Actively Learning Concepts and Conjunctive Queries under ELr-Ontologies". In: *IJCAI*. Ed. by Zhi-Hua Zhou. ijcai.org, 2021, pp. 1887–1893. DOI: 10.24963/IJCAI.2021/260.

[FM99] Michal Finkelstein-Landau and Emmanuel Morin. "Extracting Semantic Relationships between Terms: Supervised *vs.* Unsupervised Methods". In: *International Workshop on Ontological Engineering on the Global Information Infrastructure*. 1999, pp. 71–80.

[FT90] Robert B. Fraser and Stephen Z. Turney. "An expert system for the nutritional management of the critically ill". In: *Computer Methods and Programs in Biomedicine* 33.3 (1990), pp. 175–180. ISSN: 0169-2607. DOI: 10.1016/0169-2607(90)90040-G.

[Fuk80] Kunihiko Fukushima. "Neocognitron: A Self-Organizing Neural Network Model for a Mechanism of Pattern Recognition Unaffected by Shift in Position". In: *Biological Cybernetics* 36.4 (1980), pp. 193–202. DOI: 0.1007/BF00344251.

[Fun+19] Maurice Funk et al. "Learning Description Logic Concepts: When can Positive and Negative Examples be Separated?" In: *IJCAI*. Ed. by Sarit Kraus. ijcai.org, 2019, pp. 1682–1688. DOI: 10.24963/IJCAI.2019/233.

[Fun+23] Maurice Funk et al. "Towards Ontology Construction with Language Models". In: *(KBC-LM) and (LM-KBC) co-located with ISWC*. Ed. by Simon Razniewski et al. Vol. 3577. CEUR Workshop Proceedings.

CEUR-WS.org, 2023. URL: https://ceur-ws.org/Vol-3577/paper16.pdf.

[FZ11] Peter Forbes and Mu Zhu. "Content-Boosted Matrix Factorization for Recommender Systems: Experiments with Recipe Recommendation". In: *Proceedings of the Fifth ACM Conference on Recommender Systems*. RecSys '11. Chicago, Illinois, USA: Association for Computing Machinery, 2011, pp. 261–264. ISBN: 9781450306836. DOI: 10.1145/2043932.2043979.

[GC19] Aaron Gokaslan and Vanya Cohen. *OpenWebText Corpus*. http://Skylion007.github.io/OpenWebTextCorpus. 2019.

[Gel90] Tim van Gelder. "Why Distributed Representation is Inherently Non-Symbolic". In: *Konnektionismus in Artificial Intelligence und Kognitionsforschung. Proceedings 6. Österreichische Artificial Intelligence-Tagung (KONNAI), Salzburg, Österreich, 18. bis 21. September 1990*. Ed. by Georg Dorffner. Vol. 252. Informatik-Fachberichte. Springer, 1990, pp. 58–66. DOI: 10.1007/978-3-642-76070-9_6.

[Gev+21] Mor Geva et al. "Did Aristotle Use a Laptop? A Question Answering Benchmark with Implicit Reasoning Strategies". In: *Trans. Assoc. Comput. Linguistics* 9 (2021), pp. 346–361. DOI: 10.1162/TACL_A_00370.

[GF17] Bryce Goodman and Seth R. Flaxman. "European Union Regulations on Algorithmic Decision-Making and a "Right to Explanation"". In: *AI Mag.* 38.3 (2017), pp. 50–57. DOI: 10.1609/AIMAG.V38I3.2741.

[Giu+24] Mauro Giuffrè et al. "Optimizing large language models in digestive disease: strategies and challenges to improve clinical outcomes". In: *Liver International* 44.9 (2024), pp. 2114–2124. DOI: https://doi.org/10.1111/liv.15974.

[GM03] Aldo Gangemi and Peter Mika. "Understanding the Semantic Web through Descriptions and Situations". In: *On The Move to Meaningful Internet Systems 2003: CoopIS, DOA, and ODBASE - OTM Confederated International Conferences, CoopIS, DOA, and ODBASE 2003, Catania, Sicily, Italy, November 3-7, 2003*. Ed. by Robert Meersman, Zahir Tari, and Douglas C. Schmidt. Vol. 2888. Lecture Notes in Computer Science. Springer, 2003, pp. 689–706. DOI: 10.1007/978-3-540-39964-3_44.

[GR68] C. Cordell Green and Bertram Raphael. "The use of theorem-proving techniques in question-answering systems". In: *Proceedings of the 23rd ACM national conference (ACM 1968)*. Ed. by Richard B. Blue Sr. and Arthur M. Rosenberg. New York, NY, USA: ACM, 1968, pp. 169–181. DOI: 10.1145/800186.810578.

[Gra+08] Bernardo Cuenca Grau et al. "Modular Reuse of Ontologies: Theory and Practice". In: *J. Artif. Intell. Res.* 31 (2008), pp. 273–318. DOI: 10.1613/JAIR.2375.

[Gra+24] Aaron Grattafiori et al. *The Llama 3 Herd of Models*. 2024. arXiv: 2407.21783 [cs.AI]. URL: https://arxiv.org/abs/2407.21783.

[Gri+20] Sorin Mihai Grigorescu et al. "A survey of deep learning techniques for autonomous driving". In: *Journal of Field Robotics* 37.3 (2020), pp. 362–386. DOI: 10.1002/rob.21918.

[Gri10] Stephan Grimm. "Knowledge Representation and Ontologies". In: *Scientific Data Mining and Knowledge Discovery - Principles and Foundations*. Ed. by Mohamed Medhat Gaber. Springer, 2010, pp. 111–137. DOI: 10.1007/978-3-642-02788-8_6.

[Gui+19] Riccardo Guidotti et al. "A Survey of Methods for Explaining Black Box Models". In: *ACM Comput. Surv.* 51.5 (2019), 93:1–93:42. DOI: 10.1145/3236009.

[Han+23] Jiuzhou Han et al. "PiVe: Prompting with Iterative Verification Improving Graph-based Generative Capability of LLMs". In: *CoRR* abs/2305.12392 (2023). arXiv: 2305.12392.

[Hao+23] Shibo Hao et al. "BertNet: Harvesting Knowledge Graphs with Arbitrary Relations from Pretrained Language Models". In: *Findings of the Association for Computational Linguistics: ACL 2023, Toronto, Canada, July 9-14, 2023*. Ed. by Anna Rogers, Jordan L. Boyd-Graber, and Naoaki Okazaki. Association for Computational Linguistics, 2023, pp. 5000–5015. DOI: 10.18653/V1/2023.FINDINGS-ACL.309.

[Har+03] Alani Harith et al. "Automatic ontology-based knowledge extraction and tailored biography generation from the web". In: *IEEE Intelligent Systems* 18.1 (2003), pp. 14–21.

[Hay63] Louise Schmir Hay. "Axiomatization of the Infinite-Valued Predicate Calculus". In: *J. Symb. Log.* 28.1 (1963), pp. 77–86. DOI: 10.2307/2271339.

[Hez+19] Niloofar Hezarjaribi et al. "Human-in-the-Loop Learning for Personalized Diet Monitoring from Unstructured Mobile Data". In: *ACM Transactions on Interactive Intelligent Systems* 9.4 (2019). ISSN: 2160-6455. DOI: 10.1145/3319370.

[HH23] David J. Hunter and Christopher Holmes. "Where Medical Statistics Meets Artificial Intelligence". In: *New England Journal of Medicine* 389.13 (2023), pp. 1211–1219. DOI: 10.1056/NEJMra2212850.

[HHS24] Michael Townsen Hicks, James Humphries, and Joe Slater. "ChatGPT is bullshit". In: *Ethics Inf. Technol.* 26.2 (2024), p. 38. DOI: 10.1007/S10676-024-09775-5.

[Hit+16] Pascal Hitzler et al., eds. *Ontology Engineering with Ontology Design Patterns - Foundations and Applications*. Vol. 25. Studies on the Semantic Web. IOS Press, 2016. ISBN: 978-1-61499-675-0.

[Hor+06] Matthew Horridge et al. "The Manchester OWL Syntax". In: *OWLED*. Ed. by Bernardo Cuenca Grau et al. Vol. 216. CEUR Workshop Proceedings. CEUR-WS.org, 2006. URL: https://ceur-ws.org/Vol-216/submission%5C_9.pdf.

[Hor51] Alfred Horn. "On Sentences Which are True of Direct Unions of Algebras". In: *J. Symb. Log.* 16.1 (1951), pp. 14–21. DOI: 10.2307/2268661.

[Hou+17] Wenying Hou et al. "Consensus conditions for general second-order multi-agent systems with communication delay". In: *Automatica* 75 (2017), pp. 293–298. DOI: 10.1016/j.automatica.2016.09.042.

[HPS16] Moritz Hardt, Eric Price, and Nati Srebro. "Equality of Opportunity in Supervised Learning". In: *Advances in Neural Information Processing Systems 29: Annual Conference on Neural Information Processing Systems 2016, December 5-10, 2016, Barcelona, Spain*. Ed. by Daniel D. Lee et al. 2016, pp. 3315–3323. URL: https://proceedings.neurips.cc/paper/2016/hash/9d2682367c3935defcb1f9e247a97c0d-Abstract.html.

[HR24] Joschka Haltaufderheide and Robert Ranisch. "The ethics of ChatGPT in medicine and healthcare: a systematic review on Large Language Models (LLMs)". In: *npj Digital Medicine* 7.1 (2024), p. 183. DOI: 10.1038/s41746-024-01157-x.

[HS97] Sepp Hochreiter and Jürgen Schmidhuber. "Long Short-Term Memory". In: *Neural Comput.* 9.8 (1997), pp. 1735–1780. DOI: 10.1162/NECO.1997.9.8.1735.

[HSW89] Kurt Hornik, Maxwell B. Stinchcombe, and Halbert White. "Multilayer feedforward networks are universal approximators". In: *Neural Networks* 2.5 (1989), pp. 359–366. DOI: 10.1016/0893-6080(89)90020-8.

[Hu+22] Edward J. Hu et al. "LoRA: Low-Rank Adaptation of Large Language Models". In: *The Tenth International Conference on Learning Representations, ICLR 2022, Virtual Event, April 25-29, 2022*. OpenReview.net, 2022. URL: https://openreview.net/forum?id=nZeVKeeFYf9.

[Hua+18] Gao Huang et al. "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*. 2018, pp. 2752–2761. DOI: 10.1109/CVPR.2018.00291.

[Hub64] Peter J. Huber. "Robust Estimation of a Location Parameter". In: *The Annals of Mathematical Statistics* 35.1 (1964), pp. 73–101. DOI: 10.1214/aoms/1177703732.

[Jia+11] Min Jiang et al. "A study of machine-learning-based approaches to extract clinical entities and their assertions from discharge summaries". In: *J. Am. Medical Informatics Assoc.* 18.5 (2011), pp. 601–606. DOI: 10.1136/amiajnl-2011-000163.

[Jia+20] Zhengbao Jiang et al. "How Can We Know What Language Models Know". In: *Trans. Assoc. Comput. Linguistics* 8 (2020), pp. 423–438. DOI: 10.1162/TACL_A_00324.

[Jia+22] Zhimeng Jiang et al. "Generalized Demographic Parity for Group Fairness". In: *The Tenth International Conference on Learning Representations, ICLR 2022, Virtual Event, April 25-29, 2022*. OpenReview.net, 2022. URL: https://openreview.net/forum?id=YigKlMJwjye.

[Jia+23] Albert Q. Jiang et al. "Mistral 7B". In: *CoRR* abs/2310.06825 (2023). DOI: 10.48550/ARXIV.2310.06825.

[Jia+24] Albert Q. Jiang et al. "Mixtral of Experts". In: *CoRR* abs/2401.04088 (2024). DOI: 10.48550/ARXIV.2401.04088.

[KAG21] Ahlem Chérifa Khadir, Hassina Aliane, and Ahmed Guessoum. "Ontology learning: Grand tour and challenges". In: *Comput. Sci. Rev.* 39 (2021), p. 100339. DOI: 10.1016/j.cosrev.2020.100339.

[Kah11] Daniel Kahneman. *Thinking, Fast and Slow*. New York: Farrar, Straus and Giroux, 2011. ISBN: 9780374275631. URL: https://search.worldcat.org/title/thinking-fast-and-slow/oclc/815190472.

[Kan+18] Duseok Kang et al. "C-GOOD: C-code Generation Framework for Optimized On-device Deep Learning". In: *2018 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. 2018, pp. 1–8. DOI: 10.1145/3240765.3240786.

[KAS11] Toshihiro Kamishima, Shotaro Akaho, and Jun Sakuma. "Fairness-aware Learning through Regularization Approach". In: *Data Mining Workshops (ICDMW), 2011 IEEE 11th International Conference on, Vancouver, BC, Canada, December 11, 2011*. Ed. by Myra Spiliopoulou et al. IEEE Computer Society, 2011, pp. 643–650. DOI: 10.1109/ICDMW.2011.83.

[KB15] Diederik P. Kingma and Jimmy Ba. "Adam: A Method for Stochastic Optimization". In: *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*. Ed. by Yoshua Bengio and Yann LeCun. 2015. URL: http://arxiv.org/abs/1412.6980.

[KKK23] Slawomir Kierner, Jacek Kucharski, and Zofia Kierner. "Taxonomy of hybrid architectures involving rule-based reasoning and machine learning in clinical decision systems: A scoping review". In: *J. Biomed. Informatics* 144 (2023), p. 104428. DOI: 10.1016/J.JBI.2023.104428.

[KKS14] Yevgeny Kazakov, Markus Krötzsch, and Frantisek Simancik. "The Incredible ELK - From Polynomial Procedures to Efficient Reasoning with EL Ontologies". In: *J. Autom. Reason.* 53.1 (2014), pp. 1–61. DOI: 10.1007/S10817-013-9296-3.

[KLS21] Peter Kairouz, Ziyu Liu, and Thomas Steinke. "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*. Ed. by Marina Meila and Tong Zhang. Vol. 139. Proceedings of Machine Learning Research. PMLR, 2021, pp. 5201–5212. URL: http://proceedings.mlr.press/v139/kairouz21a.html.

[KMJ11] Asha Gowda Karegowda, AS Manjunath, and MA Jayaram. "Application of genetic algorithm optimized neural network connection weights for medical diagnosis of pima Indians diabetes". In: *International Journal on Soft Computing* 2.2 (2011), pp. 15–23. DOI: doi.org/10.5121/ijsc.2011.2202.

[Koh96] Ron Kohavi. *Census Income*. UCI Machine Learning Repository. 1996. DOI: https://doi.org/10.24432/C5GP7S.

[Kon+17] Boris Konev et al. "Exact Learning of Lightweight Description Logic Ontologies". In: *J. Mach. Learn. Res.* 18 (2017), 201:1–201:63. URL: http://jmlr.org/papers/v18/16-256.html.

[Kör+22] Philipp Körner et al. "Fifty Years of Prolog and Beyond". In: *Theory Pract. Log. Program.* 22.6 (2022), pp. 776–858. DOI: 10.1017/S1471068422000102.

[KOW16] Boris Konev, Ana Ozaki, and Frank Wolter. "A Model for Learning Description Logic Ontologies Based on Exact Learning". In: *AAAI*. Ed. by Dale Schuurmans and Michael P. Wellman. AAAI Press, 2016, pp. 1008–1015. DOI: 10.1609/AAAI.V30I1.10087.

[Kre+24] Simone Kresevic et al. "Optimization of hepatological clinical guidelines interpretation by large language models: a retrieval augmented generation-based framework". In: *npj Digital Medicine* 7.1 (2024), p. 102. URL: https://doi.org/10.1038/s41746-024-01091-y.

[Kum+20] Abhijeet Kumar et al. "Building Knowledge Graph using Pre-trained Language Model for Learning Entity-aware Relationships". In: *2020 IEEE International Conference on Computing, Power and Communication Technologies (GUCON)*. 2020, pp. 310–315. DOI: 10.1109/GUCON48875.2020.9231227.

[Kun+10] Gautam Kunapuli et al. "Online Knowledge-Based Support Vector Machines". In: *Machine Learning and Knowledge Discovery in Databases, European Conference, ECML PKDD 2010, Barcelona, Spain, September 20-24, 2010, Proceedings, Part II*. Ed. by José L. Balcázar et al. Vol. 6322. Lecture Notes in Computer Science.

Springer, 2010, pp. 145–161. DOI: 10.1007/978-3-642-15883-4_10.

[KW20] Alina Köchling and Marius Claus Wehner. "Discriminated by an algorithm: a systematic review of discrimination and fairness by algorithmic decision-making in the context of HR recruitment and HR development". In: *Business Research* 13.3 (2020), pp. 795–848. DOI: 10.1007/s40685-020-00134-w.

[KWH10] Bart Knijnenburg, Martijn Willemsen, and Stefan Hirtbach. "Receiving Recommendations and Providing Feedback: The User-Experience of a Recommender System". In: *E-Commerce and Web Technologies*. Vol. 61. Lecture Notes in Business Information Processing. Springer, Sept. 2010, pp. 207–216. ISBN: 978-3-642-15207-8. DOI: 10.1007/978-3-642-15208-5_19.

[Lam+20] Luís C. Lamb et al. "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*. Ed. by Christian Bessiere. ijcai.org, 2020, pp. 4877–4884. DOI: 10.24963/IJCAI.2020/679.

[LB87] Hector J. Levesque and Ronald J. Brachman. "Expressiveness and tractability in knowledge representation and reasoning". In: *Comput. Intell.* 3 (1987), pp. 78–93. DOI: 10.1111/J.1467-8640.1987.TB00176.X.

[LBH15] Yann LeCun, Yoshua Bengio, and Geoffrey E. Hinton. "Deep learning". In: *Nat.* 521.7553 (2015), pp. 436–444. DOI: 10.1038/NATURE14539.

[LDL21] Edgar Liberis, Lukasz Dudziak, and Nicholas D. Lane. "μNAS: Constrained Neural Architecture Search for Microcontrollers". In: *1st Workshop on Machine Learning and Systems (EuroMLSys@EuroSys 2021)*. Ed. by Eiko Yoneki and Paul Patras. Edinburgh, Scotland, UK: ACM, 2021, pp. 70–79. DOI: 10.1145/3437984.3458836.

[LeC+98] Yann LeCun et al. “Gradient-based learning applied to document recognition”. In: *Proceedings of the IEEE* 86.11 (1998), pp. 2278–2324. DOI: 10.1109/5.726791.

[Lew+20] Patrick Lewis et al. “Retrieval-augmented generation for knowledge-intensive NLP tasks”. In: *Proceedings of the 34th International Conference on Neural Information Processing Systems*. NIPS ’20. Vancouver, BC, Canada: Curran Associates Inc., 2020. ISBN: 9781713829546. URL: https://proceedings.neurips.cc/paper/2020/hash/6b493230205f780e1bc26945df7481e5-Abstract.html.

[LF91] Douglas B. Lenat and Edward A. Feigenbaum. “On the Thresholds of Knowledge”. In: *Artif. Intell.* 47.1-3 (1991), pp. 185–250. DOI: 10.1016/0004-3702(91)90055-O.

[LH07] Shane Legg and Marcus Hutter. “Universal Intelligence: A Definition of Machine Intelligence”. In: *Minds Mach.* 17.4 (2007), pp. 391–444. DOI: 10.1007/S11023-007-9079-X.

[LHN23] Ziyang Li, Jiani Huang, and Mayur Naik. “Scallop: A Language for Neurosymbolic Programming”. In: *Proc. ACM Program. Lang.* 7.PLDI (2023), pp. 1463–1487. DOI: 10.1145/3591280.

[Li+23] Hanzhou Li et al. “Ethics of large language models in medicine and medical research”. In: *The Lancet Digital Health* 5.6 (2023), e333–e335. DOI: 10.1016/S2589-7500(23)00083-3.

[Li+24] Dawei Li et al. “Contextualization Distillation from Large Language Model for Knowledge Graph Completion”. In: *Findings of the Association for Computational Linguistics: EACL 2024, St. Julian’s, Malta, March 17-22, 2024*. Association for Computational Linguistics, 2024, pp. 458–477. URL: https://aclanthology.org/2024.findings-eacl.32.

[Lip18] Zachary C. Lipton. “The mythos of model interpretability”. In: *Commun. ACM* 61.10 (2018), pp. 36–43. DOI: 10.1145/3233231.

[Liu+13] Chunyang Liu et al. “Convolution Neural Network for Relation Extraction”. In: *Advanced Data Mining and Applications - 9th International Conference, ADMA 2013, Hangzhou, China, December 14-16, 2013, Proceedings, Part II*. Ed. by Hiroshi Motoda et al. Vol. 8347. Lecture Notes in Computer Science. Springer, 2013, pp. 231–242. DOI: 10.1007/978-3-642-53917-6_21.

[Liu+18] Xuanqing Liu et al. “Towards Robust Neural Networks via Random Self-ensemble”. In: *Computer Vision - ECCV 2018 - 15th European Conference, Munich, Germany, September 8-14, 2018, Proceedings, Part VII*. Ed. by Vittorio Ferrari et al. Vol. 11211. Lecture Notes in Computer Science. Springer, 2018, pp. 381–397. DOI: 10.1007/978-3-030-01234-2_23.

[Liu+19] Yinhan Liu et al. “RoBERTa: A Robustly Optimized BERT Pretraining Approach”. In: *CoRR* abs/1907.11692 (2019). URL: http://arxiv.org/abs/1907.11692.

[Llo82] Stuart P Lloyd. “Least squares quantization in PCM”. In: *IEEE Transactions on Information Theory* 28.2 (1982), pp. 129–137. DOI: 10.1109/TIT.1982.1056489.

[Llo90] John W. Lloyd, ed. *Computational Logic*. Berlin, Heidelberg: Springer, 1990. DOI: 10.1007/978-3-642-76274-1.

[LNM19] Mohamed Lubani, Shahrul Azman Mohd. Noah, and Rohana Mahmud. “Ontology population: Approaches and design aspects”. In: *J. Inf. Sci.* 45.4 (2019). DOI: 10.1177/0165551518801819.

[LRU14] Jure Leskovec, Anand Rajaraman, and Jeffrey D. Ullman. *Mining of Massive Datasets, 2nd Ed.* Cambridge University Press, 2014. ISBN: 978-1107077232. URL: http://www.mmds.org/.

[Lua+18] Yi Luan et al. “Multi-Task Identification of Entities, Relations, and Coreference for Scientific Knowledge Graph Construction”. In: *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing, Brussels, Belgium, October 31 - November 4,*

2018. Association for Computational Linguistics, 2018, pp. 3219–3232. DOI: 10.18653/V1/D18-1360.

[Mag+23] Matteo Magnini et al. "Symbolic knowledge extraction for explainable nutritional recommenders". In: *Comput. Methods Programs Biomed.* 235 (2023), p. 107536. DOI: 10.1016/J.CMPB.2023.107536.

[Mag+24] Matteo Magnini et al. "Enforcing Fairness via Constraint Injection with FaUCI". In: *Proceedings of the 2nd Workshop on Fairness and Bias in AI co-located with 27th European Conference on Artificial Intelligence (ECAI 2024), Santiago de Compostela, Spain, October 20th, 2024*. Ed. by Roberta Calegari, Virginia Dignum, and Barry O'Sullivan. Vol. 3808. CEUR Workshop Proceedings. CEUR-WS.org, 2024. URL: https://ceur-ws.org/Vol-3808/paper8.pdf.

[Mag+25] Matteo Magnini et al. " Neuro-Symbolic AI for Supporting Chronic Disease Diagnosis and Monitoring ". In: *2025 IEEE International Conference on Pervasive Computing and Communications Workshops and other Affiliated Events (PerCom Workshops)*. Los Alamitos, CA, USA: IEEE Computer Society, Mar. 2025, pp. 446–451. DOI: 10.1109/PerComWorkshops65533.2025.00106.

[Mah+24] Jenish Maharjan et al. "OpenMedLM: prompt engineering can outperform fine-tuning in medical question-answering with open-source large language models". In: *Scientific Reports* 14.1 (2024), p. 14156. DOI: 10.1038/s41598-024-64827-6.

[Mai+18] David Maier et al. "Datalog: concepts, history, and outlook". In: *Declarative Logic Programming: Theory, Systems, and Applications*. Ed. by Michael Kifer and Yanhong Annie Liu. Vol. 20. ACM Books. ACM / Morgan & Claypool, 2018, pp. 3–100. DOI: 10.1145/3191315.3191317.

[Mak87] Johann A. Makowsky. "Why Horn Formulas Matter in Computer Science: Initial Structures and Generic Examples". In: *J. Comput. Syst. Sci.* 34.2/3 (1987), pp. 266–292. DOI: 10.1016/0022-0000(87)90027-4.

[Man+21] Robin Manhaeve et al. “Neural probabilistic logic programming in DeepProbLog”. In: *Artificial Intelligence* 298 (2021), p. 103504. DOI: 10.1016/j.artint.2021.103504.

[Mat75] B.W. Matthews. “Comparison of the predicted and observed secondary structure of T4 phage lysozyme”. In: *Biochimica et Biophysica Acta (BBA) - Protein Structure* 405.2 (1975), pp. 442–451. ISSN: 0005-2795. DOI: https://doi.org/10.1016/0005-2795(75)90109-9.

[MCB24] Lucas Memmert, Izabel Cvetkovic, and Eva A. C. Bittner. “The More Is Not the Merrier: Effects of Prompt Engineering on the Quality of Ideas Generated By GPT-3”. In: *57th Hawaii International Conference on System Sciences, HICSS 2024, Hilton Hawaiian Village Waikiki Beach Resort, Hawaii, USA, January 3-6, 2024*. Ed. by Tung X. Bui. ScholarSpace, 2024, pp. 7520–7529. URL: https://hdl.handle.net/10125/107289.

[McD90] Drew McDermott. “A Critique of Pure Reason”. In: *The Philosophy of Artificial Intelligence*. Ed. by Margaret A. Boden. Oxford readings in philosophy. Oxford University Press, 1990, pp. 206–230.

[McN77] George F. McNulty. “Fragments of first order logic, I: Universal Horn logic”. In: *Journal of Symbolic Logic* 42.2 (1977), pp. 221–237. DOI: 10.2307/2272123.

[MCO22a] Matteo Magnini, Giovanni Ciatto, and Andrea Omicini. “A view to a KILL: Knowledge Injection via Lambda Layer”. In: *WOA 2022 – 23rd Workshop “From Objects to Agents”*. Ed. by Angelo Ferrando and Viviana Mascardi. Vol. 3261. CEUR Workshop Proceedings. Sun SITE Central Europe, RWTH Aachen University, Nov. 2022, pp. 61–76. URL: http://ceur-ws.org/Vol-3261/paper5.pdf.

[MCO22b] Matteo Magnini, Giovanni Ciatto, and Andrea Omicini. “Bridging Symbolic and Sub-Symbolic AI: Towards Cooperative Transfer Learning in Multi-Agent Systems”. In: *Proceedings of the Discussion Papers - 22nd International Conference of the Italian Association for*

Artificial Intelligence (AIxIA 2022 DP), Udine, Italy, November 28 - December 2, 2022. Ed. by Agostino Dovier, Angelo Montanari, and Andrea Orlandini. Vol. 3419. CEUR Workshop Proceedings. CEUR-WS.org, 2022, pp. 12–22. URL: https://ceur-ws.org/Vol-3419/paper2.pdf.

[MCO22c] Matteo Magnini, Giovanni Ciatto, and Andrea Omicini. "KINS: Knowledge Injection via Network Structuring". In: *CILC 2022 – Italian Conference on Computational Logic*. Ed. by Roberta Calegari, Giovanni Ciatto, and Andrea Omicini. Vol. 3204. CEUR Workshop Proceedings. CEUR-WS, 2022, pp. 254–267. URL: http://ceur-ws.org/Vol-3204/paper_25.pdf.

[MCO22d] Matteo Magnini, Giovanni Ciatto, and Andrea Omicini. "On the Design of PSyKI: A Platform for Symbolic Knowledge Injection into Sub-symbolic Predictors". In: *Explainable and Transparent AI and Multi-Agent Systems - 4th International Workshop, EXTRAAMAS 2022, Virtual Event, May 9-10, 2022, Revised Selected Papers*. Ed. by Davide Calvaresi et al. Vol. 13283. Lecture Notes in Computer Science. Springer, 2022, pp. 90–108. DOI: 10.1007/978-3-031-15565-9_6.

[MDD21] Igor Melnyk, Pierre Dognin, and Payel Das. "Grapher: Multi-stage knowledge graph construction using pretrained language models". In: *NeurIPS 2021 Workshop on Deep Generative Models and Downstream Applications*. 2021.

[Meh+22] Ninareh Mehrabi et al. "A Survey on Bias and Fairness in Machine Learning". In: *ACM Comput. Surv.* 54.6 (2022), 115:1–115:35. DOI: 10.1145/3457607.

[MG20] Padala Manisha and Sujit Gujar. "FNNC: Achieving Fairness through Neural Networks". In: *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, IJCAI 2020*. Ed. by Christian Bessiere. ijcai.org, 2020, pp. 2277–2283. DOI: 10.24963/IJCAI.2020/315.

[MHN+13] Andrew L Maas, Awni Y Hannun, Andrew Y Ng, et al. "Rectifier nonlinearities improve neural network acoustic models". In: *Proc. icml*. Vol. 30. 1. Atlanta, GA. 2013, p. 3. URL: https://ai.stanford.edu/~amaas/papers/relu_hybrid_icml2013_final.pdf.

[Mil+19] Martijn Millecamp et al. "To Explain or Not to Explain: The Effects of Personal Characteristics When Explaining Music Recommendations". In: *IUI '19: Proceedings of the 24th International Conference on Intelligent User Interfaces*. New York, NY, USA: Association for Computing Machinery, Mar. 2019, pp. 397–407. ISBN: 9781450362726. DOI: 10.1145/3301275.3302313.

[Mil19] Tim Miller. "Explanation in artificial intelligence: Insights from the social sciences". In: *Artif. Intell.* 267 (2019), pp. 1–38. DOI: 10.1016/J.ARTINT.2018.07.007.

[Mit97] Tom M. Mitchell. *Machine learning, International Edition*. McGraw-Hill Series in Computer Science. McGraw-Hill, 1997. ISBN: 978-0-07-042807-2. URL: https://www.worldcat.org/oclc/61321007.

[MJJ20] Weiqing Min, Shuqiang Jiang, and Ramesh Jain. "Food Recommendation: Framework, Existing Solutions, and Challenges". In: *IEEE Transactions on Multimedia* 22.10 (2020), pp. 2659–2671. DOI: 10.1109/TMM.2019.2958761.

[MLP08] Diana Maynard, Yaoyong Li, and Wim Peters. "NLP Techniques for Term Extraction and Ontology Population". In: *Ontology Learning and Population: Bridging the Gap between Text and Knowledge*. Ed. by Paul Buitelaar and Philipp Cimiano. Vol. 167. Frontiers in Artificial Intelligence and Applications. IOS Press, 2008, pp. 107–127. URL: https://ebooks.iospress.nl/volume/proof-technology-and-computation.

[MN10] Patrick E. McKnight and Julius Najab. "Mann-Whitney U Test". In: *The Corsini Encyclopedia of Psychology*. John Wiley and Sons, Ltd,

2010, pp. 1–1. ISBN: 9780470479216. DOI: 10.1002/9780470479216.corpsy0524.

[MOM12] Grégoire Montavon, Genevieve B. Orr, and Klaus-Robert Müller, eds. *Neural Networks: Tricks of the Trade - Second Edition*. Vol. 7700. Lecture Notes in Computer Science. Springer, 2012. ISBN: 978-3-642-35288-1. DOI: 10.1007/978-3-642-35289-8.

[Mon+23] Sara Montagna et al. "Data Decentralisation of LLM-Based Chatbot Systems in Chronic Disease Self-Management". In: *Proceedings of the 2023 ACM Conference on Information Technology for Social Good*. GoodIT '23. Lisbon, Portugal: Association for Computing Machinery, 2023, pp. 205–212. ISBN: 9798400701160. DOI: 10.1145/3582515.3609536.

[Mon+24] Sara Montagna et al. "LLM-based Solutions for Healthcare Chatbots: a Comparative Analysis". In: *2024 IEEE International Conference on Pervasive Computing and Communications Workshops and other Affiliated Events (PerCom Workshops)*. 2024, pp. 346–351. DOI: 10.1109/PerComWorkshops59983.2024.10503257.

[Mor99] Emmanuel Morin. "Automatic acquisition of semantic relations between terms from technical corpora". In: *Proceedings of the Fifth International Congress on Terminology and Knowledge Engineering (TKE'99)*. 1999.

[MP43] Warren S McCulloch and Walter Pitts. "A logical calculus of the ideas immanent in nervous activity". In: *The bulletin of mathematical biophysics* 5 (1943), pp. 115–133. DOI: 10.1007/BF02478259.

[MP87] Marvin Minsky and Seymour Papert. *Perceptrons - an introduction to computational geometry*. MIT Press, 1987. ISBN: 978-0-262-63111-2.

[MS22] Avinash Madasu and Shashank Srivastava. "What do Large Language Models Learn beyond Language?" In: *Findings of the Association for Computational Linguistics: EMNLP 2022, Abu Dhabi, United Arab Emirates, December 7-11, 2022*. Ed. by Yoav Gold-

berg, Zornitsa Kozareva, and Yue Zhang. Association for Computational Linguistics, 2022, pp. 6940–6953. DOI: 10.18653/V1/2022.FINDINGS-EMNLP.516.

[Mua+22] Yazan Mualla et al. "The quest of parsimonious XAI: A human-agent architecture for explanation formulation". In: *Artificial Intelligence* 302 (2022). DOI: 10.1016/j.artint.2021.103573.

[Noy+08] Natalya Fridman Noy et al. "BioPortal: A Web Repository for Biomedical Ontologies and Data Resources". In: *ISWC*. Ed. by Christian Bizer and Anupam Joshi. Vol. 401. CEUR Workshop Proceedings. CEUR-WS.org, 2008. DOI: 10.1017/9781139025355.

[NS76] Allen Newell and Herbert A. Simon. "Computer Science as Empirical Inquiry: Symbols and Search". In: *Commun. ACM* 19.3 (1976), pp. 113–126. DOI: 10.1145/360018.360022.

[ODo+08] John O'Donovan et al. "PeerChooser: Visual Interactive Recommendation". In: *CHI '08: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. 2008, pp. 1085–1088. ISBN: 9781605580111. DOI: 10.1145/1357054.1357222.

[OKM21] Iñigo Orue-Saiz, Miguel Kazarez, and Amaia Mendez-Zorrilla. "Systematic Review of Nutritional Recommendation Systems". In: *Applied Sciences* 11.24 (2021). ISSN: 2076-3417. DOI: 10.3390/app112412069.

[Omi20] Andrea Omicini. "Not Just for Humans: Explanation for Agent-to-Agent Communication". In: *Proceedings of the AIxIA 2020 Discussion Papers Workshop co-located with the the 19th International Conference of the Italian Association for Artificial Intelligence (AIxIA2020), Anywhere, November 27th, 2020*. Ed. by Giuseppe Vizzari, Matteo Palmonari, and Andrea Orlandini. Vol. 2776. CEUR Workshop Proceedings. CEUR-WS.org, 2020, pp. 1–11. URL: http://ceur-ws.org/Vol-2776/paper-1.pdf.

[Ope23] OpenAI. "GPT-4 Technical Report". In: *CoRR* abs/2303.08774 (2023). DOI: 10.48550/ARXIV.2303.08774.

[OPM20] Ana Ozaki, Cosimo Persia, and Andrea Mazzullo. "Learning Query Inseparable ELH Ontologies (Extended Abstract)". In: *DL*. Ed. by Stefan Borgwardt and Thomas Meyer. Vol. 2663. CEUR Workshop Proceedings. CEUR-WS.org, 2020. DOI: 10.1609/AAAI.V34I03.5688.

[Pad+21] Ishita Padhiar et al. "Semantic Modeling for Food Recommendation Explanations". In: *37th IEEE International Conference on Data Engineering Workshops, ICDE Workshops 2021, Chania, Greece, April 19-22, 2021*. IEEE, 2021, pp. 13–19. DOI: 10.1109/ICDEW53142.2021.00010.

[Pan+23] Jeff Z. Pan et al. "Large Language Models and Knowledge Graphs: Opportunities and Challenges". In: *TGDK* 1.1 (2023), 2:1–2:38. DOI: 10.4230/TGDK.1.1.2.

[Pan+24] Shirui Pan et al. "Unifying Large Language Models and Knowledge Graphs: A Roadmap". In: *IEEE Trans. Knowl. Data Eng.* 36.7 (2024), pp. 3580–3599. DOI: 10.1109/TKDE.2024.3352100.

[Pea01] Karl Pearson. "LIII. On lines and planes of closest fit to systems of points in space". In: *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science* 2.11 (1901), pp. 559–572. DOI: 10.1080/14786440109462720.

[Pet+11] Georgios Petasis et al. "Ontology Population and Enrichment: State of the Art". In: *Knowledge-Driven Multimedia Information Extraction and Ontology Evolution - Bridging the Semantic Gap*. Ed. by Georgios Paliouras, Constantine D. Spyropoulos, and George Tsatsaronis. Vol. 6050. Lecture Notes in Computer Science. Springer, 2011, pp. 134–166. DOI: 10.1007/978-3-642-20795-2_6.

[Pet+19] Fabio Petroni et al. "Language Models as Knowledge Bases?" 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 2019, Hong Kong, China, November 3-7, 2019*. Ed. by Kentaro Inui et al. Association

for Computational Linguistics, 2019, pp. 2463–2473. DOI: 10.18653/V1/D19-1250.

[PQ95] Terence John Parr and Russell W. Quong. "ANTLR: A Predicated-*LL(k)* Parser Generator". In: *Softw. Pract. Exp.* 25.7 (1995), pp. 789–810. DOI: 10.1002/SPE.4380250705.

[PY10] Sinno Jialin Pan and Qiang Yang. "A Survey on Transfer Learning". In: *IEEE Transactions on Knowledge and Data Engineering* 22.10 (2010), pp. 1345–1359. DOI: 10.1109/TKDE.2009.191.

[Qui86] J. Ross Quinlan. "Induction of decision trees". In: *Machine Learning* 1.1 (1986), pp. 81–106. DOI: 10.1007/BF00116251.

[Rad+18] Alec Radford et al. "Improving language understanding by generative pre-training". In: (2018). URL: https://cdn.openai.com/research-covers/language-unsupervised/language_understanding_paper.pdf.

[Rad+19] Alec Radford et al. *Language Models are Unsupervised Multitask Learners*. https://d4mucfpksywv.cloudfront.net/better-language-models/language-models.pdf. 2019.

[Raf+20] Colin Raffel et al. "Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer". In: *J. Mach. Learn. Res.* 21 (2020), 140:1–140:67. URL: http://jmlr.org/papers/v21/20-074.html.

[Raf+24] Andrea Rafanelli et al. "An Empirical Study on the Robustness of Knowledge Injection Techniques Against Data Degradation". In: *Proceedings of the 25th Workshop "From Objects to Agents", Bard (Aosta), Italy, July 8-10, 2024*. Ed. by Marco Alderighi et al. Vol. 3735. CEUR Workshop Proceedings. CEUR-WS.org, 2024, pp. 20–32. URL: https://ceur-ws.org/Vol-3735/paper%5C_02.pdf.

[Raj+22] Pranav Rajpurkar et al. "AI in health and medicine". In: *Nature Medicine* 28.1 (2022), pp. 31–38. DOI: 10.1038/s41591-021-01614-0.

[Rec+19] Benjamin Recht et al. "Do ImageNet Classifiers Generalize to ImageNet?" In: *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*. Ed. by Kamalika Chaudhuri and Ruslan Salakhutdinov. Vol. 97. Proceedings of Machine Learning Research. PMLR, 2019, pp. 5389–5400. URL: http://proceedings.mlr.press/v97/recht19a.html.

[RGB23] Laura von Rüden, Jochen Garcke, and Christian Bauckhage. "How Does Knowledge Injection Help in Informed Machine Learning?" In: *International Joint Conference on Neural Networks, IJCNN 2023, Gold Coast, Australia, June 18-23, 2023*. IEEE, 2023, pp. 1–8. DOI: 10.1109/IJCNN54540.2023.10191994.

[RHW86] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. "Learning representations by back-propagating errors". In: *Nature* 323.6088 (1986), pp. 533–536. DOI: 10.1038/323533a0.

[RHW87] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. "Learning Internal Representations by Error Propagation". In: *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*. Vol. 1: Foundations. MIT Press, 1987, pp. 318–362. URL: https://ieeexplore.ieee.org/document/6302929.

[RN20] Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach (4th Edition)*. Pearson, 2020. ISBN: 9780134610993. URL: http://aima.cs.berkeley.edu/.

[Ros58] Frank Rosenblatt. "The perceptron: a probabilistic model for information storage and organization in the brain." In: *Psychological review* 65.6 (1958), p. 386. DOI: 10.1037/h0042519.

[RSG16] Marco Túlio Ribeiro, Sameer Singh, and Carlos Guestrin. ""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, San Francisco, CA, USA,*

August 13-17, 2016. Ed. by Balaji Krishnapuram et al. ACM, 2016, pp. 1135–1144. DOI: 10.1145/2939672.2939778.

[Rud+22] Cynthia Rudin et al. "Interpretable machine learning: Fundamental principles and 10 grand challenges". In: *Statistics Surveys* 16.none (2022), pp. 1–85. DOI: 10.1214/21-SS133.

[Rüd+23] Laura von Rüden et al. "Informed Machine Learning - A Taxonomy and Survey of Integrating Prior Knowledge into Learning Systems". In: *IEEE Trans. Knowl. Data Eng.* 35.1 (2023), pp. 614–633. DOI: 10.1109/TKDE.2021.3079836.

[Rud19] Cynthia Rudin. "Stop explaining black box machine learning models for high stakes decisions and use interpretable models instead". In: *Nat. Mach. Intell.* 1.5 (2019), pp. 206–215. DOI: 10.1038/S42256-019-0048-X.

[RZ09] Stephen E. Robertson and Hugo Zaragoza. "The Probabilistic Relevance Framework: BM25 and Beyond". In: *Found. Trends Inf. Retr.* 3.4 (2009), pp. 333–389. DOI: 10.1561/1500000019.

[Sab+22] Federico Sabbatini et al. "Symbolic knowledge extraction from opaque ML predictors in PSyKE: Platform design & experiments". In: *Intelligenza Artificiale* 16.1 (July 2022). Ed. by Roberta Calegari et al., pp. 27–48. ISSN: 1724-8035. DOI: 10.3233/IA-210120.

[Sav+21] Stefano Savazzi et al. "Opportunities of federated learning in connected, cooperative, and automated industrial systems". In: *IEEE Communications Magazine* 59.2 (2021), pp. 16–21. URL: https://ieeexplore.ieee.org/document/9374643.

[SB14] Shai Shalev-Shwartz and Shai Ben-David. *Understanding Machine Learning - From Theory to Algorithms*. Cambridge University Press, 2014. ISBN: 978-1-10-705713-5. DOI: 10.1017/CBO9781107298019.

[SB98] Richard S. Sutton and Andrew G. Barto. *Reinforcement learning - an introduction*. Adaptive computation and machine learning. MIT Press, 1998. ISBN: 978-0-262-19398-6. URL: http://www.incompleteideas.net/book/first/the-book.html.

[SCR15] Achille Souili, Denis Cavallucci, and Franc cois Rousselot. "Natural language processing (NLP)–a solution for knowledge extraction from patent unstructured data". In: *Procedia engineering* 131 (2015), pp. 635–643.

[Sha15] Murray Shanahan. *The technological singularity*. MIT press, 2015.

[She+22] Jianhao Shen et al. "Joint Language Semantic and Structure Embedding for Knowledge Graph Completion". In: *Proceedings of the 29th International Conference on Computational Linguistics, COLING 2022, Gyeongju, Republic of Korea, October 12-17, 2022*. Ed. by Nicoletta Calzolari et al. International Committee on Computational Linguistics, 2022, pp. 1965–1978. URL: https://aclanthology.org/2022.coling-1.171.

[She09] Sara J Shettleworth. *Cognition, Evolution, and Behavior: Cognition, Evolution, and Behavior*. Oxford University Press, Dec. 2009. ISBN: 9780195319842. DOI: 10.1093/oso/9780195319842.001.0001.

[Shi+16] Hoo-Chang Shin et al. "Deep Convolutional Neural Networks for Computer-Aided Detection: CNN Architectures, Dataset Characteristics and Transfer Learning". In: *IEEE Transactions on Medical Imaging* 35.5 (2016), pp. 1285–1298. DOI: 10.1109/TMI.2016.2528162.

[Shi02] Hideo Shimazu. "ExpertClerk: A Conversational Case-Based Reasoning Tool for Developing Salesclerk Agents in E-Commerce Webshops." In: *Artificial Intelligence Review* 18 (Dec. 2002), pp. 223–244. DOI: 10.1023/A:1020757023711.

[Shr+17] Ashish Shrivastava et al. "Learning from Simulated and Unsupervised Images through Adversarial Training". In: *2017 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017, Honolulu, HI, USA, July 21-26, 2017*. IEEE Computer Society, 2017, pp. 2242–2251. DOI: 10.1109/CVPR.2017.241.

[Shu+21] Ilia Shumailov et al. “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*. 2021, pp. 212–231. DOI: 10.1109/EuroSP51992.2021.00024.

[SK19] Connor Shorten and Taghi M. Khoshgoftaar. “A survey on Image Data Augmentation for Deep Learning”. In: *Journal of Big Data* 6.1 (2019), 60:1–60:48. DOI: 10.1186/s40537-019-0197-0.

[SKG22] Apoorv Saxena, Adrian Kochsiek, and Rainer Gemulla. “Sequence-to-Sequence Knowledge Graph Completion and Question Answering”. In: *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2022, Dublin, Ireland, May 22-27, 2022*. Ed. by Smaranda Muresan, Preslav Nakov, and Aline Villavicencio. Association for Computational Linguistics, 2022, pp. 2814–2828. DOI: 10.18653/V1/2022.ACL-LONG.201.

[SM24] Christel Sirocchi and Sara Montagna. “Integrating Symbolic Knowledge and Machine Learning in Healthcare”. In: *Companion Proceedings of the 8th International Joint Conference on Rules and Reasoning co-located with 20th Reasoning Web Summer School (RW 2024) and 16th DecisionCAMP 2024 as part of Declarative AI 2024, Bucharest, Romania, September 16-18, 2024*. Ed. by Anisa Rula et al. Vol. 3816. CEUR Workshop Proceedings. CEUR-WS.org, 2024. URL: https://ceur-ws.org/Vol-3816/paper45.pdf.

[Smi+88] Jack W Smith et al. “Using the ADAP learning algorithm to forecast the onset of diabetes mellitus”. In: *Proceedings of the annual symposium on computer application in medical care*. American Medical Informatics Association. 1988, p. 261.

[Sri+14] Nitish Srivastava et al. “Dropout: a simple way to prevent neural networks from overfitting”. In: *J. Mach. Learn. Res.* 15.1 (2014), pp. 1929–1958. DOI: 10.5555/2627435.2670313.

[SS20] Ramon Sanchez-Iborra and Antonio F Skarmeta. "TinyML-enabled frugal smart objects: Challenges and opportunities". In: *IEEE Circuits and Systems Magazine* 20.3 (2020), pp. 4–18. DOI: 10.1109/MCAS.2020.3005467.

[SSW22] Ritu Shandilya, Sugam Sharma, and Johnny Wong. "MATURE-Food: Food Recommender System for MAndatory FeaTURE Choices A system for enabling Digital Health". In: *International Journal of Information Management Data Insights* 2.2 (2022), p. 100090. ISSN: 2667-0968. DOI: 10.1016/j.jjimei.2022.100090.

[Ste00] Robert J. Sternberg, ed. *Handbook of Intelligence*. Cambridge University Press, 2000. ISBN: 9780511807947. DOI: 10.1017/CBO9780511807947.

[SW24] Vivian Magri Alcaldi Soares and Renata Wassermann. "Ontology extraction and evaluation for the Blue Amazon". In: *ONTOBRAS*. Ed. by Claudenir Morais Fonseca and Jeanne Louize Emygdio. Vol. 3905. CEUR Workshop Proceedings. CEUR-WS.org, 2024. URL: https://ceur-ws.org/Vol-3905/master5.pdf.

[Sze+14] Christian Szegedy et al. "Intriguing properties of neural networks". In: *2nd International Conference on Learning Representations (ICLR 2014), Conference Track Proceedings*. Ed. by Yoshua Bengio and Yann LeCun. Banff, AB, Canada, 2014. URL: https://openreview.net/forum?id=kklr_MTHMRQjG.

[Ta24] Gemma Team and et al. *Gemma 2: Improving Open Language Models at a Practical Size*. 2024. arXiv: 2408.00118 [cs.CL]. URL: https://arxiv.org/abs/2408.00118.

[TE17] Christoph Trattner and David Elsweiler. "Investigating the Healthiness of Internet-Sourced Recipes: Implications for Meal Planning and Recommender Systems". In: *Proceedings of the 26th International Conference on World Wide Web*. WWW '17. International World Wide Web Conferences Steering Committee, 2017, pp. 489–498. ISBN: 9781450349130. DOI: 10.1145/3038912.3052573.

[Tia+22] Yijun Tian et al. “Recipe2Vec: Multi-modal Recipe Representation Learning with Graph Neural Networks”. In: *Proceedings of the Thirty-First International Joint Conference on Artificial Intelligence*. July 2022, pp. 3448–3454. DOI: 10.24963/ijcai.2022/479.

[TL18] Trieu H. Trinh and Quoc V. Le. “A Simple Method for Commonsense Reasoning”. In: *CoRR* abs/1806.02847 (2018). URL: http://arxiv.org/abs/1806.02847.

[TM06] Hristo Tanev and Bernardo Magnini. “Weakly Supervised Approaches for Ontology Population”. In: *EACL 2006, 11st Conference of the European Chapter of the Association for Computational Linguistics, Proceedings of the Conference, April 3-7, 2006, Trento, Italy*. Ed. by Diana McCarthy and Shuly Wintner. The Association for Computer Linguistics, 2006. URL: https://aclanthology.org/E06-1003/.

[Tol+17] Gabriele Tolomei et al. “Interpretable Predictions of Tree-based Ensembles via Actionable Feature Tweaking”. In: *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Halifax, NS, Canada, August 13 - 17, 2017*. ACM, 2017, pp. 465–474. DOI: 10.1145/3097983.3098039.

[Tou+23a] Hugo Touvron et al. “Llama 2: Open Foundation and Fine-Tuned Chat Models”. In: *CoRR* abs/2307.09288 (2023). DOI: 10.48550/arxiv.2307.09288. arXiv: 2307.09288.

[Tou+23b] Hugo Touvron et al. “LLaMA: Open and Efficient Foundation Language Models”. In: *CoRR* abs/2302.13971 (2023). DOI: 10.48550/ARXIV.2302.13971.

[TP98] Sebastian Thrun and Lorien Y. Pratt. “Learning to Learn: Introduction and Overview”. In: *Learning to Learn*. Ed. by Sebastian Thrun and Lorien Y. Pratt. Springer, 1998, pp. 3–17. DOI: 10.1007/978-1-4615-5529-2_1.

[Tra+18] Thi Ngoc Trang Tran et al. “An Overview of Recommender Systems in the Healthy Food Domain”. In: *J. Intell. Inf. Syst.* 50.3 (June 2018), pp. 501–526. ISSN: 0925-9902. DOI: 10.1007/s10844-017-0469-0.

[TS94] Geoffrey G. Towell and Jude W. Shavlik. “Knowledge-Based Artificial Neural Networks”. In: *Artif. Intell.* 70.1-2 (1994), pp. 119–165. DOI: 10.1016/0004-3702(94)90105-8.

[TSN90] Geoffrey G. Towell, Jude W. Shavlik, and Michiel O. Noordewier. “Refinement of Approximate Domain Theories by Knowledge-Based Neural Networks”. In: *Proceedings of the 8th National Conference on Artificial Intelligence. Boston, Massachusetts, USA, July 29 - August 3, 1990, 2 Volumes*. Ed. by Howard E. Shrobe, Thomas G. Dietterich, and William R. Swartout. AAAI Press / The MIT Press, 1990, pp. 861–866. URL: http://www.aaai.org/Library/AAAI/1990/aaai90-129.php.

[Val84] Leslie G. Valiant. “A Theory of the Learnable”. In: *Commun. ACM* 27.11 (1984), pp. 1134–1142. DOI: 10.1145/1968.1972.

[Vas+17] Ashish Vaswani et al. “Attention is All you Need”. In: *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*. Ed. by Isabelle Guyon et al. 2017, pp. 5998–6008. URL: https://proceedings.neurips.cc/paper/2017/hash/3f5ee243547dee91fbd053c1c4a845aa-Abstract.html.

[VD01] Johan Van Benthem and Kees Doets. “Higher-Order Logic”. In: *Handbook of Philosophical Logic*. Ed. by D. M. Gabbay and F. Guenthner. Dordrecht: Springer Netherlands, 2001, pp. 189–243. ISBN: 978-94-015-9833-0. DOI: 10.1007/978-94-015-9833-0_3.

[VDH20] Sahil Verma, John P. Dickerson, and Keegan Hines. “Counterfactual Explanations for Machine Learning: A Review”. In: *CoRR* abs/2010.10596 (2020). arXiv: 2010.10596. URL: https://arxiv.org/abs/2010.10596.

[Wan+17] Quan Wang et al. “Knowledge Graph Embedding: A Survey of Approaches and Applications”. In: *IEEE Trans. Knowl. Data Eng.* 29.12 (2017), pp. 2724–2743. DOI: 10.1109/TKDE.2017.2754499.

[Wan+21a] Bo Wang et al. “Structure-Augmented Text Representation Learning for Efficient Knowledge Graph Completion”. In: *WWW ’21: The Web Conference 2021, Virtual Event / Ljubljana, Slovenia, April 19-23, 2021*. Ed. by Jure Leskovec et al. ACM / IW3C2, 2021, pp. 1737–1748. DOI: 10.1145/3442381.3450043.

[Wan+21b] Wenjie Wang et al. “Market2Dish: Health-aware Food Recommendation”. In: *ACM Trans. Multimedia Comput. Commun. Appl.* 17.1 (Apr. 2021). ISSN: 1551-6857. DOI: 10.1145/3418211.

[Wan+23] Guan Wang et al. “OpenChat: Advancing Open-source Language Models with Mixed-Quality Data”. In: *arXiv preprint arXiv:2309.11235* (2023).

[Wan+24] Andy Wang et al. “Fine-tuning large language models for rare disease concept normalization”. In: *Journal of the American Medical Informatics Association* 31.9 (June 2024), pp. 2076–2083. ISSN: 1527-974X. DOI: 10.1093/jamia/ocae133.

[Wei+22] Jason Wei et al. “Chain-of-Thought Prompting Elicits Reasoning in Large Language Models”. In: *Advances in Neural Information Processing Systems 35: Annual Conference on Neural Information Processing Systems 2022, NeurIPS 2022, New Orleans, LA, USA, November 28 - December 9, 2022*. Ed. by Sanmi Koyejo et al. 2022. URL: http://papers.nips.cc/paper%5C_files/paper/2022/hash/9d5609613524ecf4f15af0f7b31abca4-Abstract-Conference.html.

[WG21] Benedikt Wagner and Artur S. d’Avila Garcez. “Neural-Symbolic Integration for Fairness in AI”. In: *Proceedings of the AAAI 2021 Spring Symposium on Combining Machine Learning and Knowledge Engineering (AAAI-MAKE 2021), Stanford University, Palo Alto, California, USA, March 22-24, 2021*. Ed. by Andreas Martin et al.

Vol. 2846. CEUR Workshop Proceedings. CEUR-WS.org, 2021. URL: https://ceur-ws.org/Vol-2846/paper5.pdf.

[WGY24] Gail Weiss, Yoav Goldberg, and Eran Yahav. "Extracting automata from recurrent neural networks using queries and counterexamples (extended version)". In: *Mach. Learn.* 113.5 (2024), pp. 2877–2919. DOI: 10.1007/S10994-022-06163-2.

[Wil45] Frank Wilcoxon. "Individual Comparisons by Ranking Methods". In: *Biometrics Bulletin* 1.6 (1945), pp. 80–83. ISSN: 00994987. DOI: 10.2307/3001968. (Visited on 07/10/2025).

[WM97] David H. Wolpert and William G. Macready. "No free lunch theorems for optimization". In: *IEEE Trans. Evol. Comput.* 1.1 (1997), pp. 67–82. DOI: 10.1109/4235.585893.

[Wol90] WIlliam Wolberg. *Breast Cancer Wisconsin (Original)*. UCI Machine Learning Repository. 1990. DOI: 10.24432/C5HP4Z.

[Wu+18] Bichen Wu et al. "Shift: A Zero FLOP, Zero Parameter Alternative to Spatial Convolutions". In: *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 2018, pp. 9127–9135. DOI: 10.1109/CVPR.2018.00951.

[Xie+22] Xin Xie et al. "From Discrimination to Generation: Knowledge Graph Completion with Generative Transformer". In: *Companion of The Web Conference 2022, Virtual Event / Lyon, France, April 25 - 29, 2022*. Ed. by Frédérique Laforest et al. ACM, 2022, pp. 162–165. DOI: 10.1145/3487553.3524238.

[Xu+18] Jingyi Xu et al. "A Semantic Loss Function for Deep Learning with Symbolic Knowledge". In: *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*. Ed. by Jennifer G. Dy and Andreas Krause. Vol. 80. Proceedings of Machine Learning Research. PMLR, 2018, pp. 5498–5507. URL: http://proceedings.mlr.press/v80/xu18h.html.

[Xu+19] Kun Xu et al. “Cross-lingual Knowledge Graph Alignment via Graph Matching Neural Network”. In: *Proceedings of the 57th Conference of the Association for Computational Linguistics, ACL 2019, Florence, Italy, July 28- August 2, 2019, Volume 1: Long Papers*. Association for Computational Linguistics, 2019, pp. 3156–3161. DOI: 10.18653/V1/P19-1304.

[YAM22] Raciel Yera, Ahmad A. Alzahrani, and Luis Martínez. “Exploring Post-Hoc Agnostic Models for Explainable Cooking Recipe Recommendations”. In: *Knowledge-Based Systems* 251.C (Sept. 2022). ISSN: 0950-7051. DOI: 10.1016/j.knosys.2022.109216.

[Yan+23] Chengrun Yang et al. “Large Language Models as Optimizers”. In: *CoRR* abs/2309.03409 (2023). DOI: 10.48550/ARXIV.2309.03409.

[Yan+24] An Yang et al. *Qwen2 Technical Report*. 2024. arXiv: 2407.10671 [cs.CL]. URL: https://arxiv.org/abs/2407.10671.

[Yoo+07] Hee-Geun Yoon et al. “Ontology Population from Unstructured and Semi-structured Texts”. In: *Proceedings of The Sixth International Conference on Advanced Language Processing and Web Information Technology, ALPIT 2007, Luoyang, Henan, China, 22-24 August 2007*. IEEE Computer Society, 2007, pp. 135–139. DOI: 10.1109/ALPIT.2007.30.

[You23] Zhang Youheng. “A Historical Review and Philosophical Examination of the two Paradigms in Artificial Intelligence Research”. In: *European Journal of Artificial Intelligence and Machine Learning* 2.2 (Apr. 2023), pp. 24–32. DOI: 10.24018/ejai.2023.2.2.23.

[ZC20] Yongfeng Zhang and Xu Chen. “Explainable Recommendation: A Survey and New Perspectives”. In: *Foundations and Trends in Information Retrieval* 17.1 (Mar. 2020), pp. 1–101. DOI: 10.1561/1500000066.

[Zen+14] Daojian Zeng et al. “Relation Classification via Convolutional Deep Neural Network”. In: *COLING 2014, 25th International Conference on Computational Linguistics, Proceedings of the Conference: Tech-*

nical Papers, August 23-29, 2014, Dublin, Ireland. Ed. by Jan Hajic and Junichi Tsujii. ACL, 2014, pp. 2335–2344. URL: https://aclanthology.org/C14-1220/.

[Zha+22] Xiang Zhao et al. "An Experimental Study of State-of-the-Art Entity Alignment Approaches". In: *IEEE Transactions on Knowledge and Data Engineering* 34.6 (2022), pp. 2610–2625. DOI: 10.1109/TKDE.2020.3018741.

[Zha+23] Yue Zhang et al. "Siren's Song in the AI Ocean: A Survey on Hallucination in Large Language Models". In: *CoRR* abs/2309.01219 (2023). arXiv: 2309.01219.

[Zhu+15] Yukun Zhu et al. "Aligning Books and Movies: Towards Story-Like Visual Explanations by Watching Movies and Reading Books". In: *2015 IEEE International Conference on Computer Vision, ICCV 2015, Santiago, Chile, December 7-13, 2015*. IEEE Computer Society, 2015, pp. 19–27. DOI: 10.1109/ICCV.2015.11.

[Zhu+24] Yuqi Zhu et al. "LLMs for knowledge graph construction and reasoning: recent capabilities and future opportunities". In: *World Wide Web (WWW)* 27.5 (2024), p. 58. DOI: 10.1007/S11280-024-01297-W.

[ZY22] Yu Zhang and Qiang Yang. "A Survey on Multi-Task Learning". In: *IEEE Transactions on Knowledge and Data Engineering* 34.12 (2022), pp. 5586–5609. DOI: 10.1109/TKDE.2021.3070203.