



ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

DOTTORATO DI RICERCA IN

FISICA

Ciclo 38

Settore Concorsuale: 01/A2 – Geometria e Algebra

Settore Scientifico Disciplinare: MAT/03 – Geometria

ON DEEP LEARNING FOR GRAPH-STRUCTURED DATA

Presentata da: Michela Lapenna

Coordinatore Dottorato

Alessandro Gabrielli

Supervisore

Rita Fioresi

Cosupervisore

Giovanni Bruno

Esame finale anno 2026

Acknowledgments

I would like to thank Rita Fioresi, Giovanni Bruno, Francesco Faglioni, Ferdinando Zanchetta, Athanasios Tsamos, Bahman Gharesifard, and Caterina De Bacco. Working with you over these years has allowed me to grow deeply, both professionally and personally.

This thesis is dedicated to my friends (all around the world) and to my family, who have been a constant source of support during the most challenging moments. I love you.

Here's to the spirit of the Slow Science Manifesto, advocating for collaborative and sustainable research (slow-science.com).

Abstract

Graphs are a powerful and flexible data structure for representing relational data and are widely used to describe complex real-world systems. Graph Neural Networks (GNNs) have emerged as a leading machine learning paradigm for learning from such relational data, yet their operational principles, inherent limitations, and advantages over alternative models remain areas of active investigation. This thesis provides a multifaceted interrogation of GNNs to clarify their capabilities and boundaries.

First, to enhance the interpretability of neural networks, we develop a thermodynamic framework that analyzes weight dynamics during training and quantifies parameter importance through gradient activity. This framework enables structured pruning and we apply it to prune input features in GNNs, identifying prunable features as those that induce low mean squared gradient on the weights. We show that GNNs performance remains robust even after aggressively reducing high-dimensional features, with a significant decline only occurring at a critically low dimensionality.

We further pioneer the use of GNNs for the complex task of 3D multi-phase microstructure segmentation, a domain typically dominated by Convolutional Neural Networks. Our results demonstrate that the relational inductive bias of GNNs enables effective handling of irregular geometries and offers superior adaptability in data-scarce experimental settings. However, this success relies on well-engineered features, underscoring the fundamental GNN dependency on sufficiently expressive node-level attributes.

Finally, we address a critical gap in the literature by conducting a systematic comparison between GNNs and the interpretable paradigm of Probabilistic Graphical Models (PGMs). We show that PGMs are more robust in real-world scenarios characterized by low-dimensional or noisy node features and complex graph connectivity, such as heterophily. GNNs, in contrast, excel when high-dimensional features are available, and we find that their performance on heterophilic graphs significantly improves when features are strategically selected to incorporate information about the graph structure.

Collectively, this work advances our understanding of GNNs, positioning them as powerful feature-processing engines that use graph structure as a computational scaffold, rather than models that learn primarily from graph topology. The insights and methodologies developed herein highlight the advantages of GNNs while paving the way for more robust, interpretable, and data-efficient deep learning on graphs.

Contents

Abstract	2
1 Introduction	5
1.1 Overview and Motivation	5
1.2 Thesis Outline	8
2 Foundations of Deep Learning	10
2.1 Introduction to Deep Learning	10
2.2 Artificial Neural Networks	11
2.3 Loss Functions	13
2.4 Gradient Descent Optimization	16
2.5 Non-Convex Optimization Challenges	18
2.6 Insights from Statistical Physics	19
2.7 The Convolutional Mechanism	22
2.8 The Role of Residual Connections	26
3 Foundations of Graph Machine Learning	30
3.1 Graph Theory Elements	30
3.2 Geometric Inductive Biases	33
3.3 The Message Passing Neural Network Mechanism	34
3.4 From Node Embeddings to Link Predictions	38
3.5 Homophily Bias and Oversmoothing	39
3.6 Probabilistic Graphical Models	42
3.7 Generative vs. Discriminative Link Prediction	45
3.8 Statistical Inference in Graphical Models	46
3.9 Modeling Graph Communities	51
3.10 The Stochastic Block Model	52
4 Thermodynamic Approach to Neural Network Pruning	58
4.1 The Interpretability Challenge in Deep Learning	59
4.2 From Classical Dynamics to Network Temperature	61
4.3 SGD-Temperature as a Pruning Criterion	64

4.4	Comparison of Weight Dynamics in SGD and Adam	67
4.5	Experimental Setup	68
4.6	Pruning Graph Neural Network Input Features	70
4.7	Feature Redundancy via Cosine Similarity	75
4.8	Pruning Embedding Dimensions	77
4.9	Comparison with Pruning Convolutional Filters	78
4.10	Discussion and Conclusions	80
5	Graph-Based Segmentation of 3D XCT Microstructures	83
5.1	Segmenting Multi-Phase XCT via Deep Learning	84
5.2	Spatial vs. Relational Inductive Biases	86
5.3	GNN Advantages for Microstructures Segmentation	87
5.4	Synthetic and Experimental Data	89
5.5	Input Representation: Graph and Node Features	92
5.6	GNN Architectural Challenges	94
5.7	From Proof of Concept to ViG Architecture	95
5.8	Training and Evaluation	98
5.9	Segmentation Performance and Visual Quality	100
5.10	Error Analysis and Fine-Tuning Performance	104
5.11	Parameter Efficiency and Computational Cost	106
5.12	Discussion and Conclusions	107
6	Two Learning Paradigms for Graph-Structured Data	109
6.1	Conceptual Foundations	110
6.2	Comparison Framework	112
6.3	Models and Experimental Protocol	114
6.4	Synthetic and Real-World Datasets	120
6.5	The Role of Input Features	123
6.6	Robustness to Noise	127
6.7	Resilience to Increasing Heterophily	131
6.8	Interpretability	133
6.9	Computational Complexity	136
6.10	Discussion and Conclusions	137
6.11	PGM vs. GNN scatterplots	139
7	Discussion and Conclusions	144
7.1	Summary of Contributions	144
7.2	Future Directions	146
	Bibliography	148

Chapter 1

Introduction

1.1 Overview and Motivation

In recent years, Graph Neural Networks (GNNs) [1, 2] have emerged as a central paradigm in machine learning, capable of operating on relational and irregular data. The key strength of GNNs lies in the flexibility of graphs: unlike the regular grids used in Convolutional Neural Networks (CNNs) [3, 4, 5], or the sequential structures exploited by Recurrent Neural Networks (RNNs) [6, 4, 7] and Transformers [8], graphs allow arbitrary connectivity patterns. This flexibility makes GNNs particularly well-suited for domains such as molecular property prediction, physical simulations, recommendation systems, and social network analysis (see Fig. 1.1 for an example).

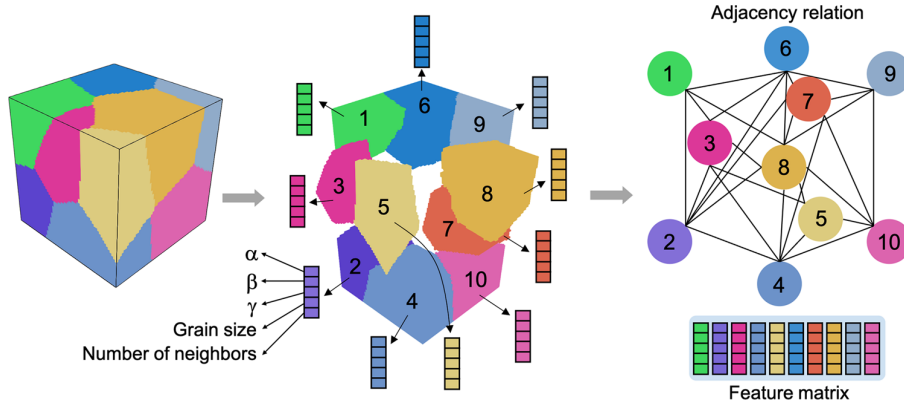


Figure 1.1: GNN framework for polycrystalline materials: grains are represented as graph nodes with features, while edges capture physical interactions [9]. The learned microstructure embedding enables accurate prediction of macroscopic properties such as magnetostriction.

The modern formulation of GNNs was introduced by Gori et al. [1] (2005) and Scarselli et al. [2] (2009), and subsequently unified under the Neural Network Message Passing framework by Gilmer et al. [10] (2017). Since then, a wide variety of architectures have been proposed, including Graph Convolutional Networks (GCNs) [11], GraphSAGE [12], and Graph Attention Networks (GATs) [13], establishing GNNs as a versatile tool for

learning from graph-structured data.

GNNs serve as a powerful unifying framework that extends and generalizes key ideas from well-known deep learning architectures [4]. For instance, CNNs can be viewed as a specialized instance of a GNN operating on a grid-structured graph. In this analogy, a convolutional kernel defines a fixed spatial pattern for message aggregation from a local pixel neighborhood. Furthermore, the Transformer architecture [8] itself can be interpreted as a form of message passing on a fully-connected graph, where each token attends to and exchanges messages with every other token in the sequence [14].

Despite their flexibility, GNNs still face several fundamental challenges that restrict their broader understanding and deployment. First, as common to deep learning models, their learned representations are not directly interpretable, because embeddings live in high-dimensional spaces where it is difficult to understand what each dimension represents, and the way interactions in the graph are encoded within these embeddings remains unclear. One has to make ad-hoc processing a posteriori to allow for interpretability [15] or design GNN models specifically to optimize for it [16]. Second, GNN architectures suffer from known structural limitations: they struggle to distinguish certain graph topologies due to the expressive limit imposed by the Weisfeiler-Lehman test [17, 18], and they are prone to phenomena such as oversmoothing [19], where node embeddings become indistinguishable across layers, and oversquashing [20], where long-range information is compressed into fixed-size vectors, leading to loss of signal. Furthermore, the oversmoothing phenomenon has been specifically identified as responsible for the poor performance of GNNs in heterophilic graph settings [21, 22, 23]. Lastly, GNNs are rarely evaluated against greedy algorithms [24] or structured probabilistic models [25], which leaves unresolved the question of when they offer a clear advantage over simpler or more interpretable alternatives.

Despite its potential, research on deep learning for graphs is still in its early stages, with the theoretical foundations, operational principles, and practical boundaries of GNNs representing major open challenges. To establish a robust understanding of their capabilities and limits, this thesis provides a structured interrogation of GNNs from three complementary viewpoints: *theoretical*, *practical*, and *comparative*.

From a *theoretical* perspective, we address the “black box” problem in deep learning by developing a novel thermodynamic framework that analyzes weights dynamics during training. This framework probes what information GNNs prioritize by examining the gradients induced on the weights, enabling the quantification of feature importance and redundancy, which is important for feature engineering [26]. As a core contribution, we apply this interpretability framework directly to structured pruning, moving beyond traditional magnitude-based approaches. Although the resulting methodology shows significant generality, extending naturally to diverse neural architectures, its application to GNNs raises a fundamental question: “How can we systematically assess feature importance to jointly guide feature engineering and model compression in GNNs?”

From a *practical* perspective, we challenge conventional GNNs usage by employing them for the computer vision task of 3D multi-phase microstructure segmentation [27, 28]. Our goal is to understand how the relational inductive bias of GNNs handles the complex and irregular geometries of microstructures compared to the fixed spatial locality of CNNs. We also evaluate how this relational bias helps GNNs in alleviating data scarcity in experimental settings compared to CNNs. Furthermore, the constrained nature of our experimental dataset, where only scalar grayscale values are available as features, directly allows us to confront the reliance of GNNs on the dimensionality and richness of node information. This investigation asks the central question: “Can the relational inductive bias of GNNs unlock superior performance on geometrically complex segmentation tasks, and what principles should guide feature engineering in data-scarce experimental settings to realize this potential?”

From a *comparative perspective*, we address a critical gap in the literature: the lack of a comprehensive, head-to-head comparison between GNNs and the well-established world of Probabilistic Graphical Models (PGMs) [29, 30]. Despite their shared focus on network-structured data, these two methodological families have evolved in relative isolation. Through controlled experiments probing feature dimensionality, robustness to feature and edge noise, and resilience to graph heterophily, we present a systematic analysis to delineate the strengths and weaknesses of each paradigm in real-world network scenarios. Our findings directly answer a critical question for the field: “How do these two paradigms leverage graph structure and node attributes, and when should one use a complex GNN over a more interpretable PGM, and vice versa?”

The convergence of evidence from our triangulated investigation clarifies the paramount role of input features in GNNs. Our theoretical framework leverages structured pruning to show that GNNs maintain strong performance even with a greatly reduced feature set when initial features are high-dimensional, although significant decline occurs at critically low dimensionality. Our practical application to microstructure segmentation demonstrates that the relational inductive bias of GNNs enables highly successful 3D segmentation and effective adaptation to limited data. However, in feature-scarce settings, their dependency on node features necessitates clever feature engineering. Finally, our comparative analysis with PGMs systematically delineates that GNNs excel when node features are rich and reliable, while their struggle with complex graph connectivity, such as in heterophilic graphs, can be mitigated through more careful feature selection.

Together, these findings posit that while GNNs are powerful function approximators, they fundamentally act as feature-processing engines. They use the graph structure primarily as a computational skeleton for message passing, rather than truly learning from its connective patterns. Ultimately, this work provides insights to move towards a more structured deep learning on graphs, where models are not only powerful but are also architected to be robust, interpretable, and truly capable of exploiting graph topology.

1.2 Thesis Outline

In this thesis, we present three research works, each discussed in a dedicated chapter. The corresponding three publications are listed below with full bibliographic details:

- **M. Lapenna**, F. Faglioni, R. Fioresi, G. Bruno, “Temperature-based pruning for input features in Graph Neural Networks”, *The European Physical Journal Plus*, vol. 140, no. 9, p. 880, 2025. [\[31\]](#)
- **M. Lapenna**, A. Tsamos, F. Faglioni, R. Fioresi, F. Zanchetta, G. Bruno, “Vision GNN (ViG) architecture for a fine-tuned segmentation of a complex Al-Si metal matrix composite XCT volume”, *Journal of Materials Science*, vol. 60, no. 16, pp. 6907–6921, 2025. [\[32\]](#)
- **M. Lapenna**, C. De Bacco, “How do probabilistic graphical models and graph neural networks look at network data?”, *Journal of Physics: Complexity*, vol. 7, no. 1, p. 015008, 2026. [\[33\]](#)

The first two works build upon earlier published studies ([\[34\]](#) and [\[35\]](#) for the thermodynamic analogy and [\[36\]](#) for the segmentation task). Below, we outline the content and objectives of each chapter. The first two chapters provide the necessary background material, while the subsequent three analyze the three publications listed above, which constitute the core contributions of this thesis.

Before focusing on graph-structured data, Chapter [2](#) provides a foundational overview of deep learning, beginning with Artificial Neural Networks (ANNs) and the gradient-based optimization algorithms used to train them. It explores specialized architectures like CNNs for spatial data and RNNs for sequences, while also drawing important theoretical parallels to statistical physics to explain training dynamics and generalization.

Following this, Chapter [3](#) provides a comprehensive introduction to machine learning on graphs. It begins with fundamental graph concepts and notation before presenting the two methodological families of GNNs and PGMs. For GNNs, we first analyze the role of symmetry and invariance in their inductive biases. Then, we present the Neural Network Message Passing framework [\[10\]](#) and detail its inherent limitations, such as homophily bias and oversmoothing effects. To provide a concrete understanding, we also discuss examples of commonly used GNN layers. We then contrast the discriminative, representation-learning approach of GNNs with the probabilistic framework of PGMs, focusing on generative models with latent variables and their associated statistical inference algorithms. This is concretized through the Stochastic Block Model (SBM) [\[37, 38, 39, 40\]](#), a fundamental generative model for community detection, for which we detail two specific variants used in our comparative analysis.

After providing the necessary background material, the thesis turns to its original contributions. First, in Chapter 4, we present the thermodynamic modeling exploited for introducing a new gradient-based pruning method, which we employ to prune node features and embedding dimensions in GNNs. This work follows and extends our previously published studies [34, 35]. Using the concept of neural network temperature, a measure linked to the gradient activity due to the loss function minimization at equilibrium, we identify and discard weight structures associated with small mean squared gradient, without degrading performance. Crucially, we identify that retaining high-redundancy features, those that are more correlated with other features, is a more effective strategy. This method, tested on multiple benchmark datasets and GNN architectures, offers a promising physics-inspired alternative to standard magnitude-based pruning techniques.

Next, in Chapter 5, we explore the application of GNNs to a challenging computer vision task: the voxel-level segmentation of complex 3D microstructures from X-ray computed tomography (XCT) scans. Building upon our previously published proof of concept study [36], we develop a Vision GNN (ViG) architecture [32] initially trained on synthetic data and then fine-tuned on experimental samples. This approach achieves segmentation accuracy comparable to U-Nets [41] on synthetic data, though U-Nets maintain higher performance on experimental data. The GNN architecture, however, demonstrates superior adaptability and robustness when fine-tuning on limited experimental data. Additionally, GNNs significantly reduce the number of trainable parameters compared to U-Nets, even though their reliance on sparse message passing, which is not as efficiently optimized in current libraries as CNNs dense operations, incurs longer training times.

As third contribution, in Chapter 6, we benchmark GNNs against interpretable PGMs, specifically SBMs, on a link prediction task. Our experiments on synthetic and real-world datasets investigate how both frameworks leverage input features and graph structure. We show that PGMs outperform GNNs when input features are low-dimensional or noisy, a common scenario in real-world networks where node attributes are often limited or unreliable. Furthermore, PGMs exhibit greater robustness to increasing graph heterophily. Interestingly, we find that GNN performance on heterophilic graphs can be significantly improved through more informed feature selection. When high-quality, high-dimensional input features are available and the graph is assortative, GNNs can achieve performance comparable to, or exceeding, that of PGMs. Additionally, we compare computational complexity and interpretability of both paradigms, clarifying when deep learning on graphs offers clear advantages over classical probabilistic methods.

Finally, Chapter 7 synthesizes the key findings from each contribution into a broader discussion on the explored research directions. It underscores the strengths of GNNs relative to CNNs and PGMs, with a specific focus on their capabilities for data fusion tasks. Building on the thesis findings, the chapter concludes by outlining promising future research directions for enhancing GNN design and extending their range of applications.

Chapter 2

Foundations of Deep Learning

This chapter establishes the theoretical foundations and methodological frameworks that underpin Deep Learning, a transformative paradigm enabling machines to learn complex hierarchical representations directly from raw data. The discussion begins in Sec. 2.1 with a high-level overview of deep learning, contrasting it with traditional machine learning approaches. We then systematically examine the mathematical principles of Artificial Neural Networks (ANNs) in Sec. 2.2, starting from the simple perceptron and progressing to multi-layer architectures. Sec. 2.3 covers the loss functions that quantify prediction error, while Sec. 2.4 details the gradient-based optimization algorithms that enable effective training. The chapter subsequently explores specialized architectures, including Convolutional Neural Networks (CNNs) [3, 4, 5] for spatial data in Sec. 2.7 and Recurrent Neural Networks (RNNs) [6, 4, 7] for sequential data in Sec. 2.8. A key focus is placed on critical innovations like skip connections, which stabilize the training of deep networks by preserving gradient flow. Furthermore, Sec. 2.6 examines the learning dynamics and generalization of these high-dimensional models through the lens of statistical physics, providing a cohesive theoretical perspective on their behavior.

2.1 Introduction to Deep Learning

Deep learning [4] is an advanced branch of machine learning characterized by models composed of many sequential processing layers, a design that gives rise to the term “deep”. This multi-layered architecture enables the system to learn directly from large-scale data by automatically detecting and hierarchically organizing patterns into increasingly abstract representations. This stands in contrast to traditional machine learning methods, which often rely on “shallow” architectures (e.g., decision trees [42], or support vector machines [43]) and, most importantly, require extensive manual effort to preprocess the data.

In traditional machine learning, domain experts must carefully design and select relevant features before the model can be trained. In deep learning, instead, the

network learns its own features hierarchically [44, 45]: early layers detect simple patterns, intermediate layers combine these into more complex structures, and deeper layers assemble these into more abstract representations. The capability to learn feature hierarchies directly from data makes deep learning exceptionally powerful for tasks such as image and speech recognition, where designing features by hand would be difficult or impractical.

The rapid advancement of deep learning has been largely enabled by increased computational power, particularly through the use of GPUs. The resource-intensive nature of training deep neural networks on massive datasets requires vast parallel processing capabilities, a task for which GPUs are uniquely well-suited due to their efficiency in performing the core matrix operations central to these algorithms. This hardware revolution has drastically reduced training times, enabling the development of models with millions or even billions of parameters and unlocking new complexities in Artificial Intelligence applications.

Another critical factor driving the success of deep learning is the modern availability of large datasets, driven by the widespread digitalization of society and declining cost of digital storage. Deep neural networks, particularly in fields like computer vision and natural language processing, inherently require massive amounts of data to learn complex patterns and avoid overfitting. However, when learning is supervised, a fundamental limitation is the need for accurately labeled data, a process that is often extremely time-consuming and limited by scarce expert knowledge. This labeling constraint has led to the exploration of synthetic data generation techniques, which utilize generative models to create realistically labeled samples from existing data, thereby bypassing the need for manual annotation.

It is important to note that this revolution comes with significant costs: extreme energy consumption, reliance on rare materials for hardware, and substantial environmental impacts, including water usage for cooling and carbon emissions, that often affect third countries disproportionately. To mitigate these effects, it is crucial to move toward more efficient algorithm design. Indeed, due to the black-box nature of deep learning models, current practices rely heavily on costly grid searches for hyperparameter optimization. Developing more transparent and theoretically grounded methods is essential to guide deep learning toward a more sustainable future.

2.2 Artificial Neural Networks

Artificial Neural Networks (ANNs) are computational models inspired by the biological neural networks of the human brain. They consist of interconnected layers of nodes, or neurons, each combining input signals, applying a non-linear transformation, and passing the result to the next layer. The simplest form of an ANN is the perceptron, a single-layer network initially proposed by Frank Rosenblatt in the 1950s [46] (see Fig. 2.1).

Mathematically, a perceptron takes a set of input values $\mathbf{x} = (x_1, x_2, \dots, x_n)$, multiplies them by corresponding weights $\mathbf{w} = (w_1, w_2, \dots, w_n)$, and applies a weighted sum followed by a non-linear activation function σ to produce an output y :

$$y = f \left(\sum_{i=1}^n w_i x_i + b \right) \quad (2.2.1)$$

where b is the bias term. The perceptron’s original purpose was binary classification, which led to the use of the Heaviside step function for σ . This function produces a binary output (e.g., 0 or 1) depending on whether the weighted sum exceeds a threshold, mimicking the “firing” of a biological neuron. However, the step function has two critical limitations. First, it is non-differentiable at zero and has a zero derivative elsewhere, which prevents the use of gradient-based learning algorithms like backpropagation [47], a technique used to efficiently calculate gradients for parameter updates that we will discuss in detail in Sec. 2.4. Second, its binary output prevents the small, continuous weight adjustments needed to learn complex functions, ultimately restricting the perceptron to solving only linearly separable problems.

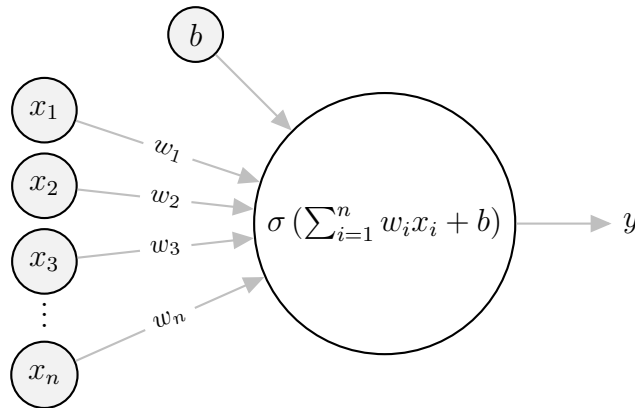


Figure 2.1: Illustration of a single perceptron with inputs $\mathbf{x}$, weights $\mathbf{w}$, bias b , and output y .

The true power of neural networks is realized by combining many perceptrons into a Multi-Layer Perceptron (MLP), also called fully-connected network. The key innovation in MLP is the introduction of one or more “hidden” layers between the input and output (see Fig. 2.2). Crucially, without non-linear activations, the entire network could be mathematically reduced to a single layer, losing its representational power. Commonly employed non-linear functions include the Sigmoid [48] and ReLU [49]. These functions are differentiable, enabling network-wide training via backpropagation, and they introduce the necessary complexity into the network’s calculations. This allows the MLP to model highly complex, non-linear relationships, transforming it from a simple linear classifier into a universal function approximator.

Mathematically, the output of an MLP with L layers is the result of a chain of function

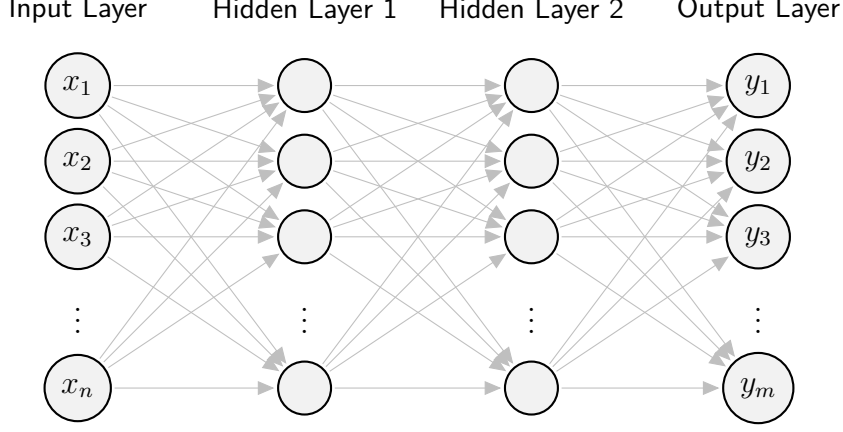


Figure 2.2: Diagram of a multilayer perceptron with input layer, two hidden layers, and output layer.

compositions:

$$y = \sigma^{(L)} (\mathbf{W}^{(L)} \dots \sigma^{(1)} (\mathbf{W}^{(1)} \mathbf{x} + \mathbf{b}^{(1)}) \dots + \mathbf{b}^{(L)}) \quad (2.2.2)$$

where $\mathbf{W}^{(l)}$ and $\mathbf{b}^{(l)}$ are the weight matrix and bias vector for the l -th layer, respectively. This multi-layered design of non-linear transformations is the core of deep learning, and the MLP serves as the foundational architecture from which all other complex neural networks have evolved.

2.3 Loss Functions

Machine learning tasks are broadly categorized into supervised and unsupervised learning based on the availability of labels and the nature of the task. *Unsupervised learning* aims to discover the inherent structure within unlabeled data. This encompasses objectives like clustering (e.g., community detection in networks, see Sec. 3.9), dimensionality reduction, and data generation. In contrast, *supervised learning* tackles the distinct problem of learning a mapping from inputs to outputs, such as predicting a discrete category in classification or a continuous value in regression. This section focuses specifically on the supervised setting, with a particular emphasis on classification tasks.

In supervised learning we are given a dataset $\mathcal{D} = \{(\mathbf{x}_i, y_i)\}_{i=1}^N$, where each input vector $\mathbf{x}_i \in \mathbb{R}^d$ contains the features of the i -th example (for instance, pixel values of an image or sensor measurements) and each label y_i is the corresponding target value, e.g., the discrete class index. We can define a neural network as a parametric function $\mathbf{x} \mapsto f_{\mathbf{w}}(\mathbf{x})$ that outputs a *score* (or a vector of scores, known as *logits*). The function is parameterized by $\mathbf{w}$, which encompasses all trainable weights and biases. To learn these parameters we specify a loss function $L(\mathbf{w})$ that quantifies the error between the model's predictions and the true targets, and we employ an optimizer to update $\mathbf{w}$ so as

to minimize this loss (see Sec. 2.4).

It is often convenient to view the network probabilistically. For each input $\mathbf{x}$ the network is assumed to define a conditional distribution $p_{\mathbf{w}}(y \mid \mathbf{x})$, the probability of observing the label y given that input. Under this assumption the likelihood of the parameters on the entire dataset is:

$$\mathcal{L}(\mathbf{w}) = \prod_{i=1}^N p_{\mathbf{w}}(y_i \mid \mathbf{x}_i),$$

and taking the negative logarithm gives the negative log-likelihood (NLL):

$$L(\mathbf{w}) = - \sum_{i=1}^N \log p_{\mathbf{w}}(y_i \mid \mathbf{x}_i)$$

Minimizing this quantity is exactly maximum-likelihood estimation: it finds the parameters that make the observed data most probable.

For binary classification tasks, where each target y_i takes values in $\{0, 1\}$, a natural generative assumption is that each label follows a Bernoulli distribution with success probability $\hat{y}_i = p_{\mathbf{w}}(y_i = 1 \mid \mathbf{x}_i)$ predicted by the network. Using the logistic sigmoid:

$$\sigma(z) = \frac{1}{1 + e^{-z}}, \quad \hat{y}_i = \sigma(\mathbf{w}^\top \mathbf{x}_i),$$

the negative log-likelihood (NLL) simplifies to the Binary Cross-Entropy (BCE) loss [50]:

$$L_{\text{BCE}}(\mathbf{w}) = - \frac{1}{N} \sum_{i=1}^N \left[y_i \log \hat{y}_i + (1 - y_i) \log(1 - \hat{y}_i) \right],$$

which serves as the standard objective function for logistic regression and for neural networks with a single probabilistic output unit.

When there are C classes, each label is represented as a one-hot vector $\mathbf{y}_i \in \{0, 1\}^C$, where $y_{i,c} = 1$ if example i belongs to class c and 0 otherwise. For example, in a classification task with $C = 3$ classes (*car*, *bicycle*, and *airplane*) the one-hot vectors could be: a *car* represented as $[1, 0, 0]$, a *bicycle* as $[0, 1, 0]$, and an *airplane* as $[0, 0, 1]$. Thus, for an image i of a bicycle, the label vector is $\mathbf{y}_i = [0, 1, 0]$, meaning $y_{i,1} = 1$ while all other elements are 0.

The network outputs a vector of logits $\mathbf{z}_i$, which are converted into class probabilities using the softmax function:

$$\hat{y}_{i,c} = \frac{e^{z_{i,c}}}{\sum_{k=1}^C e^{z_{i,k}}}$$

The corresponding negative log-likelihood yields the Categorical Cross-Entropy (CCE)

loss [50]:

$$L_{\text{CCE}}(\mathbf{w}) = -\frac{1}{N} \sum_{i=1}^N \sum_{c=1}^C y_{i,c} \log \hat{y}_{i,c}$$

This probabilistic view connects directly to information theory. Let P denote the true label distribution and Q the model’s predicted distribution. The Kullback-Leibler (KL) divergence [51]:

$$D_{\text{KL}}(P \parallel Q) = \sum_{c=1}^C P(c) \log \frac{P(c)}{Q(c)},$$

measures the extra information required to encode samples from P when using a code optimized for Q . The cross-entropy between P and Q :

$$H(P, Q) = -\sum_{c=1}^C P(c) \log Q(c),$$

satisfies

$$H(P, Q) = H(P) + D_{\text{KL}}(P \parallel Q),$$

where $H(P)$ is the Shannon entropy of the true distribution:

$$H(P) = -\sum_{c=1}^C P(c) \log P(c)$$

This term $H(P)$ is constant during training because it depends solely on the *true* underlying distribution of the labels in the dataset, which is fixed and does not change as the model’s parameters $\mathbf{w}$ are updated. Since $H(P)$ remains constant during training, minimizing the cross-entropy loss is exactly the same as minimizing the KL divergence between the empirical label distribution and the model’s predicted distribution. In this way, cross-entropy losses used for binary or multi-class classification can be interpreted both as negative log-likelihoods of simple generative models and as information-theoretic objectives that reduce the divergence between the model and the true data distribution.

While this section has focused on loss functions for supervised learning, analogous objectives are equally central to unsupervised learning. In the absence of labeled outputs y_i , the loss function must be designed to discover intrinsic structure within the input data $\{\mathbf{x}_i\}$ itself. This leads to objectives that quantify goals such as the reconstruction error in Autoencoders [52, 53], which measures fidelity in data compression, or the adversarial loss in Generative Adversarial Networks [54]. By minimizing these objectives, the model learns meaningful patterns and representations directly from the data itself.

2.4 Gradient Descent Optimization

To transition from the architectural blueprint of an MLP to more specialized deep learning architectures, we must define the process by which they learn from data. This process, known as training, typically involves optimizing the network’s parameters to minimize its prediction error (supervised learning) or reconstruction error (unsupervised learning). The ultimate goal, however, is for the model to generalize, meaning to perform well on unseen data.

A central challenge in this paradigm is overfitting, which occurs when a model learns patterns specific to the data used for training, failing to capture the broader underlying distribution. To estimate this generalization performance and guide the training process effectively, the available data is typically partitioned into three distinct subsets. The *training set* is used to directly adjust the model’s parameters by minimizing the loss function. The *validation set* provides an unbiased evaluation of the model’s performance on unseen data during training. This set serves to tune the model’s hyperparameters and to determine the optimal point to stop training, thereby directly combating overfitting. Finally, the *test set* is held in reserve until the training is complete and it is used for a final assessment of the model’s generalization error.

The most fundamental method for this optimization is Gradient Descent (GD). The core idea is to navigate the typically non-convex, high-dimensional loss landscape (see Sec. 2.5) by moving in the direction of steepest descent. The update rule for the parameters is:

$$\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} - \eta \nabla_{\mathbf{w}} L(\mathbf{w}^{(t)}) \quad (2.4.1)$$

where $\mathbf{w}^{(t)}$ represents the weights at iteration t , η is the learning rate (a crucial hyperparameter controlling the step size), and $\nabla_{\mathbf{w}} L(\mathbf{w}^{(t)})$ is the gradient of the loss function with respect to the weights. Computing this gradient efficiently for deep networks is made possible by the backpropagation algorithm [47]. Backpropagation applies the chain rule of calculus to compute the gradient for each weight layer-by-layer, starting from the output and moving backwards to the input.

Computing the gradient using the entire dataset at once is computationally expensive for large-scale problems. To improve efficiency, modern gradient descent variants estimate the gradient using randomly sampled subsets of data instead of the complete dataset. The fundamental algorithm in this family is Stochastic Gradient Descent (SGD) [55], which, in its simplest form, uses a single randomly selected data point per update. While computationally efficient, this approach suffers from high-variance updates that can lead to unstable convergence. The standard practice is therefore a compromise: mini-batch SGD [56]. This method processes small random subsets, or mini-batches, at each iteration. Formally, a mini-batch $\mathcal{B}$ of size B is sampled, typically without replacement, from the shuffled training set. This approach balances computational efficiency with convergence

stability, as larger batches provide more accurate gradient estimates while remaining memory-efficient compared to full-batch processing. The update rule for mini-batch gradient descent is:

$$\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} - \frac{\eta}{B} \sum_{b=1}^B \nabla_{\mathbf{w}} L(\mathbf{w}^{(t)}, \mathcal{B}_b) = \mathbf{w}^{(t)} - \eta \nabla_{\mathbf{w}} L_{\mathcal{B}}(\mathbf{w}^{(t)}) \quad (2.4.2)$$

where $L_{\mathcal{B}}(\mathbf{w}^{(t)}) = \frac{1}{B} \sum_{b=1}^B L(\mathbf{w}^{(t)}, \mathcal{B}_b)$ is the average loss over the batch.

This process of iterating through mini-batches continues until the entire dataset has been processed. Formally, one epoch is completed after $\lceil N/B \rceil$ (i.e., the smallest integer greater than or equal to N/B) updates, where N is the dataset size. Training typically requires multiple epochs for the model to converge.

To improve the convergence properties of SGD, advanced optimizers incorporate the concept of momentum [57]. Momentum accelerates updates in directions of persistent reduction of the loss and dampens oscillations by accumulating a moving average of past gradients, akin to a ball rolling downhill. The update rule with momentum is:

$$\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} - \eta \mathbf{v}^{(t+1)} \quad \text{with} \quad \mathbf{v}^{(t+1)} = \beta \mathbf{v}^{(t)} + (1 - \beta) \nabla_{\mathbf{w}} L_{\mathcal{B}}(\mathbf{w}^{(t)}) \quad (2.4.3)$$

where $\mathbf{v}$ is the velocity vector and $\beta \in [0, 1]$ is the momentum coefficient (typically close to 1, e.g., 0.9).

The most popular modern optimizer is Adam (Adaptive Moment Estimation) [58], which combines the concepts of momentum and per-parameter adaptive learning rates. Adam maintains two moving averages of the gradients: the first moment (the mean, $\mathbf{f}$) and the second moment (the variance, $\mathbf{s}$). These are updated at each step t using decay rates β_1 and β_2 (typically set to $\beta_1 = 0.9$ and $\beta_2 = 0.999$), which control the exponential decay of these moving averages:

$$\mathbf{f}^{(t)} = \beta_1 \mathbf{f}^{(t-1)} + (1 - \beta_1) \nabla_{\mathbf{w}} L_{\mathcal{B}}(\mathbf{w}^{(t)}), \quad \mathbf{s}^{(t)} = \beta_2 \mathbf{s}^{(t-1)} + (1 - \beta_2) (\nabla_{\mathbf{w}} L_{\mathcal{B}}(\mathbf{w}^{(t)}))^2 \quad (2.4.4)$$

Since these estimates are biased towards zero at the beginning of training, Adam applies bias correction:

$$\hat{\mathbf{f}}^{(t)} = \frac{\mathbf{f}^{(t)}}{1 - \beta_1^t}, \quad \hat{\mathbf{s}}^{(t)} = \frac{\mathbf{s}^{(t)}}{1 - \beta_2^t} \quad (2.4.5)$$

Finally, the parameters are updated by scaling the learning rate η for each parameter according to this ratio:

$$\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} - \eta \cdot \frac{\hat{\mathbf{f}}^{(t)}}{\sqrt{\hat{\mathbf{s}}^{(t)}} + \epsilon} \quad (2.4.6)$$

where ϵ is a small scalar for numerical stability.

This optimization-based approach has been the driving force behind most recent large-scale machine learning successes. However, one powerful class of methods has been largely left behind by these advances: Bayesian approaches [59]. This is primarily because traditional Bayesian methods, such as Markov Chain Monte Carlo (MCMC) [60, 61], are computationally prohibitive at scale, as each iteration typically requires processing the entire dataset. Despite this, Bayesian methods remain highly appealing due to their ability to quantify uncertainty in learned parameters and provide inherent protection against overfitting. We will explore core Bayesian methodologies for Probabilistic Graphical Models [29, 30] in Sec. 3.8.

2.5 Non-Convex Optimization Challenges

The fundamental objective of training a neural network is to find a parameter configuration $\mathbf{w}$ that minimizes a loss function $L(\mathbf{w})$ and is able to generalize to unseen data, avoiding overfitting. While gradient descent and its variants provide a mechanism for this minimization, the nature of the loss function itself presents the primary challenge. Indeed, the loss functions of deep neural networks are typically profoundly non-convex, which results in a complex, high-dimensional landscape characterized by a proliferation of suboptimal critical points (e.g., local minima, saddle points, and flat regions).

Despite the theoretical NP-hardness of this problem [62], gradient-based methods often succeed in practice. However, the optimization and generalization properties of deep neural networks are highly dependent on structural and algorithmic choices. Specifically, certain architectural choices (e.g., skip connections) facilitate training by shaping a more tractable loss landscape, while well-tuned hyperparameters (e.g., batch size, learning rate, optimizer) guide the optimization process toward minimizers whose geometry (i.e., their sharpness/flatness) leads to improved generalization.

Empirical studies of the loss landscape have revealed several key behaviors that underpin these effects [63]. Notably, as network depth increases, the loss landscape undergoes a sharp transition from being relatively smooth and nearly convex to becoming highly chaotic. This transition correlates with a severe degradation in both generalization performance and trainability.

Architectural choices, most prominently skip connections, which will be introduced in detail in Sec. 2.8, counteract this effect by promoting flatter minimizers and preserving a more tractable landscape geometry (see Fig. 2.3). Their effectiveness was first demonstrated in the ResNet architecture [64], which successfully enabled the training

of networks with hundreds of layers. Since then, this principle has been empirically validated as a general-purpose strategy across diverse neural network families, including Convolutional Neural Networks (CNNs) [64], Transformers [65], and Graph Neural Networks (GNNs) [66].

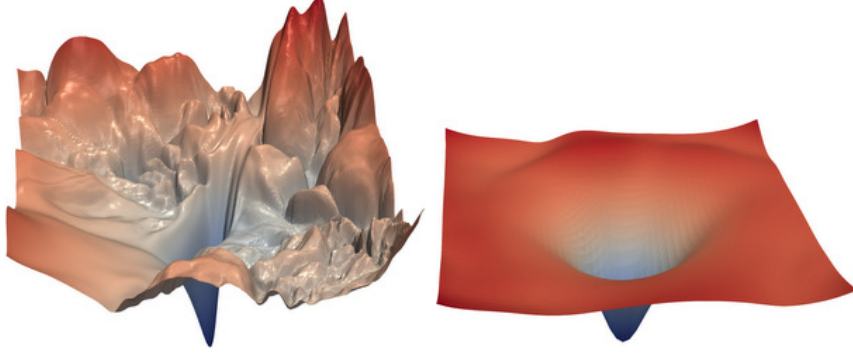


Figure 2.3: The loss surfaces of ResNet-56 without (left) and with (right) skip connections (visualizations in [63])

Furthermore, analyses of optimization trajectories suggest that, despite the high-dimensional parameter space, the path taken by gradient descent often lies on a low-dimensional manifold. This indicates that the search space, though globally non-convex, contains large regions that are locally well-behaved and nearly convex, effectively guiding the optimizer toward good solutions. Therefore, the efficacy of gradient descent depends not only on the algorithm itself but also on the geometry of the loss landscape, which is shaped by architectural design and parameter initialization. Understanding this interaction is crucial for developing models that are both easier to train and better at generalization (see Sec. 2.6 for insights on loss landscapes from a statistical physics perspective).

2.6 Insights from Statistical Physics

The theoretical modelling of ANNs finds profound grounding in statistical physics, a interdisciplinary connection with deep historical roots [67]. This intellectual cross-fertilization gained momentum in the mid-1980s as Artificial Intelligence shifted from logic-based systems toward statistical learning approaches. Seminal developments include Hopfield’s influential model of associative memory [68, 69], which initiated the application of spin glass theory concepts to neural network modeling, and Gardner’s analytical framework for precisely characterizing learning curves [70, 71], i.e., the evolution of training and generalization errors with increasing training examples.

Although peripheral to mainstream machine learning theory, this statistical physics perspective offers a framework for analyzing the high-dimensional systems that underpin deep learning. It provides crucial insights into fundamental puzzles of deep learning, such as the “overparameterization paradox” [72]. This refers to the standard yet counter-intuitive

practice of training neural networks in an overparameterized regime, where models with more parameters than training samples generalize excellently instead of overfitting catastrophically.

Glassiness of Deep Learning. Training a deep neural network via SGD involves navigating a non-convex, high-dimensional loss landscape, characterized by a profusion of local minima and saddle points analogous to the complex energy landscapes of spin glasses [73]. This process can be interpreted as the motion of a stochastic particle through a rugged terrain, searching for low-loss configurations. The connection to statistical physics is profound, as this field has long studied complex landscapes and their relation to dynamical behaviour.

A significant line of theoretical inquiry seeks to understand how the noise inherent in SGD, introduced by mini-batch sampling (see Sec. 2.4), shapes the learning process and influences a model’s ability to generalize. This effort often proceeds by formalizing a connection to Langevin dynamics from statistical physics, modeling the discrete optimization trajectory as a continuous stochastic process.

From an analytic perspective, much of this literature employs the central-limit theorem to model SGD as gradient descent perturbed by Gaussian Langevin noise, with variance dependent on both the mini-batch size $|\mathcal{B}|$ and learning rate η [74, 75, 76]. Following the treatment in [75], the discrete SGD update:

$$\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} - \frac{\eta}{B} \sum_{b=1}^B \nabla_{\mathbf{w}} L(\mathbf{w}^{(t)}, \mathcal{B}_b), \quad (2.6.1)$$

is reformulated as the following stochastic differential equation (SDE) (using the Itô formalism [77]):

$$d\mathbf{w}(t) = -\eta \nabla_{\mathbf{w}} L(\mathbf{w}) dt + \sqrt{2\zeta^{-1} D(\mathbf{w})} dW(t), \quad (2.6.2)$$

where $D(\mathbf{w})$ is a diffusion matrix capturing anisotropic noise, $W(t)$ is a Wiener process, and $\zeta^{-1} = \frac{\eta}{2|\mathcal{B}|}$ acts as an effective inverse temperature. This formulation elegantly decomposes the dynamics into a *drift term* guiding weights toward low-loss regions and a *diffusive term* representing stochastic exploration. The proof of the convergence of discrete SGD in Eq. 2.6.1 to the SDE in Eq. 2.6.2 is obtained in [74].

However, this continuous-time formulation encounters significant theoretical difficulties due to a mathematically inconsistent *hybrid limit* [78, 79]. The fundamental issue arises from conflicting scaling requirements: for the discrete updates to become truly infinitesimal, one must send the learning rate $\eta \rightarrow 0$, treating it as the continuous-time step dt . In this proper limit, however, the stochastic gradient noise would be suppressed to zero, yielding

an ordinary differential equation without diffusion. To circumvent this problem, the limit $dt \rightarrow 0$ is applied to the drift term, while simultaneously treating the prefactor $\sqrt{\eta}$ as a finite constant to maintain a non-vanishing noise term. Despite this theoretical limitation, the SDE analogy remains valuable as a conceptual framework for understanding SGD’s dynamics.

Analytically characterizing the full trajectory of the SGD algorithm without resorting to such approximations is considerably more challenging. Recent works [80, 81, 79] have shown that the dynamical trajectory of SGD can be tracked analytically using dynamical mean-field theory (DMFT) from statistical physics. Importantly, this description is not confined to a continuous-time limit: the discretized DMFT equations can accurately capture the dynamics of SGD at finite learning rates.

This conceptual link to SDEs has inspired the design of new algorithms like *Stochastic Gradient Langevin Dynamics* (SGLD) [59]. Rather than attempting to derive an SDE from standard SGD, SGLD is algorithmically constructed by injecting Gaussian noise, scaled with $\sqrt{\eta}$, into a stochastic gradient update. This specific construction ensures the discrete-time process approximates a continuous-time SDE whose stationary distribution is the Bayesian posterior. By systematically reducing the step size η towards zero over time (a process known as annealing), the parameter iterates converge to samples from the true posterior distribution rather than collapsing to a single point estimate. This provides a seamless transition from stochastic optimization to full Bayesian sampling, naturally incorporating uncertainty quantification and built-in protection against overfitting.

Geometry of Loss Landscape. Beyond stochastic dynamics, the geometry of the loss landscape is a decisive factor in deep learning. While SGD’s noise aids navigation, enabling escapes from poor local optima, the high-dimensional terrain itself dictates which solutions are found and their robustness. Understanding this geometry provides essential insights into generalization and guides the development of more effective training methods.

The local geometry of the loss landscape is characterized by critical points where $\|\nabla L(\mathbf{w})\| = 0$ and by the curvature information in the Hessian matrix $H = \nabla^2 L(\mathbf{w})$. Empirical studies reveal a striking deviation from classical convex landscapes: the Hessian spectrum exhibits a bulk of near-zero eigenvalues with only a few large positive modes [82]. A few negative modes are also present, which correspond to directions of negative curvature, indicating missed local descent directions that first-order optimizers could have exploited to escape saddle points [83]. This spectrum implies that most directions in weight space are flat, causing minimal change in the loss value, while only a few directions corresponding to large Hessian eigenvalues significantly affect the loss. Consequently, the landscape is dominated by wide, flat valleys rather than narrow, sharp minima.

This geometric insight carries profound implications. Solutions in these broad valleys demonstrate enhanced stability under perturbations, and this geometric bias is strongly

linked to improved generalization performance [84]. The observation has inspired novel optimization algorithms such as Entropy-SGD [83], which employs SGLD [59] to optimize a “local entropy” objective. Drawing from statistical physics, this objective favors states that achieve a favorable balance of low energy (low loss) and high entropy (a wide basin of surrounding low-loss configurations). By actively biasing optimization toward wide valleys, such methods demonstrably improve generalization.

In summary, analyzing neural network training through the combined lens of statistical physics reveals a compelling narrative: the interplay between noise-driven exploration and landscape geometry transforms the objective from simple loss minimization to the entropic selection of robust wide minima, capturing the very essence of generalization. This perspective not only deepens theoretical understanding but also guides the design of training procedures that consciously leverage these fundamental physical principles.

2.7 The Convolutional Mechanism

Convolutional Neural Networks (CNNs) represent a cornerstone of deep learning, having fundamentally transformed the field of computer vision. Their architecture is exceptionally well-suited for processing data with a grid-like topology, such as the matrix of pixels that constitutes a digital image. A principal strength of CNNs is their capacity for automatic, hierarchical feature extraction. As the network depth increases, successive layers learn to identify progressively more complex patterns, from simple edges to intricate object structures [85]. This capability was decisively demonstrated in 2012 when AlexNet [86], a deep CNN, achieved a landmark victory in the ImageNet competition [87], substantially outperforming all conventional methods. This success catalyzed the widespread adoption of deep learning across both academic research and industrial applications.

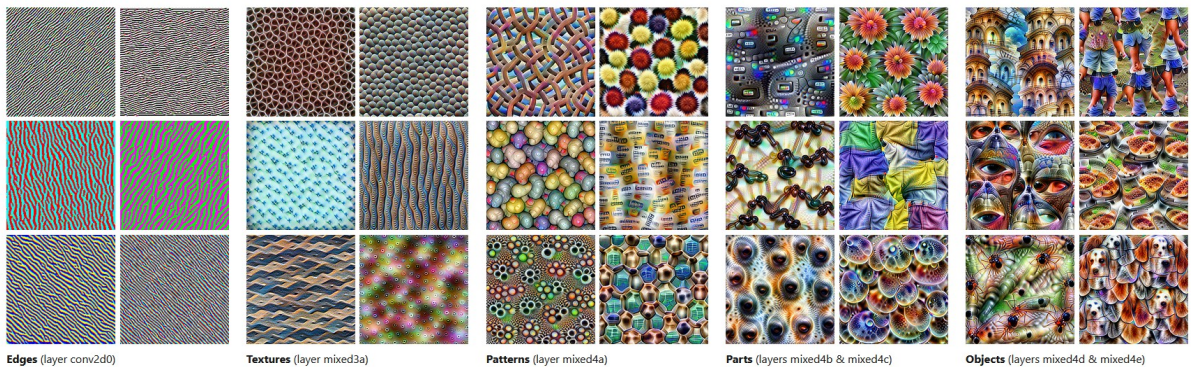


Figure 2.4: Hierarchical feature learning in a CNN progresses from simple to complex representations. Early layers detect basic local patterns such as edges and textures, while deeper layers combine these into increasingly abstract and complex features. This visualization illustrates how GoogLeNet [88], trained on ImageNet [87], constructs its understanding of images across multiple layers. Image taken from [89].

The convolutional layer forms the computational core of a CNN, an architecture expressly engineered to address the deficiencies of fully-connected networks in processing image data. Unlike a fully-connected layer, which establishes connections between every neuron in adjacent layers, the convolutional layer employs a set of learnable filters (or kernels) that perform convolution operations by scanning across the input data. Fully-connected architectures exhibit two major drawbacks for image-related tasks: poor scalability, as they lead to a computationally intractable number of parameters even for moderately-sized images, and a fundamental disregard for the intrinsic properties of visual data, such as the strong locality of pixel correlations (where adjacent pixels are highly interdependent) and translational equivariance (where semantically meaningful features may occur anywhere within the image).

Convolutional layers overcome these limitations through two foundational mechanisms: local connectivity and weight sharing. Local connectivity restricts each neuron to a small spatially localized region of the input, substantially reducing the number of parameters. Weight sharing ensures that the same set of learnable parameters, encapsulated within a filter, is applied across the entire input, enabling the detection of similar patterns irrespective of their position and thereby leading to translational equivariance. Moreover, the hierarchical arrangement of convolutional layers permits the construction of increasingly complex features: elementary patterns such as edges and textures in earlier layers are progressively combined into higher-level structures like contours and object parts in deeper layers, effectively mirroring the compositional hierarchy of visual information (see Fig. 2.4).

The input to a convolutional layer l is a volume $X^{(l)}$ of size $H^{(l)} \times W^{(l)} \times C^{(l)}$, where $H^{(l)}$ is the height, $W^{(l)}$ is the width, and $C^{(l)}$ is the number of input channels. For the first layer ($l = 1$), these channels typically correspond to the color channels of an input image (e.g., $C^{(1)} = 3$ for RGB, $C^{(1)} = 1$ for grayscale). In deeper layers, channels represent the feature maps generated by previous layers. The layer contains $K^{(l)}$ learnable filters. The application of $K^{(l)}$ such filters results in an output feature map $O^{(l)}$ of size $H^{(l+1)} \times W^{(l+1)} \times K^{(l)}$, which becomes the input $X^{(l+1)}$ to the next layer.

The discrete convolution operation for a single filter at spatial location (x, y) in the input $X^{(l)}$ computes a weighted sum $z_{(x,y)}^{(l)}$ as defined in Eq. 2.7.1. Each filter is defined by a weight tensor $I^{(l)}$ containing the layer's learnable parameters, forming a $F^{(l)} \times F^{(l)} \times C^{(l)}$ volume (see Fig. 2.5), where $F^{(l)}$ is the spatial size of the filter kernel.

$$z_{(x,y)}^{(l)} = \sum_{i=1}^{F^{(l)}} \sum_{j=1}^{F^{(l)}} \sum_{c=1}^{C^{(l)}} \left(I_{i,j,c}^{(l)} \cdot X_{(x+i,y+j,c)}^{(l)} \right) + b^{(l)}, \quad (2.7.1)$$

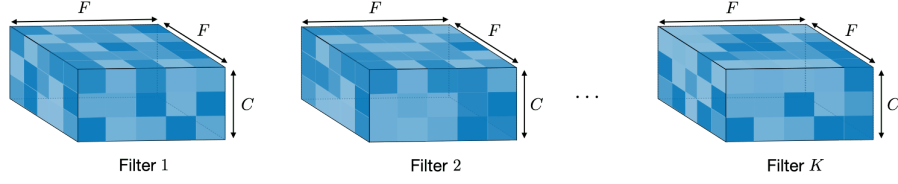


Figure 2.5: Visualization of a convolutional layer containing K distinct filters, each with dimensions $F \times F \times C$. Image taken from [90].

Following this linear transformation, a non-linear activation function (e.g., ReLU) is applied element-wise to the output feature map, introducing the non-linearity necessary for learning complex functions.

The key hyperparameters of a convolutional layer are the filter size $F^{(l)}$ and the stride $S^{(l)}$, which denotes the number of pixels by which the filter window moves. The spatial dimensions of the output feature map are determined by the input size $H^{(l)}$, filter size $F^{(l)}$, stride $S^{(l)}$, and any zero-padding $P^{(l)}$ (the process of adding zeros to each side of the input boundaries). The output height size $H^{(l+1)}$ is given by:

$$H^{(l+1)} = \left\lfloor \frac{H^{(l)} - F^{(l)} + P_{\text{start}}^{(l)} + P_{\text{end}}^{(l)}}{S^{(l)}} \right\rfloor + 1 \quad (2.7.2)$$

Typically, padding is chosen so that $P_{\text{start}}^{(l)} = P_{\text{end}}^{(l)} = P^{(l)}$ (see Fig. 2.6). The same formula applies for the output width size $W^{(l+1)}$.

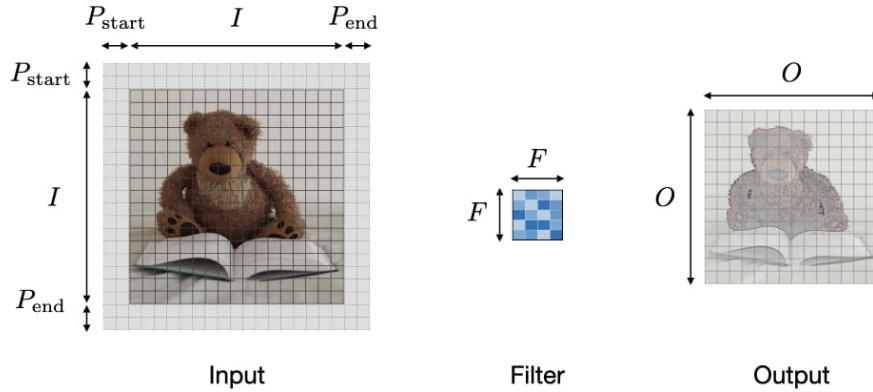


Figure 2.6: The effect of zero-padding on an input volume. Padding prevents excessive shrinkage, controls the size of the output volume, and helps retain information at the borders. Image taken from [90].

As the network deepens, the effective receptive field of neurons expands, enabling the composition of low-level features into higher-level representations. The receptive field $R^{(l)}$ at layer l is the area $R^{(l)} \times R^{(l)}$ of the input that each neuron of the l -th activation map

can see. It is computed as:

$$R^{(l)} = 1 + \sum_{j=1}^l (F^{(j)} - 1) \prod_{i=0}^{j-1} S^{(i)} \quad (2.7.3)$$

where $F^{(j)}$ is the filter size at layer j , $S^{(i)}$ is the stride value of layer i , and $S^{(0)} = 1$ by convention.

To enforce translational invariance and progressively reduce spatial dimensionality, convolutional layers are typically interspersed with pooling layers, a downsampling operation. Pooling layers make the network more computationally efficient and less sensitive to minor input shifts. The most prevalent form, max pooling, operates by dividing a feature map into smaller regions and outputting only the maximum value from each region (see Fig. 2.7). This process effectively summarizes the most prominent feature in each region, reducing the spatial dimensions of the data while retaining the most salient information.

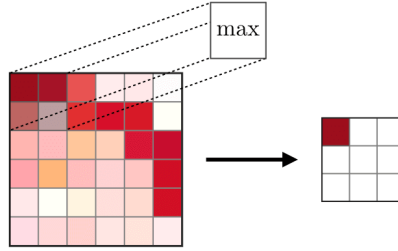


Figure 2.7: An illustration of the max pooling operation, which reduces the spatial dimensions while retaining the most activated features. Image taken from [90].

A canonical CNN architecture is typically constructed from multiple convolutional and pooling layers (see Fig. 2.8). This backbone progressively transforms the raw input pixels into increasingly abstract and semantically meaningful feature representations. The final part of the architecture is, however, highly task-dependent. In image classification, the network usually concludes with one or more fully connected layers that map the extracted features into a single vector of class scores, or logits (see Sec. 2.3). The dimensionality of this output corresponds to the total number of classes. In contrast, semantic segmentation aims to assign a class label to every pixel in the input image, thereby producing a dense, pixel-wise prediction map rather than a single global label. To achieve this, the final layers of the network remain convolutional. The last convolutional layer is configured with a number of filters equal to the number of target classes, producing an output volume where each spatial location contains a vector of class scores.

For both tasks, these logits represent the model’s unnormalized predictions. To convert them into a probability distribution, they are passed through a softmax function. This yields a final vector of probabilities for each class, whether for the entire image

(classification) or for each individual pixel (segmentation). The model is trained by minimizing a task-specific loss function, most commonly categorical cross-entropy (see Sec. 2.3). This loss quantifies the difference between the predicted probability distribution and the true target distribution. Through gradient-based optimization, the network learns to adjust all its parameters to produce accurate predictions.

These principles are powerfully employed in architectures like U-Net [41], introduced in 2015 and designed for image segmentation of anatomical structures in biomedical images. U-Net employs a symmetric encoder-decoder structure, depicted in Fig. 2.8. The encoder pathway utilizes successive convolution and pooling operations to capture multi-scale contextual information and compress spatial resolution. The decoder pathway then uses upsampling or transposed convolutions to precisely localize these features by recovering spatial detail. To preserve fine-grained information that may be lost during pooling, the network incorporates skip connections, concatenating feature maps from the encoder with the corresponding upsampled maps in the decoder. This allows the model to synthesize high-level context with high-resolution detail for accurate pixel-wise prediction. In Chap. 5, we will compare the performance of a U-Net architecture with a Graph Neural Network (GNN) on a difficult semantic segmentation task.

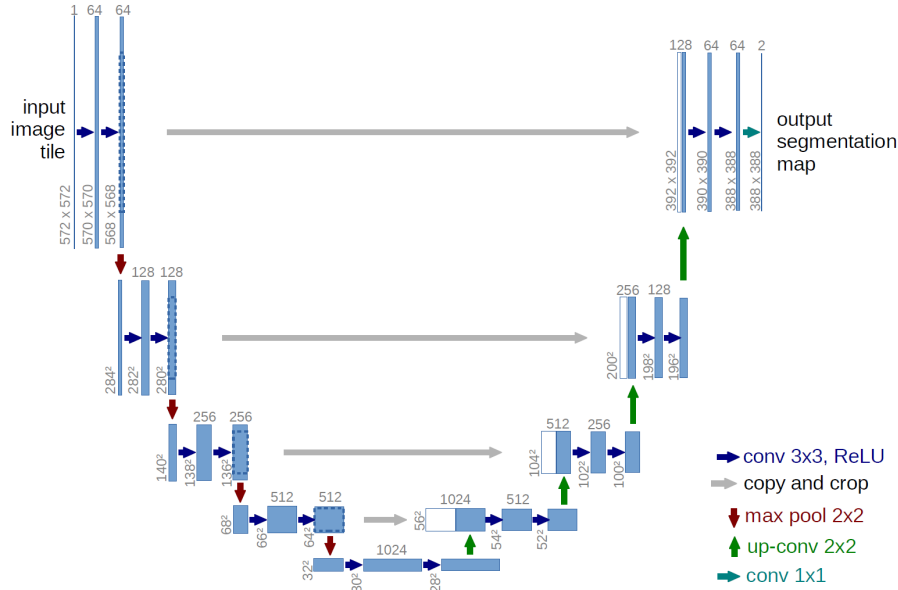


Figure 2.8: Architectural diagram of the U-Net model taken from [41]. Blue boxes represent multi-channel feature maps, with the channel depth annotated above and the spatial dimensions annotated at the bottom left. White boxes indicate feature maps copied from the encoder pathway via skip connections.

2.8 The Role of Residual Connections

A pivotal advancement in convolutional architectures is the Residual Network (ResNet) [64], which was specifically designed to overcome the limitations of training very deep

models. The core innovation of ResNet is the introduction of skip connections (also known as residual connections), which directly address the optimization challenges that plague deep networks. As demonstrated in Sec. 2.5, these connections effectively smooth the otherwise chaotic and rough loss landscape characteristic of deep architectures, facilitating more stable and efficient training.

The primary obstacle that ResNet mitigates is the vanishing gradient problem. This phenomenon arises during backpropagation in deep networks, where the gradients used to update parameters become exponentially smaller as they are propagated backward through successive layers. The chain rule of calculus, essential to backpropagation, involves multiplying many partial derivatives (often with values < 1), causing the overall gradient magnitude to diminish rapidly with depth. Consequently, the early layers in a deep network receive negligible updates, effectively ceasing to learn and making the training of very deep models infeasible.

ResNet’s skip connection mechanism counteracts this by providing a direct identity pathway through the additive formulation:

$$\mathbf{y} = \mathbf{x} + f_{\mathbf{w}}(\mathbf{x}),$$

where $f_{\mathbf{w}}$ represents learned transformation within a layer or block. This shortcut connection allows gradients to propagate more directly during backpropagation, bypassing the non-linear transformations that typically lead to multiplicative decay. By preserving gradient magnitude, skip connections ensure that even the earliest layers receive sufficient signal for effective learning.

The vanishing gradient problem is particularly acute in Recurrent Neural Networks (RNNs), which are designed to process sequential data $\{\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_T\}$, where the order and context of inputs are paramount. Unlike fully-connected layers, RNNs maintain a latent internal state $\mathbf{h}_t$ that evolves over time (see Fig. 2.9). This allows the network to integrate information from the entire history of the sequence, making it a powerful tool for tasks with inherent temporal structure, such as time-series forecasting, speech recognition, and machine translation.

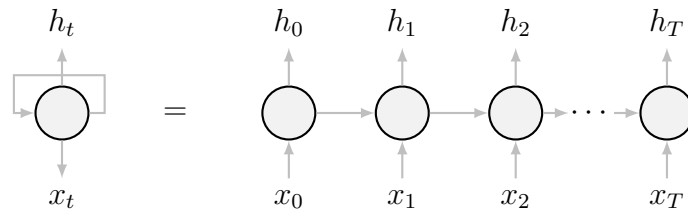


Figure 2.9: Unfolding a RNN through time. The left side shows the recurrent cell with its feedback connection, while the right side shows the same cell unfolded into a computational graph across multiple time steps.

This capability stems from a recurrent connection, a feedback loop within the network’s architecture. This loop can be conceptually “unfolded” over time to visualize the RNN as a deep network, where the same set of parameters is shared across all time steps. The standard state update at each time step t is:

$$\mathbf{h}_t = \sigma(\mathbf{W}_{hh}\mathbf{h}_{t-1} + \mathbf{W}_{xh}\mathbf{x}_t + \mathbf{b}_h), \quad (2.8.1)$$

where σ denotes a non-linear activation function, and $\theta = \{\mathbf{W}_{hh}, \mathbf{W}_{xh}, \mathbf{b}_h\}$ are the shared model parameters learned during training. The resulting state $\mathbf{h}_t$ is a compressed representation or learned embedding of the entire sequence history $\{\mathbf{x}_1, \dots, \mathbf{x}_t\}$.

RNNs are trained via Backpropagation Through Time (BPTT) [91], which presents a fundamental challenge: calculating the gradient with respect to parameters at distant time steps requires multiplying many Jacobian matrices from the intermediate steps. As this product often involves norms less than one, the gradient can vanish exponentially with the sequence length T , severely impairing the model’s capacity to learn long-range dependencies.

Inspired by ResNet, incorporating a skip connection transforms the standard RNN update into a Residual RNN, directly mitigating the vanishing gradient problem:

$$\mathbf{h}_t = \mathbf{h}_{t-1} + \sigma(\mathbf{W}_{hh}\mathbf{h}_{t-1} + \mathbf{W}_{xh}\mathbf{x}_t + \mathbf{b}_h) \quad (2.8.2)$$

This additive formulation ensures an unobstructed gradient path through the identity term $\mathbf{h}_{t-1}$, which stabilizes training and facilitates learning across extended sequences.

Beyond its immediate practical benefits, this residual update possesses a profound mathematical interpretation as an Euler discretization of a continuous transformation [92]. This perspective generalizes the discrete recurrence into a continuous-time model defined by a Neural Ordinary Differential Equation (Neural ODE):

$$\frac{d\mathbf{h}(t)}{dt} = f(\mathbf{h}(t), \mathbf{x}(t), t, \theta), \quad (2.8.3)$$

where the final state $\mathbf{h}(T)$ is defined as the solution to this ODE initial value problem at time T , computed by a black-box differential equation solver. This approach offers several benefits, such as memory-efficient training and the ability to natively model data observed at irregular time intervals, presenting a powerful alternative to discrete sequence models.

The transformative impact of skip connections reaches its zenith in the Transformer architecture [8]. Although the Transformer abandons recurrence in favor of self-attention for parallel sequence processing, it crucially retains and relies upon skip connections around every layer. These connections are indispensable for stabilizing the training of its deep structure and preserving information flow across numerous layers of computation. Indeed, as shown in [65], skip connections help preventing the rank collapse of the

self-attention matrix along the layers. The synergistic combination of self-attention and residual connections has enabled Transformers to comprehensively surpass traditional RNNs, forming the foundational core of all modern Large Language Models (LLMs) [93, 94].

Consequently, the skip connection has evolved from a specific optimization technique in computer vision to a ubiquitous, indispensable component that underpins the stability and performance of contemporary deep learning architectures. Indeed, in Chapter 3 we will further discuss their importance in the context of Graph Neural Networks (GNNs).

Chapter 3

Foundations of Graph Machine Learning

Graphs provide a powerful framework for representing relational data, capturing entities and their interactions in domains where conventional grid-based or sequential neural architectures face limitations. This challenge motivates the development of Graph Neural Networks (GNNs) [1, 2], which extend deep learning to non-Euclidean domains. We first establish graph fundamentals in Sec. 3.1, then analyze the role of symmetry and invariance as core inductive biases in Geometric Deep Learning [95, 96] in Sec. 3.2. Sec. 3.3 details the GNN message-passing mechanism [10, 97] and provides examples of common GNN layers, while Sec. 3.4 covers GNN applications to node classification and link prediction. A significant challenge for GNNs lies in homophily bias and oversmoothing effects, particularly in heterophilic graphs and deep architectures, which we examine in Sec. 3.5. Finally, to provide an interpretable, mechanistic counterpart for graph-structured data analysis, Sec. 3.6 introduces Probabilistic Graphical Models (PGMs) [29, 30], focusing specifically on generative models with latent variables. We begin by detailing statistical inference methods for these models and comparing them with neural network training in Sec. 3.8. We then introduce the concept of graph communities in Sec. 3.9 and describe the Stochastic Block Model (SBM) [37, 40] in Sec. 3.10 as a canonical example of a generative model for community detection. Sec. 3.10 also presents two SBM variants that are later used in the comparative analysis between PGMs and GNNs in Chap. 6. The complementary perspectives of these two paradigms are directly contrasted in Sec. 3.7 which compares generative and discriminative approaches to the link prediction task.

3.1 Graph Theory Elements

A graph or *network* is a mathematical structure that models relationships between entities, formally defined as an ordered pair $G = (V, E)$. Here, V represents the set of nodes (or vertices) with cardinality $|V| = N$, while $E = \{(i, j) : i, j \in V\} \subseteq V \times V$ constitutes the set of edges that capture interactions between nodes (see Fig. 3.2).

The most fundamental representation of a graph's connectivity is its *adjacency matrix* $A = \{A_{ij}\} \in \mathbb{R}^{N \times N}$, a square matrix providing a complete description of the edge set:

$$A_{ij} = \begin{cases} 1, & \text{if } (i, j) \in E \\ 0, & \text{otherwise} \end{cases}$$

In practice, most large graphs are *sparse*: the number of actual edges $|E|$ is far smaller than the possible N^2 . This sparsity makes adjacency matrices predominantly zero-valued, a property crucially exploited for computational efficiency through specialized data structures that store only non-zero entries.

The fundamental types of graphs, such as undirected, directed, and weighted, are characterized by distinct structural properties that are directly reflected in their adjacency matrices, as shown in Figure 3.1

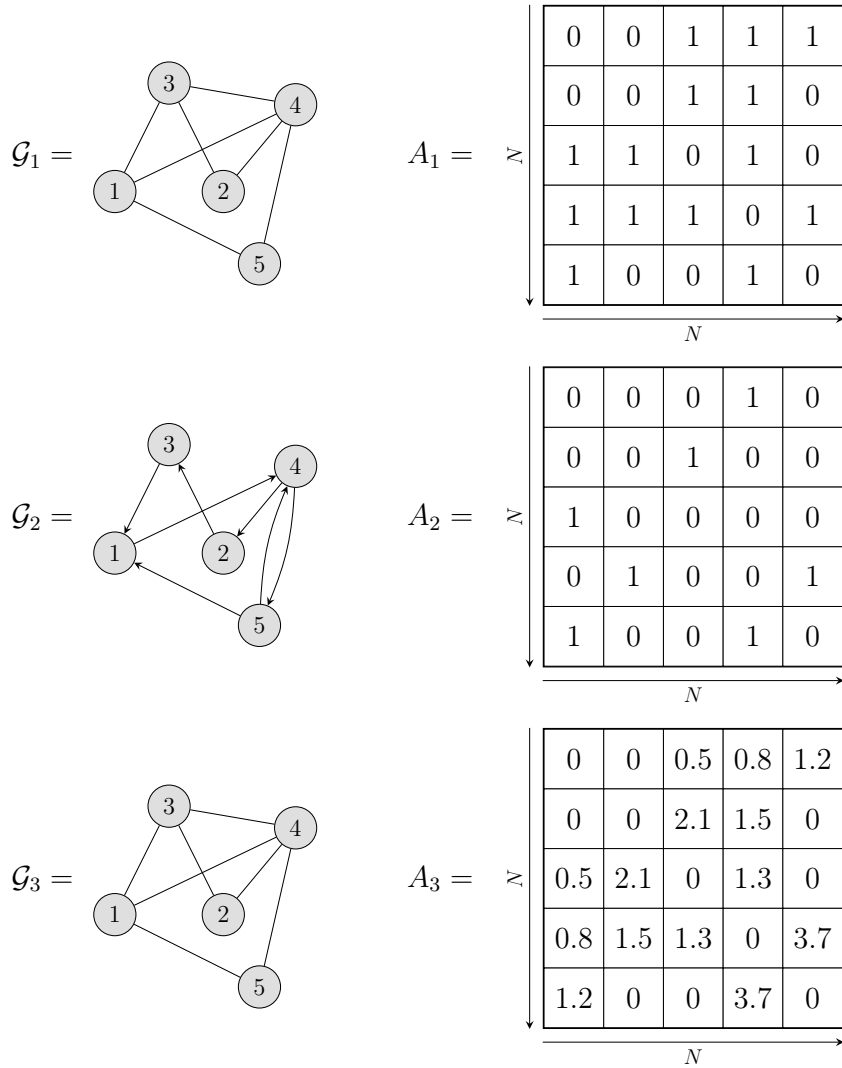


Figure 3.1: Three types of graphs with identical connectivity: $\mathcal{G}_1$ (undirected) with symmetric A_1 , $\mathcal{G}_2$ (directed) with asymmetric A_2 , and $\mathcal{G}_3$ (weighted) with real-valued A_3 . All matrices are $N \times N$ where $N = 5$.

In *undirected graphs*, relationships are mutual: an edge between i and j implies connectivity in both directions. This symmetry is captured by a symmetric adjacency matrix where $A_{ij} = A_{ji}$, perfectly modeling reciprocal relationships like friendships in social networks or bonds in molecular structures.

In contrast, *directed graphs* capture asymmetric relationships where an edge from i to j does not guarantee the reverse connection. This results in an asymmetric adjacency matrix where A_{ij} and A_{ji} are independent, essential for modeling directional interactions like citations between papers or financial transactions between accounts.

Beyond mere connectivity, many real-world networks require quantifying relationship strength. This motivates the extension to *weighted graphs*, where $A_{ij} \in \mathbb{R}_{\geq 0}$ represents connection strength rather than binary existence. For example, in transportation networks, weights can encode distances or travel times, enabling shortest-path calculations that would be impossible with simple binary edges.

From the adjacency matrix, we can derive several fundamental concepts that characterize node connectivity. The *degree matrix* D is a diagonal matrix where each entry $D_{ii} = \sum_j A_{ij}$ quantifies the total number of neighbors of a node i , essentially the number of its connections, also referred to as its *degree*. In undirected graphs, this simply counts the edges incident to node i , while in directed graphs it generalizes to two distinct measures: the out-degree ($\sum_j A_{ij}$), counting edges originating from i , and the in-degree ($\sum_i A_{ij}$), counting edges pointing to j . Closely related to degree is the concept of *neighborhood*. We define the neighborhood $\mathcal{N}_i$ of node i as the set of all vertices connected to it.

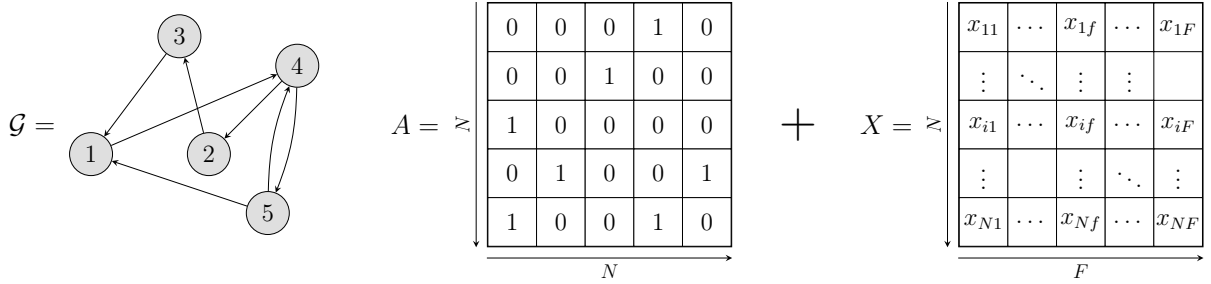


Figure 3.2: The fundamental components of a graph $\mathcal{G}$ for machine learning: the $N \times N$ adjacency matrix A encoding its connectivity, and the $N \times F$ node attribute matrix X containing feature vectors.

Finally, in real-world datasets nodes are typically associated with feature or attribute vectors describing their properties, forming an *attributed graph*. The *feature matrix* $X \in \mathbb{R}^{N \times F}$ collects all node features, with row $X_i^\top$ representing the F -dimensional attribute vector of node i :

$$X = \begin{bmatrix} X_1^\top \\ X_2^\top \\ \vdots \\ X_N^\top \end{bmatrix} \in \mathbb{R}^{N \times F}$$

This matrix X , combined with the adjacency matrix A , provides the complete input for modern graph machine learning models that leverage both structural connectivity and node attributes (see Fig. 3.2).

3.2 Geometric Inductive Biases

While fully connected networks are universal approximators, the remarkable success of deep learning in specific domains is largely attributable to the design of model architectures that incorporate powerful inductive biases, i.e., assumptions about the structure of the data. The inductive bias lets the model favour certain solutions over others, and if the bias is indeed consistent with the target task, it helps in better generalization [98]. Geometric Deep Learning [95, 96] formalizes this approach by explicitly encoding geometric principles, such as symmetry and invariance, directly into the model’s architecture. This prevents the model from having to learn these fundamental properties from scratch, drastically improving generalization.

The most common and powerful geometric prior is the encoding of symmetries. A symmetry is a transformation that leaves an object’s essential properties unchanged. Building models that are equivariant or invariant to these transformations ensures they respect the underlying structure of the data. For a function $f : \mathcal{X} \rightarrow \mathcal{Y}$ and a group of transformations G acting on $\mathcal{X}$ and $\mathcal{Y}$, we say:

- f is *invariant* to G if $f(g \cdot x) = f(x)$ for all $x \in \mathcal{X}$ and all $g \in G$. The output is unchanged by the transformation of the input.
- f is *equivariant* to G if $f(g \cdot x) = g \cdot f(x)$ for all $x \in \mathcal{X}$ and all $g \in G$. Transforming the input results in an identical transformation of the output.

Grids (CNNs). The domain of images and other grid-like data exhibits symmetry under the group of translations T . CNNs leverage this translational symmetry as a fundamental design principle:

- **Equivariant feature extraction:** Convolutional layers are equivariant to T . For a translation $t \in T$, a shift in the input x produces a corresponding shift in the output feature map:

$$f(t \cdot x) = t \cdot f(x)$$

This is achieved via weight sharing, i.e., the same kernel applied across the entire grid ensures a feature is detected regardless of its position.

- **Invariant Prediction:** Subsequent pooling layers build invariance to T . This means that:

$$f(t \cdot x) = f(x)$$

This hierarchical structure (equivariant layers followed by invariant pooling) is ideal for tasks like image classification, where the goal is to identify an object irrespective of its location.

Graphs (GNNs). Graphs represent a more complex domain where the fundamental symmetry is under the group of *node permutations* Π . Unlike images or sequences, there is no fixed notion of locality or sliding window on a graph, and graphs do not impose any inherent order on their vertices. The structure of a graph is defined solely by its connectivity: any permutation $\pi \in \Pi$ of the node indices that preserves the adjacency structure describes an identical graph.

Consider learning a function f on a graph $G := (A, X)$. As detailed later in Sec. 3.3, GNNs consist of multiple layers that are either permutation equivariant (for node updates) or permutation invariant (for graph-level readout):

- **Equivariant node processing:** GNNs achieve *equivariance* to Π through message passing, where the same local aggregation rule is applied to each node’s neighborhood. Formally, for a permutation π :

$$f(\pi \cdot (A, X)) = \pi \cdot f(A, X),$$

where $\pi \cdot (A, X) = (\pi A \pi^\top, \pi X)$ denotes the group action on the graph. This ensures that internal node representations transform consistently under any relabeling.

- **Invariant readout:** For graph-level predictions, the equivariant node embeddings are aggregated via a *permutation-invariant* readout function, ensuring that:

$$f(\pi \cdot (A, X)) = f(A, X),$$

so that the final prediction remains identical across all equivalent representations of the graph.

In summary, both CNNs and GNNs are quintessential examples of Geometric Deep Learning. The key difference lies in the underlying symmetry group they are designed to respect: the translation group T for CNNs and the permutation group Π for GNNs.

3.3 The Message Passing Neural Network Mechanism

Unlike the cross-correlation-based convolutions used in CNNs for grid-like data [4], most GNNs are formulated within the *Message Passing Neural Network (MPNN)* framework [10]. This paradigm provides a powerful and unified approach to graph representation learning, where the goal is to automatically learn a vector representation, i.e., an embedding, for each node or the entire graph. These embeddings are ideally designed to encode network

information, such that nodes with similar structural roles have similar embeddings in the vector space, making them useful for downstream tasks like node classification or link prediction.

As previously noted, GNNs operate on an attributed graph, defined by its adjacency matrix A and node feature matrix X . The model processes this input through multiple computational layers, each of which transforms and enriches the node representations by combining information from local neighborhoods. Consider the node feature matrix $X \in \mathbb{R}^{N \times F}$, where $X_i \in \mathbb{R}^F$ represents the feature vector of node i . We can view these features as a vector-valued function $h : V \rightarrow \mathbb{R}^F$ defined on the graph nodes [99], while the underlying graph structure serves as the pathway for information propagation.

More specifically, following the message-passing formulation of Bronstein et al. [96], at each layer ℓ , the hidden representation $\mathbf{h}_i^{(\ell)} \in \mathbb{R}^{d^{(\ell)}}$ of node i is updated by aggregating information from its neighborhood $\mathcal{N}_i = \{j \in V \mid (i, j) \in E\}$, with the initial state $\mathbf{h}_i^{(0)}$ set equal to the input feature vector $X_i \in \mathbb{R}^F$. The choice of dimensionality $d^{(\ell)}$ at each layer allows the model to compress, maintain, or expand representation sizes throughout the network.

The core message-passing update follows this general form:

$$\mathbf{h}_i^{(\ell)} = \phi^{(\ell)} \left(\mathbf{h}_i^{(\ell-1)}, \bigoplus_{j \in \mathcal{N}_i} \psi^{(\ell)}(\mathbf{h}_i^{(\ell-1)}, \mathbf{h}_j^{(\ell-1)}) \right) \quad (3.3.1)$$

The three key components are:

- $\psi^{(\ell)} : \mathbb{R}^{d^{(\ell-1)}} \times \mathbb{R}^{d^{(\ell-1)}} \rightarrow \mathbb{R}^{m^{(\ell)}}$ is a learnable *message function* that computes messages between connected nodes. This is typically implemented as a linear transformation $W_\psi^{(\ell)} \in \mathbb{R}^{m^{(\ell)} \times d^{(\ell-1)}}$ or an MLP;
- $\bigoplus$ denotes a permutation-invariant *aggregation operator* (such as sum, mean, or max) that combines messages from all neighbors while remaining invariant to their ordering;
- $\phi^{(\ell)} : \mathbb{R}^{d^{(\ell-1)}} \times \mathbb{R}^{m^{(\ell)}} \rightarrow \mathbb{R}^{d^{(\ell)}}$ is a learnable *update function* that combines the node's current state with aggregated neighborhood messages. A common implementation first concatenates the node's state with the aggregated message, then applies a non-linear transformation such as an MLP.

Several GNN layers have been proposed within the message passing framework. We now describe five of the most widely used, whose details are summarized in Table 3.1.

Graph Convolutional Networks (GCN). The Graph Convolutional Network [11] employs a normalized averaging scheme for aggregation. Its update rule is:

$$\mathbf{h}_i^{(\ell+1)} = \text{ReLU} \left(\sum_{j \in \mathcal{N}_i \cup \{i\}} \frac{1}{\sqrt{\tilde{d}_i \tilde{d}_j}} W^{(\ell)} \mathbf{h}_j^{(\ell)} \right),$$

where $W^{(\ell)}$ is a single learnable weight matrix shared across both the message function $\psi^{(\ell)}$ and the update function $\phi^{(\ell)}$, i.e., $W_\psi^{(\ell)} = W_\phi^{(\ell)} = W^{(\ell)}$. The adjacency matrix is augmented with self-loops, $\tilde{A} = A + I$, and $\tilde{d}_i$ denotes the degree of node i in the graph with self-loops added.

GraphSAGE. This approach is akin to GCN [11], but it offers a more flexible framework for customizing the convolution steps. Specifically, the SAGEConv implementation in PyTorch [12] is:

$$\mathbf{h}_i^{(\ell+1)} = \text{ReLU} \left(W_\phi^{(\ell)} \cdot \mathbf{h}_i^{(\ell)} + W_\psi^{(\ell)} \cdot \frac{1}{|\mathcal{N}_i|} \sum_{j \in \mathcal{N}_i} \mathbf{h}_j^{(\ell)} \right),$$

where the center node representation is decoupled from the aggregated neighborhood information by learning two distinct $W_\phi^{(\ell)}$ and $W_\psi^{(\ell)}$ weight matrices.

Graph Attention Networks (GAT). Graph Attention Networks [13] employ attention coefficients to assign different weights to neighbors:

$$\alpha_{ij}^{(\ell)} = \frac{\exp \left(\text{LeakyReLU} \left(\mathbf{a}^\top [W_\psi^{(\ell)} \cdot \mathbf{h}_i^{(\ell)} \parallel W_\psi^{(\ell)} \cdot \mathbf{h}_j^{(\ell)}] \right) \right)}{\sum_{k \in \mathcal{N}_i} \exp \left(\text{LeakyReLU} \left(\mathbf{a}^\top [W_\psi^{(\ell)} \cdot \mathbf{h}_i^{(\ell)} \parallel W_\psi^{(\ell)} \cdot \mathbf{h}_k^{(\ell)}] \right) \right)},$$

where $\parallel$ denotes concatenation, and $\mathbf{a}$ is a learnable attention vector. The node update is then:

$$\mathbf{h}_i^{(\ell+1)} = \text{ELU} \left(\sum_{j \in \mathcal{N}_i \cup \{i\}} \alpha_{ij}^{(\ell)} W_\psi^{(\ell)} \cdot \mathbf{h}_j^{(\ell)} \right)$$

Multi-head attention is often used to stabilize training and improve expressiveness.

Graph Isomorphism Network (GIN). A Graph Isomorphism Network (GIN) [100] layer aggregates information from a node's neighbors and updates its representation in a way that is theoretically as powerful as the *Weisfeiler-Lehman* (WL) test [17, 18], a traditional method for checking graph isomorphism.

GIN uses a sum aggregation function which allows to capture subtle details of the graph structure that may be overlooked with other aggregation methods. Once the node

aggregates the features of its neighbors, the result is passed through a MLP to update the node’s representation. While GIN primarily uses sum aggregation, the MLP allows for additional non-linear processing of this aggregated information.

The update in PyTorch GINConv [100] can be expressed as:

$$\mathbf{h}_i^{(\ell+1)} = \text{MLP}^{(\ell)} \left((1 + \epsilon) \cdot \mathbf{h}_i^{(\ell)} + \sum_{j \in \mathcal{N}_i} \mathbf{h}_j^{(\ell)} \right),$$

where the learnable parameter ϵ controls the balance between the node’s own feature and its neighbors, allowing flexibility in how much influence a node’s own features have in relation to its neighbors during the update process.

H₂GCN. The Heterophily-aware Graph Convolutional Network (H₂GCN) [101] extends traditional GNN frameworks to better handle graphs with heterophilic connections, where neighboring nodes may belong to different classes (see Sec. 3.5). H₂GCN is distinguished by three key design principles:

- **Separate ego and neighbor embeddings:** Encode each node’s own embedding independently from the aggregated embeddings of its neighbors, since neighbors may have dissimilar labels in heterophilic graphs.
- **Higher-order neighborhood aggregation:** Aggregate information from beyond immediate neighbors, incorporating higher-order neighborhoods at each propagation step to capture richer structural context. This is motivated by the observation that, in heterophilic graphs, while immediate neighborhoods may exhibit heterophily, higher-order neighborhoods often display homophilic patterns and can thus provide more informative context.
- **Intermediate representation combination:** Combine node embeddings from all layers at the final layer, to preserve both local and global information. This is achieved by concatenating the intermediate self-representations from each layer with the final neighborhood-aggregated representation, effectively creating skip connections throughout the network.

The convolutional update can be expressed as:

$$\mathbf{h}_i^{(\ell+1)} = \text{ReLU} \left(W_{\phi}^{(\ell)} \cdot \mathbf{h}_i^{(\ell)} + W_{\psi}^{(\ell)} \cdot \frac{1}{|\mathcal{N}_i|} \sum_{j \in \mathcal{N}_i} \mathbf{h}_j^{(\ell)} + W_{2\text{-hop}}^{(\ell)} \cdot \frac{1}{|\mathcal{N}_i^{(2)}|} \sum_{k \in \mathcal{N}_i^{(2)}} \mathbf{h}_k^{(\ell)} \right),$$

where $W_{\phi}^{(\ell)}$, $W_{\psi}^{(\ell)}$, and $W_{2\text{-hop}}^{(\ell)}$ are learnable weight matrices for the central node, first-order neighbors, and second-order neighbors, respectively, and $\mathcal{N}_i^{(2)}$ denotes the 2-hop

neighborhood of node i . These three design principles enable the model to distinctly preserve node-specific information while integrating neighbor information, helping to mitigate the oversmoothing effect [102, 103, 104, 105], which can degrade GNN performance in heterophilic settings. We will provide a detailed analysis of oversmoothing and related challenges in Sec. 3.5.

Table 3.1: Popular GNNs instances (see Eq. 3.3.1).

GNN	$\psi^{(\ell)}$	$\oplus$	$\phi^{(\ell)}$
GCN	linear	mean	linear + ReLU
GraphSAGE	linear	mean	linear + ReLU
GAT	linear	weighted mean	linear + ELU
GIN	identity	sum	MLP (ReLU)
H ₂ GCN	linear	mean	linear + ReLU

3.4 From Node Embeddings to Link Predictions

Graph learning tasks can be broadly categorized by their prediction target. Node-level prediction infers attributes of individual nodes, such as predicting atomic properties in a molecular graph. Edge-level prediction involves forecasting the existence or type of relationships between node pairs, crucial for applications like recommendation systems and drug interaction discovery. Finally, graph-level prediction aims to infer global properties of the entire structure, for instance, predicting a molecule’s toxicity or its binding affinity to a protein target. In the following, we will focus on node classification and link prediction.

In node classification, the goal is to assign a categorical label y_i to each node i based on its features and the graph structure. Let $\mathbf{h}_i^{(L)}$ denote the final embedding of node i after L layers of a GNN. The predicted label is obtained by applying a classifier to the node embedding:

$$\hat{y}_i = \text{softmax}(\mathbf{W}_c \cdot \mathbf{h}_i^{(L)}),$$

where $\mathbf{W}_c$ is a learnable weight matrix mapping embeddings to the output label space. The model is trained using a cross-entropy loss computed over the labeled subset of nodes:

$$\mathcal{L}_{\text{node}} = - \sum_{i \in V_{\text{label}}} y_i \log \hat{y}_i,$$

where V_{label} denotes the set of nodes with known labels, corresponding to the *training set*.

During *training*, the network learns to produce embeddings that capture both node features and structural context. Once trained, the model is evaluated on the *unlabeled nodes* (the test set), where it predicts their classes using the learned embeddings. Node classification thus focuses on *per-node representation learning*: each prediction depends on a

single embedding that encodes information aggregated from the node’s local neighborhood. The objective encourages embeddings to be discriminative for their respective classes, enabling accurate generalization from labeled to unlabeled nodes.

Link prediction, in contrast, concerns the presence or absence of edges between pairs of nodes. Instead of treating nodes independently, this task operates on pairs of node embeddings. Let $\mathbf{h}_i^{(L)}$ and $\mathbf{h}_j^{(L)}$ denote the final embeddings of nodes i and j . The probability that an edge exists between them can be computed as:

$$\hat{y}_{ij} = \sigma(f(\mathbf{h}_i^{(L)}, \mathbf{h}_j^{(L)})),$$

where σ is the sigmoid function and f is a scoring function. Typical choices include a simple inner product:

$$f(\mathbf{h}_i^{(L)}, \mathbf{h}_j^{(L)}) = (\mathbf{h}_i^{(L)})^\top \mathbf{h}_j^{(L)},$$

or a learnable combination via a multi-layer perceptron (MLP):

$$f(\mathbf{h}_i^{(L)}, \mathbf{h}_j^{(L)}) = \text{MLP}([\mathbf{h}_i^{(L)} \parallel \mathbf{h}_j^{(L)}]),$$

where $[\mathbf{h}_i^{(L)} \parallel \mathbf{h}_j^{(L)}]$ denotes the concatenation of the two embeddings.

Each pair of nodes (i, j) is associated with a ground-truth label $y_{ij} \in \{0, 1\}$, which indicates the *existence* ($y_{ij} = 1$) or *non-existence* ($y_{ij} = 0$) of an edge between them. A critical distinction with node classification is the necessity of negative samples when training. Because real-world graphs are typically sparse, training requires both *positive examples* (observed edges) and *negative examples* (non-existent edges). The model is optimized using a binary cross-entropy loss defined over both sets:

$$\mathcal{L}_{\text{link}} = - \sum_{(i,j) \in E^+ \cup E^-} \left[y_{ij} \log \hat{y}_{ij} + (1 - y_{ij}) \log(1 - \hat{y}_{ij}) \right],$$

where E^+ and E^- denote the sets of positive and negative edges, respectively. During *training*, the model learns to assign high probabilities to existing edges and low probabilities to non-existent ones. At *test time*, the trained model predicts the likelihood of missing or future links based on the learned node embeddings, after combining them with a scoring function. Link prediction therefore emphasizes *inter-node relationships* rather than individual node labels (see Sec. 3.7 and Chap. 6.3 for a more detailed discussion).

3.5 Homophily Bias and Oversmoothing

The design and effectiveness of GNNs are closely linked to the underlying structure of the graph, which governs how information flows between nodes. The core inductive bias of most message-passing GNNs is homophily, the principle that connected nodes tend to

be similar and therefore share attributes or labels. Under this assumption, by iteratively aggregating information from local neighborhoods (first-order neighbors; see Eq. 3.3.1), nodes build representations that capture their immediate structural context. However, this assumption becomes problematic in heterophilic graphs, where connected nodes may belong to different classes or have dissimilar features (see Fig. 3.3). Since a graph’s connectivity pattern is typically unknown beforehand, it is essential to design robust GNN architectures that do not rely on strong assumptions about the underlying graph’s structure. Furthermore, increasing model depth to capture long-range dependencies introduces critical pathologies that are a direct consequence of the local message-passing framework. These include oversmoothing [19] and oversquashing [20], which will be described later in this section.

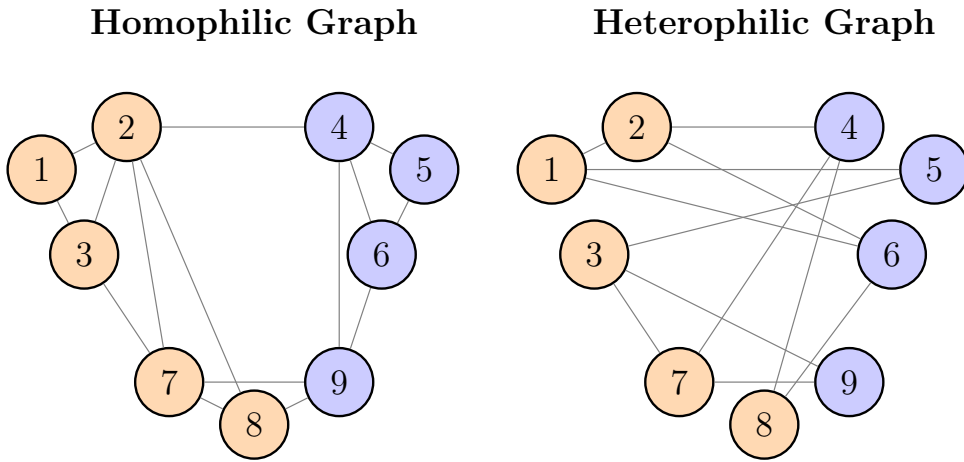


Figure 3.3: Comparison of homophilic and heterophilic graph structures with two classes (orange and purple). The homophilic graph (left) exhibits dense connections within classes, while the heterophilic graph (right) shows predominantly cross-class connections.

The Homophily bias. A graph is said to exhibit *homophily* (or *assortative mixing*) [106] when nodes with similar features or labels have a high probability of being connected. This principle is often summarized by: “birds of a feather flock together”. For instance, in academic citation networks, papers primarily cite other works on a similar topic. Conversely, a graph exhibits *heterophily* (or *disassortative mixing*) when nodes tend to connect to others that are dissimilar with respect to their features or labels, i.e., “opposites attract”. A simple example are transaction networks for fraud detection, where fraudulent accounts are typically more connected to legitimate customers, rather than with other fraudsters.

Early GNN research and canonical benchmark datasets (e.g., Cora [107], Citeseer [108]) predominantly featured homophilic graphs. This led to design architectures with a strong homophily bias. This bias is fundamentally rooted in the local neighborhood aggregation scheme of message passing: a node’s representation is updated using features from its

first neighbors (see Eq. 3.3.1). In homophilic settings, this acts as a beneficial smoothing operation, where neighbors reinforce the node’s own label, providing strong classification signals. In heterophilic graphs, however, this same process becomes detrimental. Aggregating features from dissimilar neighbors injects noise and obfuscates class boundaries, as the most relevant information for classification may lie beyond the immediate neighborhood. This bias limits the robustness and generalizability of GNNs, leading to poor performance on inherently heterophilic real-world networks [21, 22, 23]. Addressing this bias requires moving beyond the assumption that locality implies similarity. As discussed in Sec. 3.3, architectures such as the Heterophily-aware Graph Convolutional Network (H₂GCN) [101] adopt higher-order neighborhood designs precisely to overcome this limitation.

Oversmoothing and Oversquashing. Beyond the homophily bias, the fundamental reliance on first-neighbor aggregation in message passing creates additional challenges when building deeper networks to capture long-range interactions. These include oversmoothing [19], where node representations become indistinguishable, and oversquashing [20], where information from a large receptive field is compressed into a fixed-size vector. Recent studies show that these two phenomena are closely related [109], prompting significant research into understanding and mitigating their effects [104, 105, 110].

Oversmoothing [19] occurs when node representations quickly converge to each other and become indistinguishable as the number of layers increases. In other words, as information propagates through many layers, embeddings of neighboring nodes tend to converge to similar representations, making them hard to distinguish. This explains why very deep GNNs often struggle to separate nodes belonging to different classes, even if the underlying graph structure contains enough information. This issue puts a limit on the depth of a GNN, prohibiting the model’s capability of capturing complex relationships in the data. Formally, following the treatment in [111], for every pair of neighboring nodes i and j , we have:

$$\mathbf{h}_i^{(\ell)} - \mathbf{h}_j^{(\ell)} \rightarrow 0 \quad \text{as } \ell \rightarrow \infty \quad (3.5.1)$$

Equation (3.5.1) captures the local smoothing effect observed between two connected nodes i and j . A global perspective can be obtained by aggregating these differences over all edges in the graph. Specifically, we define:

$$\sum_{(i,j) \in E} \mathbf{h}_i^{(\ell)} - \mathbf{h}_j^{(\ell)} \rightarrow 0 \quad \text{as } \ell \rightarrow \infty \quad (3.5.2)$$

If the sum in Equation (3.5.2) converges to zero, the model is said to experience oversmoothing. This formulation aligns with the definition based on node-wise Dirichlet energy used in [112], which was first proposed as a measure of over-smoothing by [105].

Given node embeddings $H^{(\ell)} = [\mathbf{h}_1^{(\ell)}, \dots, \mathbf{h}_N^{(\ell)}]^\top$, the Dirichlet energy is:

$$\mathcal{E}(H^{(\ell)}) = \frac{1}{|V|} \sum_{i \in V} \sum_{j \in \mathcal{N}_i} \left\| \mathbf{h}_i^{(\ell)} - \mathbf{h}_j^{(\ell)} \right\|_2^2, \quad (3.5.3)$$

and the associated measure:

$$\mu(H^{(\ell)}) = \sqrt{\mathcal{E}(H^{(\ell)})} \quad (3.5.4)$$

serves as a valid node-similarity score: oversmoothing is reflected by $\mu(H^{(\ell)})$ decreasing toward zero as ℓ grows. Thus, the decline of Dirichlet energy provides a principled, graph-level indicator of the loss of discriminative power in deep GNNs.

A widely used strategy to alleviate oversmoothing is the incorporation of residual connections, inspired by the success of ResNets [64] as detailed in Sec. 2.8. The key idea is to preserve information from earlier layers so that node embeddings do not collapse to an indistinguishable subspace as depth increases [113]. Residual connections are particularly valuable for graphs with heterophily, where oversmoothing is a major source of performance degradation [102, 103, 104, 105], as it further erodes the already weak signal from a node’s immediate neighbors. As explained in detail in Sec. 3.3, H₂GCN [101], a heterophily-aware GNN, exemplifies this principle by explicitly separating the representations of a node’s own features and those of its neighbors, while adding skip connections that propagate embeddings from earlier layers directly to deeper ones.

Oversquashing [20], on the other hand, arises from the difficulty of compressing a large amount of information into fixed-size node embeddings. From Equation 3.3.1, we can observe that messages can only be transmitted by a distance of 1 at each layer. Hence, two nodes separated by distance l can exchange information if and only if the GNN has at least l layers. For any node i , the receptive field is defined as the set of nodes whose messages can reach i [111]. As the number of layers l increases, the size of each node’s receptive field grows exponentially [114]. Consequently, messages from an exponentially increasing number of distant nodes are compressed into fixed-size vectors, limiting the model’s ability to capture long-range dependencies [20], effectively “squashing” them. This bottleneck makes it hard for the network to propagate distant signals, especially in graphs with high connectivity, limiting the ability of the GNN to capture global structure.

3.6 Probabilistic Graphical Models

GNNs provide powerful tools for learning node and graph representations, however they often function as “black boxes”, offering limited interpretability into the underlying mechanisms governing network formation and structure. This stands in contrast to the rich tradition of probabilistic modeling that has long guided network science, where analyzing a system’s representative network (or graph structure) is key to understanding its intricate

structure and behavior [115]. Within this tradition, *Probabilistic Graphical Models (PGMs)* [29, 30] serve as a unifying framework, offering a principled language to encode complex dependencies among random variables through graph-structured representations.

This probabilistic perspective offers a complementary paradigm to GNNs. While GNNs learn complex nonlinear transformations through layered message passing, PGMs provide explicit probabilistic semantics and uncertainty quantification. PGMs offer interpretable building blocks for understanding network mechanisms, while GNNs provide powerful function approximation for learning from graph-structured data. A growing body of research now seeks to bridge the gap between these interpretable probabilistic approaches and modern deep learning on graphs [25].

Generative Models. Within the PGM broad framework, *Generative Models* are a particularly influential class designed to capture the underlying processes that produce observed network data by representing it in terms of an unknown and approximate probability distribution $P(A)$, where A is the graph’s adjacency matrix. The generative approach offers several distinctive advantages: it allows not only to describe and analyze an observed network but also to generate realistic synthetic graphs, model hypothetical scenarios (e.g., “what if” a connection is removed), and handle missing data systematically.

The power of these generative models lies in their interpretability, as they can directly encode specific hypotheses about a system’s hidden mechanisms into the model’s assumptions, e.g., community structure (see Sec. 3.10). This approach stands in contrast to discriminative methods like GNNs: while GNNs excel at mapping observed structures to predictions, they typically do not model the data-generating mechanism itself, see Fig. 3.4, inspired by [116].

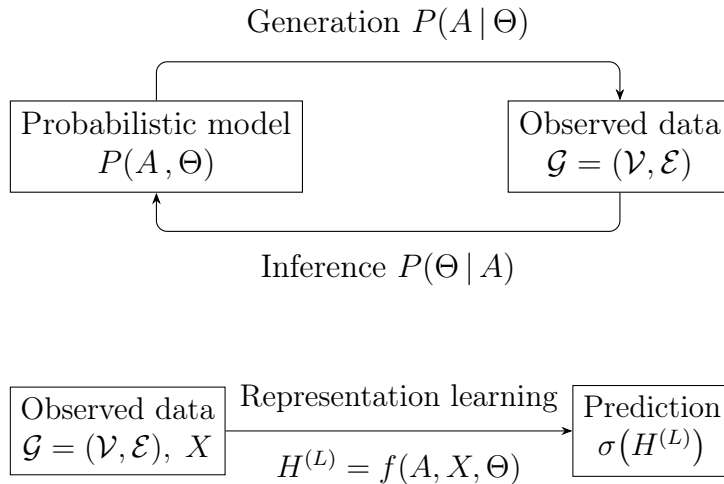


Figure 3.4: Generative models like PGMs (top) establish a bidirectional relationship between the model and observed data through generation and inference. Discriminative models like GNNs (bottom) learn representations from observed data via a feedforward process, culminating in predictions.

A canonical example of generative model is the Erdős-Rényi model [117, 118], which is employed to generate random graphs. This model assumes that all graphs with a fixed vertex set V and a predetermined number of edges $|E|$ are equally probable, selecting a graph uniformly at random from this collection. Although such models serve as valuable theoretical tools for analysis, their underlying assumptions are too simplistic to capture the complexities of real-world data.

This need is met by a powerful class of generative models known as *latent variable models* [119]. These models provide a theoretical foundation for understanding complex networks by positing that the observed graph structure A is shaped by a set of underlying, unobserved factors, known as the latent variables Θ . In statistics, a latent variable is a variable that is not directly measured but is inferred through a mathematical model from other observed variables, representing a hidden underlying factor. The core assumption is that the observed connectivity patterns arise from these hidden variables. Inferring Θ thus enables a deeper understanding of the hidden factors that govern the network's structure [119, 120], e.g., node communities (see Sec. 3.10).

The methodology follows a principled probabilistic cycle, which can be summarized in three key stages:

1. **Model Specification:** Initially, we make assumptions about the hidden variables that might influence the data generation process, and we represent these relationships in a joint probability distribution of both observed and hidden random variables, denoted as $P(A, \Theta)$.
2. **Statistical Inference:** Subsequently, when we have an observed dataset, such as a graph (represented by its adjacency matrix A), we perform inference on the latent variables. This is the process of learning or deducing the most plausible latent variables Θ that explain the observed network (see Sec. 3.8).
3. **Model Utilization and Validation:** Once these variables are learned, the model can be used to predict missing data (as in link prediction) or generate new synthetic data. This also serves as model validation: if the synthetic data exhibits properties similar to those of the observed data, it signifies that our initial assumptions about the generative model were reasonable. Conversely, discrepancies necessitate model refinement.

This entire cycle is formally grounded in the specification of the generative model, a joint distribution $P(A, \Theta)$ canonically factorized using Bayes' rule as $P(A, \Theta) = P(A | \Theta) P(\Theta)$. Here, $P(A | \Theta)$ is the *likelihood* function, modeling the probability of observing network data A given latent parameters Θ ; and $P(\Theta)$ is the *prior distribution*, encoding initial assumptions about the latent variables before observing any data.

To make these models computationally tractable, a common assumption is that of *conditional independence* [121, 122]. This principle states that, conditioned on the latent variables Θ , the observed edges A_{ij} are independent of each other. This implies that all interactions are independent and identically distributed given Θ , leading to the simplified joint probability:

$$P(A, \Theta) = \prod_{i,j} P(A_{ij} \mid \Theta) P(\Theta)$$

The specific choice of $P(A_{ij} \mid \Theta)$ and $P(\Theta)$ depends on the nature of the data. For instance, binary edge weights are naturally modeled using a Bernoulli likelihood $P(A_{ij} \mid \Theta)$, while nonnegative discrete edge weights are better captured by a Poisson distribution (see Sec. 3.10 for a concrete example).

3.7 Generative vs. Discriminative Link Prediction

The practical utility of this generative framework is powerfully demonstrated in tasks like *link prediction*. Since the model is generative, it provides a natural mechanism for predicting missing links from partially-observed data [121]: the learned parameters define a complete probabilistic model of the network. This capability is typically evaluated through a standard hold-out test: a portion of the entries in the observed adjacency matrix A (e.g., 20%) is hidden, forming a *test* set, while the remaining entries serve as a *training* set A_{train} for model inference.

Crucially, *training* here refers to the statistical inference of the model’s latent parameters. This process, detailed in Sec. 3.8, can result in either a single point estimate of the parameters $\hat{\Theta}$ or a full posterior distribution $P(\Theta \mid A_{\text{train}})$ that captures the uncertainty in their values. The probability of existence of a link between node i and j in the held-out *test set* is then derived as $P(A_{ij} \mid \hat{\Theta})$ for point estimates, and as $\mathbb{E}_{P(\Theta \mid A_{\text{train}})}[P(A_{ij} \mid \Theta)]$ for full posteriors. Sec. 3.10 provides a concrete illustration of this using the Stochastic Block Model.

This approach fundamentally differs from discriminative methods like GNNs. In GNNs, *training* involves optimizing the model’s weights to obtain a single, point estimate for each parameter. This process learns to transform input node features and the graph structure into informative node embeddings. The resulting link probability is then calculated by applying a heuristic scoring function (e.g., a dot product followed by a sigmoid) to these final-layer node embeddings $(h_i^{(L)}, h_j^{(L)})$ (see Sec. 3.4 and Sec. 6.3).

The trade-off is clear: the generative PGM provides a probabilistically coherent framework where predictions are directly governed by interpretable parameters, naturally allowing for uncertainty quantification. In contrast, the GNN offers a flexible, feature-driven approach that excels at capturing complex non-linear interactions but often lacks the same level of explicit mechanistic interpretability.

3.8 Statistical Inference in Graphical Models

The ability to predict missing links in a graph, as described above, hinges on a crucial preliminary step: learning the model parameters from the observed data. This section details this core learning problem, known as *statistical inference*, which constitutes the second stage of the probabilistic cycle introduced in Sec. 3.6. For latent variable models, this means inverting the generative model $P(A, \Theta) = P(A | \Theta)P(\Theta)$ to deduce the latent variables Θ that best explain the observed network A .

The central object of interest for this learning process is the *posterior distribution*, derived from the joint distribution $P(A, \Theta) = P(A | \Theta)P(\Theta)$ via Bayes' theorem:

$$P(\Theta | A) = \frac{P(A | \Theta)P(\Theta)}{P(A)} \quad (3.8.1)$$

The posterior $P(\Theta | A)$ represents the updated belief about the latent variables Θ after accounting for the evidence provided by the observed graph A . The normalizing constant $P(A)$, known as the *model evidence*, ensures the posterior is a valid probability distribution.

This formulation parallels neural network training but with a fundamental philosophical difference. In probabilistic generative models, the joint distribution $P(A, \Theta)$ formally assumes how data A is generated from interpretable latent factors Θ , and inference on observed data then systematically refines beliefs about these hidden variables. In contrast, neural networks find point estimates of weights optimized to minimize a task-specific loss function. Although their internal representations can capture complex statistical patterns, the lack of explicit assumptions about the data-generating process results in weights and representations that are typically not directly interpretable.

The spectrum of inference methods for characterizing the posterior $P(\Theta | A)$ ranges from finding single point estimates to fully approximating the entire distribution, each with distinct advantages depending on the analytical goals and computational constraints.

Point estimate. For many practical applications, particularly with large-scale networks, *point estimation* provides a computationally efficient approach by seeking a single optimal value for Θ . The most fundamental point estimate is the *Maximum Likelihood Estimation (MLE)*, which finds the optimal parameters $\hat{\Theta}^{\text{MLE}}$ by maximizing the probability of the observed data under the assumed statistical model:

$$\hat{\Theta}^{\text{MLE}} = \arg \max_{\Theta} P(A | \Theta) \quad (3.8.2)$$

Notably, MLE is equivalent to assuming uniform priors over the parameters Θ , meaning $P(\Theta)$ is treated as constant. This makes the posterior in (3.8.1) proportional to the likelihood: $P(\Theta | A) \propto P(A | \Theta)$. Therefore, we can directly use the likelihood $P(A | \Theta)$ to estimate the parameters Θ . An alternative approach is *Maximum A Posteriori (MAP)*

estimation, which seeks the most probable parameter values by considering both the likelihood of the data and a prior distribution over the parameters:

$$\hat{\Theta}^{\text{MAP}} = \arg \max_{\Theta} P(A \mid \Theta) P(\Theta) \quad (3.8.3)$$

Both methods are conceptually straightforward when the likelihood $P(A \mid \Theta)$ is tractable. However, in many latent variable models, this likelihood is analytically intractable and computationally difficult to manipulate directly. To simplify the expression of the likelihood, a set of unobserved or missing variables Z is introduced. In this framework, the observed network adjacency matrix A is considered as *incomplete data*, while the pair (A, Z) constitutes the *complete data*. The motivation for this construction is profound: if the unobserved variables Z were known, the complete-data likelihood $P(A, Z \mid \Theta)$ would become mathematically tractable. By explicitly accounting for Z , the likelihood is rewritten as:

$$P(A \mid \Theta) = \sum_Z P(A, Z \mid \Theta), \quad (3.8.4)$$

which leads to the corresponding log-likelihood:

$$\mathcal{L}(\Theta) = \log P(A \mid \Theta) = \log \sum_Z P(A, Z \mid \Theta) \quad (3.8.5)$$

The *log-likelihood* converts the product of probabilities into a sum of log-probabilities, which is numerically more stable and mathematically easier to optimize. Similarly, neural networks optimize log-probabilities via cross-entropy loss rather than raw probabilities, see Sec. 2.3.

In Eq. 3.8.5, Z is treated as a set of discrete random variables for notational clarity, but the concept is readily extended to the continuous case by substituting the summation with an integral. The introduction of Z creates a fundamental computational challenge: the logarithm of a summation generally prevents analytical simplification, making direct optimization of the likelihood computationally infeasible.

To address this difficulty, the *Expectation-Maximization (EM) algorithm* [123, 124] offers a principled iterative strategy. Its key idea is to replace the intractable direct maximization of the log-likelihood in Equ. (3.8.5) with a sequence of tractable optimizations, achieved by constructing and iteratively tightening a lower bound on the log-likelihood, using the current best guess for the distribution of the missing data Z . The EM algorithm is fundamentally a method for Maximum Likelihood Estimation, and therefore operates under the implicit assumption of uniform priors discussed above.

First, to handle the summation inside the logarithm, we introduce an auxiliary distribution $q(Z)$ over the missing variables Z , which satisfies $q(Z) \geq 0$ and $\sum_Z q(Z) = 1$.

Using this distribution, we can rewrite the log-likelihood as:

$$\mathcal{L}(\Theta) = \log \sum_Z q(Z) \frac{P(A, Z | \Theta)}{q(Z)} \quad (3.8.6)$$

Since the logarithm is concave, we can apply *Jensen's inequality* [125], which states that for any concave function f and any probability distribution q :

$$f\left(\mathbb{E}_q[X]\right) \geq \mathbb{E}_q[f(X)], \quad (3.8.7)$$

to obtain a tractable lower bound:

$$\log \sum_Z q(Z) \frac{P(A, Z | \Theta)}{q(Z)} \geq \sum_Z q(Z) \log \frac{P(A, Z | \Theta)}{q(Z)} := \mathcal{L}(q, \Theta) \quad (3.8.8)$$

Therefore, Equation (3.8.8) serves as a lower bound for the log-likelihood, and maximizing $\mathcal{L}(\Theta)$ is equivalent to maximizing $\mathcal{L}(q, \Theta)$. The EM algorithm offers a practical approach for performing this optimization by iteratively alternating between two steps:

- **E-step (Expectation):** Given the current parameter estimates $\Theta^{(t)}$, the E-step maximizes the lower bound $\mathcal{L}(q, \Theta^{(t)})$ with respect to the distribution $q(Z)$. The solution to this maximization problem is achieved when the lower bound matches the log-likelihood, meaning that the distribution $q^{(t+1)}(Z)$ equals the posterior distribution:

$$q^{(t+1)}(Z) = P(Z | A, \Theta^{(t)})$$

This step fills in the missing variables Z by calculating their distribution based on the observed graph A and the current parameters $\Theta^{(t)}$.

- **M-step (Maximization):** Holding the distribution $q^{(t+1)}(Z)$ fixed, the M-step updates the model parameters Θ by maximizing the lower bound $\mathcal{L}(q^{(t+1)}, \Theta)$ with respect to Θ :

$$\Theta^{(t+1)} = \arg \max_{\Theta} \mathbb{E}_{q^{(t+1)}(Z)} [\log P(A, Z | \Theta)]$$

This maximizes the log-likelihood, where the expectation is taken over the missing variables Z using the distribution from the E-step.

The EM algorithm achieves particular efficiency when both the E-step and M-step permit closed-form updates. However, a fundamental characteristic of EM is its convergence to local optima, without guarantees of reaching the global optimum. To mitigate this dependence on initialization, practitioners typically employ multiple restarts with different initial conditions, allowing the algorithm to explore various local solutions. The final

model is then selected as the realization achieving the highest likelihood value upon convergence. In Chapter 6, Section 6.3, we employ this approach to infer parameters for two of the Stochastic Block Models under analysis.

Full posterior distribution. For high-consequence applications in fields like medicine and finance, relying on a single point estimate is risky, as it ignores the uncertainty inherent in parameter estimation. A more comprehensive approach is to estimate the full *posterior distribution* $P(\Theta | A)$ and use it to compute expectations. This is vital for quantifying uncertainty and making robust predictions.

However, directly computing the posterior in (3.8.1) is often intractable because it requires calculating the normalizing constant $P(A)$, which involves a high-dimensional integral that is typically infeasible to evaluate. To address this challenge, we turn to powerful approximation methods. *Variational Inference (VI)* [126, 127] transforms the inference problem into an optimization one, seeking a tractable distribution that is as close as possible to the true posterior. Conversely, *Markov Chain Monte Carlo (MCMC)* [60, 61] is a sampling-based approach that constructs a Markov chain which eventually converges to the true posterior, allowing for a numerical approximation through the collected samples.

The central idea of VI [126, 127] is to replace the true, intractable posterior $P(\Theta | A)$ with a simpler, tractable distribution $q(\Theta)$ drawn from a chosen family $\mathcal{Q}$. The optimal approximation $q^*(\Theta)$ is obtained by minimizing the *Kullback-Leibler (KL)* divergence [51] from $q(\Theta)$ to $P(\Theta | A)$:

$$q^*(\Theta) = \arg \min_{q \in \mathcal{Q}} D_{\text{KL}}(q(\Theta) \parallel P(\Theta | A)) = \arg \min_{q \in \mathcal{Q}} \mathbb{E}_{q(\Theta)} \left[\log \frac{q(\Theta)}{P(\Theta | A)} \right]$$

Expanding the posterior reveals the core difficulty:

$$D_{\text{KL}}(q(\Theta) \parallel P(\Theta | A)) = \mathbb{E}_{q(\Theta)} [\log q(\Theta)] - \mathbb{E}_{q(\Theta)} [\log P(A, \Theta)] + \log P(A)$$

This objective remains unfeasible to minimize directly because it explicitly depends on the intractable model evidence $\log P(A)$. To overcome this, VI optimizes an alternative objective function called the *Evidence Lower Bound (ELBO)*.

By definition, the ELBO can be expressed as:

$$\begin{aligned} \text{ELBO}(q(\Theta)) &= \mathbb{E}_{q(\Theta)} [\log P(A, \Theta)] - \mathbb{E}_{q(\Theta)} [\log q(\Theta)] \\ &= \log P(A) - \text{KL}(q(\Theta) \parallel P(\Theta | A)) \end{aligned} \tag{3.8.9}$$

Since $\log P(A)$ is constant with respect to Θ , minimizing the KL divergence between $q(\Theta)$ and $P(\Theta | A)$ is equivalent to maximizing the ELBO [128].

Unlike variational inference, which approximates the posterior through optimization over a simplified distribution, MCMC methods [60, 61] address the sampling problem directly by constructing a Markov chain. MCMC generates a sequence of S correlated samples $\{\Theta^{(s)}\}_{s=1}^S$ that collectively represent the target posterior $P(\Theta | A)$. This chain is designed with a transition kernel $T(\Theta^{(s+1)} | \Theta^{(s)})$ that ensures the target posterior $P(\Theta | A)$ is its unique stationary distribution. The method’s theoretical foundation rests on *asymptotic exactness*: as $S \rightarrow \infty$, the empirical distribution of the samples converges to the true posterior. This sequence of samples can then be used to approximate the posterior distribution (e.g., to generate a histogram) or to compute integrals such as posterior expectations (e.g., an expected value) empirically.

A cornerstone of the MCMC framework is the Metropolis-Hastings algorithm [129, 130], which operates through a two-step process. First, a *proposal distribution* q suggests new candidate states, then an *acceptance step* determines whether to incorporate these candidates into the sample sequence. Conceptually, the Markov Chain explores the target posterior landscape, guided by the proposal distribution that suggests its next move. The choice of proposal is critical, as a distribution narrowly concentrated around the current state $\Theta^{(s)}$ leads to high acceptance but slow exploration, while a broad one yields low acceptance with redundant computation.

The algorithm proceeds iteratively as follows. At step s , given the current state $\Theta^{(s)}$:

1. **Propose** a candidate $\Theta^{(s+1)}$ from the proposal distribution $q(\Theta^{(s+1)} | \Theta^{(s)})$
2. **Accept** the candidate with probability α , where:

$$\alpha = \min \left(1, \frac{P(\Theta^{(s+1)} | A)}{P(\Theta^{(s)} | A)} \cdot \frac{q(\Theta^{(s)} | \Theta^{(s+1)})}{q(\Theta^{(s+1)} | \Theta^{(s)})} \right) \quad (3.8.10)$$

The acceptance ratio α quantifies the relative probability of $\Theta^{(s+1)}$ versus $\Theta^{(s)}$ under the target posterior $P(\Theta | A)$. This creates an efficient exploration: moves to higher-probability regions ($\alpha > 1$) are always accepted, while moves to lower-probability regions are accepted with probability α , ensuring occasional exploration of less probable areas. In practice, a uniform random number $u \sim U(0, 1)$ is drawn, and the next state is set to $\Theta^{(s+1)}$ if $u \leq \alpha$, otherwise it remains $\Theta^{(s)}$.

While MCMC’s flexibility enables sampling from complex distributions that challenge variational methods, significant computational limitations remain. High-dimensional posteriors often require long chains, careful proposal tuning, and substantial resources to ensure convergence. The required number of iterations depends on multiple factors, particularly the relationship between target and proposal distributions, necessitating problem-specific adjustment for practical success. In Chapter 6, Section 6.3, we employ this approach to infer parameters for one of the Stochastic Block Models under analysis.

3.9 Modeling Graph Communities

A fundamental concept in network analysis is *communities*, also referred to as *clusters* or *modules* [131]. These are groups of vertices within a network that typically share common properties or perform similar functional roles. Identifying these communities is crucial as it reveals the hidden organizational principles and functional subunits embedded within complex networks [132].

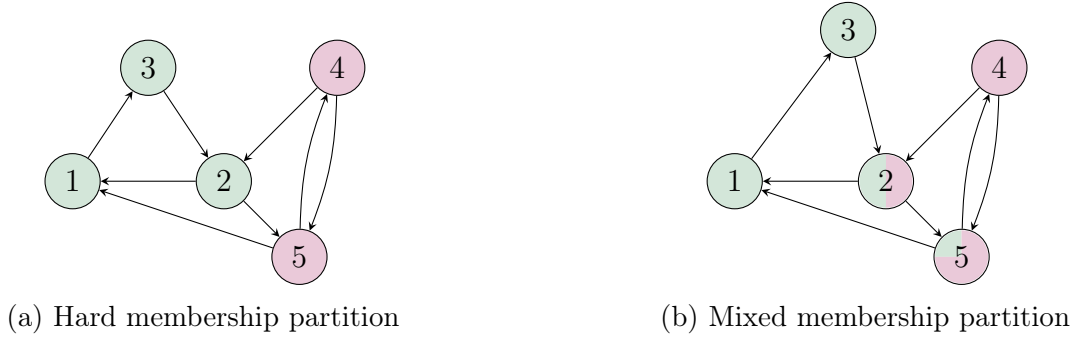


Figure 3.5: Visual comparison of community membership types. (a) Hard membership partition where each node belongs exclusively to one community. (b) Mixed membership partition where nodes exhibit overlapping community affiliations.

Node Membership in Community Detection. The assignment of nodes to communities is formally described by their *membership*, a concept visually represented in Figure 3.5. The simplest approach, known as *hard membership* (or disjoint communities), assigns each node exclusively to a single community, resulting in a partition with strictly non-overlapping boundaries. While conceptually straightforward, this rigid assignment often proves inadequate for capturing the complexity of real-world systems, where nodes frequently maintain multiple affiliations.

This limitation is addressed by *mixed membership* models, which allow nodes to participate simultaneously in multiple communities with varying degrees of association [133, 134]. This framework, giving rise to *overlapping communities*, provides a more nuanced and accurate representation of systems where entities naturally exhibit multiple affiliations. For instance, individuals in social networks typically belong to diverse circles (family, work, hobbies), proteins in biological networks participate in multiple functional pathways, and researchers in academic collaboration networks work across different scientific domains.

Mathematically, this distinction is captured through different representations of the membership vector. In hard membership models, each node i is associated with a one-hot vector $u_i \in \{0, 1\}^K$ satisfying $\sum_{k=1}^K u_{ik} = 1$, where $u_{ik} = 1$ indicates exclusive assignment to community k . Conversely, mixed membership models represent each node's affiliations through a probability vector $u_i \in [0, 1]^K$, also normalized such that $\sum_{k=1}^K u_{ik} = 1$. Here,

the elements u_{ik} quantify graded affiliation strengths, enabling the representation of partial memberships across multiple communities.

Affinity Matrix and Community Structure. Regardless of whether communities are defined through hard or mixed-membership partitions, they fundamentally group nodes that display similar connectivity patterns relative to nodes in other groups [120]. The global community structure of a network is consequently characterized by the interaction patterns between these different groups. Community detection models typically capture this information through an *affinity matrix* W of dimensions $K \times K$, where each element w_{kl} quantifies the connection density between communities k and l .

$$\begin{array}{cc}
 W_{\text{assortative}} \in \mathbb{R}^{K \times K} & W_{\text{disassortative}} \in \mathbb{R}^{K \times K} \\
 \begin{bmatrix} 0.8 & 0.1 & 0.1 & 0.1 \\ 0.1 & 0.9 & 0.1 & 0.1 \\ 0.1 & 0.1 & 0.7 & 0.1 \\ 0.1 & 0.1 & 0.1 & 0.8 \end{bmatrix} & \begin{bmatrix} 0.1 & 0.8 & 0.7 & 0.6 \\ 0.8 & 0.1 & 0.9 & 0.7 \\ 0.7 & 0.9 & 0.1 & 0.8 \\ 0.6 & 0.7 & 0.8 & 0.1 \end{bmatrix} \\
 \text{Strong diagonal elements} & \text{Strong off-diagonal elements} \\
 w_{kk} \gg w_{kl} \text{ for } k \neq l & w_{kl} \gg w_{kk} \text{ for } k \neq l
 \end{array}$$

Figure 3.6: Affinity matrices $W \in \mathbb{R}^{K \times K}$ for different community structures. The assortative matrix (left) features strong diagonal elements, promoting dense connections within communities. The disassortative matrix (right) exhibits strong off-diagonal elements, encouraging connections between different communities. Here, $K = 4$ represents the number of communities.

Real-world networks exhibit diverse structural patterns that are directly reflected in their affinity matrices, as shown in Figure 3.6. Among the most common is the *assortative or homophilic structure* [106], characterized by communities with dense internal connections and sparser links between groups. This pattern emerges when the affinity matrix W has strong diagonal entries, promoting higher connection probabilities within the same community. Conversely, *disassortative or heterophilic structures* feature higher connection densities between different groups than within them, arising from affinity matrices with prominent off-diagonal elements that encourage cross-community links. Beyond these basic patterns, more complex organizational forms such as core-periphery structures can also be captured through tailored configurations of W , demonstrating the model's flexibility in representing varied relational structures.

3.10 The Stochastic Block Model

Since community structures are generally unobserved in empirical data, the primary goal of community detection algorithms is to uncover these latent patterns to enable

meaningful interpretation of network interactions [135]. While numerous algorithmic approaches exist, generative PGMs offer significant advantages: they impose minimal structural assumptions, support rigorous statistical inference, and naturally accommodate overlapping communities.

Foundational SBM Framework. Generative PGMs characterize network dependencies probabilistically by learning latent variables Θ that explicitly govern the probability of observing each edge A_{ij} . The *Stochastic Block Model* (SBM) [37, 38, 39, 40] stands as a cornerstone among these generative models for networks exhibiting community structure. The model posits that each node i belongs to one of K latent communities, with memberships encoded in a matrix $U \in \mathbb{R}^{N \times K}$ where u_{ik} represents the probability or strength of node i 's membership in community k . For directed networks, separate membership matrices U and V may be used to model outgoing and incoming connections respectively. The SBM implements this framework through the general formulation:

$$P(A_{ij} \mid \Theta) = P(A_{ij} \mid f(u_i, v_j, W)), \quad (3.10.1)$$

where u_i and v_i represent the K -dimensional *latent* membership vectors of node i for outgoing and incoming connections, respectively ($u_i = v_i$ for undirected graphs); W is a $K \times K$ *latent* affinity matrix regulating edge densities between communities; and f is a function combining these parameters into a scalar value (typically modeling $\mathbb{E}[A_{ij}]$).

From a statistical inference perspective, the primary task involves estimating the posterior distribution of the *latent variables* Θ given the observed network A . This represents a classic latent variable inference problem that can be addressed through different computational frameworks. Point estimates of the parameters can be obtained via the Expectation-Maximization algorithm [121, 39], while full posterior distributions are inferred using Bayesian methods such as variational inference [134, 136] or Markov Chain Monte Carlo sampling [137, 138] (see Sec. 3.8).

A critical aspect in using the SBM lies in *model selection*, the task of identifying the correct number of communities K . In network modeling, prevalent strategies for this task include the minimum description length principle [139, 140] and cross-validation techniques [141, 121]. The latter approach involves partitioning the network data into training and test sets: the model parameters are inferred from the training edges, while the model's predictive performance is evaluated on the held-out test set. This procedure is repeated over multiple data splits and for different candidate values of K , and the optimal K is then selected based on the best average predictive performance on the test sets. Alternatively, Bayesian nonparametric methods [139] offer a different solution by placing appropriate prior distributions on the model, allowing the number of communities to be inferred automatically during the posterior inference process.

Extensions with Node Covariates. The SBM framework has been extended in numerous directions to enhance its modeling capabilities. A particularly valuable extension incorporates *node covariates*, which are observed attributes associated with each node. This approach leverages both the network’s adjacency matrix A and a feature matrix X containing node attributes, enabling a more comprehensive analysis that combines structural and feature-based information.

A prominent example is the Contextual Stochastic Block Model (CSBM) [142, 143]. In this model, node covariates are accounted for by modeling them as a high-dimensional Gaussian mixture, where the cluster labels in the feature space coincide with the community labels in the network, and the centroids of these distributions are themselves latent variables. This embodies a general and intuitive assumption: that the covariates are generated from the community memberships, meaning a node’s latent group helps predict its attributes (see also [144, 122]).

In these covariate-enhanced models, the edge probability depends not only on the latent community structure but also on the observed node features. This joint modeling approach can be formalized in various ways. One common method introduces a correlation matrix $\beta \in \mathbb{R}^{K \times Z}$ that captures relationships between the K communities and Z attribute categories [122], explicitly modeling how node covariates influence community formation and connectivity patterns. This joint modeling approach can be formalized by modifying the edge probability function to:

$$P(A_{ij} \mid \Theta) = P(A_{ij} \mid f(u_i, v_j, W, \beta)), \quad (3.10.2)$$

Alternative approaches, however, challenge this dependency assumption. Some models [145, 146] instead posit independence between attributes and community membership. This follows a different generative intuition: an interaction between individuals is likely if they belong to the same community, if they share a relevant attribute, or both.

Crucially, the efficacy of integrating covariates is not guaranteed, as it relies on the availability of meaningful node covariates, an assumption that should not be taken for granted. As proven in several works, extra information on nodes can be non-informative for learning from graphs [147, 148, 122, 149], and can, counterproductively, degrade model performance. Therefore, the unified framework of combining topological and nodal information only improves community detection accuracy when the covariates are genuinely informative. This underscores the importance of developing models that can not only integrate node attributes but also automatically assess their relevance and potentially discard non-informative features, thus ensuring robust performance.

The MULTITENSOR (MT) Model. To provide concrete examples of the SBM framework discussed above, we now describe two of the generative models used in the

comparative study in Sec. 6. The MULTITENSOR (MT) model [121] provides a flexible framework for modeling multilayer networks [150], where multiple types of relationships (e.g., “friendship” and “colleague” ties in a social network) are represented as separate *relationship layers*, each with its own adjacency matrix. The term *layer* in this context refers specifically to the network’s adjacency matrix, not to be confused with neural network layers in GNNs. This formulation accommodates both directed and undirected networks, allowing for arbitrary mixtures of assortative, disassortative, core-periphery and other complex structural patterns.

In our implementation we focus on the single-layer case ($L = 1$), which reduces the model to a powerful, mixed-membership SBM for a single network represented by one adjacency matrix. The following formulas present the single-layer case; for the complete multilayer formalism see [121, 122]. The expected number of edges from node i to node j is given by a bilinear form:

$$M_{ij} = \sum_{k,l=1}^K u_{ik}v_{jl}w_{kl} \quad (3.10.3)$$

Here, u_{ik} and v_{jl} represent K -dimensional membership vectors for outgoing and incoming connections respectively, while w_{kl} are the affinity matrix entries encoding connection densities between communities.

The model assumes that each edge A_{ij} is drawn from a Poisson distribution with mean M_{ij} . This choice leads to tractable and efficient algorithmic implementations [151, 152, 124] compared to approaches using Bernoulli random variables [144, 148], while accommodating both binary and integer-weighted networks. Under the conditional independence assumption that edges are independent given the memberships and affinity matrices, the likelihood becomes:

$$P_G(A \mid U, V, W) = \prod_{i,j=1}^N \frac{e^{-M_{ij}} (M_{ij})^{A_{ij}}}{A_{ij}!} \quad (3.10.4)$$

which leads to the network log-likelihood:

$$\mathcal{L}_G(U, V, W) = \sum_{i,j} \left[A_{ij} \log \sum_{k,l} u_{ik}v_{jl}w_{kl} - \sum_{k,l} u_{ik}v_{jl}w_{kl} \right] \quad (3.10.5)$$

where constant terms independent of the parameters have been omitted as they depend only on the observed data and do not affect the optimization.

The parameters of the MULTITENSOR model are estimated using the EM algorithm described in Sec. 3.8. This involves introducing a variational distribution ρ_{ijkl} over pairs of groups (k, l) for each edge A_{ij} , which acts as the auxiliary distribution $q(Z)$ in the general EM framework. This distribution represents the probability that an edge exists

due to node i belonging to group k and node j belonging to group l .

As a generative model, MULTITENSOR naturally provides a framework for link prediction with partially observed data. The procedure, as outlined in Sec. 3.7 and Sec. 6.3, involves first using the observed edges to estimate the parameters $\Theta = (U, V, W)$ via the EM algorithm. The expected number of edges M_{ij} is then computed for all node pairs (i, j) using these parameters. Missing links are predicted by ranking unobserved node pairs according to their M_{ij} values, under the premise that true missing links will receive a higher expected value than non-existent ones.

The MULTITENSOR with Covariates (MTCOV) Extension. As a concrete implementation of the SBM extension with node covariates discussed previously, the MTCOV model builds directly upon the MULTITENSOR foundation by incorporating node attributes through a principled probabilistic framework. The key innovation is the joint modeling of the network structure and node attributes under a unified likelihood function. The model introduces a correlation matrix $\beta \in \mathbb{R}^{K \times Z}$ that captures relationships between the K communities and Z attribute categories.

For attribute modeling, MTCOV assumes that node covariates are generated from the membership vectors. The observed node attributes are organized in a binary feature matrix $X^{N \times Z}$, where each row is a one-hot encoded vector $x_i = (x_{i1}, \dots, x_{iZ})$ with $x_{iz} = 1$ if node i belongs to category z and 0 otherwise, ensuring $\sum_{z=1}^Z x_{iz} = 1$.

The probability of observing category z for node i is given by:

$$\pi_{iz} = \frac{1}{2} \sum_{k=1}^K \beta_{kz} (u_{ik} + v_{ik}) \quad (3.10.6)$$

where β_{kz} represents the probability of observing category z given membership in community k . This formulation ensures $\pi_i = (\pi_{i1}, \dots, \pi_{iZ})$ forms a valid probability vector where $\pi_{iz} \in [0, 1]$ and $\sum_{z=1}^Z \pi_{iz} = 1$ for all nodes i .

The attributes are modeled using a multinomial distribution with parameter π_i , yielding the likelihood:

$$P_X(X_i = x_i \mid U, V, \beta) = \prod_{z=1}^Z \pi_{iz}^{x_{iz}} \quad (3.10.7)$$

This formulation assumes attributes can be binned into a finite number Z of unordered categories. Multiple attributes can be combined by encoding each combination as a distinct category.

To satisfy the normalization constraints $\sum_{z=1}^Z \pi_{iz} = 1$, the parameters are constrained such that $\sum_{z=1}^Z \beta_{kz} = 1$ for all communities k , and $\sum_{k=1}^K u_{ik} = \sum_{k=1}^K v_{ik} = 1$ for all nodes i . These constraints provide a probabilistic interpretation for both the correlation matrix β and the membership vectors U and V . Assuming conditional independence of attributes

across nodes leads to the attribute log-likelihood:

$$\mathcal{L}_X(U, V, \beta) = \sum_{i=1}^N \sum_{z=1}^Z x_{iz} \log(\pi_{iz}) = \sum_{i,z} x_{iz} \log \left(\frac{1}{2} \sum_{k=1}^K \beta_{kz} (u_{ik} + v_{ik}) \right) \quad (3.10.8)$$

The complete model combines both the network structure likelihood $\mathcal{L}_G(U, V, W)$ and the node attribute likelihood $\mathcal{L}_X(U, V, \beta)$ using a scaling parameter $\gamma \in [0, 1]$ that balances their contributions:

$$\mathcal{L}(U, V, W, \beta) = (1 - \gamma) \mathcal{L}_G(U, V, W) + \gamma \mathcal{L}_X(U, V, \beta) \quad (3.10.9)$$

This formulation allows flexible interpolation between purely topological analysis ($\gamma = 0$) and purely attribute-based analysis ($\gamma = 1$), with the optimal value determinable through cross-validation based on the specific application goals.

Similarly to MULTITENSOR , the MTCOV model is trained using the same EM framework, but extends it by introducing an additional variational distribution h_{izk} for the node attributes. Here, ρ_{ijkl} again handles the edge assignments, while h_{izk} represents the probability that node i 's attribute z is explained by its membership to community k .

Chapter 4

Thermodynamic Approach to Neural Network Pruning

This chapter presents the core theoretical contribution of this thesis by developing a thermodynamic framework for model interpretability and its application to structured pruning [31], extending previous works [34, 35]. While focused on feature importance in Graph Neural Networks (GNNs), the methodology demonstrates remarkable generality.

We begin by framing the problem in Sec. 4.1, which details the interpretability challenge in deep learning models and reviews existing pruning approaches. In Sec. 4.2, we then draw an analogy between Stochastic Gradient Descent (SGD) and the dynamics of a thermodynamic system, formally defining a temperature for neural network parameters. The core of our approach is established in Sec. 4.3, where we introduce the SGD-temperature parameter. Sec. 4.4 then reveals a fundamental difference in parameter dynamics between optimizers, which not only motivates our generalized SGD-temperature for consistent pruning but also provides a novel lens through which to interpret and contrast the behavior of different optimizers.

Our empirical analysis begins in Sec. 4.6, which details how the SGD-temperature serves as a robust criterion for pruning input features in GNNs. We further quantify feature redundancy via cosine similarity in Sec. 4.7, providing a statistical perspective that corroborates our thermodynamic findings. The framework is then extended to the pruning of internal GNN embedding dimensions in Sec. 4.8, demonstrating its versatility beyond the input layer. Finally, in Sec. 4.9, we conduct a comparative analysis by applying the same temperature-based pruning to filters in Convolutional Neural Networks (CNNs), highlighting the general applicability of our method.

Overall, this work moves beyond traditional, magnitude-based pruning strategies by incorporating a dynamic measure of the parameters' role during training. This not only yields effective model compression but also provides a deeper, more mechanistic understanding of neural network learning.

4.1 The Interpretability Challenge in Deep Learning

Deep Learning and neural networks [3, 4, 153] are among the most successful tools in machine learning. Neural networks have largely been used as *black boxes*, deep and complex models able to adapt more and more precisely to data, despite the lack of a clear relationship between the input and the output representation, making it increasingly difficult to understand how the learning occurs. Addressing the problem of *interpretability* and *explainability* of deep learning algorithms has now become crucial for building trust and guiding the responsible development of the technology [154, 155].

One way to interpret neural networks is through the lens of statistical physics (see Sec. 2.6). Since the pioneering works by Jaynes [156] and Hopfield [68], up to the theory of (deep) Boltzmann machines [157, 158], the language of statistical mechanics has proven successful in describing inference and learning. Nowadays, many researchers are trying to exploit the connections between statistical mechanics and artificial neural networks to theoretically describe their phase transition during the optimization process of Stochastic Gradient Descent (SGD) (see for example [159, 160]). Some others have focused instead on the theory of stochastic processes to study the dynamics of the parameters under the SGD mechanism [75, 83, 161, 59]. Along the same vein, a more geometrical modeling via thermodynamics concepts appears in [162, 163, 164].

Besides the physically-inspired theoretical efforts, a more empirical way to interpret neural networks is through the practice of pruning [165, 166]. Pruning has the aim of removing unimportant weights from a deep learning model to reduce the model’s memory footprint and accelerate inference performance. It is well-known that neural networks usually work with an excess of parameters with respect to the ones actually needed for a good generalization [167]. This is because, to this day, there are no established theoretical guidelines in assigning the size of a neural network layer, e.g., the kernel size and the number of filters of a CNN or the embedding dimension in a GNN, and those are finally chosen by means of a blind and expensive grid search. Hence, the design of a pruning technique inspired by mathematical and physical reasoning is crucial to move towards a more accurate understanding of neural networks. At the same time, cleverly deciding the size of the model will make deep learning training more sustainable [168]. Common pruning techniques are *magnitude-based* [169, 170, 171] and *gradient-based* [172, 173, 174].

Magnitude-based Pruning. *Magnitude-based* pruning removes weights with small magnitude. For example, [171] prunes CNN filters based on the sum of their absolute kernel weights, i.e., their L_1 -norm. The motivation is that a convolutional kernel with a low norm detects less important features than those with a high norm [174]. This effect can be amplified during training by applying L_1 or L_2 regularization, which pushes unimportant kernels toward smaller values.

Gradient-based Pruning. In contrast, *gradient-based* pruning assesses importance by measuring the sensitivity of the loss to changes in each weight. A small gradient indicates that a change in that parameter has a low direct effect on the loss, making it a candidate for removal. The foundation for this approach was laid by Optimal Brain Damage (OBD) [172], which uses a second-order Taylor expansion and the diagonal of the Hessian to estimate the change in loss caused by removing a parameter. This was later refined by the Optimal Brain Surgeon (OBS) algorithm [173], which uses the full Hessian at equilibrium to more accurately identify weights whose removal minimally increases training loss. While more precise, this method is computationally expensive. To reduce this overhead, modern methods like [174] return to a first-order Taylor expansion.

The core challenge with a first-order approach is that after sufficient training, the gradient tends to zero, suggesting it offers little information. This is precisely why OBD and OBS regard the first-order term as zero and focus on the second-order term. However, differently from OBD and OBS, [174] proposes looking at the absolute difference in loss rather than the signed one, thus using the absolute value of the first-order term, whose expectation is stated to be proportional to the variance of the gradients. This variance proves empirically more informative for pruning than the mean, while entirely avoiding the computational burden of the second-order term. The method in [174] implements this with an iterative schedule: the model is first trained to convergence, and then undergoes alternating steps of pruning and fine-tuning. Crucially, at each pruning step, it greedily removes weights that exhibit the smallest gradients calculated from a single training step, continuing this cycle until a target efficiency is met.

A Physics-Informed Pruning Criterion. Our work unites the statistical physics perspective with the practical goal of pruning. We draw a stringent analogy between a neural network’s parameters and a thermodynamic system to formulate a novel, physics-inspired pruning criterion. Rather than relying on instantaneous gradient snapshots [174] or static weight magnitudes [169, 170, 171], we define a *temperature* parameter for weight structures, proportional to their average squared gradient over many epochs at equilibrium. Independently on the optimizer employed, weight structures with persistently low squared gradient prove to be the ones to prune in order to maintain the performance.

To demonstrate the efficacy of our temperature-based criterion, we employ a non-iterative procedure (see Sec. 4.5 for details). First, we train the network to equilibrium and identify persistently low squared gradient weight structures from their average temperature profile. We then perform a single pruning step and evaluate the resulting performance. The thermodynamic analogy underpinning this approach is detailed in Sec. 4.2, while our generalized pruning method, the SGD-temperature, is formally introduced in Sec. 4.3

4.2 From Classical Dynamics to Network Temperature

The starting point for our thermodynamic modelling is the observation that when a neural network is trained with Stochastic Gradient Descent (SGD) [56], or derived methods such as SGD with momentum [57] and Adam [58], the parameters change at each step as the loss function is minimized. After training has converged, i.e., when further optimization no longer improves performance, continued training still results in parameter changes due to the stochasticity of the optimizer. This behavior reminds us of individual atomic or molecular positions once thermodynamic equilibrium is reached. To establish a thermodynamic framework for neural networks, we draw an analogy between the dynamics of network parameters during optimization and particles in a statistical mechanical system. This allows us to define a temperature for neural networks based on parameter dynamics.

The Kinetic Definition of Temperature. To ground our thermodynamic analogy, we begin with the fundamentals of classical dynamics. We consider a system of N particles, where each particle k has a mass m_k and a position coordinate $\mathbf{x}_k$ in 3-dimensional space. Forces F_k are commonly regarded as gradients of a potential V and coordinates x_k evolve according to Newton’s law:

$$F_k = -\frac{\partial V}{\partial x_k} \quad a_k = \frac{d^2 x_k}{dt^2} = \frac{F_k}{m_k} \quad (4.2.1)$$

When discretized for numerical integration, coordinate evolution at each time-step t takes the form:

$$\begin{aligned} x_k(t) &= x_k(t-1) + v_k(t-1)\Delta t + \frac{1}{2}a_k(t-1)\Delta t^2 = \\ &= x_k(t-1) + v_k(t-1)\Delta t - \frac{1}{2m_k} \left(\frac{\partial V}{\partial x_k} \right)_{t-1} \Delta t^2 \end{aligned} \quad (4.2.2)$$

where m_k is the mass associated to each degree of freedom, and $v_k(t)$ is the instantaneous velocity. This expression describes constant energy dynamics. The instantaneous temperature $\mathcal{T}(t)$ is defined as the mean kinetic energy:

$$\mathcal{T}(t) = \frac{\mathcal{K}(t)}{k_B} = \frac{1}{k_B d} \sum_{k=1}^d \frac{1}{2} m_k v_k(t)^2 \quad (4.2.3)$$

where $\mathcal{K}(t)$ is the (instantaneous) kinetic energy, d is the number of degrees of freedom,

and k_B is Boltzmann's constant. The thermodynamic quantities are obtained as the time-averages of the corresponding instantaneous forms. For average kinetic energy K and temperature T we have:

$$K = \frac{1}{\tau} \int_0^\tau \mathcal{K}(t) dt = \frac{1}{\tau} \sum_{t=0}^\tau \mathcal{K}(t) \quad (4.2.4)$$

$$T = \frac{K}{k_B} = \frac{1}{\tau} \sum_{t=0}^\tau \mathcal{T}(t) = \frac{1}{k_B \tau d} \sum_{t=0}^\tau \sum_{k=1}^d \frac{1}{2} m_k v_k(t)^2 \quad (4.2.5)$$

where the averages are computed over a long enough sampling time τ to ensure fluctuations in the resulting values are negligible.

To perform constant temperature dynamics, contact with a heat reservoir is mimicked by periodically rescaling the velocity of each particle in order to adjust the average kinetic energy to the desired temperature, leading to the expression:

$$x_k(t) = x_k(t-1) + \beta v_k(t-1) \Delta t - \frac{1}{2m_k} \left(\frac{\partial V}{\partial x_k} \right)_{t-1} \Delta t^2, \quad (4.2.6)$$

where $\beta > 1$ ($\beta < 1$) is used to warm (cool), i.e., to transfer heat into (out of) the system.

Thermodynamic Analogy for SGD Optimizer. We now demonstrate how SGD update rule naturally fits into the thermodynamic picture we discussed above. In our thermodynamic analogy, we draw a parallel between the optimization of a neural network and the dynamics of a physical system. Specifically, we treat each network parameter w_k for $k = 1, \dots, N$ as a “particle” with a one-dimensional degree of freedom. The collection of all parameters defines the system's state in a high-dimensional “parameter space” of dimension N . The loss function $L(\mathbf{w})$ acts as the potential energy V governing the dynamics of these particles.

In NNs which employ SGD optimizer, both with and without momentum (see Sec. 2.4), parameters evolve according to the gradient of the mini-batch loss function L_B as:

$$m_k(t) = \beta m_k(t-1) + \eta \left(\frac{\partial L_B}{\partial w_k} \right)_{t-1} \quad w_k(t) = w_k(t-1) - m_k(t) \quad (4.2.7)$$

where $\beta = 0$ describes SGD without momentum and η denotes the learning rate. This corresponds to a discretized parameter velocity:

$$v_k(t) = w_k(t) - w_k(t-1) = -m_k(t) = \beta v_k(t-1) - \eta \left(\frac{\partial L_B}{\partial w_k} \right)_{t-1} \quad (4.2.8)$$

or

$$w_k(t) = w_k(t-1) + \beta v_k(t-1) - \eta \left(\frac{\partial L_{\mathcal{B}}}{\partial w_k} \right)_{t-1} \quad (4.2.9)$$

which is similar to eq. (4.2.6) above, if we let $\Delta t = 1$, $\eta = \frac{1}{2m_k}$, and identify L with a conservative potential driving the dynamics. Since in NN optimizations $\beta \in [0, 1)$, the evolution of the NN parameters driven by loss minimization is analogous to the evolution of particles driven by potential minimization when the particle velocities are quenched at each time-step, i.e., when heat is extracted from the system. In molecular dynamics this leads to constant temperature dynamics, which does not minimize the average energy but the Helmholtz or Gibbs potential, depending on volume or pressure constraints [175, 176].

For the time being, we assume that all parameters have the same mass, set to $m_k = 1$. This assumption is not necessary, but assigning masses would require further assumptions, such as thermal equilibrium within the system, leading to a less straightforward description. This leads to the slightly simplified form for the NN temperature, equivalent to the one in eq. (4.2.3) and (4.2.5) for a thermodynamical system:

$$\mathcal{T}(t) = \frac{\mathcal{K}(t)}{k_B} = \frac{1}{k_B N} \sum_{k=1}^N \frac{1}{2} v_k(t)^2 \quad (4.2.10)$$

$$T = \frac{1}{\tau} \sum_{t=1}^{\tau} \mathcal{T}(t) = \frac{1}{k_B \tau N} \sum_{t=1}^{\tau} \sum_{k=1}^N \frac{1}{2} v_k(t)^2 \quad (4.2.11)$$

where k_B is a constant to provide the correct physical dimensions, equivalent to Boltzmann's constant in thermodynamics. Velocities are computed as variations between steps of the corresponding parameters w :

$$v_k(t) = w_k(t) - w_k(t-1) \quad (4.2.12)$$

In the case of SGD, eq. (4.2.12) corresponds to eq. (4.2.8), implying that the temperature defined in eq. (4.2.11) directly measures the mean squared gradient of the loss.

The same applies when weight regularization is added to the loss:

$$L_{\text{reg}}(w) = L(w) + \frac{\lambda}{2} \sum_{k=1}^N w_k^2,$$

where λ is the weight decay factor. This term acts like an additive conservative force, pulling the parameters towards the origin.

4.3 SGD-Temperature as a Pruning Criterion

The thermodynamic framework provides more than just an analogy, we now explain why temperature can be used as a practical pruning criterion and we then introduce our proposed method: the *SGD-temperature*.

Temperature as a Pruning Criterion. Our temperature metric (Eq. 4.2.11) can be measured either between parameter updates or between epochs. While the average temperatures from both methods are proportional, we argue that the inter-epoch dynamics are more representative of the network’s stable state. This is because intra-epoch parameter evolution is influenced by mini-batch variations within the epoch. Specifically, we measure temperature between epochs in the equilibrium phase of the training, when loss and accuracy have converged. In this phase, even though the performance does not change much, weights continue to exhibit fluctuations as gradients are not precisely zero.

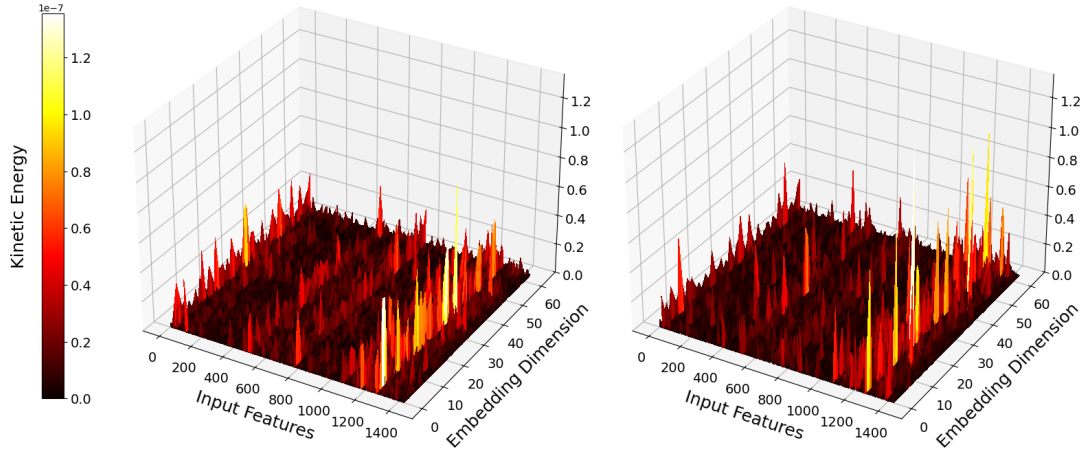


Figure 4.1: Thermal agitation at equilibrium for the two heads of the first GAT layer in the case of Cora dataset (see Sec. 4.5). These heatmaps correspond to a single training with SGD with momentum and without weight decay.

A key point is that these fluctuations are not uniform: as shown in Fig. 4.1, different weight groups exhibit distinct thermal agitation levels. We therefore propose temperature as a pruning criterion and find that “cold” weights, with small average squared gradient, are candidates for removal, while “hot” weights, characterized by larger average squared gradient, should be retained. An interpretation for this is the following. Since we are at equilibrium, we could assume the average gradient to vanish. In this case, the mean squared gradient would become proportional to the stochastic gradient variance and “hot” weights correspond to directions exhibiting larger variance. This effect is most pronounced in early layers, with deeper layers showing more homogeneous thermal patterns [177] (Sec. 4.8 and Sec. 4.9).

When pruning, we assess temperature in relative terms. This involves comparing the

thermal activity of different weight structures, defined as groups of weights, both against each other and against the entire network. This is applied by calculating the temperature for coherent parameter groups, serving as a structured pruning technique. For instance, in a CNN, one can prune entire filters by calculating the temperature for each filter, where N in Eq. (4.2.11) is the number of kernel weights within that filter (Sec. 4.9). Similarly, when pruning input features in a GNN, the temperature for each row of the first layer embedding matrix is considered (see Sec. 4.5 and Sec. 4.6), with N set equal to the embedding dimension (see Fig. 4.1).

The SGD-Temperature. The preceding analysis suggests that temperature could serve as an effective pruning criterion: different weight structures exhibit distinct levels of thermal agitation, and those with consistently low temperature, could be pruned while maintaining performance. However, for a practical and general-purpose pruning criterion, we require consistency across optimizers. The temperature defined in Eq. 4.2.11 fails this requirement, as the physical interpretation established in Sec. 4.2 breaks down when considering adaptive optimization methods like Adam. To achieve optimizer-agnostic pruning, we therefore introduce a more robust definition of temperature, which we call *SGD-temperature*. Before introducing it, we first examine how Adam’s update rule differs from SGD and why it alters the thermodynamic analogy established in Section 4.2

Adam incorporates both momentum $\hat{f}_k$ and per-parameter adaptive scaling [58], resulting in updates normalized by the square root of the uncentered gradient variance $\hat{s}_k$ (second moment) in the denominator:

$$w_k(t) = w_k(t-1) - \eta \frac{\hat{f}_k(t)}{\sqrt{\hat{s}_k(t)} + \epsilon}, \quad (4.3.1)$$

therefore

$$v_k(t) = w_k(t) - w_k(t-1) = -\eta \frac{\hat{f}_k(t)}{\sqrt{\hat{s}_k(t)} + \epsilon}, \quad (4.3.2)$$

where

$$f_k(t) = \beta_1 f_k(t-1) + (1 - \beta_1) \frac{\partial L_{\mathcal{B}}}{\partial w_k}, \quad s_k(t) = \beta_2 s_k(t-1) + (1 - \beta_2) \left(\frac{\partial L_{\mathcal{B}}}{\partial w_k} \right)^2,$$

and

$$\hat{f}_k(t) = \frac{f_k(t)}{1 - \beta_1^t}, \quad \hat{s}_k(t) = \frac{s_k(t)}{1 - \beta_2^t}$$

with standard values $\beta_1 = 0.9$, $\beta_2 = 0.999$, and $\epsilon = 10^{-8}$.

In [178], where the sign and magnitude of Adam’s update are studied separately, the

update magnitude is stated to be governed by the global learning rate η and the factor:

$$\frac{\hat{f}_k(t)}{\sqrt{\hat{s}_k(t)}} = \sqrt{\frac{1}{1 + \frac{\hat{s}_k(t) - \hat{f}_k(t)^2}{\hat{f}_k(t)^2}}}, \quad (4.3.3)$$

where $\hat{s}_k(t) - \hat{f}_k(t)^2$ is interpreted as an estimate of the stochastic gradient variance and $\frac{\hat{s}_k(t) - \hat{f}_k(t)^2}{\hat{f}_k(t)^2}$ is called *relative variance*. The smaller the relative variance, the higher the signal-to-noise ratio. With respect to the SGD update in eq. (4.2.7), the factor $\frac{\hat{f}_k(t)}{\sqrt{\hat{s}_k(t)}}$ is indeed proved to be responsible for shortening the update in directions of large relative variance. A detailed comparison of SGD and Adam is beyond the scope of this thesis. We observe, however, that Adam forces higher velocities on less informative parameters, even if their mean squared gradient is smaller, disrupting the analogy with statistical thermodynamics. This effect is illustrated in Fig. 4.2, which shows how groups of weights identified as “cold” under SGD appear “hot” under Adam and vice versa, when measuring temperature as the mean kinetic energy in eq. (4.2.11).

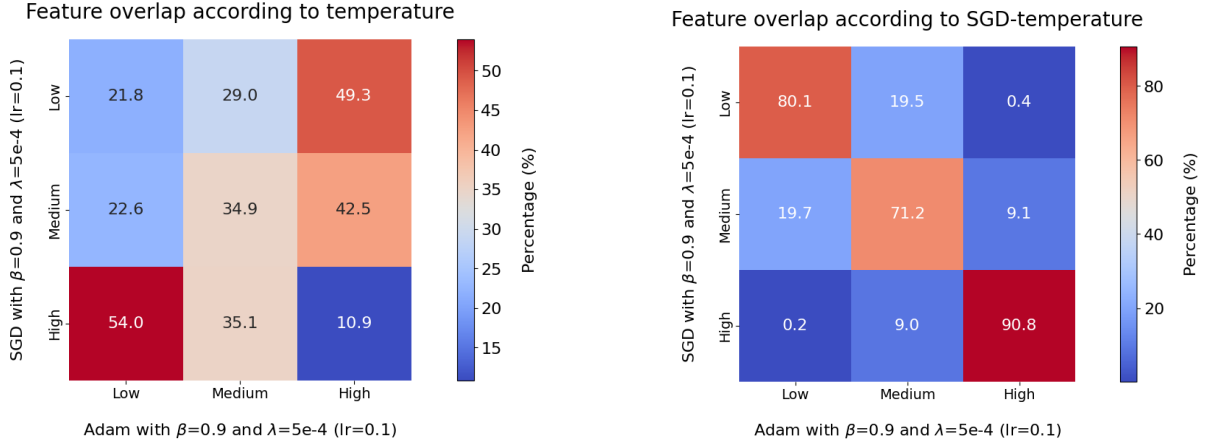


Figure 4.2: Heatmaps showing the overlap of Cora input features grouped into three quartiles of temperature in SGD and Adam, each feature corresponding to a row of weights in the GNN embedding matrix. On the left, the temperature calculated using eq. (4.2.11), while on the right, the SGD-temperature defined in eq. (4.3.4). Results from a single training.

To ensure a consistent pruning criterion across optimizers, since in the SGD regime, with or without momentum, our definition of thermodynamic temperature in eq. (4.2.11) is directly proportional to the squared gradient of the loss function, we generalize our notion of temperature and we call *SGD-temperature* the mean squared gradient of the loss:

$$T_{\text{SGD}} \stackrel{\text{def}}{=} \frac{1}{\tau N} \sum_{t=1}^{\tau} \sum_{k=1}^N \left(\frac{\partial L_{\mathcal{B}}}{\partial w_k}(t) \right)^2 \quad (4.3.4)$$

Under the SGD-temperature, weight structures identified as “cold” by SGD correspond

to “cold” ones in Adam, vice versa for the “hot” ones (see Fig. 4.2). We will see that the SGD-temperature serves as a consistent criterion for effective pruning across optimizers.

4.4 Comparison of Weight Dynamics in SGD and Adam

We want to further investigate and compare the parameter dynamics under SGD and Adam, by examining velocity patterns and Adam’s internal scaling factors. In Fig. 4.3 we plot the velocities for the weights in a single row of the GNN embedding matrix over 200 equilibrium epochs, specifically the row corresponding to the feature identified as the “coldest” or “hottest” based on the temperature in eq. (4.2.11) at equilibrium. As expected, under SGD the “cold” feature induces lower velocities compared to the “hot” one, whereas under Adam the opposite behaviour is observed. This divergence clearly demonstrates that the naive temperature definition in eq. (4.2.11) fails to provide uniform behavior across optimizers, a desirable feature for a reliable pruning criterion.

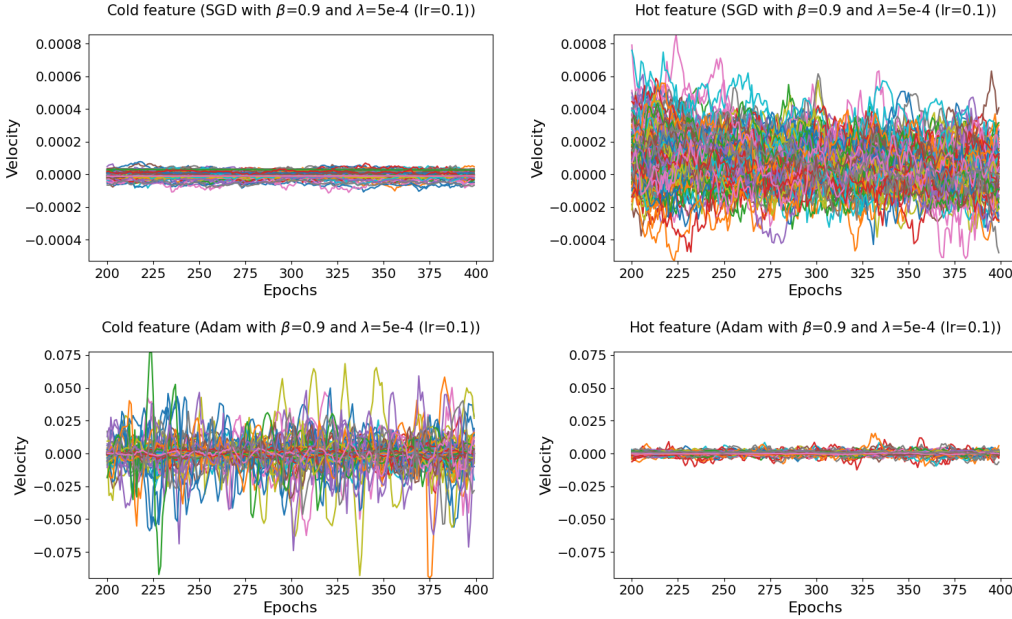


Figure 4.3: Velocity of row weights corresponding to the “coldest” (left) and “hottest” (right) Cora input features for SGD (top) and Adam (bottom).

We trace back this behavior to Adam’s scaling factor $\frac{\hat{f}_k(t)}{\sqrt{\hat{s}_k(t)}}$, which, as shown in Eq. (4.3.2), is proportional to the velocity of the weight update under Adam. In contrast, for SGD, the velocity is directly proportional to the gradient. In Fig. 4.4, we report the absolute rescaled learning rate $\frac{\hat{f}_k(t)}{\sqrt{\hat{s}_k(t)}}$ and the relative variance $\frac{\hat{s}_k(t) - \hat{f}_k(t)^2}{\hat{f}_k(t)^2}$ along the equilibrium epochs for the “coldest” and “hottest” features, when measuring the SGD-temperature in eq. (4.3.4). At each epoch, these values are obtained by averaging across the embedding dimension for the row of weights corresponding to each feature. As expected, the rescaled learning rate is consistently higher for the “coldest” feature, while

the relative variance is more than two orders of magnitude larger for the “hottest” one. This indicates that Adam assigns larger rescaled learning rates, i.e., larger velocities, to “colder” parameters with smaller average squared gradient, i.e., smaller gradient variance if we assume the average gradient to be zero. This suggests that the relative variance in Eq. (4.3.3) may contribute to amplifying updates for less informative features (with low SGD-temperature) and suppressing them for more informative ones. We hypothesize that this mechanism underlies the discrepancy between SGD and Adam when using Eq. (4.2.8) to define temperature, and we leave a deeper investigation of this effect to future work.

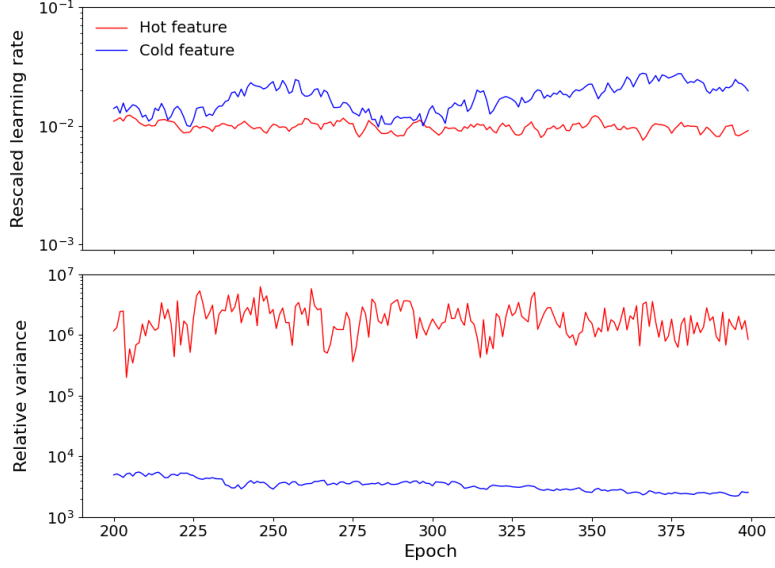


Figure 4.4: Absolute rescaled learning rate and relative variance for the “coldest” (blue) and “hottest” (red) Cora input features when training with Adam with weight decay.

4.5 Experimental Setup

Following the theoretical framework established in previous sections, we now detail the experimental methodology, including datasets, model architectures, and training procedures used to validate our approach. We analyze two GNN architectures, one to perform link prediction on Cora [107] and Citeseer [108] datasets and the other one to solve a node classification task on PubMed [179] and PPI [180] (see Fig. 4.5). For the link prediction task, we employ Graph Attention layers (GAT) [13], while for node classification a GraphSAGE [181] model. We design different GNN architectures for different tasks to prove the generality of our temperature-based pruning technique. For GNN state-of-the-art performances see [182] for link prediction and [183] for node classification.

Datasets Description. Cora, Citeseer and PubMed datasets are single-graph citation networks where each node represents a scientific publication. In Cora and Citeseer, the input feature for the node is a 0/1-valued word vector indicating the absence/presence of

Link Prediction:	Node Classification:
– GATConv(input $\rightarrow$ 64, heads=2)	– SAGEConv(input $\rightarrow$ 64)
– BatchNorm1d(64 * 2)	– Dropout $p = 0.05$
– LeakyReLU	– LeakyReLU
– GATConv(64 * 2 $\rightarrow$ 128, heads=2)	– SAGEConv(64 $\rightarrow$ 128)
– BatchNorm1d(128 * 2)	– Dropout $p = 0.05$
	– LeakyReLU
	– SAGEConv(128 $\rightarrow$ classes)

Figure 4.5: GNN architectures used for Link Prediction (Cora and Citeseer) and Node Classification (Pubmed and PPI).

the corresponding word in a dictionary extracted from the dataset: the dictionary consists of 1433 unique words in Cora and 3703 in Citeseer. For what concerns PubMed, each node is described by a TF/IDF weighted word vector from a dictionary of 500 unique words and is classified into one of 3 classes. TF/IDF, which stands for Term Frequency-Inverse Document Frequency, is a statistical measure used to evaluate the importance of a word in a document relative to a collection of documents. On the other side, PPI is a Protein-Protein Interaction dataset consisting of 20 graphs. Each node represents a protein and the edges in the graph represent the interactions between proteins. The input vector for each node consists of 50 features obtained by dimensionality reduction from a longer vector including biological and biochemical properties of the proteins. Each node can belong to multiple classes, representing different biological functions, and the total number of classes is 102. We test our temperature-based pruning technique on these datasets to distinguish between more and less useful node features, with the goal of reducing input dimensionality and explain how this information is used by GNNs in the learning process. The selected datasets, spanning binary bag-of-words, TF/IDF-weighted, and real-valued features across a broad range of dimensionalities, provide the diversity needed to rigorously assess the robustness of our method across different input representations and scales.

Training Details. We experiment with both SGD and Adam optimizers, setting the learning rate to 0.1. For SGD, we explore configurations with momentum ($\beta = 0.9$) and with or without L_2 weight decay ($\lambda = 5 \times 10^{-4}$). For Adam, we employ the standard first and second moment estimates ($\beta_1 = 0.9$ and $\beta_2 = 0.999$) and consider setups with or without L_2 weight decay ($\lambda = 5 \times 10^{-4}$).

For the link prediction task, since we are solving a binary classification problem on the edges, we use a binary cross-entropy loss. Indeed, we want to predict whether the

edge exists (label 1) or does not exist (label 0). The edge embedding is computed by applying the scalar product to the final layer embeddings of the connected nodes, and the likelihood of edge existence is scored via softmax. Also, in order to train and test our binary classifier, the dataset is enriched with non-existent edges with label 0, following the standard practice for link prediction with GNNs (see Sec. 3.4). For node classification, instead, we employ the negative log-likelihood loss on the number of classes. For all datasets, the split ratio is of 85% for the training set, 5% for the validation set and 10% for the test set. We stress that in the case of single-graph datasets (Cora, Citeseer and PubMed), the split is done in a transductive way on the edges (link prediction) or on the nodes (node classification), while for PPI dataset the split is inductive on the number of graphs. The model is trained 10 times with a fixed random seed and for each training we save the weights for the model performing the best on the validation set.

Experimental Methodology. As stated in Sec. 4.3, we measure the thermodynamic temperature of a neural network by averaging on the equilibrium epochs. We train the models in Fig. 4.5 until loss and accuracy converge, then we compute the mean squared gradients of the weights over 200 equilibrium epochs. To implement the temperature-based pruning for the input features, we measure each feature’s temperature by averaging the squared gradients across the output embedding dimension of the first GAT and GraphSAGE layers (see Fig. 4.1).

We test the effects of pruning the input features with two different experiments. First, we just zero-out the entries in the input vectors, by maintaining their original dimension, and we evaluate the model again on the pruned test set. Then, we suppress the features by reducing the dimensionality of the input vectors, and we retrain the model from scratch on the new graph. Furthermore, we compare our gradient-based SGD-temperature criterion against a magnitude-based approach [171] to assess its consistency for GNN pruning across optimizers. We also evaluate it against the original mean squared velocity temperature to justify our generalized definition. Finally, in Sec. 4.7, we compare our method with a straightforward statistical correlation analysis of input features.

4.6 Pruning Graph Neural Network Input Features

We now present a comprehensive analysis of the results obtained by applying our thermodynamic pruning framework to the input features of GNNs across four benchmark datasets: Cora [107], Citeseer [108], PubMed [179], and PPI [180]. Our investigation demonstrates how the SGD-temperature effectively identifies feature importance, enabling structured pruning while enhancing interpretability.

From the diagram of the link prediction architecture in Fig 4.5, we see how each head of the first GAT layer embeds the vector of features for each node from its input

dimension (1433 in Cora and 3703 in Citeseer) to a dimension of 64. Similarly, in the node classification case, the first GraphSAGE layer embeds PubMed and PPI respectively 500-dimensional and 50-dimensional input vectors to a dimension of 64.

If we measure the mean kinetic energy of the weights belonging to the embedding matrix of the first GAT and GraphSAGE layers, we notice that some features in the input vector induce higher thermal agitation than others (see Fig. 4.1 for a visualization in the case of Cora). Hence, we extract the temperature for each input feature by averaging on the output dimension of the first GNN layer and we use this as the criterion to prune useless input features. In the case of GAT layer, we average on both heads.

Feature Masking without Retraining. The first pruning experiment consists in zeroing-out a varying number of entries in the input vector of each node, going from the “hottest” to the “coldest” and vice versa. After pruning, we evaluate again the model (on the pruned test set) and we look at how the performance has changed with respect to the original setting.

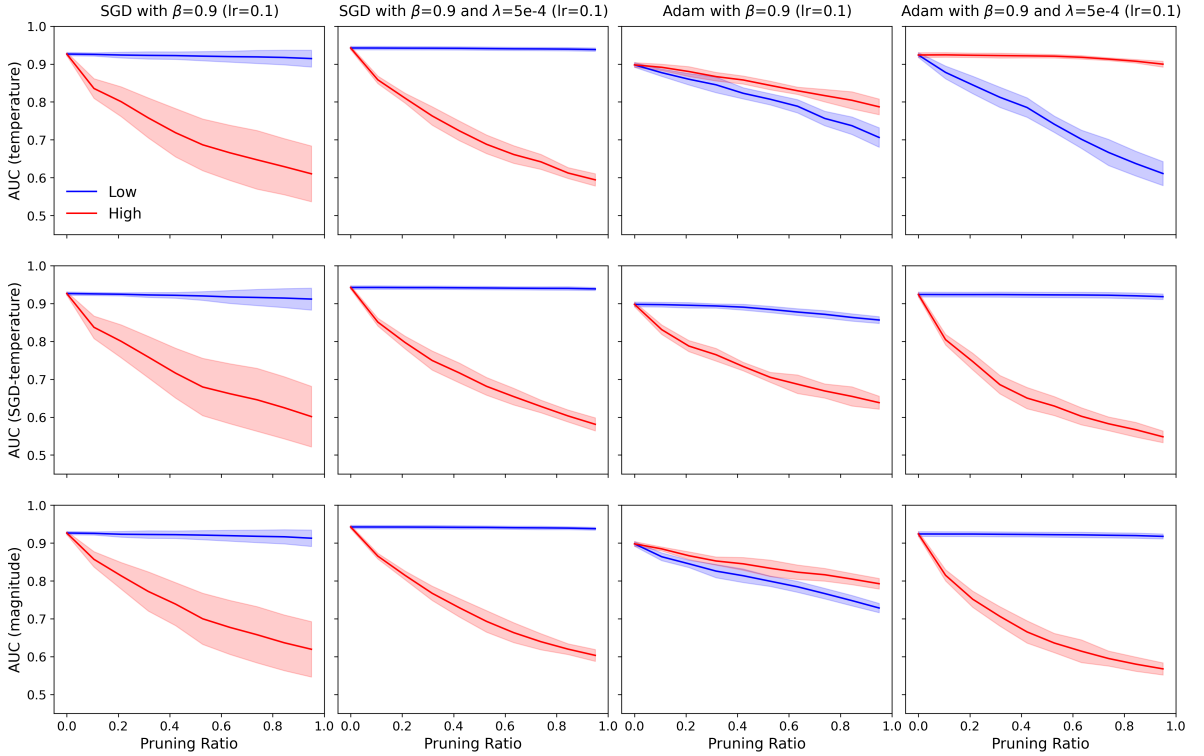


Figure 4.6: Performance after zeroing-out features and re-evaluating the model on the pruned test set based on temperature, SGD-temperature and magnitude (Cora dataset). Features are pruned by increasing ratio from the “hottest” to the “coldest” (red curve) and vice versa (blue curve). Shaded areas show standard deviation over 10 trained models. AUC indicates area under the curve.

In Fig. 4.6, we present the pruning curves obtained on Cora using the following different criteria:

1. the SGD-temperature (eq. 4.3.4), proportional to the mean squared gradient of the loss;
2. the temperature defined as proportional to the mean kinetic energy (eq. 4.2.11);
3. the magnitude, i.e., the mean absolute value of the weights.

We stress that when measuring the magnitude of each feature, we first average on the equilibrium epochs and then on the output dimension of the GNN embedding matrix, as for temperature and SGD-temperature.

Our analysis shows that SGD generally selects “cold” features for pruning, whereas Adam tends to select the “hot” ones, based on temperature, i.e., criterion (2). However these differences reconcile once we take into exam the SGD-temperature, i.e., criterion (1). On the other hand, magnitude proves to be an inconsistent indicator of feature importance across different optimizers and does not always suggest pruning features with low magnitude, as demonstrated for CNN filters in [171]. Notice that SGD-temperature yields consistent results across different optimizers: the features selected for pruning are those associated with lower mean squared gradient.

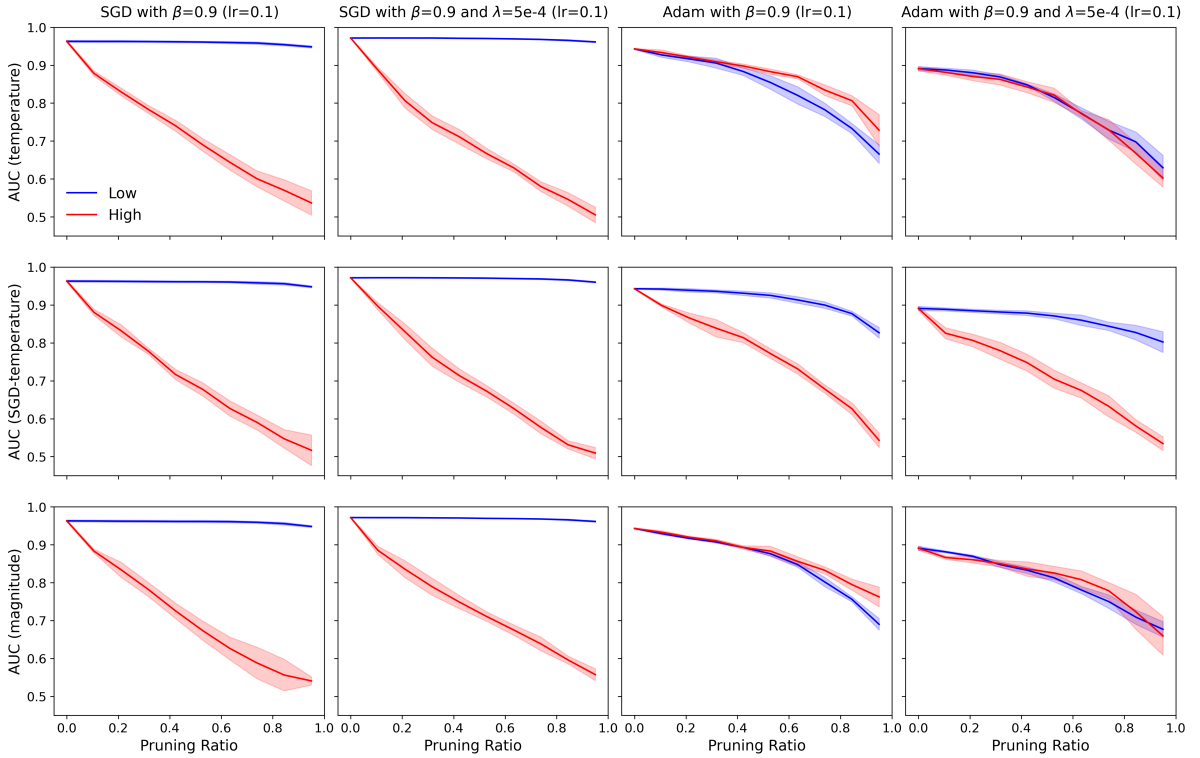


Figure 4.7: Performance after zeroing-out features and re-evaluating the model on the pruned test set based on temperature (Citeseer dataset).

Very similar results to those shown in Fig. 4.6 are observed for Citeseer dataset in Fig. 4.7. Pruning based on magnitude and temperature yields inconsistent results across optimizers as shown in Fig. 4.6. In contrast, SGD-temperature provides a clear behavior, consistently identifying low SGD-temperature features as the ones to prune. Furthermore, we observe that both temperature and magnitude align well with SGD-temperature when using the SGD optimizer. This behavior is also evident in Fig. 4.6 for the Cora dataset. However, this correspondence breaks down when using the Adam optimizer.

In our experiments on both Cora and Citeseer, where input features are binary and highly sparse, we find that pruning the least important features does not significantly impact performance until a large proportion of features is removed. We attribute this to the nature of the input features in these two datasets: feature vectors are represented as bag-of-words, with each dimension being binary (1 indicating the presence of a word and 0 its absence). Due to the high sparsity (about 90% of zeros), many features are effectively inactive (zero) and, as shown in Sec. 4.7, those are usually the ones deemed as useless. Consequently, pruning these features typically amounts to zeroing out elements that were already inactive, causing minimal performance impact. In contrast, pruning highly useful (active or nonzero) features leads to rapid drop in performance.

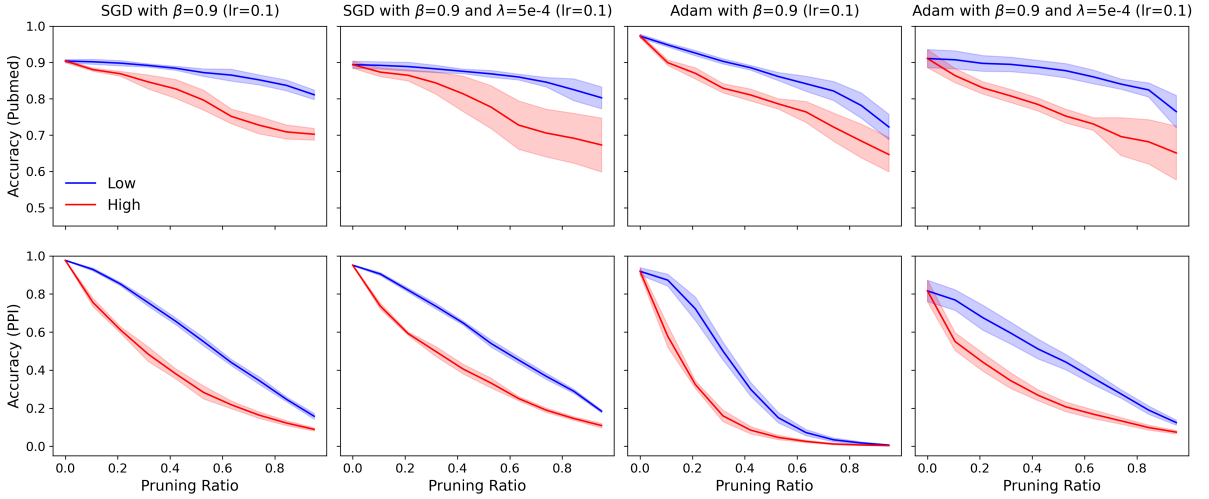


Figure 4.8: Performance after zeroing-out features and re-evaluating the model on the pruned test set based on SGD-temperature (Pubmed and PPI datasets).

For the node classification case, as shown in Fig. 4.8 for Pubmed and PPI datasets, we observe once again that performance remains closer to the original when pruning the features with low rather than high SGD-temperature. However, when pruning “colder” features, the performance degrades more rapidly in Pubmed and PPI compared to Cora and Citeseer. Although the feature vectors in Pubmed are also highly sparse (about 90% of zeros), the input dimensionality in Pubmed and PPI datasets is considerably lower

than that in Cora and Citeseer. Notably, the gap between the “hot” and “cold” pruning curves is especially narrow in PPI dataset (see Fig. 4.8), where the input vectors have a dimensionality of just 50, and all features seem to be important for the node classification task. Furthermore, in PPI, the features are real-valued rather than sparse binary, which may further contribute to the observed behavior.

Dimensionality Reduction with Retraining. Additionally, we broaden our analysis by applying temperature-based pruning to reduce the dimensionality of the input feature vectors, instead of simply zeroing out features, and then retraining the model from scratch on the pruned data. This process is repeated for progressively larger sets of features, removing them in order from “hottest” to “coldest” and vice versa. The model is retrained following the same procedure as before, and we compare its new performance to the original. It is important to highlight that reducing the dimensionality of the input vectors also reduces the size of the embedding matrices in the first layers of GAT and GraphSAGE, thereby decreasing the total number of trainable parameters in the model.

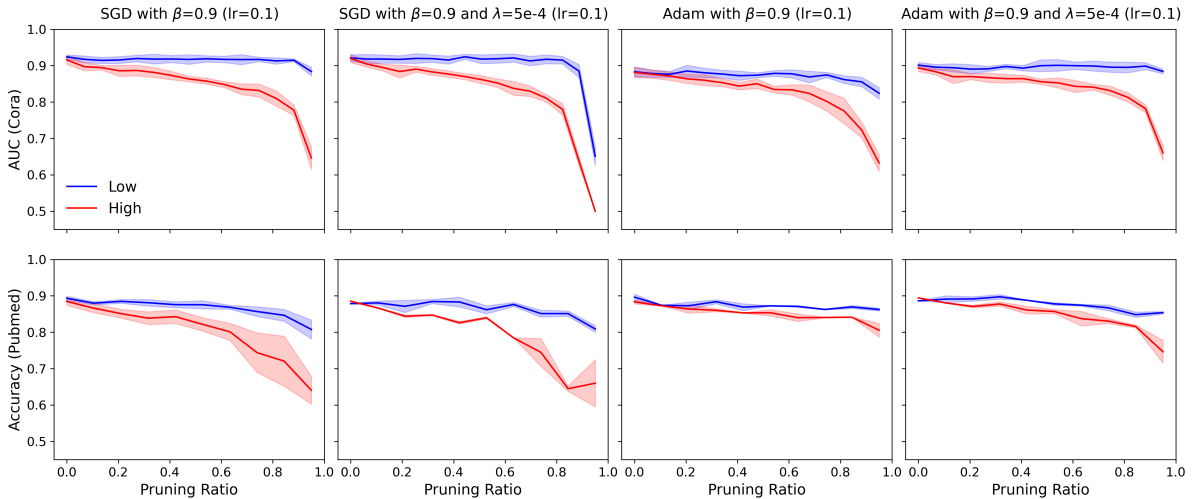


Figure 4.9: Performance after re-training and re-evaluating the model on the pruned dataset based on SGD-temperature (Cora and Pubmed datasets).

In Fig. 4.9, we present the performance curves on Cora and Pubmed datasets, when pruning based on SGD-temperature. In line with our initial pruning experiment, we find that pruning “colder” features has little effect on performance, whereas pruning “hot” features causes a more substantial degradation. Even in this scenario, we observe that a substantial portion of “cold” features can be pruned without significantly affecting performance in the highly sparse Cora and Pubmed datasets. Moreover, compared to the case where the model is not retrained, pruning “hot” features now leads to a smaller performance drop. We attribute this improvement to the model’s ability to adapt to the pruned dataset through retraining.

This further demonstrates that the model can maintain strong performance even with a greatly reduced input feature dimension, although performance inevitably declines when the feature dimensionality becomes very low (as also observed in [184, 33]). These observations are consistent with prior work on pruning GNN input features [26], which showed that in node classification tasks with GNNs on datasets like Cora, Citeseer, and Pubmed, accuracy decreases by only about 15% even after reducing the feature dimensionality by over 80%.

4.7 Feature Redundancy via Cosine Similarity

We want to quantify the informational redundancy among input features, which is the degree to which different features provide similar information across the dataset. For instance, in the Cora dataset where each node has 1433 possible features, we aim to identify which of these features are redundant with others. To this end, we construct a *feature redundancy matrix*. This symmetric matrix captures the pairwise cosine similarity between all feature vectors $\mathbf{f}_i$ and $\mathbf{f}_j$, where each $\mathbf{f}_i \in \mathbb{R}^{1 \times N}$ represents the values of the i -th feature across all N nodes. The cosine similarity between two features is given by:

$$\frac{\mathbf{f}_i \cdot \mathbf{f}_j}{\|\mathbf{f}_i\| \|\mathbf{f}_j\|}$$

which is the cosine of the angle between the two vectors $\mathbf{f}_i$ and $\mathbf{f}_j$. A value near 1 signifies the vectors are nearly parallel, meaning the two features exhibit almost identical patterns of variation across the node set and thus provide overlapping information. A value near 0 signifies the vectors are nearly orthogonal, meaning the features vary independently across nodes and provide distinct, non-redundant information.

The resulting redundancy matrix has shape $F \times F$, where F is the total number of features. To quantify how redundant a given feature is, we compute the average cosine similarity between that feature and all the others by averaging its corresponding row in the redundancy matrix, excluding the diagonal (i.e., self-similarity is ignored by setting diagonal entries to zero). This approach captures the global co-variation structure of features across nodes, providing a robust measure of redundancy for both binary and real-valued features.

In Fig. 4.10, we present the pruning curves derived from redundancy analysis on Cora and PPI datasets. These curves are generated by progressively zeroing out features, from highest to lowest redundancy and vice versa, and subsequently re-evaluating the model on the pruned test set. Interestingly, both figures reveal that pruning low-redundancy features, i.e., those that are more unrelated to others, results in better performance. Notably, these pruning trends closely mirror those observed when using SGD-temperature, as shown in Fig. 4.6 and Fig. 4.8.

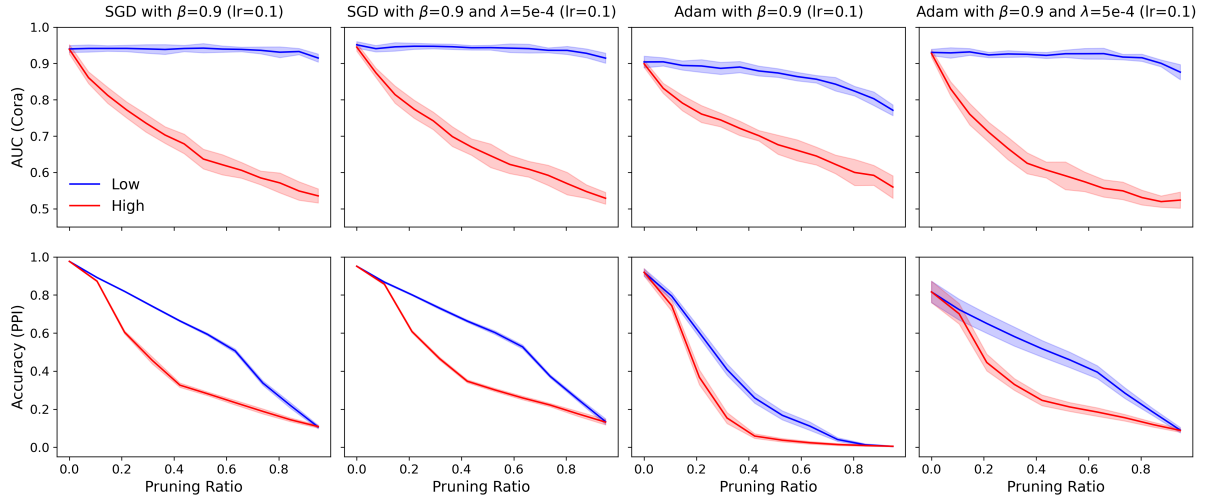


Figure 4.10: Performance after zeroing-out features and re-evaluating the model on the pruned test set based on redundancy (Cora and PPI datasets).

It is important to note, however, that redundancy analysis applies strictly to input features, whereas pruning based on T_{SGD} is more general and can be applied across all weight structures, as further discussed in Sec. 4.8 (embedding dimensions in GNNs) and Sec. 4.9 (CNN filters). Nonetheless, the agreement between these distinct methods on which features are candidates for pruning provides statistical corroboration for the temperature-based approach.

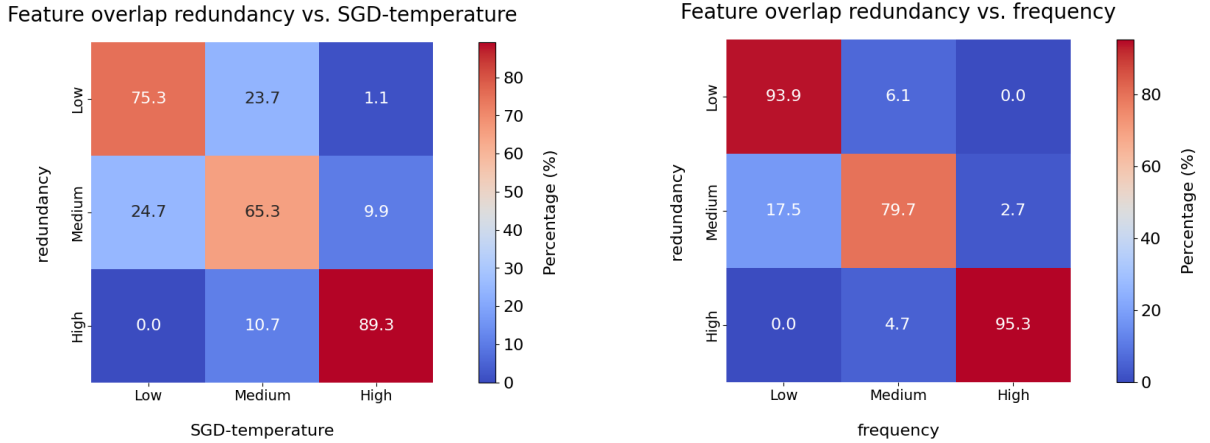


Figure 4.11: Heatmaps showing the overlap of Cora input features grouped into quartiles: redundancy vs. SGD-temperature (left) and redundancy vs. frequency (right).

As shown in Fig. 4.11 for the Cora dataset, features classified as “cold” by SGD-temperature generally align with those exhibiting lower redundancy. The same figure further reveals that low redundancy features also tend to be low frequency. Since features in Cora are binary, we quantify feature frequency simply as the average value of $\mathbf{f}_i$, which corresponds to the proportion of nonzero entries in $\mathbf{f}_i$. Consequently, low frequency features are those

that are predominantly zero throughout the dataset. This observation suggests that, in Cora, features which are often zero tend to be the less informative. This analysis provides compelling evidence that GNN’s message passing mechanism relies more on enhancing co-varying, widely distributed signals than on exploiting unique, node-specific information. Features that are rarely active (mostly zero across the nodes) contribute little to performance and can be pruned. This could explain why preserving redundant features is crucial for maintaining GNN performance when pruning.

4.8 Pruning Embedding Dimensions

After pruning rows in the GNN embedding matrix, each corresponding to a feature, we further extend the analysis by pruning columns, effectively reducing the embedding dimension of the GNN layer. This is done by computing the SGD-temperature of columns of weights, averaging this time over the input dimension. We show the corresponding pruning curves for Cora dataset in Fig. 4.12 and for Pubmed and PPI datasets in Fig. 4.13

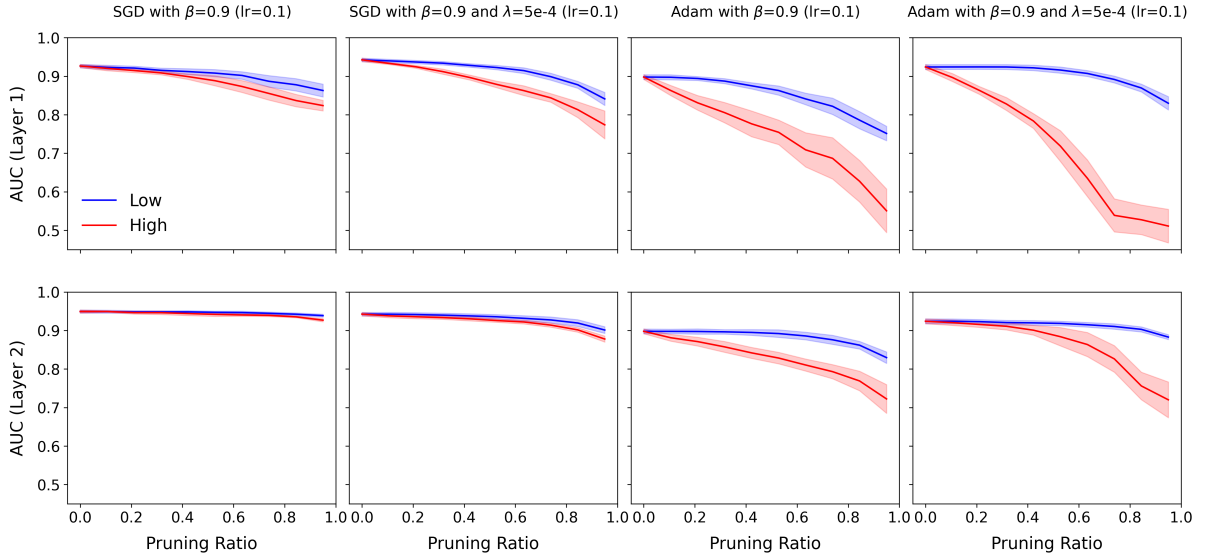


Figure 4.12: Performance after zeroing-out embedding dimensions in the first or second GNN layers and re-evaluating the model on the pruned test set based on SGD-temperature (Cora dataset).

Interestingly, while the tendency to prune “cold” structures persists, the pruning curves vary depending on the initial dimensionality of the input features. In Cora, which has high-dimensional input features, the performance difference between pruning “hot” and “cold” embedding dimensions is less pronounced compared to input feature pruning, especially under the SGD optimizer. Pruning “hot” embedding dimensions has a smaller impact on performance compared to pruning “hot” input features. This is likely due to

the embedding space having significantly lower dimensionality than the input space. The effect becomes even more evident in deeper layers: when examining the second GNN layer instead of the first one (see Fig. 4.12), the performance gap between the pruning curves narrows further. We attribute this behavior to the progressive reduction in temperature observed in deeper layers of the network. On the other side, for Pubmed, which has a lower input dimensionality than Cora, pruning “hot” embedding dimensions leads to a more noticeable performance drop with respect to pruning “cold” embedding dimensions, as in the case of pruning input features. Furthermore, in PPI, where the input dimensionality is even smaller and the embedding dimension exceeds the input dimension, pruning “cold” embedding dimensions in PPI performs even worse than pruning “cold” input features.

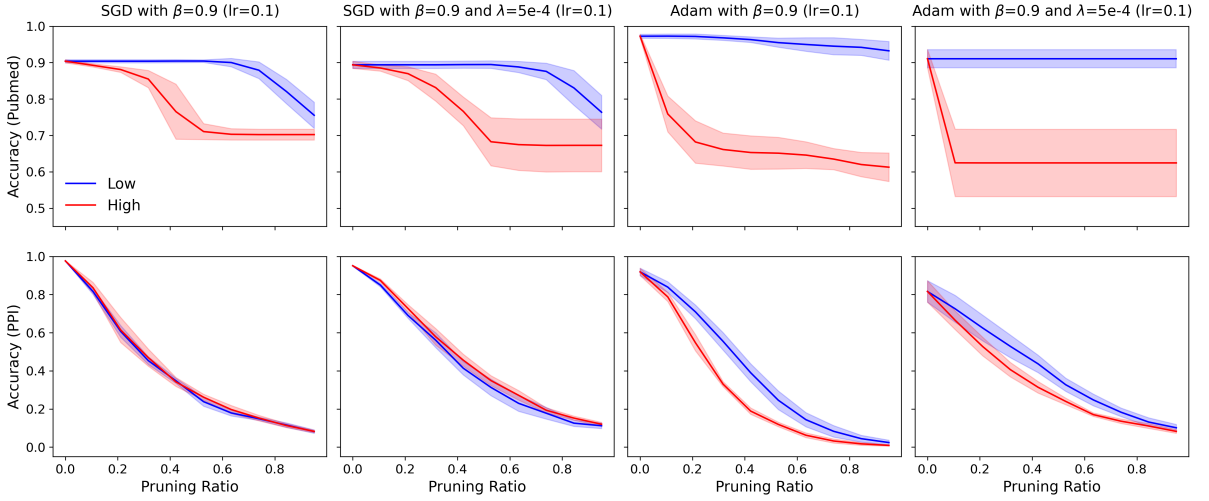


Figure 4.13: Performance after zeroing-out embedding dimensions in the first GNN layer and re-evaluating the model on the pruned test set based on SGD-temperature (Pubmed and PPI datasets).

4.9 Comparison with Pruning Convolutional Filters

We conduct a comparative analysis of the three pruning criteria (temperature, T_{SGD} , and magnitude), when pruning filters from the CNN architecture shown in Fig. 4.14. This architecture is the same we employed in our previous work on filter pruning [34] and we train it on MNIST dataset [185]. We experiment with both SGD and Adam optimizers, setting the learning rate to 0.1. For SGD, we explore configurations with momentum ($\beta = 0.9$) and with or without L_2 weight decay ($\lambda = 5 \times 10^{-4}$). For Adam, we employ the standard first and second moment estimates ($\beta_1 = 0.9$ and $\beta_2 = 0.999$) and consider setups with or without L_2 weight decay ($\lambda = 5 \times 10^{-4}$). Training is performed using cross-entropy loss over the 10 classes and repeated 10 times with a fixed random seed, with the number of equilibrium epochs set to 20.

CNN Architecture:

- Conv2d(input $\rightarrow$ 12, kernel=5)
- ReLU
- Conv2d(12 $\rightarrow$ 12, kernel=3)
- ReLU
- MaxPool2d(2)
- Dropout(0.25)
- Conv2d(12 $\rightarrow$ 24, kernel=5)
- ReLU
- Conv2d(24 $\rightarrow$ 24, kernel=3)
- ReLU
- MaxPool2d(2)
- Dropout(0.25)
- Linear(24 * 2 * 2 $\rightarrow$ 10)

Figure 4.14: CNN architecture for image classification on MNIST.

In Fig. 4.15 and 4.16, we show the pruning curves obtained when zeroing out filter weights for an increasing number of filters, going from the “hottest” to the “coldest” and vice versa. After pruning, we evaluate again the pruned model on the test set and we look at how the performance has changed with respect to the original setting. Fig. 4.15 corresponds to pruning the first layer, while Fig. 4.16 to the second one.

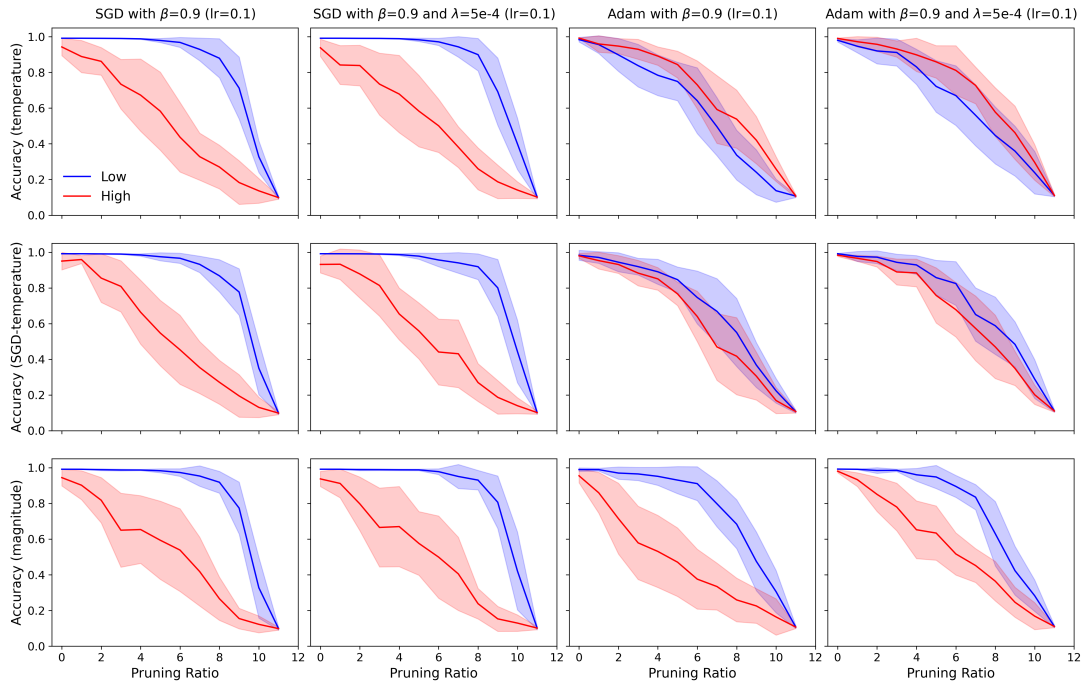


Figure 4.15: Performance after zeroing-out filters in the first CNN layer and re-evaluating the pruned model on the test set (MNIST dataset).

The magnitude criterion, measuring the absolute value of filter weights, exhibits

consistent behavior across optimizers: pruning filters with the lowest magnitude yields the best results, in agreement with the literature [171]. On the other side, when using temperature in eq. 4.2.11 as a pruning criterion, we again observe a divergence between optimizers: for SGD, pruning filters with the lowest kinetic energy (“cold” filters) is optimal, whereas for Adam, pruning filters with the highest kinetic energy (“hot” filters) performs better. This observation aligns with our earlier findings on feature pruning in GNNs in Sec. 4.6. Finally, the SGD-temperature criterion shows consistent behavior across optimizers, indicating that pruning filters with low mean squared gradient leads to the best performance. As noted in the previous sections, the performance difference between pruning strategies diminishes in deeper layers, with the gap between pruning curves narrowing significantly when moving from the first (Fig. 4.15) to the second layer (Fig. 4.16).

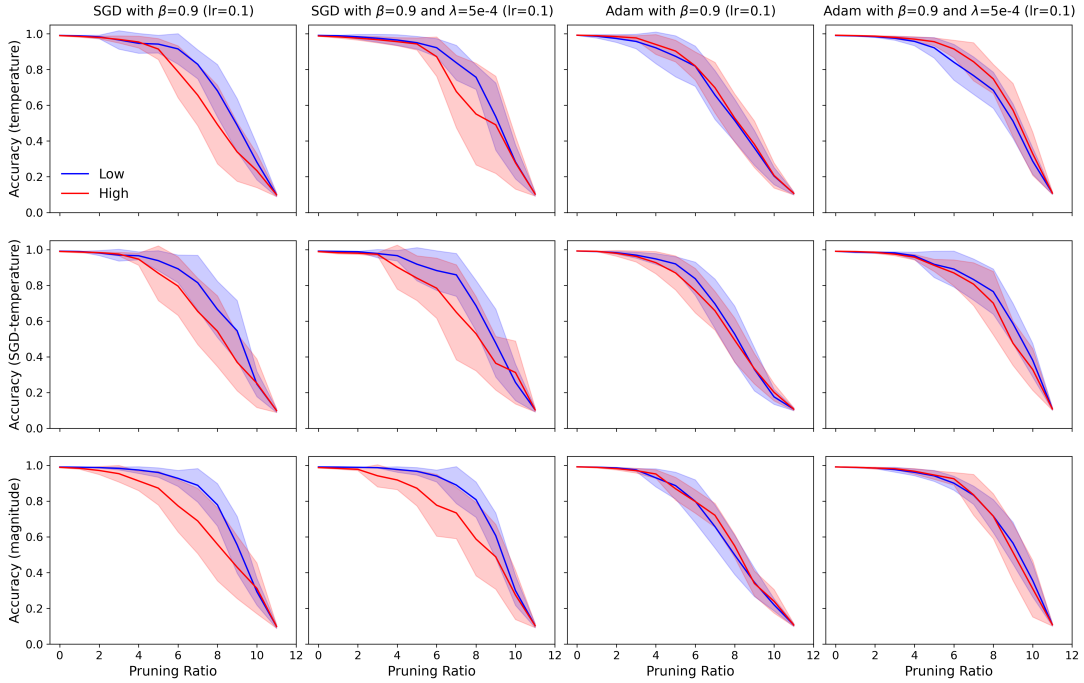


Figure 4.16: Performance after zeroing-out filters in the second CNN layer and re-evaluating the pruned model on the test set (MNIST dataset).

4.10 Discussion and Conclusions

This chapter has demonstrated how a thermodynamic analogy can be exploited to reduce the computational and storage cost of neural networks through a novel gradient-based pruning technique. Rather than relying on instantaneous gradient snapshots as in [174], our generalized notion of temperature, the SGD-temperature, is defined as the mean squared gradient of the loss over equilibrium epochs, when loss and accuracy have converged. This provides a robust, dynamic measure that consistently identifies

for pruning those “cold” weight structures characterized by low mean squared gradient, independently of the optimizer employed. Specifically, unlike pruning individual weights, this approach offers a structured method that avoids introducing sparsity [171, 186] and demonstrates general applicability to different neural networks and weight structures.

We employ this criterion to highlight which features in a GNN’s input node vectors are important and which can be discarded, by analyzing the gradients induced by the features on the weights of the first GNN layer. Our investigation reveals that the impact of pruning is governed by the initial dimensionality of input features. We find that GNNs with high-dimensional input features can be pruned to a large extent, whereas those relying on reduced-dimension features are more severely affected. A critical dimensionality threshold exists, below which performance decreases significantly. This principle holds true across different evaluation methods, whether we simply set features to zero and reevaluate the model on the test set, or actively reduce the feature dimensionality and retrain the neural network.

Furthermore, our analysis reveals that “cold” features, characterized by inducing low mean squared gradients on the weights, correspond to non-redundant ones, where redundancy measures the degree of co-variation or informational overlap between input feature dimensions across nodes in the dataset. This leads to the counterintuitive conclusion that successful GNN input feature pruning must preserve redundant, strongly co-varying features. Consequently, GNN’s message passing mechanism appears to rely more on enhancing widely-distributed, correlated signals than on exploiting unique, node-specific information. This mechanistic insight challenges conventional pruning wisdom that prioritizes the removal of redundancy.

The framework’s applicability extends effectively to GNN embedding dimensions, where the tendency to prune “cold” structures remains, although the distinction between “cold” and “hot” curves diminishes. We observe a nuanced effect dependent on the ratio between the dimensionality of the input features and the one of the embedding space. Pruning “hot” embedding dimensions has a smaller impact on performance compared to pruning “hot” input features when the embedding space has significantly lower dimensionality with respect to the input feature space. Conversely, when the input feature vector dimensionality is lower, pruning hot embedding dimensions has a stronger detrimental effect on the performance.

Further validating the method’s generality, we apply the pruning strategy to filters in CNNs. The results confirm that the SGD-temperature provides a consistent criterion across optimizers and leads to pruning weights with small mean squared gradients. In this context, we also verify that magnitude-based pruning remains consistent across optimizers, successfully targeting weights with small magnitudes as established in the literature [169, 170, 171].

Overall, our findings highlight a crucial distinction: while both SGD-temperature

and magnitude-based pruning are effective and consistent for CNNs, considering only weight magnitude is misleading for GNNs, where it fails to be consistent across optimizers when pruning input features. The gradient-based method provided by SGD-temperature directly solves this inconsistency, leading to a unified and robust pruning criterion. More fundamentally, by incorporating a thermodynamic perspective, we achieve not only effective model compression but also a deeper, more mechanistic understanding of the network’s learning process, ultimately improving the interpretability and effectiveness of pruning strategies across diverse neural network architectures and learning problems.

Chapter 5

Graph-Based Segmentation of 3D XCT Microstructures

This chapter presents the core practical contribution of this thesis by developing a novel graph-based framework for 3D XCT microstructures segmentation. We recast this task as a node classification problem, exploring the generalizability of Graph Neural Networks (GNNs) for segmenting irregular geometric shapes in computer vision.

We begin in Sec. 5.1 by establishing the context of deep learning for materials analysis through XCT segmentation, underscoring the role of 3D geometry and synthetic data for intricate, multi-phase microstructures. Sec. 5.2 then motivates our graph-based approach as a powerful alternative to conventional grid-based models, with Sec. 5.3 detailing core GNN advantages like permutation equivariance and semi-supervised learning.

Sec. 5.4 describes the experimental and synthetic datasets of a challenging six-phase Al-Si alloy composite, while Sec. 5.5 details our graph construction and node feature engineering. After discussing key GNN architectural challenges in Sec. 5.6, Sec. 5.7 first introduces a simpler proof of concept architecture [36] and then describes our custom Vision GNN (ViG) model [66]. Our training methodology, including pre-training on synthetic data and fine-tuning on experimental data, is outlined in Sec. 5.8.

A comprehensive evaluation benchmarks our GNN models against a state-of-the-art 3D U-Net [28]. The analysis is multi-faceted: Sec. 5.9 assesses segmentation performance using quantitative metrics and qualitative visual inspections; Sec. 5.10 examines per-class accuracy through confusion matrices and discuss the models' adaptability via fine-tuning; and Sec. 5.11 compares the computational cost of both approaches.

This comparison reveals a key trade-off between initial and final performance. While the 3D U-Net and GNN perform similarly on synthetic data, the U-Net generalizes better to experimental data before fine-tuning. Crucially, this advantage is reversed after fine-tuning on limited experimental data, where the GNN achieves higher final performance with far greater parameter efficiency, highlighting its remarkable adaptability.

5.1 Segmenting Multi-Phase XCT via Deep Learning

The quantitative analysis of XCT is undergoing a transformative era, driven by the successful application of deep learning algorithms [187, 188]. For material qualification, which depends on quantitative microstructural analysis, the workflow of XCT imaging paired with automatic segmentation represents a significant gain in efficiency compared to traditional time-consuming manual segmentation [189, 27, 28].

This work focuses on segmenting a challenging six-phase Al–Si alloy composite (a detailed description of these phases is provided in Sec. 5.4). Segmenting multi-phase composites is particularly difficult because different phases can share similar grayscale values in XCT data due to comparable X-ray attenuation coefficients [28], a challenge visualized in the experimental XCT slice of Fig. 5.1.

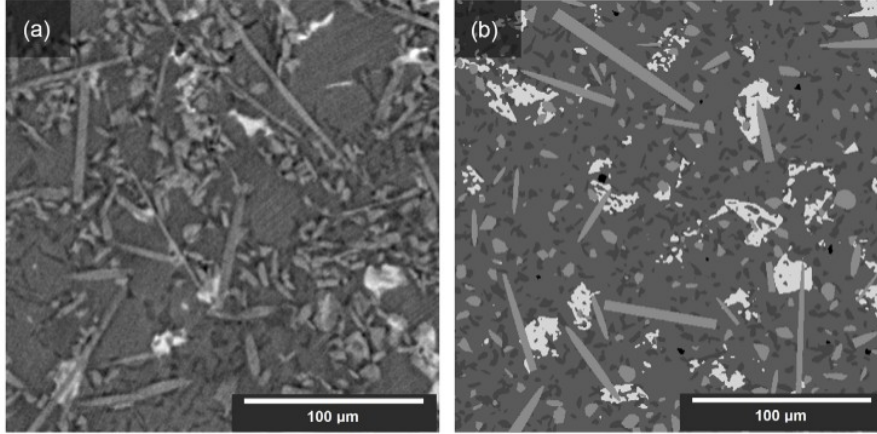


Figure 5.1: A comparison between experimental and synthetic data using 2D slices from the respective 3D volumes: (a) XCT reconstruction of an experimental Al–Si Metal Matrix Composite (MMC), and (b) a synthetically generated Al–Si MMC with limited resemblance. Image taken from [190].

This grayscale ambiguity causes traditional thresholding methods like Otsu’s [191] to fail, establishing Convolutional Neural Networks (CNNs) as the modern standard [27, 28]. Furthermore, as highlighted in [27, 28], when many phases have similar gray levels or the microstructures are geometrically complex, the 3D shape and spatial context of a phase become the most discriminative features for accurate identification. Consequently, segmenting the entire 3D volume is fundamentally more effective than analyzing independent 2D slices, as it allows the model to leverage this crucial geometric information.

A paramount challenge in supervised 3D deep learning for XCT segmentation is the profound scarcity of accurately labeled volumetric data. While for simple materials 2D CNNs achieve good segmentation results when trained on a few manually annotated slices, their performance degrades with increasing geometric complexity and number of phases, making 3D models essential. Training these 3D models, however, requires annotated 3D

volumes, demanding orders of magnitude more labeled data than their 2D counterparts. Manually annotating voxels across experimental volumes is prohibitively time-consuming and requires expert knowledge, creating a critical bottleneck for model development. To circumvent this limitation, the field increasingly relies on synthetically generated datasets [192, 193, 194, 28]. The fidelity of these synthetic volumes is crucial: the more they resemble real microstructures in terms of texture, morphology, and statistical properties, the better the model will generalize to experimental data [195]. Fig. 5.4 illustrates an example of this fidelity discrepancy for the Al-Si composite analyzed in this study, further details of which are provided in Section 5.4.

These challenges of data scarcity and geometric complexity motivate a critical examination of model architectures and their inductive biases, i.e., the fundamental assumptions a model makes about data structure (see Sec. 3.2). The current dominant paradigm for dense segmentation tasks, including XCT analysis [196], is built upon the U-Net architecture [41], a convolutional encoder-decoder network (see Sec. 2.7) originally developed for medical imaging [196]. Its success has cemented CNNs, including 3D [197] and 2D [198] variants, as the established paradigm for modern computer vision tasks such as classification, object detection, and semantic segmentation [199, 86, 200]. More recently, Vision Transformers have emerged as a powerful alternative for various visual tasks [201, 202, 203].

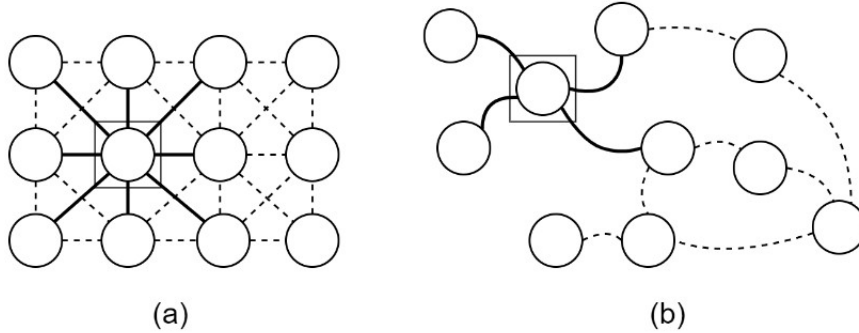


Figure 5.2: A conceptual illustration of different data representations: (a) a Euclidean graph (grid graph), such as the pixels of an image, and (b) an arbitrary graph with an irregular, non-Euclidean structure. Image from [204].

While CNNs view an image as a regular grid of pixels and Transformers treat it as a sequence of patches, we propose a different inductive bias: viewing the volume as a graph and performing segmentation through Graph Neural Networks (GNNs). Although GNNs have been predominantly applied to point cloud data in computer vision [205, 206, 207], their application to the dense semantic segmentation of volumetric imaging data, such as XCT, remains less explored. This approach offers a flexible representation capable of modeling both complex, irregular structures and regular grids (see Fig. 5.2). In this work, we specifically adopt a grid-as-graph perspective, treating the volume as a graph where voxels are nodes connected to their spatial neighbors (implementation details are

in Sec. 5.4. We systematically compare the inductive biases of GNNs and CNNs in Sec. 5.2, and detail the core advantages of the graph-based approach for segmenting complex microstructures in Sec. 5.3.

5.2 Spatial vs. Relational Inductive Biases

The performance of a deep learning model is profoundly guided by its inductive bias. This refers to the fundamental assumptions the model makes about the structure of the data it will process, which are built into its architecture and shape how it generalizes from examples. A well-aligned inductive bias, one that matches the underlying structure of the problem, is crucial for effective learning, especially with limited data [98] (a broader discussion is in Sec. 3.2).

CNNs, particularly the U-Net architecture [41], constitute the current state-of-the-art for semantic segmentation of volumetric images. Their success is largely attributable to a powerful *spatial inductive bias*: by processing data as a regular grid, they are exceptionally suitable for learning local, translation-invariant patterns. However, the complex and often irregular geometries inherent in 3D microstructures suggest that a different inductive bias, one focused on relational structure rather than fixed spatial locality, may be more beneficial.

This motivates our investigation of GNNs as a promising alternative, rooted in their *relational inductive bias*, i.e., the core principle that a node’s properties are shaped by its connections and neighbors.. In this section, we examine their fundamental operational differences. A foundational way to understand this is to compare their operations on a shared grid-graph structure (see Fig. 5.2), where each pixel/voxel is a node. While both architectures perform local feature aggregation, their parameterization schemes reveal fundamentally different inductive biases.

In a 3D CNN, each convolutional filter in layer l is defined by a weight tensor $I^{(l)} \in \mathbb{R}^{F^{(l)} \times F^{(l)} \times F^{(l)} \times C^{(l)}}$ containing $F^{(l)} \times F^{(l)} \times F^{(l)} \times C^{(l)}$ learnable parameters (e.g., $3 \times 3 \times 3 \times 1 = 27$ parameters for a $3 \times 3 \times 3$ kernel applied to a grayscale input where $C^{(l)} = 1$). The filter operates on an input $X^{(l)} \in \mathbb{R}^{D^{(l)} \times H^{(l)} \times W^{(l)} \times C^{(l)}}$, where $D^{(l)}$ is the depth, $H^{(l)}$ is the height, and $W^{(l)}$ is the width, which for the first layer ($l = 1$) contains the raw grayscale voxel intensities. This operation produces $K^{(l)}$ output channels through a position-dependent, weighted aggregation, with each channel corresponding to one of the layer’s $K^{(l)}$ distinct filters. This directly extends the 2D convolution defined in Sec. 2.7 to three spatial dimensions. For a central voxel at (x, y, z) , the output is computed as:

$$z_{(x,y,z)}^{(l)} = \sum_{i=1}^{F^{(l)}} \sum_{j=1}^{F^{(l)}} \sum_{k=1}^{F^{(l)}} \sum_{c=1}^{C^{(l)}} \left(I_{i,j,k,m}^{(l)} \cdot X_{(x+i,y+j,z+k,m)}^{(l)} \right) + b^{(l)},$$

where indices i, j, k traverse the $F^{(l)} \times F^{(l)} \times F^{(l)}$ spatial neighborhood and c indexes the input channels. Critically, while the same kernel is convolved across all spatial locations (weight sharing), the weights $I_{i,j,k,c}^{(l)}$ are unique for each combination of relative spatial position (i, j, k) and input channel c within the kernel’s local neighborhood. This implements a spatially-aware (direction-dependent) message passing that gives CNNs a strong inductive bias for learning grid-aligned features.

The GNN operates on a fundamentally different principle, which we implement using a grid-as-graph perspective. Specifically, we construct a graph from the volumetric data by treating each voxel as a node and connecting it to its 6 nearest neighbors (see Sec. 5.5). This reformulation preserves the inherent spatial connectivity of the original grid while recasting it into a graph structure, which is then passed as input to the GNN. A graph convolution, such as in GraphSAGE [181] (see Sec. 3.3), performs an update for a node i using shared weight matrices that are agnostic to the specific spatial arrangement of neighbors. Let $\mathbf{h}_i^{(\ell)} \in \mathbb{R}^{d_\ell}$ be the feature vector (or embedding) of node i at layer ℓ , and $\mathcal{N}_i$ its set of neighbors. The update to $\mathbf{h}_i^{(\ell+1)} \in \mathbb{R}^{d_{\ell+1}}$ is:

$$\mathbf{h}_i^{(\ell+1)} = \text{ReLU} \left(\mathbf{W}_\phi^{(\ell)} \mathbf{h}_i^{(\ell)} + \mathbf{W}_\psi^{(\ell)} \cdot \frac{1}{|\mathcal{N}_i|} \sum_{j \in \mathcal{N}_i} \mathbf{h}_j^{(\ell)} \right),$$

where $\mathbf{W}_\phi^{(\ell)}$ and $\mathbf{W}_\psi^{(\ell)}$ are the shared learnable weight matrices applied to every node in the graph. Critically, these matrices transform the node embeddings from dimension d_ℓ to $d_{\ell+1}$, operating purely in the feature space. This differs fundamentally from the CNN’s paradigm, where the transformation is defined across spatial positions and input/output channels. The aggregation function (here, a mean) is permutation-invariant, making it agnostic to the directional relationships between the central node and its neighbors. This implements a spatially-agnostic (direction-independent) message passing that learns from the relations between connected nodes in the feature space rather than their fixed spatial coordinates.

This fundamental shift, from many direction-specific weights to shared weight matrices with invariant aggregation, defines the GNN’s more flexible but less spatially explicit inductive bias, which directly underlies the trade-offs in segmentation performance explored in this work.

5.3 GNN Advantages for Microstructures Segmentation

We now elaborate on the specific advantages of GNNs for segmenting complex microstructures: an inherent flexibility to geometric transformations that is less constrained by a fixed grid, and superior data efficiency through relational learning that can help alleviate data scarcity.

GNN Flexibility for Irregular Structures. A core limitation of CNNs is their reliance on a fixed, Euclidean grid. The convolutional kernel’s operation exhibits a powerful translational equivariance (see Sec. 3.2), meaning that a shift in the input leads to a corresponding shift in the output feature map. This bias is highly effective for recognizing shapes that are simple translated instances of a learned pattern. However, this same bias can be a constraint for segmenting the intricate and often non-grid-aligned morphologies found in microstructures (see Fig. 5.4), as these complex shapes are not mere translations of a kernel-sized template.

GNNs overcome this limitation by representing the 3D volume as a graph, liberating the model from the rigid grid. Rather than learning translation-invariant patterns, the graph convolution learns the relations between connected nodes in the feature space. This operation is inherently permutation equivariant (see Sec. 3.2), ensuring predictions depend solely on which other nodes each node is linked to, not the arbitrary order of its neighbors, a property we posit is better suited for learning irregular shapes. Furthermore, the GNN’s operational principle offers a more flexible form of feature aggregation. Although we employ a grid-as-graph representation with a fixed number of neighbors per node, i.e., 6, the core GNN operation is inherently adaptable to heterogeneous neighborhood structures where nodes can have varying numbers of connections. This foundational flexibility contrasts sharply with CNNs, where kernels are fundamentally tied to a fixed grid and process a neighborhood of predetermined size.

Furthermore, when node features are coordinate-frame independent, such as scalar properties like grayscale intensity, the resulting GNN segmentations exhibit classification invariance. This ensures a microstructural phase is consistently identified regardless of its position or orientation in the volume. This stands in sharp contrast to applications where node features are vectors in a global coordinate system, e.g., 3D atom positions in a molecular graph. In such cases, as GNN’s predictions are not inherently rotation-invariant, the model’s predictions become tied to the molecule’s orientation, as the node interactions depend on the coordinate values in their feature vectors. This limitation has motivated the development of specialized $SO(3)$ -equivariant GNNs [208].

GNN Data Efficiency. A powerful advantage of GNNs is their innate suitability for semi-supervised learning [11], a paradigm where labels are available for only a small subset of nodes. GNNs excel in this setting by leveraging the graph’s connectivity to propagate and enrich node representations through their neighbors, effectively inferring labels for unmarked nodes based on the learned contextual relationships between nodes. This capability stands in stark contrast to CNNs, where the convolution of kernels across the grid requires dense, extensive labeling for effective learning. This process is illustrated in Fig. 5.3 which shows a multi-layer Graph Convolutional Network (GCN) [11] architecture and demonstrates how the model successfully separates distinct classes in its learned

representation space, even when trained with very few labels.

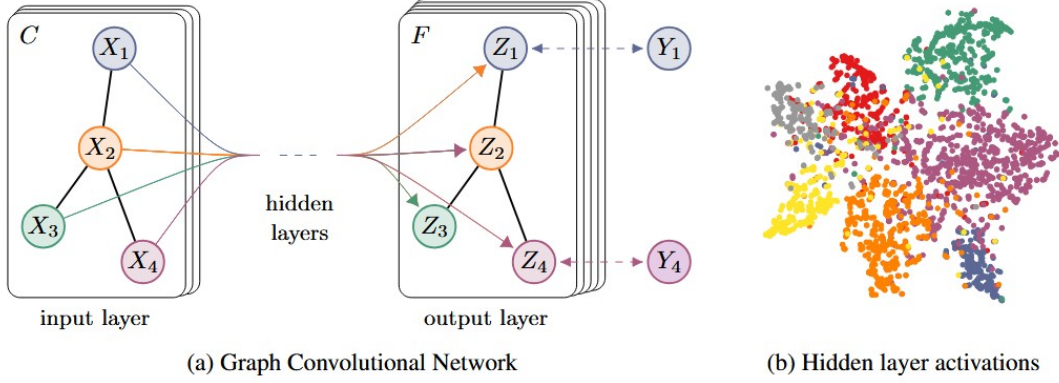


Figure 5.3: Semi-supervised learning with a Graph Convolutional Network (GCN) [11]. (a) Multi-layer GCN, which takes C input feature channels and produces F output feature maps. The graph structure (edges shown as black lines) is shared across all layers, and node labels are denoted by Y_i . (b) Low-dimensional t-SNE visualization [209] of hidden layer representations from a two-layer GCN trained on the Cora dataset [107] using only 5% of the labels, demonstrating effective class separation (colors indicate classes). Image source: [11].

Although our framework is not semi-supervised strictly speaking, as we perform standard supervised fine-tuning on a fully-labeled sub-volume, we directly exploit this same relational inductive bias. We leverage this capability to compensate for the limited resemblance between our synthetic training data and real experimental volumes. Our strategy follows a two-stage process. First, the model is pre-trained on large, fully-labeled synthetic volumes to learn fundamental segmentation features. Then, it is adapted to real experimental data through fine-tuning on a very small labeled volume.

This approach drastically reduces the annotation burden, as the GNN can generalize from a handful of annotated voxels to segment an entire volume by leveraging the learned contextual relationships between nodes in the sub-volume. Our specific fine-tuning methodology is detailed in Sec. 5.8, with the resulting performance gains presented and discussed in Sec. 5.9 and 5.10.

5.4 Synthetic and Experimental Data

We segment an AlSi12CuMgNi Metal Matrix Composite (MMC) reinforced with 7%vol Al_2O_3 short fibers and 15%vol SiC particles, as detailed in [28]. The Synchrotron X-ray Computed Tomography (SXCT) data were collected at the BAMline beamline at BESSY II synchrotron in Berlin, Germany. More details about the SXCT imaging and the analysis of the microstructures are contained in [27]. Fig. 5.4 shows 512×512 pixel cross-sections of the AlSi12CuMgNi MMC, contrasting experimental and synthetic data.

The experimental slices illustrate the core segmentation challenge: distinct microstructural phases can exhibit nearly identical gray levels due to similar X-ray attenuation coefficients, making them difficult to distinguish by intensity alone.

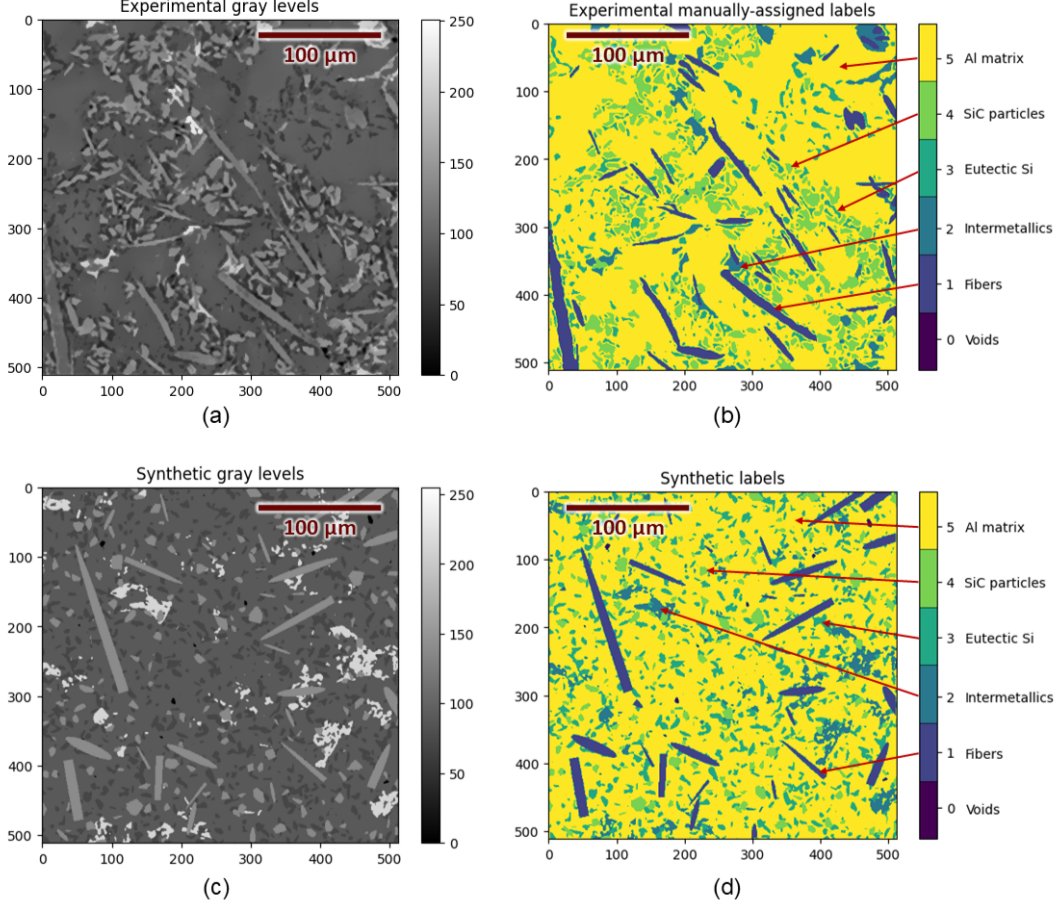


Figure 5.4: Experimental (top: (a), (b)) and synthetic (bottom: (c), (d)) 512×512 pixel cross-sections of the AlSi12CuMgNi Metal Matrix Composite (MMC). On the left the gray level (an integer between 0 and 255), on the right the label corresponding to its class (a number ranging from 0 to 5). Arrows point to the six phases for clarity.

Synthetic Data Generation. As established in Sec. 5.1, 3D shape and spatial context are crucial for identifying phases with similar grayscale values or complex geometries, making full 3D segmentation more effective than 2D analysis. However, this approach requires substantial labeled data. To meet this demand, we train our GNN on a synthetic dataset (Fig. 5.4), generated using the procedure detailed in [28]. Here, we provide a brief summary: synthetic Al–Si MMC microstructures are created using BAM’s in-house MATLAB library (BAM SynthMAT [28]). This generation process accounts for structural and grayscale resemblance, positioning the microstructures into a single volume using a priority function [28]. The priority function is necessary due to the inhomogeneous spatial distribution of particles and volume fractions in the experimental data. Furthermore,

voxel grayscales are assigned by modeling local contrast, noise, and blur to mimic the spatial and intensity characteristics of the experimental phases.

The final synthetic volume comprises two raw 8-bit binary files: one containing the grayscale values (ranging from 0 to 255) and the other containing labels for the various synthetic phases (with labels ranging from 0 to the number of phases-1). For the Al–Si MMC under investigation, there are six distinct phases: voids, Al_2O_3 fibers, intermetallics (IMs), eutectic Si, SiC particles, and the Al matrix. Figure 5.4 displays a cross-section from one of the synthetic volumes along with its corresponding labels. Since the phases are not homogeneously distributed in the volume, we carefully weight the contribution of each phase when computing the loss function (see Sec. 5.8). Specifically, we take into account the average occurrence of each phase across the synthetic volumes (see Table 5.1). This is computed as the average voxel count per class, normalized by the sum of all class averages.

Voids	IMs	Al_2O_3 Fibers	SiC Particles	Si	Al Matrix
0.13 %	3.90 %	4.32 %	4.41 %	10.31 %	76.92 %

Table 5.1: Mean occurrence of the 6 phases among the 8 volumes of the synthetic dataset (in increasing order).

The synthetic dataset comprises 8 Al–Si MMCs volumes, each with dimensions of $512 \times 512 \times 512$ voxels and generated using various parameters such as fiber sizes, lengths, orientations, grayscales, and volume fractions. Out of these, 7 volumes are designated for training and validation, while one is reserved for testing (see Sec. 5.8).

Furthermore, since the synthetic grayscale distributions are not perfect models of the experimental ones, we augment the synthetic training dataset to enhance model generalization when used to segment experimental data [28]. Four augmentations techniques are applied: contrast and brightness variations ($\pm 10\%$ each); Gaussian noise (random standard deviation of 0 – 8); and 3D Gaussian spatial blur (random standard deviation of 0 – 1). The brightness/contrast augmentations improve recognition of material interfaces and consequently of their 3D geometry. The noise and blur augmentations create a more challenging training environment that promotes robust feature learning. These augmentations are applied randomly to the sub-volumes used for training (whose creation from full volumes is detailed in Sec. 5.8), rather than to the 7 full synthetic volumes. This approach maximizes augmentation diversity by generating unique training samples from each base volume, thereby significantly improving generalization capability.

Experimental Data Preprocessing. The experimental dataset used to evaluate the trained model includes 4 XCT volumes, also sized at $512 \times 512 \times 512$ voxels, each containing a single manually-labeled slice (see Figure 5.4). These slices serve as ground truth for

assessing the model’s performance on the experimental volumes.

To enhance the final segmentation quality, we reduce noise and blur in the experimental data, a preprocessing step we refer to as *conditioning* because it adapts the real data to match the model’s synthetic training domain. For this, we apply two specific techniques: a Non-local Means filter [210] and BAM SynthCOND, a deep conditioning framework from [211]. Since BAM SynthCOND trains conditioning CNNs on synthetic data, and our segmentation model is likewise trained on analogous synthetic data, we expect a synergistic improvement in performance when segmenting experimental data conditioned with this approach [190]. The effectiveness of the different conditioning methods is qualitatively assessed in Fig. 5.5, which displays an experimental slice before and after conditioning,

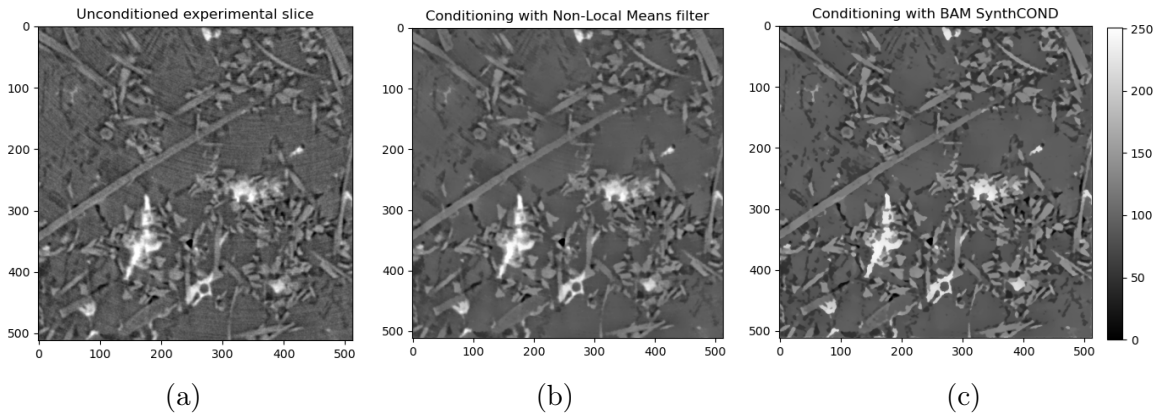


Figure 5.5: Conditioned experimental slices of the AlSi12CuMgNi MMC. From left to right: (a) the unconditioned experimental slice, (b) the Non-local means filter conditioning [210], (c) the conditioning with BAM SynthCOND [211].

5.5 Input Representation: Graph and Node Features

This section describes how we transform the XCT volume into a graph-structured input for the GNN, specifying both the connectivity criterion between nodes and the feature engineering process for node representations.

Graph Construction from Volumetric Data. The XCT volume is converted into a graph where each voxel becomes a node (see Fig. 5.6), with its grayscale intensity (0-255) as one possible immediate feature and its material phase (labels 0-5) as its class label. To define the graph’s topology, we connect each node to its 6 nearest neighbors in 3D space. This spatial proximity-based connectivity provides a deliberate inductive bias, encoding the assumption that local geometric context, i.e., the immediate voxel neighborhood, is meaningful for classifying a voxel’s phase within a microstructure. This construction of the graph preserves the spatial relationships of the original volume while liberating the data from the constraints of a rigid grid. For computational tractability, as detailed in Sec.

[5.8] this graph is constructed on sub-volumes of $64 \times 64 \times 64$ voxels, which are extracted from the original $512 \times 512 \times 512$ volumes for both training and testing.

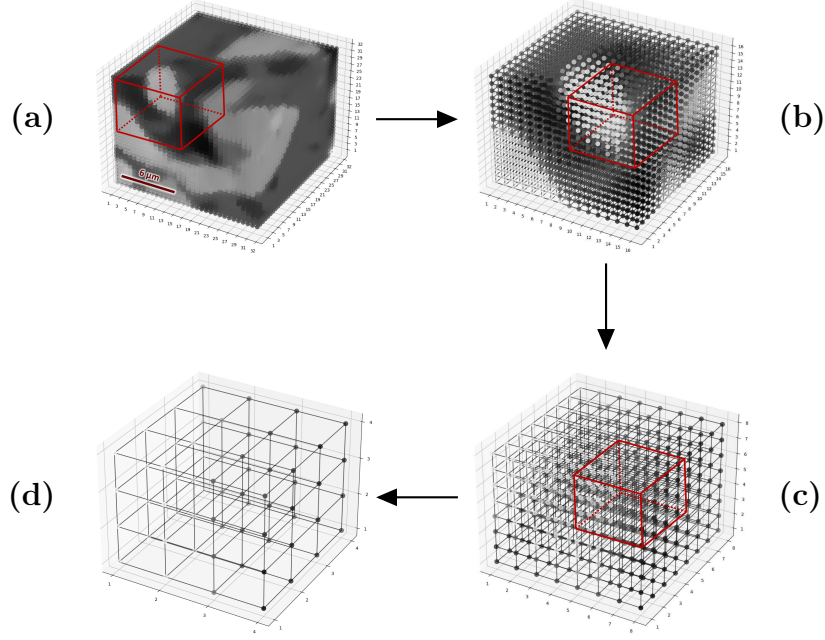


Figure 5.6: Successive zoom on the underlying graph structure. From (a) to (d) the side of the sub-volume goes from 32 to 16, from 16 to 8, and from 8 to 4.

Node Feature Engineering. The choice of node features is crucial, as it shapes the GNN’s reasoning process by defining its initial input information. While scalar grayscale intensity is a natural starting point, serving as a direct measure of X-ray attenuation, microstructural properties extend beyond single-voxel values. In principle, a more powerful feature set could incorporate hand-crafted local texture descriptors, quantifying contrast, homogeneity, or entropy within a neighborhood. However, such feature engineering requires significant preprocessing and introduces its own assumptions.

As a deliberate simplification to establish a baseline, our initial proof of concept work [36] uses only the scalar grayscale value of the voxel as the node input feature. This tests whether a GNN can learn useful node textural features directly through node interactions, thereby placing the burden of feature extraction solely on the message-passing mechanism. Building upon this foundation, we then introduce a significant enhancement in [32]: we enrich the initial node features by processing each node’s local neighborhood with a CNN, a method inspired by [66] and detailed in Sec. 5.7. As our results in Sec. 5.9 will demonstrate, this approach leads to a marked improvement in performance.

This methodological shift is strongly supported by the literature. It has been proven that GNNs struggle with low-dimensional features on node classification tasks [184], a limitation we will explore in greater depth in Chapter 6. Our hybrid architecture directly

addresses this limitation by leveraging the CNN’s powerful inductive bias for spatial feature extraction to generate higher-dimensional feature vectors. These enhanced feature vectors are then fed into the GNN, which can focus on its core strength: propagating and refining this information through the graph structure.

5.6 GNN Architectural Challenges

Semantic segmentation of volumetric XCT reconstructions is a fundamental task in materials science, aiming to identify and delineate microstructural constituents. This process classifies each voxel with a material label, transforming a 3D grayscale intensity field into an interpretable map of material phases. We frame this as a supervised node classification problem using GNNs, where voxels are treated as nodes and edges connect neighboring voxels to encode the spatial structure of the microstructure (See Sec. 3.4 for a general introduction to node classification with GNNs).

Common GNN architectures follow an *encoder-decoder* structure, where the encoder comprises Graph Convolutional Layers (GCLs) and the decoder is a Multi-Layer Perceptron (MLP). Unlike CNNs, where each position in the receptive field is multiplied by a unique kernel weight (see Sec. 5.2), in GCLs the same weight matrix is applied to each node in the neighborhood of a given vertex. This generally makes CNNs more expressive for visual tasks. To enhance GNN expressiveness, several methods have been proposed, such as the introduction of an attention mechanism in Graph Attention (GAT) layers [212] and MLP-based transformations in Graph Isomorphism Network (GIN) layers [100]. A detailed discussion of these layers is provided in Sec. 3.3.

The core GNN mechanism is formalized by the Message Passing Neural Network (MPNN) framework [10], where a node’s embedding is iteratively updated through consecutive Graph Convolutional Layers (GCLs) by aggregating information from its neighbors (see Sec. 3.3 for details). After L layers, a node’s representation captures the structural and feature information from its entire L -hop neighborhood.

This process introduces a powerful inductive bias for node classification, known as *homophily*, which posits that connected nodes are likely to be similar, e.g., belong to the same class. By sharing information, neighboring nodes develop similar embeddings, which in turn lead to similar predicted class labels. This directly leverages the homophily bias to simplify the classification task, a beneficial effect often referred to as *smoothing*. We posit that this is particularly well-suited for our microstructure segmentation problem, as we construct the graph from the XCT volume by linking each node (voxel) only to its 6 nearest neighbors: a voxel and its immediate neighbors are highly likely to belong to the same material phase and thus share the same class label.

However, this strength becomes a problem in deep GNN architectures. As the number of layers increases, each node’s receptive field expands to include distant and potentially

dissimilar nodes. The relentless aggregation of messages can then over-homogenize the feature space, causing the embeddings of distinct nodes to become indistinguishable, a phenomenon known as *oversmoothing* [213]. Consequently, the pursuit of a more expressive model with a larger receptive field can paradoxically reduce its discriminative power. We provide a detailed discussion of the homophily bias and the oversmoothing problem in Sec. 3.5. We address this critical challenge when designing our GNN architectures in Sec. 5.7.

5.7 From Proof of Concept to ViG Architecture

This section details our graph-based segmentation architecture, beginning with a simple proof of concept GNN to establish a baseline. We then introduce our enhanced Vision Graph Neural Network (ViG) architecture, which addresses the limitations of the initial model through enriched node features and mechanisms to combat oversmoothing.

Proof of Concept Architecture. Our approach in [36] segments the Al-Si MMC using a simple GNN trained on non-augmented synthetic data, serving as a proof of concept baseline. This baseline model employs single-voxel grayscale values as node features, testing whether message-passing alone can learn complex textural properties directly from the information exchange between neighboring nodes.

While this model successfully captures basic microstructure semantics, its performance deteriorates significantly when trained on the augmented synthetic dataset (see Sec. 5.4). This performance drop reveals a dual limitation: GNNs’ inherent difficulty with low-dimensional scalar features [184], compounded by a shallow architecture lacking capacity for robust representation learning. Consequently, the model cannot learn sufficiently complex embeddings to handle the enlarged range of gray levels, being constrained by both poor input features and insufficient network depth.

Architecturally, this simple GNN reflects these constraints. Its encoder comprises just three distinct Graph Convolutional Layers (GCLs): GAT [212], GraphSAGE [12], and GCN [11], each followed by a non-linearity. A GAT layer is strategically placed as first layer to compute attention coefficients directly from the raw scalar inputs, while the decoder consists of a simple two-layer MLP. This compact design, while sufficient for a proof of concept and characterized by a low parameter count (see Sec. 5.11), underscores the need for a more powerful approach.

Vision Graph Neural Network Architecture. We then develop a deeper architecture in [32], diagrammed in Fig. 5.7. This model draws inspiration from Vision Graph Neural Network (ViG) blocks [66], a design that has proven successful in segmentation domains such as histopathology for cancer diagnosis and prognosis [214].

The core ViG block comprises two modules: a Grapher that performs structural reasoning by propagating and aggregating information between connected nodes via Graph Convolutional Layers (GCLs), and a Feed Forward Network (FFN), a simple MLP, for further feature transformation. The FFN is crucial for enhancing model capacity and mitigating oversmoothing in deeper architectures [213], as it interrupts the message-passing flow that causes feature homogenization by projecting node embeddings into a new space. ViG architecture also employs skip connections to alleviate oversmoothing, a widely used technique (as detailed in Section 2.8). These allow to preserve information from earlier layers, preventing node embeddings from collapsing into an indistinguishable subspace as model depth increases [113]. The final architecture is constructed by stacking multiple ViG blocks.

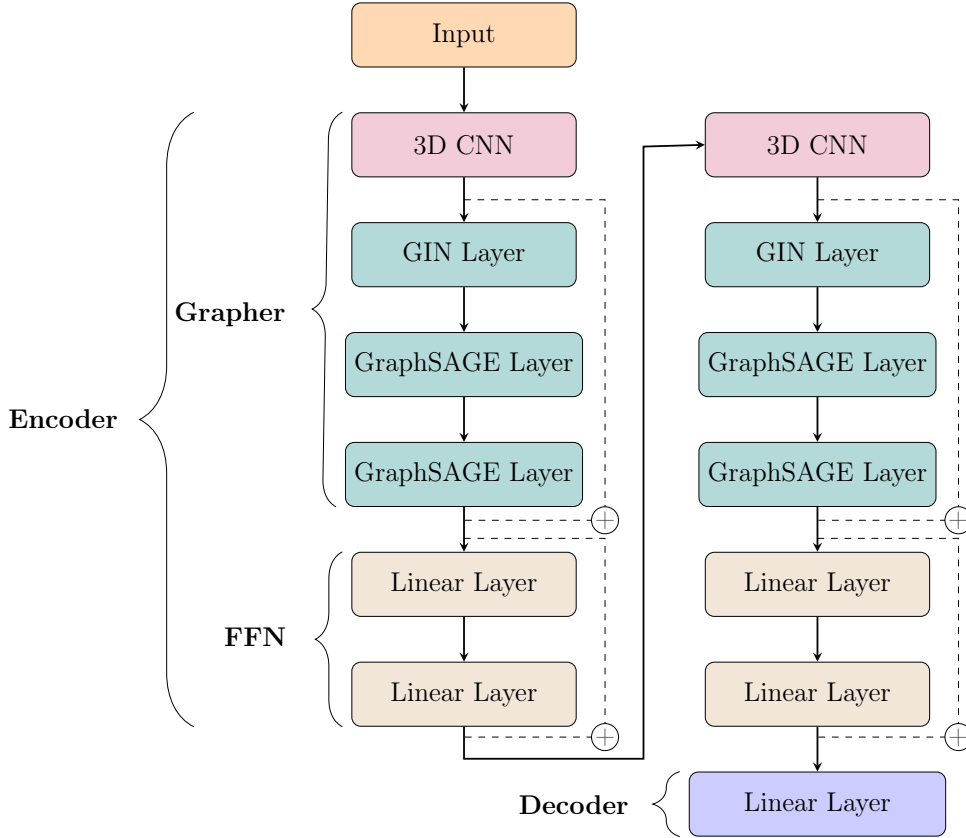


Figure 5.7: Diagram of the ViG architecture. The $64 \times 64 \times 64$ sub-volume goes through an encoder composed of 2 ViG blocks with a Grapher and a FFN module each. The Grapher comprises a first 3D CNN and three following GCLs; the FFN simply consists of two feed-forward layers. The final decoder is a feed-forward layer. Skip connections (dashed lines) sum the 3D CNN output to the Grapher output and the result of this first sum to the FFN output.

Our custom ViG architecture, diagrammed in Fig. 5.7 stacks two adapted ViG blocks to form the encoder. Each block’s Grapher module first leverages a 3D CNN to extract rich, localized features from each node’s neighborhood. This is followed by three GCLs:

one GIN layer [100] with a 2-layer MLP and two consecutive GraphSAGE layers [12], all followed by SiLU activation [215] and batch normalization. The subsequent FFN module in each block consists of two linear layers separated by SiLU activation.

Feature diversity is preserved through skip connections that form residual pathways, sequentially combining the 3D CNN output with the Grapher output, and then this sum with the FFN output. Throughout this process, the graph’s size and topology remain fixed, while the node representations are progressively enriched. The initial node features, embedded in a 50-dimensional space by the 3D CNN, are transformed through the ViG blocks before a final MLP decoder projects them onto the 6 target classes.

A pivotal advantage of our ViG architecture over our simpler GNN in [36] is its superior computational efficiency. This is achieved by leveraging GCLs, namely GIN [100] and GraphSAGE [12], whose mathematical formulation is inherently compatible with highly optimized sparse matrix multiplication. This compatibility stems from a key architectural design: the decoupling of a node’s feature transformation from the aggregation of its neighbors’ features. While the FFN modules and skip connections are explicitly designed to mitigate oversmoothing, as previously discussed, this decoupling in the GNN layers provides a complementary benefit by helping to preserve node identity and combat feature homogenization (see Sec. 3.3).

Crucially, this decoupling allows the aggregation step to be expressed as a single, highly efficient sparse matrix multiplication. This is evident in the GIN layer’s update function, detailed in Section 3.3, which can be reformulated as:

$$\mathbf{H}^{(l+1)} = \text{MLP}^{(l)} \left((1 + \epsilon) \cdot \mathbf{H}^{(l)} + \mathbf{A}\mathbf{H}^{(l)} \right) \quad (5.7.1)$$

Here, $\mathbf{A}$ is the adjacency matrix of the graph, a highly sparse matrix for our 3D grid graph, where each node (voxel) is connected only to its 6 nearest neighbors, and $\mathbf{H}^{(l)}$ is the $N \times d_l$ matrix of node features, with N the total number of nodes and d_l the embedding dimension from layer l . This explicitly maintains a separation between the ego embedding $\mathbf{H}^{(l)}$ and the neighbor embeddings $\mathbf{A}\mathbf{H}^{(l)}$, a design that naturally helps preserve node identity. Furthermore, the subsequent MLP allows for a further enriched feature transformation. GraphSAGE operates on a similar principle, using separate weight matrices for the center node and its neighbors (see Sec. 3.3).

This formulation allows the node features for the entire graph to be updated in a single, parallelized operation through sparse matrix multiplication, resulting in a dramatically lower memory footprint and faster execution. This memory-efficient sparse aggregation is natively available in PyTorch Geometric for GIN and GraphSAGE layers. In contrast, such an implementation is not feasible for the GAT [212] and GCN [11] layers used in our simpler GNN model, which lack this efficient decoupling. We provide a quantitative comparison of the execution time and RAM usage for both architectures in Section 5.11

5.8 Training and Evaluation

This section provides a comprehensive overview of our complete training and evaluation pipeline. Our methodology follows a three-stage approach: we first train our models on synthetic data, then fine-tune them on a small portion of experimental data, and finally conduct comprehensive evaluations on both synthetic and experimental test sets. In Section 5.9, we will compare the performance of our simple GNN and enlarged ViG models against the 3D U-Net implementation from [28] across all evaluation scenarios.

Training on Synthetic Data. We begin by partitioning our synthetic dataset, allocating 7 volumes for training and validation and one held-out volume for testing. To ensure a fair comparison with the prior U-Net work by [28], our GNN models are evaluated on the identical test volume used in that study. To maintain manageable memory requirements during both training and inference, we subdivide each volume into smaller, overlapping sub-volumes of $64 \times 64 \times 64$ voxels. A graph structure is then constructed for each of these sub-volumes (see Sec. 5.5). The sub-volumes from the 7 training/validation volumes are then randomly shuffled and split, with 80% allocated for training and 20% for validation. This dataset partitioning strategy is summarized in Table 5.2

Training	Validation	Test
70%	17%	13%

Table 5.2: Dataset split ratio between Training, Validation and Test.

We optimize the model using the Adam optimizer [58] with an initial learning rate of 0.001, an exponential decay rate of 0.96, and a weight decay regularization of 0.001. Training proceeds for 50 epochs with a batch size of 64 graphs. To prevent overfitting, we employ an early stopping criterion, saving the model weights that minimize the loss on the validation set with a patience of 20 epochs. The model uses a cross-entropy loss function, where the contribution of each class is weighted by the inverse of its average occurrence across the 8 synthetic volumes (see Table 5.1) to mitigate class imbalance. Furthermore, since voids constitute only 0.13% of voxels on average and are not annotated in the experimental slices, we set their contribution to the total loss to zero.

Evaluation Methodology and Metrics. To ensure statistical robustness in our evaluation, we conduct 10 training runs with a fixed random seed for each model and report the mean performance along with the standard deviation corresponding to a 95% confidence interval. This approach for assessing model stability is common practice in the literature (see, for example, [216]).

The evaluation process begins with the reconstruction of the segmented volume. For a given test volume, the predicted probabilities from all $64 \times 64 \times 64$ sub-volumes are aggregated to reconstruct 6 full-scale $512 \times 512 \times 512$ probability volumes, one for each material class. To combine predictions in the overlapping regions between sub-volumes, the final probability for each voxel is computed by summing the probabilities from all overlapping sub-volume predictions. The final, discrete segmentation is then generated by assigning each voxel the class label with the highest summed probability.

We assess segmentation accuracy using the Dice score (also known as the $F1$ score), a widely adopted metric for image segmentation tasks (see, for example, [189], [28], [214]). The Dice score provides a balanced measure of segmentation quality by rewarding correctly segmented voxels while penalizing both false positives and false negatives. The score ranges from 0 to 1 (typically reported as a percentage from 0% to 100%), where a value of 0 indicates no spatial overlap between the predictions and the ground truth labels, and a value of 1 signifies a perfect match. It is defined in terms of True Positive (TP), False Positive (FP), and False Negative (FN) voxels as follows:

$$\text{Dice score} = \frac{2 \times TP}{2 \times TP + FP + FN} \quad (5.8.1)$$

A key aspect of our evaluation protocol is the dimensionality of the Dice score calculation, which is determined by the nature of the available ground truth data. For the synthetic test volume, where a complete 3D label set exists, the Dice score is calculated directly over the entire 3D volume. This provides the most comprehensive assessment of segmentation quality, as it fully captures the spatial consistency of the 3D microstructures. The reported score for synthetic data is the average across the 10 trained models evaluated on the 3D test volume.

For the 4 available experimental volumes, we follow the same reconstruction procedure to generate a full $512 \times 512 \times 512$ 3D segmentation but restrict our evaluation to 512×512 2D slices. This strategy is employed to leverage existing annotated 2D slices from prior work in [28], thereby enabling a direct and consistent comparison with established benchmarks. The final Dice score is derived by first calculating the score for each annotated slice across all 10 models, then averaging these results per slice, and finally computing the mean of these 4 slice averages. Since voids are not annotated in the experimental slices, the evaluation focuses on the remaining 5 material classes. While this 2D evaluation provides a practical and relevant measure of performance, it is important to acknowledge this methodological distinction when interpreting and comparing results between the synthetic and experimental domains.

Fine-Tuning on Experimental Data. To adapt our synthetically-trained model to real experimental data, we perform a fine-tuning step on experimental volumes. This

step is critical because our synthetic data, while useful for initial training, exhibits a low degree of resemblance to complex experimental samples and fails to fully capture their specific noise, textures, and morphological features (see 5.4). Fine-tuning offers a valuable mechanism to bridge this domain gap, allowing the model to adjust its parameters to the realistic characteristics of the experimental data.

Since the model is trained on 3D synthetic data, this fine-tuning must also be performed in 3D to be effective. To minimize the prohibitive cost required for manual voxel labeling of 3D experimental volumes, we employ a two-step process: a $128 \times 128 \times 128$ voxel sub-volume from an experimental sample is first segmented by the pre-trained model, and this output is then manually corrected to serve as the fine-tuning ground truth. This sub-volume is divided into overlapping $64 \times 64 \times 64$ voxel sub-volumes, and the model is trained for an additional 50 epochs with a reduced batch size of 4. Given the limited amount of data, we omit a validation set and save the weights from the final epoch. The performance of the fine-tuned models is then evaluated using the same protocol described above, reconstructing the full 3D volume but calculating the Dice score exclusively on the available 2D experimental annotated slices. Further details and results of this procedure are provided in Sec. 5.10.

Because the model is initially trained on synthetic data with low realism, we fine-tune it on experimental data conditioned with BAM SynthCOND [211] (see Sec. 5.4). This conditioning method is particularly valuable here, as it constrains the experimental data to be entropically analogous to the synthetic data, providing a crucial bridge between the two domains and resulting in higher performance compared to fine-tuning on data conditioned with a Non-local means filter [210].

5.9 Segmentation Performance and Visual Quality

This section presents a comprehensive evaluation of our GNN architectures for 3D XCT microstructures segmentation. We compare the performance of our simple GNN and enlarged ViG models against the 3D U-Net implementation from [28] across both synthetic and experimental datasets. The analysis examines quantitative results through Dice scores, along with qualitative assessments of segmentation quality and spatial coherence.

Quantitative Performance Analysis. Table 5.3 summarizes the Dice scores obtained from evaluating the three architectures on synthetic test data and manually-labeled experimental slices, comparing different experimental conditioning methods.

The performance gap between synthetic and experimental data is substantial and consistent across all architectures, clearly reflecting the domain gap between our low-resemblance synthetic training data and real experimental samples. On synthetic data, all models achieve high performance, with Dice scores predominantly above 90% and often exceeding

95%. The ViG model demonstrates a marked improvement over the simple GNN, nearly matching the Single U-Net’s performance on all phases except Al_2O_3 Fibers, where the U-Net holds a distinct advantage (97.2% vs. 91.1%).

However, this strong performance does not translate directly to experimental data. When evaluated on experimental slices, Dice scores drop precipitously. For instance, the ViG model’s score for Al_2O_3 Fibers fibers goes from 91.1% on synthetic data to just 33.1% on experimental data conditioned with a Non-local means (NLM) filter, and to 40.5% when conditioned with BAM SynthCOND. This pattern of significant performance degradation is consistent across nearly all material classes and models, underscoring that the synthetic data cannot fully capture the intricate grayscale distributions and structural details present in experimental microstructures.

	Al_2O_3 Fibers (%)	IMs (%)	Eutectic Si (%)	SiC Particles (%)	Al Matrix (%)
Synthetic					
Simple GNN	91.5 ± 0.6	90.2 ± 1.2	88.4 ± 0.5	89.8 ± 0.6	95.8 ± 0.1
ViG	91.1 ± 1.0	98.9 ± 0.1	93.0 ± 0.9	96.4 ± 0.3	98.1 ± 0.2
Single U-Net	97.2 ± 0.5	98.1 ± 1.5	93.3 ± 0.2	96.4 ± 0.7	98.5 ± 0.1
NLM Experimental					
Simple GNN	31.1 ± 0.8	61.7 ± 0.8	62.6 ± 1.1	40.3 ± 2.0	75.5 ± 1.0
ViG	33.1 ± 2.7	39.5 ± 3.0	68.4 ± 1.0	61.4 ± 1.1	83.4 ± 0.6
Single U-Net	55.1 ± 0.4	53.4 ± 1.8	68.6 ± 0.2	67.2 ± 0.7	85.7 ± 0.1
SynthCond Experimental					
Simple GNN	32.7 ± 0.9	64.8 ± 2.1	62.9 ± 1.0	59.1 ± 1.3	77.5 ± 1.0
ViG	40.5 ± 2.0	61.0 ± 2.2	72.2 ± 1.1	65.0 ± 0.9	85.7 ± 0.5
Single U-Net	57.8 ± 3.0	68.5 ± 1.9	75.1 ± 1.8	70.5 ± 1.5	87.4 ± 0.6
Fine-Tuned (SynthCond)					
ViG	47.8 ± 1.0	69.9 ± 1.3	73.0 ± 1.0	67.8 ± 1.1	85.6 ± 0.5
Single U-Net	45.5 ± 2.7	68.0 ± 2.3	65.9 ± 0.8	65.5 ± 2.8	82.0 ± 0.5

Table 5.3: Dice score on the synthetic test volume and experimental slices. The values and their errors represent the mean and standard deviation over 10 training runs. The performance is compared to the Single U-Net model proposed in [28].

Our enlarged ViG architecture, trained on the augmented synthetic dataset, achieves improved overall segmentation performance on experimental data compared to our simpler GNN from [36], trained on the non-augmented synthetic dataset. The ViG model shows particular gains in segmenting Eutectic Si and SiC Particles. However, we observe a decrease in accuracy for Intermetallics (IMs), a performance drop that is most pronounced on experimental data conditioned with a NLM filter [210]. This reduction in accuracy for IMs is notably mitigated when using BAM SynthCOND [211].

Despite these improvements, the Single U-Net [28], when trained on the same augmented synthetic dataset, achieves superior performance across most material classes, with the

most pronounced advantage in segmenting Al_2O_3 Fibers. This highlights a key finding: while the ViG’s performance in segmenting synthetic data is highly competitive, its ability to directly generalize to the out-of-domain experimental data is currently less robust than the U-Net’s. This performance gap can be attributed to several factors inherent to the architectural paradigms.

We first posit that a fundamental performance difference stems from what each architecture learns from the data. The U-Net, a CNN-based architecture, employs learnable kernels that are convolved across the input to detect local patterns, specific arrangements of pixel intensities and textures that characterize each material phase. This allows it to identify these learned visual motifs in the experimental data with some resilience, even when the overall image statistics shift. Conversely, a GNN’s decision-making is more directly tied to the values of the node features themselves. Its strength lies not in spatial convolution, but in applying a matrix of weights to transform and aggregate the feature embeddings of neighboring nodes, thereby learning the relational structure between them (for a detailed comparison of these approaches, see Sec. 5.2). Therefore, when the grayscale distributions, the fundamental node features, change significantly from the synthetic to the experimental domain, the GNN’s carefully learned relational model is applied to a shifted and unfamiliar feature space, which can cause a more severe breakdown in performance. Furthermore, the discrepancy in fiber segmentation may stem from the limited connectivity in our graph construction, where each node connects only to its 6 nearest neighbors. This sparse connectivity is potentially insufficient for capturing the elongated, continuous morphology of fibrous structures, which extend beyond the immediate local neighborhood.

More fundamentally, the superior performance of the U-Net underscores its innate expressiveness for dense visual prediction tasks. As detailed in Sec. 2.7, the U-Net’s encoder-decoder structure is highly effective at capturing multi-scale contextual information. The downsampling (encoder) pathway progressively builds a robust feature hierarchy, while the upsampling (decoder) pathway enables precise spatial localization necessary for voxel-wise segmentation.

Finally, the choice of conditioning method significantly impacts experimental data segmentation. Both GNN architectures show improved Dice scores when processing data conditioned with BAM SynthCOND [211] compared to NLM filter [210]. This enhancement aligns with the theoretical foundation of BAM SynthCOND, which constrains experimental data to be entropically analogous to synthetic data [190], thereby bridging the domain gap more effectively.

Qualitative Segmentation Assessment. Figure 5.8 illustrates the segmentation results on two manually-labeled experimental slices conditioned with BAM SynthCOND [211]. Both GNN models successfully capture the semantic of the microstructures, despite

the quantitative performance limitations noted in Table 5.3. The ViG model demonstrates improved discrimination between Eutectic Si and SiC Particles, correctly identifying the Aluminum Matrix regions that the simpler GNN misclassifies as Eutectic Si. This improvement is reflected in the enhanced Dice scores for Eutectic Si, SiC Particles, and Aluminum Matrix in Table 5.3. Figure 5.8 reveals distinct segmentation patterns between the models. The simple GNN excels at recognizing thicker Al_2O_3 Fibers (highlighted in red) but suffers from a significant drawback: it often misclassifies SiC Particles and regions of the Aluminum Matrix as Fibers (areas in light blue). The ViG model solves this issue, resulting in a more balanced segmentation across all material classes and a correspondingly higher Dice score for Fibers. Despite this improvement, the accurate segmentation of fibrous geometries continues to present a challenge.

Crucially, the spatial coherence of these segmentations successfully extends into three dimensions. The 3D segmentation visualization in Figure 5.9 demonstrates that our approach successfully captures the volumetric nature of microstructures, confirming that the geometric understanding observed in 2D translates consistently in 3D, despite the model being evaluated on 2D slices for experimental data.

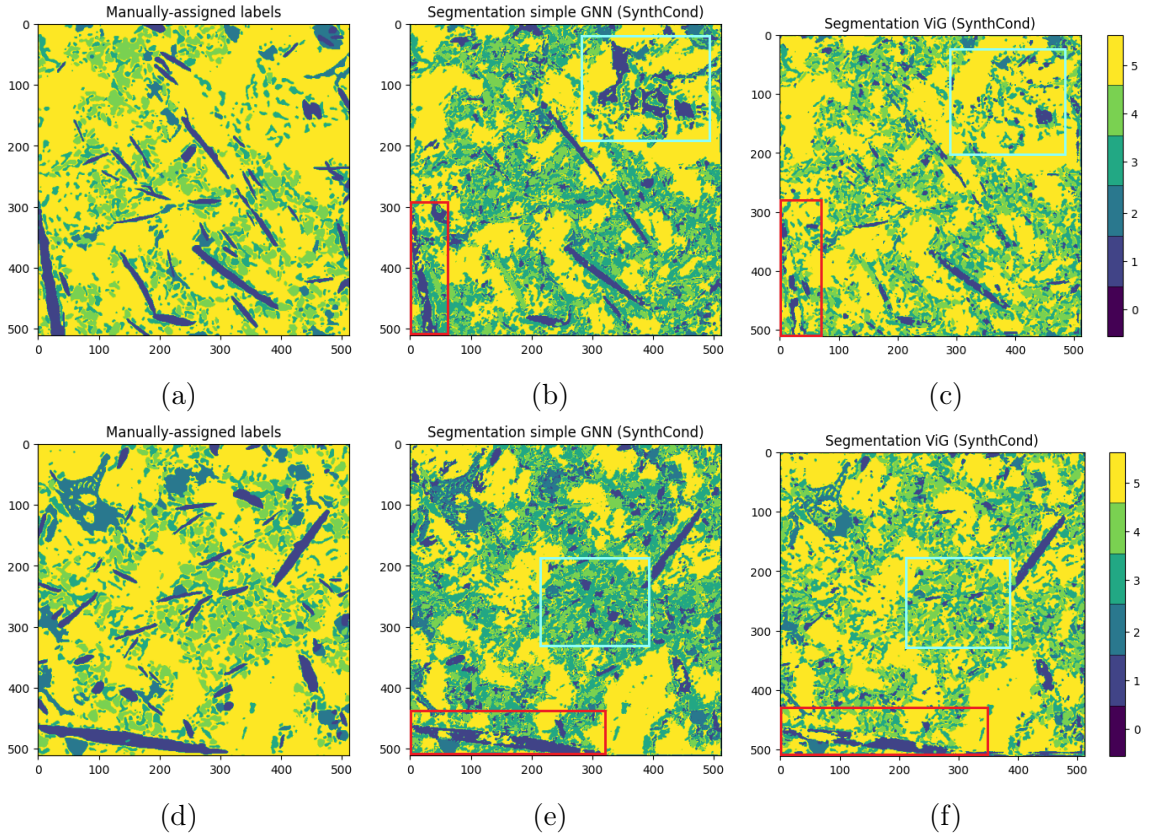


Figure 5.8: Resulting segmentation of two experimental slices conditioned with BAM SynthCOND [211]. On the left the manually-labeled ground truths ((a),(d)), in the middle the segmentation by the simple GNN model ((b),(e)), on the right the one by ViG model ((c),(f)). The color bar shows the label for each class, from 0 to 5: Voids, Al_2O_3 fibers, Intermetallics (IMs), eutectic Si, SiC particles, and Al matrix.

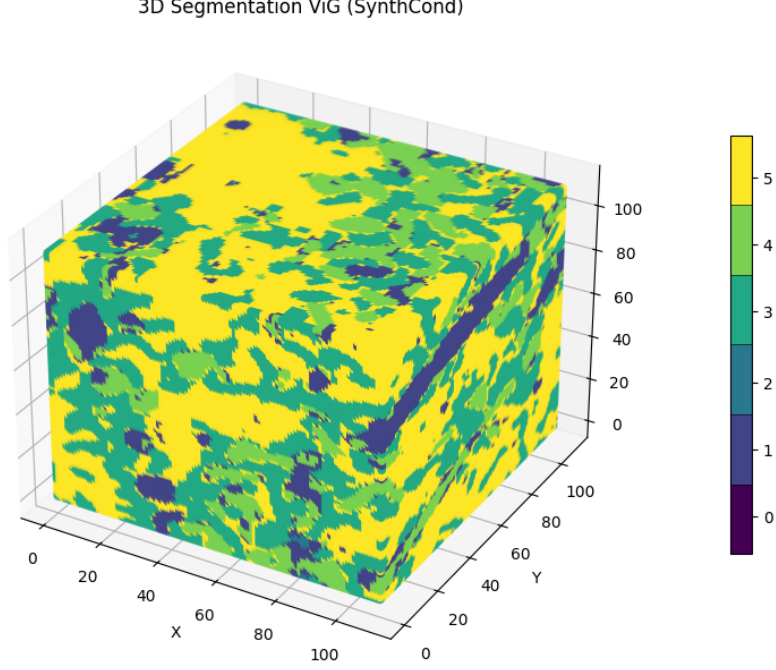


Figure 5.9: 3D visualization of the segmentation on an experimental sub-volume conditioned with BAM SynthCOND [211]. The color bar shows the label for each class, from 0 to 5: Voids, Al₂O₃ fibers, Intermetallics (IMs), eutectic Si, SiC particles, and Al matrix.

5.10 Error Analysis and Fine-Tuning Performance

This section provides a detailed analysis of segmentation errors through confusion matrices, moving beyond aggregate Dice scores. We then evaluate the efficacy of fine-tuning as a strategy to bridge the domain gap between synthetic and experimental data, analyzing both the U-Net and ViG architectures and revealing a notable divergence in their ability to adapt to limited experimental annotations.

Error Analysis via Confusion Matrices. The confusion matrices for our GNN models in Fig. 5.10 complement the aggregate Dice scores in Table 5.3 by providing a detailed breakdown of segmentation errors and revealing the specific nature of inter-class confusions. These matrices compare model predictions against ground truth labels, showing the frequency of correct classifications (diagonal entries) and misclassifications (off-diagonal entries) for each material phase. The distribution of True Positives along the diagonal aligns with the overall performance trends observed in the Dice scores of Table 5.3. For clarity, we present results for a single representative experimental slice under both conditioning methods (NLM filter [210] and BAM SynthCOND [211]) as the confusion patterns remain consistent across all analyzed slices.

A direct comparison of the matrices for the two conditioning methods confirms that BAM SynthCOND enhances segmentation performance over the NLM filter for both GNN architectures, corroborating the Dice score trends. This improvement manifests as generally stronger diagonal concentrations and reduced off-diagonal confusion. Furthermore, the matrices highlight the ViG model’s superior discrimination capability. Most notably, the ViG model substantially reduces the number of voxels belonging to the Aluminum Matrix (label 5) and misclassified with Eutectic Si (label 3), while also showing markedly fewer voxels belonging to Al_2O_3 Fibers (label 1) being confused with SiC Particles (label 4) or the Aluminum Matrix (label 5). This indicates that the enlarged ViG architecture develops more distinctive feature representations, enabling it to better differentiate between these challenging, often co-located microstructural phases.

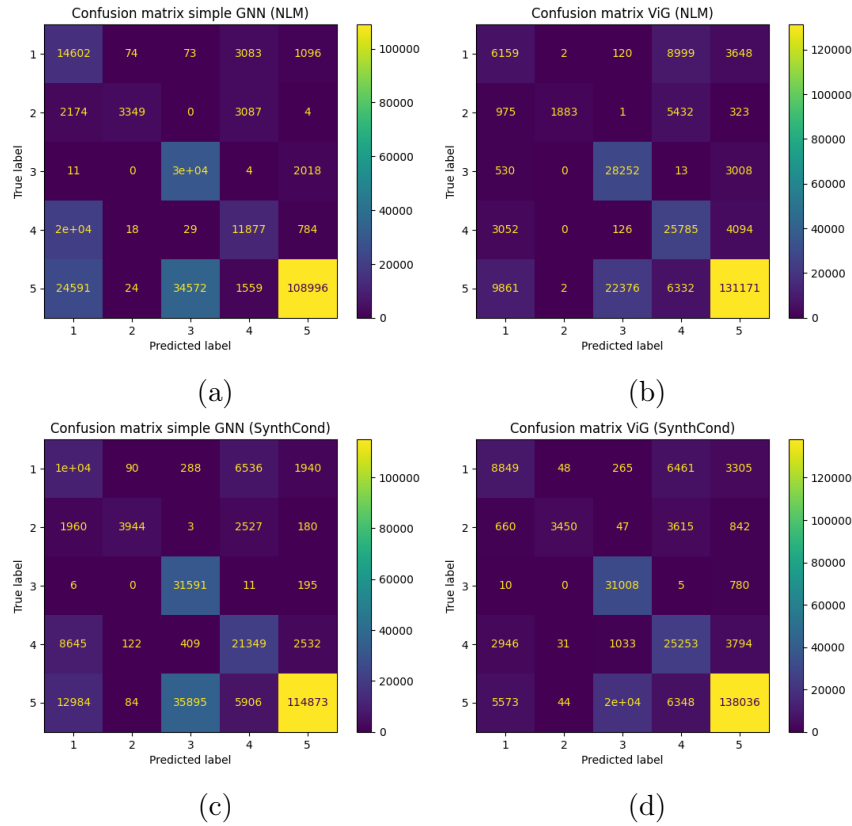


Figure 5.10: Confusion matrices resulting from the segmentation of one experimental slice conditioned with a Non-local means filter [210] (top: (a), (b)) and with BAM SynthCOND [211] (bottom: (c), (d)). The x and y axis indicate the class, from 1 to 5: Al_2O_3 fibers, Intermetallics (IMs), eutectic Si, SiC particles, and Al matrix. We discard voids since they are not annotated in the manually-labeled experimental slice.

Fine-Tuning Efficacy. To address the domain gap between synthetic and experimental data, we investigate fine-tuning strategies by further training on a small labeled experimental dataset. Given the Single U-Net’s superior baseline performance on experimental segmentation, we leverage it to create this labeled dataset. Specifically, we use the Single U-Net to

segment a $128 \times 128 \times 128$ voxel sub-volume, we manually correct the segmentation errors, and we finally fine-tune both the Single U-Net and our ViG model on this annotated data (see Sec. 5.8 for training and evaluation details).

The results in Table 5.3 reveal a significant divergence in how the architectures respond to fine-tuning. Fine-tuning improves the ViG model’s segmentation of Al_2O_3 Fibers and Intermetallics (IMs) while maintaining performance on other classes. In contrast, the Single U-Net experiences performance degradation across multiple material classes, particularly for Al_2O_3 Fibers, IMs, and SiC Particles. This inverts their pre-fine-tuning relationship: the ViG, which generalizes less effectively to experimental data when trained on synthetic data, now successfully adapts to the experimental feature space and matches or even surpasses the fine-tuned U-Net’s performance, demonstrating a remarkable capacity to adapt successfully when further trained on limited experimental data.

This divergent behavior suggests fundamentally different generalization mechanisms between GNNs and CNNs when learning from limited data. CNNs, which excel at extracting local patterns through convolution, typically require extensive labeled data for supervised 3D segmentation to build a comprehensive model of visual features. Consequently, when adapting a model pre-trained on synthetic data to a new experimental domain, fine-tuning on a very small volume may be insufficient to generalize, potentially degrading performance. In contrast, GNNs are inherently well-suited for data-efficient learning, a characteristic rooted in their semi-supervised capabilities [11]. By leveraging graph connectivity to propagate and enrich node representations through their neighbors, they can effectively generalize from sparse annotations. As discussed in Sec. 5.3, their core relational inductive bias, that a node’s properties are shaped by its neighbors, enables them to encode both the voxel-level features and, crucially, the spatial relationships between material phases directly into node representations, going beyond mere local pattern recognition. For instance, they may learn that SiC Particles are typically adjacent to Eutectic Si, or that certain phases rarely coexist.

This relational learning capability, foundational to semi-supervised learning in graphs, proves equally valuable in our fully supervised fine-tuning scenario with minimal experimental annotations. We believe that this “relational prior” enables more robust generalization during experimental fine-tuning compared to the CNN’s reliance on a “local pattern prior”, particularly when adapting from scarce labeled data. While this interpretation is consistent with our results, it requires further validation through experiments with larger labeled experimental volumes.

5.11 Parameter Efficiency and Computational Cost

An advantage of our proposed GNN architectures is their high parameter efficiency. As reported in Table 5.4, both GNN models possess significantly fewer trainable parameters

than the benchmark Single U-Net [28]. While the enlarged ViG architecture is more elaborate, with additional graph convolution and feedforward layers compared to our simpler GNN [36], and contains 6 times as many parameters, it still uses almost half the parameters of the U-Net.

Architecture	# Parameters	RAM (Train)	Time (Train)	RAM (Test)	Time (Test)
Simple GNN	61,406	4,155 MB	19 minutes	8,924 MB	12 minutes
ViG	401,006	6,570 MB	30 minutes	10,362 MB	15 minutes
Single U-Net	667,446	6,542 MB	8 minutes	10,333 MB	10 minutes

Table 5.4: Comparison of the three architectures under analysis in terms of number of trainable parameters, RAM usage and time during training and test.

This inherent parameter efficiency stems from fundamental architectural differences. In a CNN like the U-Net, the extensive downsampling and upsampling pathways necessitate numerous layers and feature channels, significantly increasing the model size. In contrast, GNNs typically employ shallower architectures to avoid the oversmoothing problem. Furthermore, our GNN models lack the multi-scale encoder-decoder structure that characterizes the U-Net, a primary source of the U-Net’s high parameter count.

Beyond parameter counts, Table 5.4 also details computational metrics including RAM usage and execution time for one training epoch and the segmentation of one experimental volume. The ViG model and Single U-Net show comparable RAM usage during both training and testing. However, the U-Net trains significantly faster, likely benefiting from highly optimized CNN libraries. Notably, while the ViG has 6.5 times the parameters of the simpler GNN, its RAM usage and training time increase by less than double. We attribute this favorable sub-linear scaling to our implementation of the ViG’s GNN layers using sparse matrix multiplication (see Sec. 5.7), which reduces memory footprint and accelerates computation. We posit that as GNN software libraries mature to the optimization level of their CNN counterparts, their inherent parameter efficiency will translate into analogous advantages in computational cost during both training and inference.

5.12 Discussion and Conclusions

This chapter introduces and validates a novel graph-based deep learning framework for segmenting complex, multi-phase 3D XCT microstructures. By reframing volumetric segmentation as a node classification problem on graphs, we show that GNNs provide a promising alternative to conventional CNN-based approaches, offering greater parameter efficiency and an enhanced capacity for data-efficient learning.

Our comprehensive evaluation reveals a nuanced performance landscape. On synthetic

data, the proposed Vision Graph Neural Network (ViG) model [32] achieves performance comparable to the benchmark U-Net [28] across most material classes, with Dice scores consistently above 90%. This demonstrates the capability of GNNs for 3D segmentation tasks. The main exception is the Al_2O_3 fibers, where the U-Net’s multi-scale receptive field offers a clear advantage in capturing elongated, continuous morphologies.

A more challenging scenario emerges when segmenting experimental data after training on low-resemblance synthetic data. While all models experience a performance drop due to domain shift, the U-Net generalizes more robustly in the absence of fine-tuning. However, this relationship reverses when limited fine-tuning on experimental data is introduced: the ViG model consistently improves, whereas the fine-tuned U-Net often degrades its performance. This finding indicates that while U-Net learned patterns are initially more transferable, the graph-based learning mechanism of the GNN exhibits greater adaptability when provided with minimal target-domain supervision, an important advantage for data-scarce experimental settings.

A central theme emerging from this work, and directly connected to the comparative study in Chapter 6, is the decisive influence of input feature quality and dimensionality on GNN performance. The progression from a simpler GNN using scalar grayscale inputs [36] to the ViG architecture with CNN-enriched embeddings demonstrates a marked performance improvement. We posit that this gain stems from two complementary aspects of ViG’s design: first, the integration of CNN-derived embeddings, which provide spatially aware and semantically rich node features; and second, the inclusion of feed-forward blocks and skip connections, which enhance feature transformation, preserve diversity during message passing, and mitigate oversmoothing in deeper networks. These results emphasize that richer and well-distinguishable features are crucial for effective graph-based learning with GNNs, directly shaping the model’s ability to represent and capture distinct material phases in the node classification task.

Looking ahead, several directions for improvement emerge. The primary challenge remains the limited resemblance between synthetic and experimental data, which leads to performance degradation during domain transfer. Closing this gap requires more realistic digital twins of materials, potentially generated using advanced generative models such as GANs [217, 218] or diffusion models [219], capable of reproducing the complex textures and morphologies of experimental microstructures. Architecturally, extending the ViG framework toward a Graph U-Net design [220] would enable a more direct comparison with CNN benchmarks by equipping GNNs with hierarchical encoder–decoder structures, helping disentangle the benefits of graph-based representation from U-Net’s inherent multi-scale design. Additionally, extending graph connectivity beyond nearest neighbors could improve modeling of long-range dependencies in fibrous or interconnected structures, further enhancing segmentation quality.

Chapter 6

Two Learning Paradigms for Graph-Structured Data

This chapter presents the core comparative contribution of this thesis through a rigorous benchmark of Graph Neural Networks (GNNs) against the well-established framework of Probabilistic Graphical Models (PGMs) [33], specifically focusing on generative models with latent variables like Stochastic Block Models (SBMs). Our analysis is deliberately designed to address the challenges prevalent in real-world networks, where node attributes are often missing or noisy and graph connectivity exhibits complex patterns like heterophily.

The analysis of network-structured data is fundamental to modern data science, yet these two methodological families have developed in relative isolation with limited direct comparison. While PGMs offer interpretable, mechanism-driven insights through probabilistic modeling, GNNs provide powerful, data-driven representation learning. A systematic analysis that treats them as competing paradigms remains absent from the literature. This chapter presents a comprehensive empirical study to bridge this gap.

We begin by framing the problem in Sec. 6.1, which introduces the fundamentals of PGMs and GNNs and explains the research philosophy behind our comparative approach. Sec. 6.2 then details our comparison setup and motivates link prediction as the shared evaluation task. Sec. 6.3 introduces the specific models selected for our analysis, while Sec. 6.4 outlines our experimental methodology using synthetic and real-world datasets.

Our investigation is organized as follows. Sec. 6.5 examines how the type and dimensionality of node attributes impact model performance. Sec. 6.6 evaluates model robustness to noise injected either into node features or graph edges, and Sec. 6.7 assesses resilience to increasing graph heterophily. Finally, Sec. 6.8 and Sec. 6.9 compare the two paradigms in terms of interpretability and computational complexity.

Our findings highlight fundamental differences in how these two approaches leverage graph structure and node attributes, moving toward a better understanding of their respective roles in graph machine learning.

6.1 Conceptual Foundations

This section establishes the conceptual foundations for our comparative study of PGMs and GNNs. We first outline their core philosophies and methodological principles, focusing on generative PGMs with latent variables (see Sec. 3.6). We then position our work within the broader research landscape, highlighting how our systematic head-to-head comparison addresses a critical gap in the literature.

Fundamentals of PGMs and GNNs. PGMs combine principles from graph theory and probability theory to model complex dependencies among random variables [29, 30]. They offer a structured way to encode prior knowledge and incorporate uncertainty, enabling efficient learning and inference in probabilistic terms. This work specifically focuses on *generative models with latent variables*, a powerful subclass of PGMs designed to capture the underlying data-generating process in a graph (see Sec. 3.6). The modeling framework typically employs linear parametric forms with a limited number of parameters, which are inferred through statistical methods (see Sec. 3.8 and Sec. 6.3 for more details).

These models are designed from interpretable, mechanistic principles guided by domain knowledge, which makes them inherently interpretable. A canonical example that we will focus on in our analysis is the Stochastic Block Model (SBM) [37, 38, 39, 40] (see Sec. 3.10). The SBM assumes that the mechanism guiding edge formation is driven by an underlying (latent) community structure that is not directly observed. Each node belongs to one or more latent communities, and the probability of an edge between two nodes depends only on their *community memberships*. By inferring these latent memberships from the observed network, the model reveals meaningful groups of nodes, provides directly interpretable parameters that naturally cluster them, and offers a probabilistic basis for predicting missing or future links (see Sec. 6.3 for more details).

In contrast, GNNs are designed to extend traditional neural networks to operate on graph-structured data [2, 221, 222]. GNNs are primarily *discriminative* models that learn to map input data to predictions by leveraging relational information. Their core operation is a message-passing mechanism [97] (see Sec. 3.3), where each node iteratively aggregates information from its local neighborhood to build increasingly sophisticated representations. Unlike generative PGMs, GNNs make no assumptions about the underlying probabilistic data-generating process. Instead, they utilize non-linear transformations and substantial parameterization to learn highly expressive node representations through gradient-based optimization on a task-specific loss function (see Sec. 6.3 for more details). These learned embeddings can be directly used for various downstream tasks including node classification, graph classification, and link prediction.

While their foundational principles are distinct, PGMs and GNNs are often applied to similar tasks. Link prediction, for example, is a natural task for generative models

(see Sec. 3.7), where it is often adopted as a model validation routine. Similarly, the node representations learned by GNNs are often used post-hoc for clustering, mimicking the explicit clustering of generative models, and efforts are made to interpret them a posteriori. Hence, in this work we compare them on common grounds aiming at exploring their strengths and weaknesses, by providing insights into the role of the input features and graph structure.

Scope and Contribution. This work presents a direct and comprehensive comparison of PGMs and GNNs as distinct frameworks for network, i.e., graph, analysis. Our primary contribution is the systematic, multi-faceted nature of this comparison itself.

While PGMs are occasionally used as baselines in machine learning papers [223, 224], these comparisons are far from systematic. Concurrently, a growing body of research explores hybrid models that integrate PGM and GNN components [225, 226, 227], such as leveraging PGMs to enhance GNN explainability [228] or employing GNNs to improve PGM scalability [229]. However, a systematic analysis that treats them as competing paradigms, evaluating their core strengths, fundamental limitations, and robustness across diverse settings, is still lacking in the literature.

This gap persists because the two model families have traditionally been developed within different scientific communities, each with distinct languages and validation criteria, making direct comparison challenging [25]. We argue that a head-to-head comparison is crucial to move beyond this fragmented understanding and provide deeper, more general insights into the inherent strengths and weaknesses of each paradigm.

Our investigation is timely, as it addresses the growing recognition of specific GNN limitations and a renewed interest in the principled insights PGMs can offer to enhance GNNs [25]. Prior work has shown that GNNs often struggle with uninformative input features and heterophilic graph structures, as they cannot fully leverage graph topology alone to achieve robust performance and strongly rely on input features [230, 231]. Furthermore, in unsupervised node clustering, GNNs have exhibited weaker performance compared to supervised approaches [232]. Notably, even in a semi-supervised setting [11], GNNs have been outperformed by the Contextual Stochastic Block Model (CSBM) [143] on node classification [224], underscoring the need for the broader investigation we present. It is important to note that while studies like [232] and [224] highlight specific performance gaps, they do not provide the extensive comparison we undertake here.

Consequently, the contribution of this work lies not in proposing another incremental architecture, but in conducting a systematic comparison. This approach reflects a research philosophy grounded in critically examining foundational knowledge through rigorous comparative analysis [24], moving beyond the prevailing machine learning paradigm of incremental model development. Notably, when GNNs truly offer an advantage over simpler or more interpretable alternatives remains underexplored. By addressing this

gap, our study seeks to provide both conceptual clarity and practical guidance, helping practitioners make more informed decisions about model selection and deployment.

6.2 Comparison Framework

Building on the conceptual foundations established in the previous section, this section introduces the framework used to systematically compare PGMs and GNNs. We first motivate our choice of link prediction as the evaluation task best suited for a fair and informative comparison between these paradigms. We then describe the experimental setup designed to probe their respective strengths and limitations.

Motivation for Link Prediction. We choose to compare the two frameworks on a link prediction task. Link prediction is a fundamental task in network analysis, where the goal is to infer missing links or predict future connections between nodes in a graph (see Sec. 3.4 and Sec. 3.7 for details about link prediction with GNNs and PGMs). Link prediction better serves our objective of comparing PGMs and GNNs than node and graph classification because it directly aligns with both models’ capabilities, thus allowing for a fairer comparison.

On one hand, differently from a node classification task, link prediction does not require explicit labels on the nodes, making it less reliant on external annotations. Indeed, PGMs are often deployed on input data that only contain network edges, with no extra information on nodes, e.g., metadata like features or labels. Moreover, node classification often assumes that network structure is correlated with node labels, to be able to learn from both types of input data. This makes it less general than link prediction, as it relies on observing meaningful node labels, which should not be taken for granted in real datasets, as proved in several works where extra information on nodes was shown not informative in learning from graphs [122, 147, 148, 149].

On the other hand, graph classification involves predicting a single label for an entire graph. While this is useful for understanding a graph as a whole, it aggregates information across nodes and edges, which can obscure model-specific performance nuances. Furthermore, graph classification is often domain-specific (e.g., molecule classification in chemistry), whereas link prediction represents a universal graph problem applicable across domains.

In addition, link prediction is often deployed as a model validation task in PGMs, in particular in clustering nodes (e.g., community detection problem) in the absence of ground truth information (see Sec. 3.6 and Sec. 3.9). Also, as link prediction often involves a large number of node pairs, it is a valuable benchmark for assessing scalability, efficiency, and the ability to handle sparsity or large graphs.

Ultimately, this choice of looking at link prediction is strategic because it highlights a

fundamental operational difference between PGMs and GNNs. GNNs naturally excel at tasks like node classification that leverage their learned node representations directly. Link prediction, however, poses a distinct challenge: it requires an additional step of mapping node embeddings to edge-level predictions. This contrasts with PGMs, which directly model edge probabilities (see Sec. 6.3). While this two-step process may disadvantage GNNs, we focus on link prediction because it represents a cornerstone task in network science. It directly assesses a model’s capacity to capture pairwise relationships, which are the essential building block of network structure. Crucially, despite its foundational role in network analysis, link prediction often occupies a peripheral role in GNN research. This discrepancy makes it particularly revealing for assessing whether GNNs fundamentally operate as sophisticated feature processors or as genuine structure learners. We therefore seek to determine to what extent GNNs truly leverage relational topology, versus merely using the graph as a routing mechanism for message passing between feature vectors.

Comparison Setup. A fundamental difference between PGMs and GNNs lies in their handling of node input features. PGMs do not necessarily require node attributes, as they can rely solely on the graph’s edge list. This allows PGMs to function effectively even when node attributes are sparse or unavailable. However, if node features are available and informative, there is no unique recipe for PGMs to incorporate them and exploit their information. Instead, one has to make ad-hoc modeling extensions to be able to accommodate them as input. A common approach, which we adopt in our PGM baseline (see Sec. 6.3), is to model the features as being generated by the latent community memberships, as discussed in Sec. 3.10. In contrast, GNNs’ message passing algorithm explicitly requires as input node features, also when they may not be available or informative. We thus evaluate how the performance of these approaches is affected by performing experiments where we tune the informativeness, the dimensionality and the availability of input features.

Then, we investigate the robustness of the models when increasing the heterophily of the graph, a structural measure that has been investigated in applications of both modeling approaches and has been shown to increase the difficulty of inference tasks [121, 233, 234, 235]. For this, we consider networks with community structures of different types, from highly cohesive communities (*homophilic* or *assortative*) to more dispersed and interconnected ones (*heterophilic* or *disassortative*) (see Sec. 3.9 for more details about community structure in networks). Since GNNs leverage the structural locality in graphs, meaning they aggregate information from the first neighbors, we expect that high mixing between communities makes it harder for the model to generalize, as it has been already discussed in [236, 237, 238, 239] (see Sec. 3.5). In addition to varying the level of heterophily, we also corrupt the graph connectivity by injecting noise into the edges, to evaluate robustness against structural perturbations.

To provide a comprehensive evaluation, we conduct experiments on both synthetic

and real-world datasets. Synthetic datasets offer controlled environments where we can manipulate network properties and systematically study the models’ behaviors under various conditions. Real-world datasets, on the other hand, typically come equipped with pre-engineered node features. By testing on both types of datasets, we aim to gain a holistic understanding of each model’s strengths and limitations across different contexts. Table 6.1 provides a high-level summary of our main findings comparing PGMs and GNNs.

Table 6.1: Summary of empirical findings comparing PGMs and GNNs across different conditions. Symbol meaning: $\uparrow$ AUC > 0.85 or noisy $\Delta\text{AUC} > 0.05$; $\uparrow$ AUC $\in [0.75, 0.85]$ or noisy $\Delta\text{AUC} \in [0, 0.05]$; $\downarrow$ AUC $\in [0.65, 0.75]$ or noisy $\Delta\text{AUC} \in [-0.05, 0]$; $\downarrow$ AUC < 0.65 or noisy $\Delta\text{AUC} < -0.05$; all values denote the mean AUC across folds and datasets (real and synthetic). Here, Z denotes the number of attribute categories in MTCOV, while F denotes the input feature dimension for GNNs.

Criterion	Condition	Community structure	PGM	GNN
Feature type	Structure-based	Assortative	$\uparrow$	$\uparrow$
		Disassortative	$\uparrow$	$\downarrow$
	Attribute-based	Assortative	-	$\uparrow$
		Disassortative	-	$\downarrow$
Feature dimensionality	High-dimensional	Any	-	$\uparrow$
	Low-dimensional	Any	$\uparrow$	$\downarrow$
Noisy input	Features	Any	$\downarrow$	$\downarrow$
	Edges	Assortative	$\downarrow$	$\downarrow$
		Disassortative	$\downarrow$	$\uparrow$
Computational complexity	Network size		linear	linear
	Feature dimensionality		small Z	large F
	Hyperparameters		fewer than 4	more than 4

6.3 Models and Experimental Protocol

This section details the specific models from both the PGM and GNN families used in our comparative study. We explicitly frame each model by detailing their input, output, parameter inference, and their specific operational mechanisms for link prediction, establishing a clear technical basis for understanding their comparative performance in our experiments.

PGMs and GNNs constitute distinct methodological approaches for modeling data structured as networks, i.e., graphs. Both frameworks take as input the network’s adjacency matrix A of dimension $N \times N$, where N denotes the number of nodes and each entry A_{ij}

represents the edge weight between nodes i and j (see Sec. 3.1 for graphs fundamentals and notation). A key differentiator emerges in their handling of node features. When available, node features are represented as F -dimensional vectors X_i , where F is the number of features. While PGMs can optionally incorporate these features through specific model extensions, they are not inherently designed to leverage them. GNNs, in contrast, fundamentally require a feature matrix X as essential input for model training.

Our experimental design mirrors a common practical scenario: a practitioner, agnostic to model families, must choose between PGM and GNN approaches to analyze a network dataset without prior knowledge of its underlying properties. Accordingly, we primarily evaluate off-the-shelf implementations of established models from both families, excluding architectures specifically engineered for particular graph structures (e.g., heterophilic or disassortative networks). However, to investigate how specialized architectures compare to general-purpose ones, we deliberately include one heterophilic-adapted GNN variant. This controlled inclusion allows us to discern when architectural specialization provides substantial benefits compared to what might be achieved through alternative strategies, such as thoughtful feature engineering with general-purpose models (see Sec. 6.7).

PGMs: A Generative Approach. This comparison focuses on a specific class of PGMs, i.e., *generative models with latent variables*, with the Stochastic Block Model (SBM) serving as our primary framework (see Sec. 3.10). These models capture the dependencies among nodes via the network’s edges in a probabilistic way, through a joint distribution $P(A, \Theta)$ that defines a complete data-generating process (see Sec. 3.6 for more details). This generative approach is characterized by the following elements:

- **Input:** The primary input is the observed graph structure, represented by the adjacency matrix A , typically divided into a *training* set A_{train} and a *test* set A_{test} for link prediction tasks (see Sec. 3.7). Optionally, node attributes (covariates) can be incorporated with ad-hoc modeling extensions.
- **Output:** Through statistical inference on A_{train} , the model produces estimates of the latent variables Θ , either as point estimates $\hat{\Theta}$ or full posterior distributions $P(\Theta \mid A_{\text{train}})$ quantifying parameter uncertainty. The latent variables Θ control explicitly the probability of observing a given edge weight A_{ij} in the hold-out set A_{test} , either as $P(A_{ij} \mid \hat{\Theta})$ for point estimates or as $\mathbb{E}_{P(\Theta \mid A_{\text{train}})}[P(A_{ij} \mid \Theta)]$ for full posteriors (see Sec. 3.7).
- **Parameters:** These are the interpretable, latent variables Θ that govern the network’s generative mechanism. For models assuming an underlying community structure in the network like the SBM, this typically includes:

- **Node Memberships (u_i):** A K -dimensional vector for each node i , where K represents the number of latent communities, defining each node’s affiliation to these communities (often with positive entries for interpretability, e.g., for node clustering), see Sec. 3.9 for more details. For directed networks, distinct *outgoing* and *incoming* membership vectors (u_i, v_i) are used, while for undirected graphs we set $u_i = v_i$.
- **Affinity Matrix (W):** A $K \times K$ matrix controlling the density of edges between and within communities, see Sec. 3.9 for more details. Its structure determines whether connections are mainly within (*assortative*) or across (*disassortative*) communities.

The SBM models the probability of observing a given edge weight A_{ij} as:

$$P(A_{ij}|\Theta) = P(A_{ij}|f(u_i, v_j, W)), \quad (6.3.1)$$

where $P(A_{ij}|\Theta)$ represents the likelihood of observing the edge between nodes i and j given the model parameters, and f is a function (often linear) combining the membership and affinity parameters into a scalar value, for instance, to model the expected value $\mathbb{E}[A_{ij}]$. We consider three SBM variants that implement this framework differently:

- **MULTITENSOR**¹ [121] and **MTCOV**² [122]: These are *parametric models* that use tensor factorization to parameterize f and employ the Expectation-Maximization (EM) algorithm [123, 124] for inference, yielding point estimates of the latent variables. The term “parametric” here specifically means that the number of communities K is a fixed model parameter that must be specified in advance and chosen via cross-validation, thereby functioning as a hyperparameter of the model. Both models incorporate a *mixed-membership* formulation, meaning that each node i can belong to multiple communities simultaneously, with its membership vector u_i containing multiple non-zero entries reflecting the degree of affiliation to each community. MTCOV builds directly upon the MULTITENSOR framework, extending it to jointly model the network structure and observed node covariates (see Sec. 3.10 for more details).
- **BNP**³ [139]: This is a Bayesian *non-parametric model* that uses Markov Chain Monte Carlo (MCMC) sampling [60, 61] for inference, estimating the full posterior distribution of the latent variables. The posterior is maximized following the agglomerative heuristic proposed in [240]. In our experiments, we employ the `minimize_blockmodel_dl()` function from the graph-tool library.

¹<https://github.com/MPI-IS/multitensor>

²<https://github.com/MPI-IS/probinet>

³<https://graph-tool.skewed.de/>

The term “non-parametric” here specifically means the number of communities K is not a fixed parameter but is instead inferred directly from the data during the inference process. This model employs hard-membership with one-hot encoded u_i .

Notably, MTCOV is the only variant that incorporates node attributes as input, thus allowing to assess the impact of using this extra information in addition to the network adjacency matrix. As detailed in Section 3.10, this covariate-enhanced model follows a specific generative assumption: node attributes are generated from the community memberships, meaning a node’s latent group helps predict its attributes. This relationship is formalized through a correlation matrix $\beta \in \mathbb{R}^{K \times Z}$ that captures associations between the K communities and Z attribute categories [122], with β representing additional learnable latent variables. MTCOV also includes a scaling parameter $\gamma \in [0, 1]$ that explicitly controls the relative contribution of network topology versus attribute information in the likelihood when inferring the latent variables. Varying γ from 0 to 1 enables interpolation between analyzing data purely through network structure ($\gamma = 0$) or purely through attributes ($\gamma = 1$). This parameter can be fixed a priori based on application goals or treated as a hyperparameter estimated via cross-validation.

GNNs: A Representation Learning Approach. GNNs are *discriminative* models that learn node representations from graph-structured data through neural network architectures. GNNs operate via message-passing mechanisms [97], where each node iteratively aggregates information from its neighbors to build increasingly sophisticated embeddings $h_i^{(\ell)}$ across layers (see Sec. 3.3 for more details). This representation learning approach is characterized by the following elements:

- **Input:** The input consists of both the graph structure, represented by the adjacency matrix A , and the node feature matrix $X \in \mathbb{R}^{N \times F}$, where each row $X_i \in \mathbb{R}^F$ is an F -dimensional feature vector associated with node i . The initial node embeddings are therefore initialized as $h_i^{(0)} = X_i$. For prediction tasks, the adjacency matrix is typically divided into *training* set A_{train} , *validation* set A_{val} and *test* set A_{test} .
- **Output:** The output comprises learned node embeddings $h_i^{(L)} \in \mathbb{R}^{d^{(L)}}$ at the final layer L , where the output dimension $d^{(L)}$ is a key hyperparameter defining the size of the final representation. These embeddings can be used for various downstream tasks including link prediction and node clustering.
- **Parameters:** These are the learnable neural network weights that govern the node transformation functions at each layer, typically involving non-linear activations. These parameters are learned through gradient descent optimization by minimizing a task-specific loss function. For link prediction, this loss is typically the *binary cross-entropy*, estimating the probability of edge existence versus non-existence.

Following the message-passing formulation [96], the core operation at each layer ℓ is:

$$h_i^{(\ell)} = \phi^{(\ell)} \left(h_i^{(\ell-1)}, \bigoplus_{j \in \mathcal{N}_i} \psi^{(\ell)}(h_i^{(\ell-1)}, h_j^{(\ell-1)}) \right), \quad (6.3.2)$$

where $\psi^{(\ell)}$ is a learnable *message function* that computes messages between connected nodes; $\bigoplus$ denotes a permutation-invariant *aggregation operator* (e.g., sum, mean, or max) over the first-order neighbors $\mathcal{N}_i = \{j \mid A_{ij} > 0\}$; and $\phi^{(\ell)}$ is a learnable *update function* that combines the node’s current state with aggregated neighborhood messages.

For link prediction, GNNs employ a two-step process that fundamentally differs from PGMs. Rather than directly parameterizing the edge probability through a true likelihood function $P(A_{ij} \mid \Theta)$, GNNs first learn node representations and then produce an *empirical estimate* of the edge existence probability by applying a scoring function to pairs of final-layer embeddings $(h_i^{(L)}, h_j^{(L)})$ (see Sec. 3.4 for details). In our setup, we use a simple and efficient scoring function based on the *sigmoid of the dot product*:

$$\hat{A}_{ij} = \sigma(h_i^{(L)\top} h_j^{(L)}), \quad (6.3.3)$$

which outputs a value in $[0, 1]$, interpreted as the predicted probability of edge existence (1 if an edge exists, 0 otherwise). Alternative scoring mechanisms exist, ranging from concatenation to complex non-linear methods [241, 242, 223]. For node clustering, the resulting final embeddings are utilized directly, typically by applying K-means [243, 244] to assign cluster labels.

A key difference from the PGMs discussed above is that standard GNNs implicitly treat the graph as undirected during message-passing: information is aggregated symmetrically using the node’s neighbors, without distinguishing between incoming and outgoing edges. This common practice is proven convenient for node classification in homophilic networks, while it may be detrimental in heterophilic settings [245]. To properly model directed graphs, the message-passing scheme must incorporate directional structure with ad-hoc choices, for example by using separate aggregations for incoming and outgoing neighborhoods [245, 246, 247], by processing source and target representations through separate graph convolutional layers [248], or by designing message functions that explicitly encode edge direction as part of the edge features [249]. We do not explore them here. Note also that the dot product used in Equation (6.3.3) is inherently symmetric. As a consequence, the reconstructed adjacency matrix satisfies $\hat{A}_{ij} = \hat{A}_{ji}$, which implicitly matches the undirected assumption of standard message-passing GNNs. When using GNN architectures with directional message passing, the decoder must also reflect edge asymmetry, for instance through parametric asymmetric scores. Exploring the impact of these choices in link prediction when modeling directed graphs is an interesting direction

for future work.

We consider four GNN architectures that implement this framework through distinct choices for the message $\psi^{(\ell)}$, aggregation $\oplus$, and update $\phi^{(\ell)}$ functions:

- **Graph Attention Network (GAT)** ⁴ [13]: Incorporates an attention mechanism, allowing each node to dynamically learn the importance of its different neighbors during message aggregation (see Sec. 3.3 for more details).
- **Graph Autoencoder (GAE) [251] and Graph Variational Autoencoder (VGAE)** ⁵ [251]: Utilize an autoencoder framework with GCN [252] layers, where GAE serves as the non-probabilistic counterpart to VGAE. The VGAE objective includes an additional KL divergence term alongside the binary cross-entropy loss.
- **H₂GCN** ⁶ [101]: Explicitly designed for heterophilic settings, it employs separate transformations for ego and neighbor embeddings, incorporates skip connections, and leverages higher-order neighborhoods to counteract oversmoothing [102, 103, 104, 105] (see Sec. 3.3 for more details).

Notably, H₂GCN addresses the fundamental challenge of heterophilic networks, where connected nodes often possess dissimilar features and labels, contrary to the homophily assumption inherent in standard GNNs (see Sec. 3.5). This specialized design allows us to evaluate when explicit architectural adaptations are essential compared to potential gains from feature engineering with general-purpose models (see Sec. 6.7).

Training and Evaluation Protocol. To ensure fair comparison, both PGM and GNN models are evaluated using identical experimental protocols. For each input network, we employ 5-fold cross-validation with an 80% – 20% train-test split of edges. GNN models additionally extract from the training set a validation set, corresponding to 10% of the entire dataset. We use the area under the receiver operating curve (AUC) as link prediction metric. This tells the probability that a randomly selected existing edge in the test set is assigned a higher probability of existing than a randomly selected non-existing edge. Its value ranges in $[0, 1]$, with 1 corresponding to perfect reconstruction and 0.5 to random chance. The training and tuning procedures differ between paradigms:

- **PGMs:** For MULTITENSOR and MTCOV, we choose the number of communities K by cross-validation. The EM algorithm [124] runs for a maximum of 50,000 iterations, and this entire fitting process is repeated 10 times with different random initializations of the parameters to mitigate local optima. For MTCOV, we additionally tune

⁴GATConv layer from https://github.com/pyg-team/pytorch_geometric [250]

⁵https://github.com/DaehaKim/vgae_pytorch/

⁶<https://github.com/GitEventhandler/H2GCN-PyTorch>

$\gamma \in [0, 1]$, which controls the relative contribution of network structure versus node attributes in the likelihood. For BNP, which infers K automatically, we use a MCMC method [60] and consider 100 samples. We perform posterior sampling via repeated calls to the graph-tool function `mcmc_equilibrate()`, where each of the 100 iterations executes 10 Metropolis-Hastings sweeps [129, 130] to propose and stochastically accept or reject a new community for every single node. Throughout this process, we collect edge probabilities using the `get_edges_prob()` function from graph-tool to infer the final community structure.

- **GNNs:** We perform 10 independent training runs with different random seeds for each hyperparameter configuration, including learning rate $\in \{0.001, 0.01, 0.1\}$, weight decay $\in \{0.0001, 0.001, 0.01\}$, dropout $\in \{0.0, 0.3, 0.5\}$, hidden layer dimension $\in \{32, 64, 128\}$, and number of layers $\in \{1, 2\}$. Models are trained for 200 epochs using Adam optimizer [253] with binary cross-entropy loss. We employ an early stopping criterion and we save the weights of the model minimizing the loss on the validation set with a patience of 50 epochs.

6.4 Synthetic and Real-World Datasets

We employ both synthetic and real-world networks to evaluate our models from complementary perspectives. The synthetic networks are generated through the SBM itself, allowing us to systematically control key structural properties, such as size, sparsity, and community mixing patterns, within a controlled environment where the ground-truth community structure is known. In contrast, the real-world datasets are selected from established GNN benchmarks, enabling a practical assessment of performance across diverse domains and under varying conditions of feature and label availability.

Synthetic Data Generation. We generate synthetic networks by sampling adjacency matrices according to the probabilistic mechanism of the MULTITENSOR model [121] (see Sec. 3.10). The key idea is that the observed network structure, represented by its adjacency matrix A , arises from a set of latent parameters that determine how communities interact (the *affinity matrix*) and how strongly each node participates in them (the *membership vectors*) (see Sec. 3.9 and Sec. 3.10). The model defines a complete probabilistic process for generating the entries of A from latent community structure. Accordingly, we proceed in three stages: (1) we specify the latent community parameters; (2) we sample the entries of the adjacency matrix according to the generative model; (3) we generate networks with varying size, sparsity, and assortativity.

(1) Latent parameters. Each node i is associated with a K -dimensional *mixed-membership vector* u_i describing its degree of participation across communities. For directed networks,

we assign two distinct membership vectors: u_i governing outgoing links and v_i governing incoming links; for undirected networks, we simply set $u_i = v_i$. Membership vectors are drawn from a symmetric Dirichlet distribution:

$$u_i, v_i \sim \text{Dirichlet}(\alpha \mathbf{1}),$$

with $\alpha = 0.05$ encouraging sparsity so that each node primarily belongs to a few communities, and where $\mathbf{1}$ denotes a K -dimensional vector of ones.

The strength of interaction between communities is governed by a $K \times K$ *affinity matrix* W , whose entries determine the expected edge density between and within communities. We consider both *assortative* and *disassortative* regimes. In the assortative case, W is diagonal, leading to stronger connections within communities. In the disassortative case, W is fully dense, allowing significant cross-community connectivity. In both cases, entries are independently drawn from a Gamma distribution:

$$w_{k\ell} \sim \begin{cases} \text{Gamma}(a, b) & \text{for all } k, \ell \in \{1, \dots, K\} \quad (\text{disassortative}), \\ \text{Gamma}(a, b) \delta_{k\ell} & (\text{assortative}), \end{cases}$$

with $a = b = 1.0$ and $\delta_{k\ell}$ the Kronecker delta enforcing assortative diagonal structure.

(2) Edge generation. Given the latent parameters, the expected number of edges between nodes i and j is:

$$M_{ij} = \sum_{k, \ell=1}^K u_{ik} v_{j\ell} w_{k\ell}$$

Each edge A_{ij} is modeled as a Poisson random variable with mean M_{ij} ,

$$A_{ij} \sim \text{Poisson}(M_{ij}),$$

meaning that the probability of generating (or observing) a particular edge A_{ij} is:

$$P(A_{ij} \mid u_i, v_j, W) = \frac{e^{-M_{ij}} (M_{ij})^{A_{ij}}}{A_{ij}!}$$

In practice, we use this formulation to generate the adjacency matrix A : for each pair of nodes (i, j) , we first compute the expected rate M_{ij} from the latent parameters and then draw a random sample A_{ij} accordingly. We focus on the sparse regime where M_{ij} is small, and finally binarize the network by clipping A_{ij} to $\{0, 1\}$. As outlined in Sec. [3.10](#), when using the generative model for inference instead of generation, these expected edge counts M_{ij} serve as link-prediction scores: missing links are identified by ranking unobserved node pairs according to their M_{ij} values, under the assumption that true links correspond to higher expected values. Importantly, this bilinear form M_{ij} corresponds precisely to

the function $f(u_i, v_j, W)$ introduced in Eq. 6.3.1.

While we ultimately binarize A_{ij} to indicate the presence or absence of an edge, we use the Poisson distribution because it produces nonnegative integer values and thus provides a natural way to model degree heterogeneity. Indeed, the membership vectors u_i and v_j are nonnegative but not normalized, allowing their overall magnitude (i.e., their L_1 norm) to vary across nodes. As a result, nodes with larger membership magnitudes tend to have higher expected values of M_{ij} , and consequently a greater number of connections, without changing their relative affinities toward different communities. This property enables the model to capture variations in node degree without requiring an explicit degree-correction mechanism. In contrast, a Bernoulli formulation would restrict each potential edge to a binary outcome (0 or 1) and require normalization of the membership vectors to ensure probabilities remain bounded between 0 and 1. This normalization artificially constrains degree variability, making degree heterogeneity more difficult to represent.

(3) Network configurations. We generate directed networks with $N \in \{100, 1000\}$ nodes, $K = 5$ communities, and average degree $\langle k \rangle \in \{5, 20\}$. For each combination of N and $\langle k \rangle$, we create both assortative and disassortative networks and sample 10 realizations per configuration. Since these networks are generated according to the same probabilistic mechanism as MULTITENSOR, we expect MULTITENSOR and its covariate-extended variant MTCOV to perform strongly on this data, potentially outperforming GNNs that lack explicit generative assumptions.

Real Datasets. We evaluate our models on a collection of 10 real-world datasets from different domains, which vary in size, sparsity, and homophily level, as reported in Table 6.2. This collection comprises both directed and undirected networks. Notably, three datasets lack pre-engineered node features and consist solely of the graph structure. This reflects a common real-world scenario where it is not immediately clear what information should be used as input features for a GNN when they are not available. We will explicitly investigate this question and the role of input features in our experiments in Sec. 6.5.

Table 6.2: Dataset statistics for the 10 real datasets analyzed. N is the number of nodes; $\langle k \rangle$ is the average degree of a node; h is the edge homophily ratio; F is the dimension of the node feature (when available).

Data	Cora	Citeseer	PPI	Peptides	Reddit	PolBlogs	Email	Chameleon	Wisconsin	Texas
Type	Citation	Citation	Biology	Biology	Information	Information	Information	Web	Web	Web
Directed	Yes	Yes	No	No	No	Yes	Yes	Yes	Yes	Yes
N	2708	3327	1750	338	228	1490	1005	2277	251	183
$\langle k \rangle$	3.90	2.74	18.47	2.02	2.20	12.77	25.44	15.85	2.05	1.77
h	0.81	0.74	-	-	-	0.91	0.36	0.23	0.21	0.11
F	1433	3703	50	9	-	-	-	2325	1703	1703

The datasets are as follows:

- **Citation Networks:** Cora and Citeseer [254], where nodes represent papers connected by citation edges, and features are bag-of-words abstracts. Although the original graphs are directed, we employ the undirected versions [254] provided in PyTorch Geometric [250].
- **Biological Networks:** Protein-Protein Interaction (PPI) [180] and Peptides [255], where edges represent interactions between proteins and similarity relationships between peptides, respectively, and features contain various protein and peptide properties.
- **Information Networks:** Reddit [256], PolBlogs [257], and Email [258]. These describe comments between users on Reddit, hyperlinks between political blogs, and email communications within a European institution, respectively (no features).
- **Web Page Networks:** Chameleon, Wisconsin, and Texas [259], where edges represent hyperlinks between web pages and node features are bag-of-words.

Seven of these datasets contain node labels, which we use to estimate the level of homophily. We measure the homophily ratio h as the fraction of edges in a graph that connect nodes with the same class label, as done in [237]. A decreasing value of h indicates increased heterophily.

6.5 The Role of Input Features

One of the main differences between PGMs and GNNs is that the latter require node features X as input, in addition to a graph adjacency matrix A . The choice of what features to give as input impacts performance on downstream tasks [260] and it is not clear how to make this choice in general. This is particularly relevant in datasets where there is no extra information besides the adjacency matrix, and one has to come up with dummy features to feed into the GNN. For instance, in social network analysis, user demographic data may be obscured due to privacy concerns. In such cases, structural features derived solely from the input graph, such as node degree and total node count, can be used as node attributes [261, 181, 262]. Hence, our first step is to evaluate how the nature of node features influences learning.

Experimental Design. For this, we run a first experiment where we change the type of information and the dimensionality of features given as input to GNNs and evaluate their impact on the performance of GNNs compared to that of PGMs. We experiment with two kinds of features:

- (1) *Structure-based features*, which are derived from the graph’s connectivity.

- (2) *Attribute-based features*, which leverage existing node attributes (when available).

For the *structure-based* case (1), we represent each node i by its corresponding row in the adjacency matrix, $A_i = [A_{i1}, \dots, A_{iN}]$. This information is readily available as it is a quantity simply derived from the input matrix A . We stress that, when splitting the graph in training, validation and test set in our cross-validation link prediction routine, we take into account the masking of the edges, and we feed the node with the masked adjacency matrix row. Indeed, the training set feature only contains the links between nodes in the training set graph, while for validation and test we progressively include the validation and test set links. For the *attribute-based* case (2), pre-existing features are directly available for only 7 of the 10 real-world datasets. For the remaining datasets without engineered features, we evaluate performance using the *structure-based* features described above.

For PGMs, which do not inherently require input features, we include them only in the model specifically designed to handle node attributes, namely MTCOV. In this case, we use a one-dimensional, *structure-based* feature. For synthetic data, this feature corresponds to the node’s true community label, defined as the index of the maximum entry in its ground-truth membership vector u_i . For real-world data, where ground-truth is unavailable, we assign each node to a cluster obtained by applying K-means to its adjacency matrix row $A_i = [A_{i1}, \dots, A_{iN}]$.

Findings I: The Equivalence of Feature Types. We find that GNNs’ performance is generally comparable in the two cases, see Figure 6.1, with the notable exception of Peptides dataset, where AUC is substantially higher with *structure-based* features. This is also the only dataset with relatively low-dimensional features ($F = 9$). Feeding the high-dimensional adjacency matrix row improves performance, which is generally comparable to that of PGMs. We find similar results for the synthetic data, see Figure 6.3

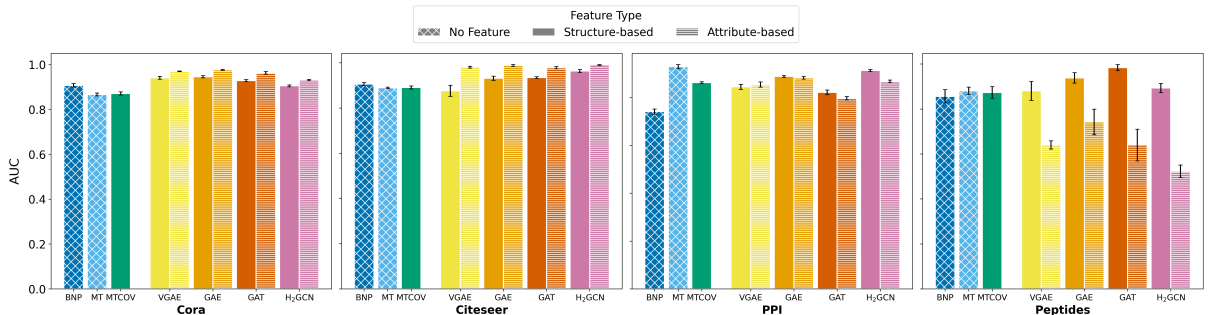


Figure 6.1: *Structure-based* vs. *attribute-based* features. AUC scores of link prediction in real datasets when giving as input different types of features. For the PGMs, only MTCOV uses input features. Bars and errors are mean and standard deviation over 5 cross-validation folds.

A closer inspection of the per-fold performance (see Section 6.11) reveals that on real

assortative datasets, GNNs outperform PGMs in roughly 80% of the folds, whereas on synthetic assortative graphs PGMs outperform GNNs in about 65% of the folds. This last result is expected since synthetic graphs are generated from one of the SBM variants.

This experiment shows that there are no clear advantages in utilizing an extra engineered input feature X for link prediction. Instead, one has to be careful in using all the information available, because adding a feature can even hurt performance. Similar results have been found in previous works using PGMs [147, 148, 122, 149]. We stress that prior studies [263, 264, 265] have proposed strategies to handle missing features in GNNs, specifically focusing on structural and positional encodings.

Findings II: The Curse of Low Dimensionality. Motivated by the above results, we assess the impact of the feature dimensionality F by reducing it to one, clustering the feature vectors with K-means and giving the cluster label as feature. This mimics situations where availability of node attributes is limited and one has access only to one type of information, e.g., age or income in a social network. We find that GNNs’ performance is significantly reduced when the input feature is clustered and this happens for every dataset studied, both real and synthetic, see Figures 6.2 to 6.4. Notably, the performance of H₂GCN often drops to levels close to random chance ($\text{AUC} \approx 0.5$) when using these low-dimensional features, highlighting a severe limitation in its ability to function with minimal attribute information.

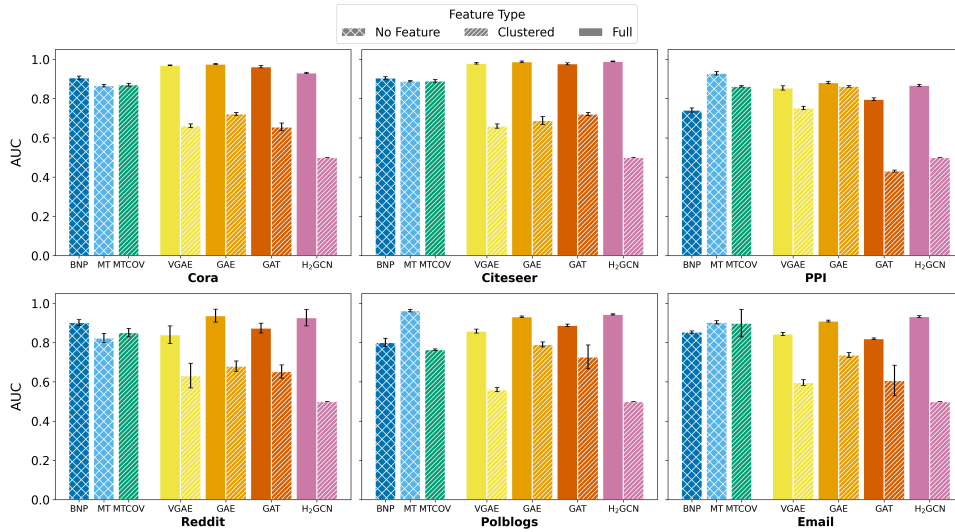


Figure 6.2: High vs. low dimensional features. AUC scores of link prediction in real datasets when giving as input either the original feature of dimension F (Full) or the one clustered with K-means (Clustered), i.e. $F = 1$. For the PGMs, MTCOV takes one-dimensional (Clustered) features by design and the other two PGMs no feature at all. Bars and errors are mean and standard deviation over 5 cross-validation folds.

Moreover, in Figure 6.3 we observe that on synthetic assortative graphs GNN performance increases with graph size, which we attribute to the adjacency-row feature becoming higher

dimensional. We stress that the difference in performance between full and clustered features tends to be smaller when the input is already low-dimensional, as in the case of Peptides ($F = 9$) (see Figure 6.4), or when the full feature performance on the dataset is already low, such as in the disassortative synthetic setting (see Figure 6.4). Instead, MTCOV, which takes as input one-dimensional features by design, shows comparable performance to that of the other PGMs and GNNs with full features. This performance is expected in the synthetic data, but we find similar results in the real datasets as well.

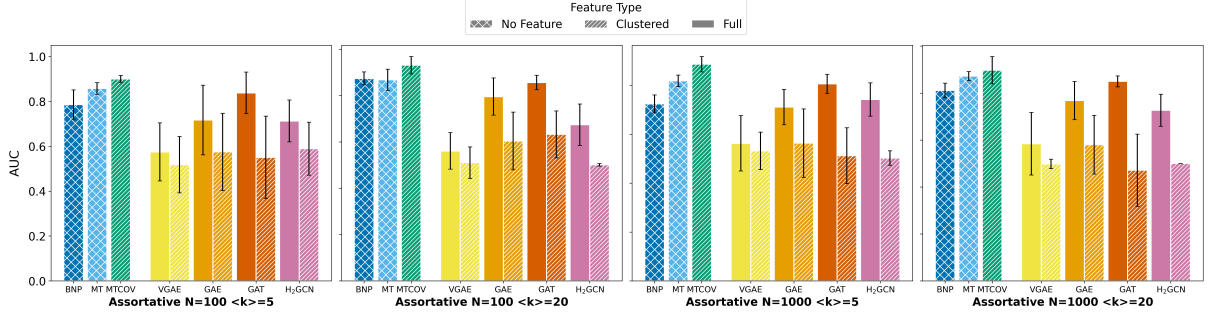


Figure 6.3: High vs. low dimensional features. AUC scores of link prediction in synthetic datasets when giving as input either the original *structure-based* feature of dimension F (Full) or the one clustered with K-means (Clustered), i.e., $F = 1$. For the PGMs, MTCOV takes one-dimensional (Clustered) features by design and the other two PGMs no feature at all. Bars and errors are mean and standard deviation over 5 cross-validation folds.

This shows the GNN inability to either properly utilize a low-dimensional feature or to discard it in favor of, for instance, a dummy high-dimensional feature like the *structure-based* one considered above. This was also observed in a node classification task [184], where it was shown how GNNs struggle in handling low-dimensional input features, and emphasized the benefits of graph-related features over engineered node features based on external information. Furthermore, [266] demonstrates that using the adjacency matrix row as an input feature substantially improves GNN performance in graph classification tasks, compared to simpler structural features such as the scalar node degree.

In practice, this issue may not occur frequently, as it is standard practice when using GNNs to employ high dimensional features, and when these are not available, one can always design them artificially, e.g., using the row of the adjacency matrix. However, the question is what to do when a low-dimensional feature is indeed available and a practitioner expects this to be informative. This situation is not uncommon when handling network datasets. While the underlying reasons for GNNs’ limited ability to leverage low-dimensional features remain unclear, assessing suitable ways to combine high and low-dimensional features to enhance prediction performance is an interesting direction for future work.

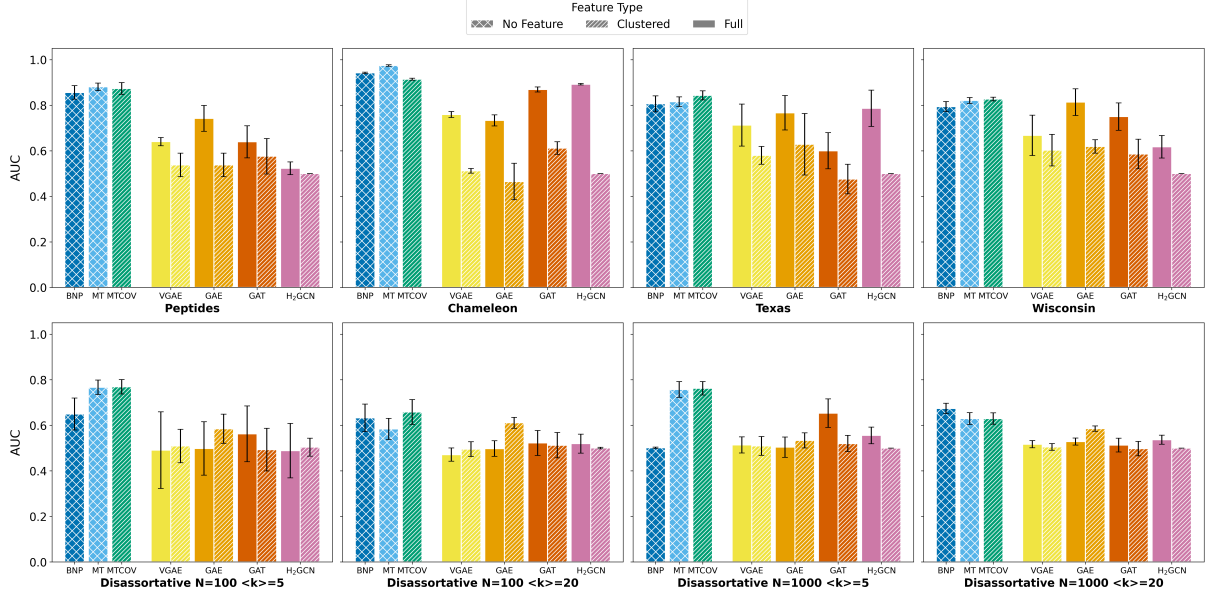


Figure 6.4: High vs. low dimensional features. AUC scores of link prediction in real and synthetic datasets when giving as input either the original feature of dimension F (Full) or the one clustered with K-means (Clustered), i.e., $F = 1$. For the PGMs, MTCOV takes one-dimensional (Clustered) features by design and the other two PGMs no feature at all. Bars and errors are mean and standard deviation over 5 cross-validation folds.

6.6 Robustness to Noise

Having investigated the impact of feature *type*, whether derived from graph connectivity or additional node attributes, and feature *dimensionality*, we now assess the robustness of the PGM and GNN models to noise injected into node features (feature *quality*) or into graph structure. In real-world scenarios, node attributes are often incomplete, corrupted, or gathered from unreliable sources [147]. Similarly, graph connectivity may include missing or spurious edges due to measurement errors [120]. While GNNs are known to be strongly dependent on input features, the extent of their vulnerability to noise compared to PGMs remains an open empirical question. Intuitively, one might expect PGMs, which do not forcibly rely on features and reconstruct the graph through statistical inference, to demonstrate superior robustness. This experiment aims to provide a comprehensive empirical evaluation to validate this expected behavior and highlight novel insights into how different models handle corruption. By providing clear empirical evidence, we offer valuable insights for a broad audience, confirming what may be intuitive only to experts.

Experimental design (noisy features). We investigate the robustness to feature noise by systematically randomizing the features of a randomly selected subset of nodes and observing the impact on each model’s performance. This simulates real-world scenarios where node attributes may be missing or corrupted. We perform this experiment on the real-world datasets equipped with *attribute-based* features.

In the case of GNNs, we randomly shuffle the entries in the input vector of features $X_i = (X_{i1}, X_{i2}, \dots, X_{iF})$, by applying a random permutation function π to its indices: $X_i^{\text{shuffled}} = (X_{i\pi(1)}, X_{i\pi(2)}, \dots, X_{i\pi(F)})$. We apply an independent random permutation π to the feature indices of each node i , so that the ordering of features differs across nodes. In the case of MTCOV, instead, where the original feature corresponds to the cluster assignment obtained by applying K-means to the node’s adjacency matrix row, we replace this feature with a random integer drawn uniformly from the range of possible cluster indices, i.e., from $\{1, \dots, C\}$, where C is the number of clusters identified by K-means.

We evaluate model robustness under increasing levels of feature randomization by independently retraining all models for each noise level: 0% (original input features are intact); 50% (features are randomized for half of the nodes); and 100% (all nodes have randomized features). Crucially, MTCOV incorporates the scaling parameter γ , which we tune separately for each randomization level.

Findings: Superior PGM Robustness to Feature Noise. While it is true that GNNs inherently rely on node features whereas PGMs do not, our comparison with MTCOV (which explicitly uses features) reveals a crucial nuance: the architectural differences in how the two models process features, not merely whether they use them, significantly impact robustness. A key advantage of MTCOV is its tunable scaling parameter γ , which is systematically optimized to balance the influence of features against topology when inferring its optimal latent parameters through the EM algorithm (see Sec. 3.10). This provides an explicit mechanism to reduce the weight of features in the likelihood as they become noisy, a capability absent in standard GNN architectures. We find that PGMs are more robust when increasing the percentage of nodes whose features are randomized, highlighting the stronger reliance of GNNs on input feature quality, see Figure 6.5

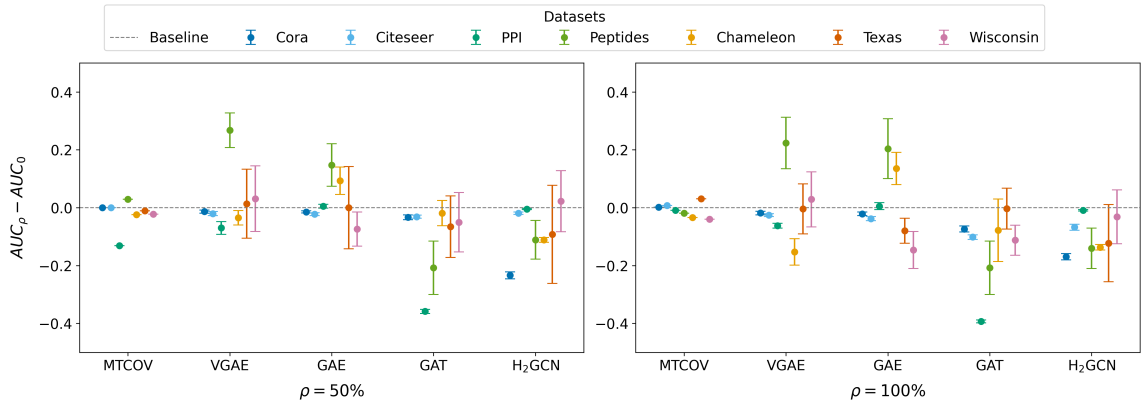


Figure 6.5: AUC score variation in real datasets when increasing the percentage of nodes ρ whose *attribute-based* features are randomized. The AUC score variation is calculated as the difference between the AUC_ρ and the original AUC_0 , corresponding to the case where 0% of nodes have randomized features. Markers and errors are mean and standard deviation over 5 cross-validation folds.

This effect, however, is modulated by the nature of the dataset. A particularly revealing case is that of Cora and Citeseer, whose node features are binary and highly sparse bag-of-words vectors. In these datasets, each feature dimension indicates the presence (1) or absence (0) of a specific word in the corresponding document. The shuffling procedure replaces zeros with zeros most of the time but randomizes the positions of the few non-zero entries, independently for each node, thereby destroying the global correspondence between feature indices and specific words. Intuitively, one might expect such disruption to cause a substantial performance drop. The fact that the decline is relatively modest is therefore interesting: it suggests that, for link prediction on these networks, GNN performance may be driven by permutation-invariant statistics of the feature vectors, such as the number of non-zero entries, which are preserved under shuffling. Conversely, the effect of noise is especially pronounced when the input features have lower dimensionality, as in Peptides ($F = 9$), or when the dataset is heterophilic, as in Chameleon and Wisconsin.

We also observe distinct noise sensitivity across GNN architectures. In particular, GAT and H₂GCN appear to be overall more sensitive to noise compared to autoencoder-based architectures (GAE, VGAE). This enhanced sensitivity in GAT and H₂GCN can be attributed to their specific design: noisy features are directly used to compute attention coefficients in GAT, and are propagated through higher-order neighborhood aggregations in H₂GCN, mechanisms that can amplify the effect of noise.

This result suggests to investigate ways to incorporate a similar hyperparameter γ into GNNs, with the goal of learning directly from data what weight to assign to structure versus node features when performing a given predictive task. One possible approach for addressing this issue is to consider feature-propagation methods to handle missing or unreliable features in GNNs [267], but this applies only under homophily and breaks down on heterophilic graphs. Hence, a main challenge is to devise methods that can be robustly applied on different underlying structures.

Experimental design (noisy edges). We then evaluate the robustness of the models to noise in the graph structure by corrupting the adjacency matrix portion used for training, while keeping the test set unchanged. This experiment is performed on both real and synthetic datasets and allows us to assess how well each method generalizes when trained on a perturbed structure but evaluated on the original unperturbed one.

To inject noise, we randomly select 20% of the positive entries ($A_{ij} = 1$) and an equal number of negative entries ($A_{ij} = 0$) in the training adjacency matrix. For each selected entry, we flip the value of A_{ij} (i.e., removing an existing edge or creating a spurious new one) with probability 0.5, and leave it unchanged otherwise. This results on an average of 10% corrupted entries.

For real datasets where *attribute-based* features are available, GNNs are fed these

instead of *structure-based* features. Conversely, for datasets that rely on *structure-based* features, we feed the GNN the original uncorrupted adjacency rows as node features. This ensures that performance changes reflect robustness to noise in the graph structure rather than in the input feature vectors.

Findings: Comparable Robustness to Edge Noise. Results are shown in Figure 6.6 for real datasets and in Figure 6.7 for synthetic ones. Resilience to randomly perturbed edges is overall comparable across models, as the difference between AUC values in the unperturbed and perturbed datasets is distributed in the same range of values. In particular, we observe that performance is generally better, i.e., models are more resilient to noisy edges, in synthetic data than in real data. In several cases, the difference in AUC calculated on perturbed and on unperturbed data is around 0 (accounting for error bars), see Figure 6.7. Also, performance on assortative data generally suffers more than on disassortative one, as its decrease is larger. This is more pronounced in GNNs, with the AUC difference being generally positive in the synthetic disassortative case, see Figure 6.7.

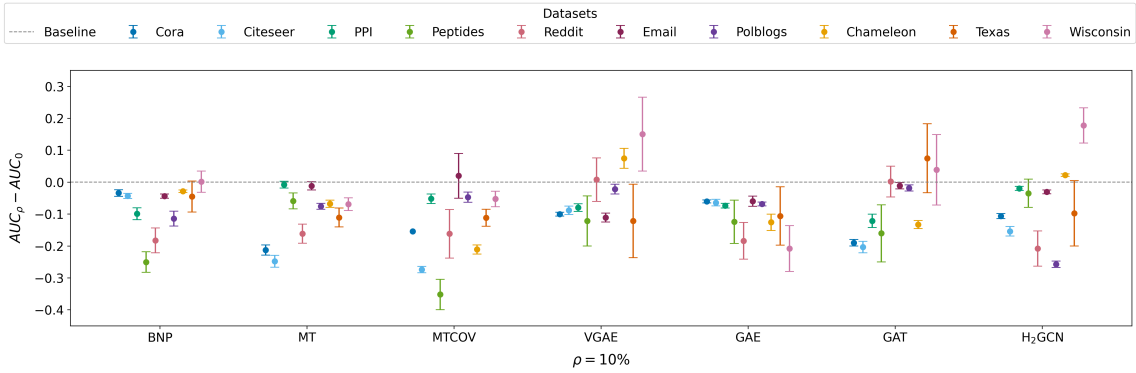


Figure 6.6: AUC score variation in real datasets when increasing the average percentage of corrupted edges ρ in the *training* graph. The AUC score variation is calculated as the difference between the AUC_ρ and the original AUC_0 , corresponding to the case where 0% of edges are corrupted. Markers and errors are mean and standard deviation over 5 cross-validation folds.

However, the per-fold scatterplots in Section 6.11 reveal how edge noise affects the relative performance between PGMs and GNNs. On real assortative datasets, where GNNs outperform PGMs in the unperturbed dataset in about 80% of the folds, their advantage decreases under noisy edges to roughly 60%. On synthetic assortative datasets, the PGM advantage increases, reflecting the same trend. For real disassortative datasets, where PGMs hold a 60% advantage (see Section 6.7), this increases to 70%, whereas on synthetic disassortative graphs PGMs become slightly less dominant (see Section 6.7). Overall, edge noise weakens the regime in which GNNs perform better, while amplifying the regimes where PGMs hold an advantage (except in the synthetic disassortative case).

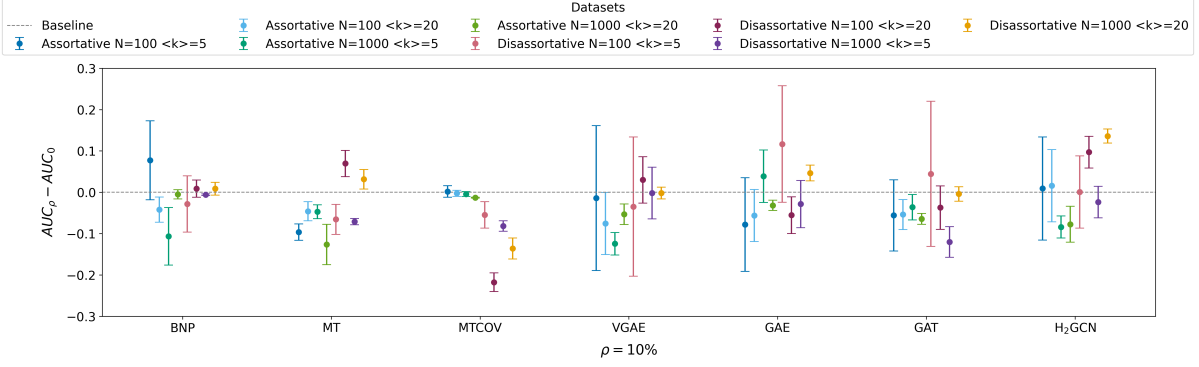


Figure 6.7: AUC score variation in synthetic datasets when increasing the average percentage of corrupted edges ρ in the *training* graph. Results correspond to one synthetic sample. The AUC score variation is calculated as the difference between the AUC_ρ and the original AUC_0 , corresponding to the case where 0% of edges are corrupted. Markers and errors are mean and standard deviation over 5 cross-validation folds.

6.7 Resilience to Increasing Heterophily

We now focus on comparing the two frameworks under different types of network clustering structures. Specifically, the third experiment explores the impact of increasing the heterophily (or disassortativeness) of the graph. This is the property of observing nodes in different clusters more likely to be connected, which contrasts with homophilic (or assortative) networks, where we expect nodes in the same cluster to have higher chances of being connected. In PGMs, learning in disassortative networks has often been shown to be a more challenging task than in assortative networks [121, 233, 234, 235]. Similar declines in performance in heterophilic networks are also now increasingly investigated in GNNs [236, 237, 238, 239].

Experimental Design. In the synthetic case, we can directly manipulate the level of heterophily by varying the density of edges between nodes in different clusters in the affinity matrix [121] (see Sec. 6.4). In real-world datasets, when extra node labels are available, and assumed to be informative, we measure the homophily edge ratio h as the fraction of edges in a graph which connect nodes that have the same class label, as done in [237]. A decreasing value of h indicates increased heterophily. Four of the real datasets have $h < 0.4$. For the synthetic networks, h is computed by assigning each node its true community label, defined as the index of the maximum entry in its ground-truth membership vector u_i .

Findings: Structural Features Enhance GNN Performance in Heterophily.

We find that PGMs generally outperform GNNs when increasing the heterophily of the graph, when GNNs are given as input the full *attribute-based* feature, see Figure 6.8. However, we also find that by giving as input to the GNN the row of the adjacency matrix

(*structure-based* feature) instead of the *attribute-based* feature, the situation changes favorably. With this input, GNNs show similar performance to PGMs in two real datasets (Email and Chameleon). This improvement is consistent across all GNN models, including H₂GCN, which is specifically designed for heterophily, though its performance remains comparable to or lower than that of PGMs. This suggests that architectural adaptations alone may not fully address the challenges posed by heterophilic graphs.

Furthermore, we stress that on synthetic networks, even if GNNs are provided with the *structure-based* feature, PGMs consistently outperform them. This is expected, as the synthetic data are generated according to the same probabilistic mechanism underlying these two models, giving them a natural advantage in recovering the latent structure. Differently from the synthetic assortative case in Figure 6.3, increasing the graph size, and thus the dimensionality of the adjacency-row feature, does not generally improve the performance here. Consistent with these results, the per-fold scatterplots in Section 6.11 show that on real disassortative datasets, PGMs hold a 60% advantage, while on synthetic disassortative dataset the advantage grows to 90%.

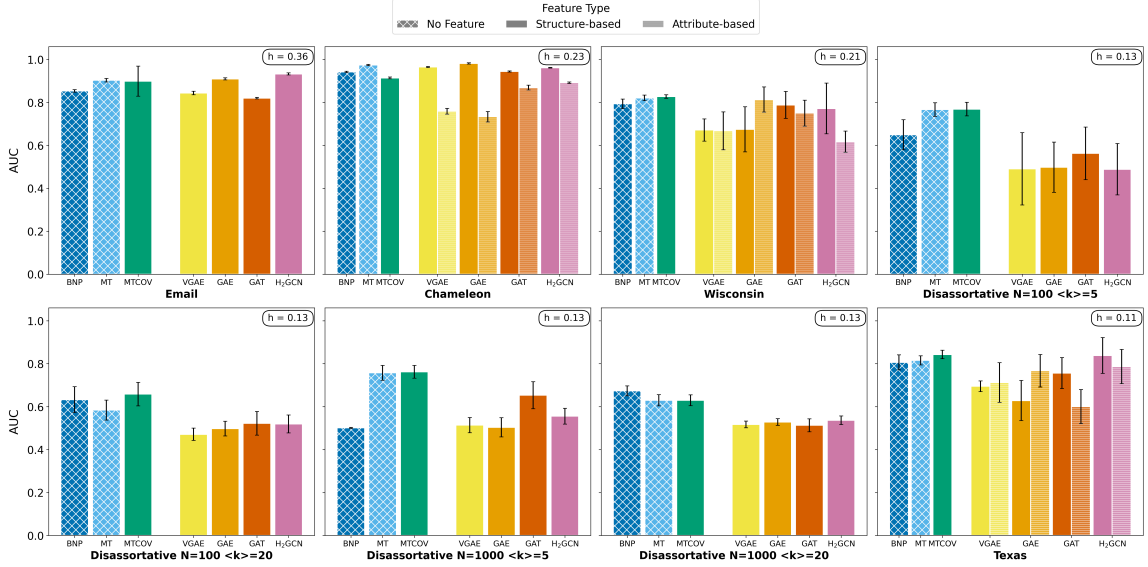


Figure 6.8: AUC scores of link prediction in real and synthetic heterophilic datasets when using the full feature. For the real dataset, we compare the AUC when feeding the GNN with *structure-based* or *attribute-based* features. For the PGMs, only MTCOV uses input features. Bars and errors are mean and standard deviation over 5 cross-validation folds.

These findings highlight that while GNNs are expected to be generally less robust in heterophilic graphs, performance could be improved by a better choice of input features, rather than acting on a change in architecture design as proposed in [22, 101, 268]. A similar conclusion was also observed in node classification tasks [269], where GNN architectures were shown to significantly improve their performance under good conditions of heterophily, specifically when nodes sharing the same label also exhibit similar embeddings and neighbor

distributions. Moreover, the phenomenon of oversmoothing [102, 103, 104, 105], which is commonly associated with poor performance on heterophilic graphs [22, 269, 270] and which H₂GCN is designed to mitigate, has been shown to be avoidable without altering the GNN architecture, simply by increasing the variance of weight initialization [271].

6.8 Interpretability

Beyond predictive performance, practical deployment also requires considering model interpretability. In this respect, PGMs and GNNs differ fundamentally in how their internal representations can be understood and explained.

The PGMs analyzed in this study, which are variants of the SBM (see Sec. 6.3), learn community membership vectors for each node. These can be either *hard* or *soft*, depending on whether nodes are allowed to belong to multiple communities. Such memberships are directly interpretable, as they provide clear insights into the network’s community structure. Moreover, the linearity of these models enables a transparent mapping between the latent parameters and the probability of observing an edge between two nodes, a design choice usually motivated by mechanistic principles.

Unlike PGMs, GNNs learn embedding vectors, quantities that are not directly interpretable. Instead, one has to make ad-hoc processing a posteriori to allow for interpretability [15] or design GNN models specifically to optimize for it [16]. A common choice is to cluster them with K-means [225, 232, 272], thus assigning them one-dimensional labels. This is a simple and direct way to analyze GNN outputs and matches our approach of prioritizing general-purpose GNN designs. Alternative and more advanced interpretability techniques exist, but they require modifying the neural network architecture itself or the loss function employed [273, 274]. Hence, we do not explore them here. Similarly, it is not directly clear how to combine node embeddings to estimate the probability of an edge [223]. One has to explicitly choose a function that maps pairs of embeddings to edge scores (see Sec. 6.3). Possible choices are the dot product or a concatenation [241, 242].

Visualizing Latent Spaces on Synthetic Data. Community structure is a direct and interpretable output of PGMs, whereas GNNs learn opaque embeddings whose relationship to communities is unclear. To enable a direct comparison with the interpretable latent parameters of PGMs, we analyze how both frameworks cluster nodes in their latent spaces. Specifically, we project the SBM membership vectors $u_i \in \mathbb{R}^K$ and the GNN final-layer embeddings $h_i^{(L)} \in \mathbb{R}^{d^{(L)}}$ into two dimensions using t-SNE [209]. This non-linear dimensionality reduction technique computes 2D spatial positions for nodes based on their associated latent parameters, of dimension K for PGMs and $d^{(L)}$ for GNNs, while preserving local neighborhood relationships, such that nodes close in the high-dimensional

space remain close in the 2D projection. The resulting visualization creates a direct parallel between the explicit community structure captured by PGMs and the implicit clustering learned by GNN embeddings, allowing for a qualitative comparison of their latent representations.

We conduct this analysis on a synthetic network generated with MULTITENSOR (see Sec. 6.4), as it provides access to the ground-truth (GT) communities, and thus a reference for evaluating how well each model’s inferred latent parameters recover the known generated community structure. In Figure 6.9, we show these t-SNE projections, where nodes are colored according to their ground-truth mixed memberships, allowing us to assess how well the spatial clustering in each model’s latent space aligns with the true community structure.

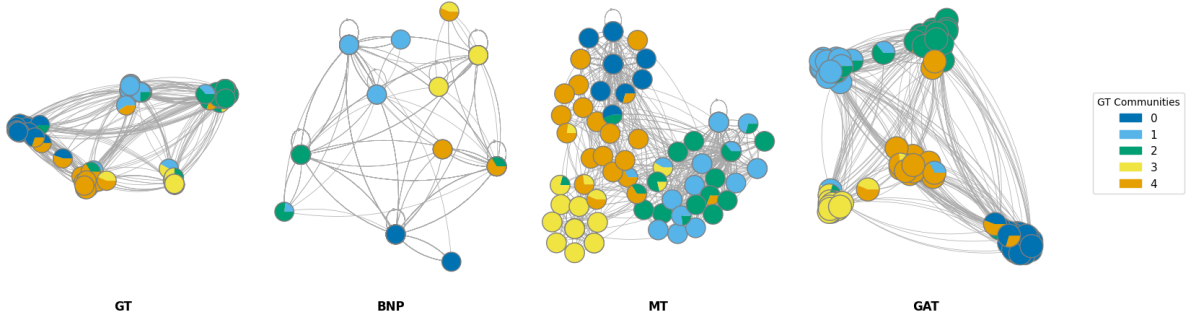


Figure 6.9: Comparing the parameters inferred. Marker positions are extracted by applying t-SNE to either the memberships u_i (PGMs) or the embeddings $h_i^{(L)}$ (GNNs), and we compare with the ones in ground truth (GT). We consider an example synthetic network, with $N = 100$ and $\langle k \rangle = 20$. Colors represent GT mixed-memberships. All the AUC values are between 0.85 and 0.88.

We first notice how nodes that have a more pronounced mixed-membership are placed on the border of their main community, closer to the other communities they also participate in, see Figure 6.9. This is somehow expected when plotting the ground-truth memberships u_i , whereas the preservation of this geometric property in the models’ inferred parameters is non-trivial. For SBMs, accurately recovering this geometric arrangement is a key indicator of successful learning. While for GNNs, it is particularly noteworthy because their high-dimensional, real-valued embeddings lack any built-in concept of communities. We find that both MULTITENSOR and GAT are able to place more mixed nodes closer to the border, as observed for the ground-truth. For instance, nodes that belong to a large extent to two communities are placed in between the two main clusters. This suggests that the GNN learns to organize its latent space in a way that implicitly reflects the underlying community structure, even without explicit architectural guidance.

However, a fundamental interpretability gap remains: while mixed-membership PGMs allow for a direct estimation of the mixed-membership, as this is their natural output, it

is not clear how to extract this information automatically and in a principled way (beside visual inspection) in GNNs from their output embeddings, as they lack the innate ability to represent these memberships explicitly. To enhance the interpretability of GNNs, one promising direction is to develop methods that enable the extraction of interpretable mixed-membership information from their embeddings. Achieving this, however, would require architectural modifications to GNNs, as done in [225].

Divergent Partitions on Real-World Data. Beyond synthetic benchmarks, we investigate whether models converge to similar node clusters (or communities) on real-world data when predictive performance is comparable. We find that despite achieving similar AUC scores (between 0.89 and 0.93) on the Citeseer dataset, indicating equivalent capability in reconstructing the adjacency matrix A , the models learn fundamentally different partitions of the network. Figure 6.10 visualizes these divergent solutions by plotting the inferred membership vectors. For PGMs, we directly visualize their native outputs: mixed memberships for MTCOV and hard assignments for BNP. For GNNs, which do not produce explicit memberships, we obtain comparable hard-membership vectors by applying K-means clustering to the final-layer embeddings, setting the number of clusters to 6 to match the number of classes in Citeseer.

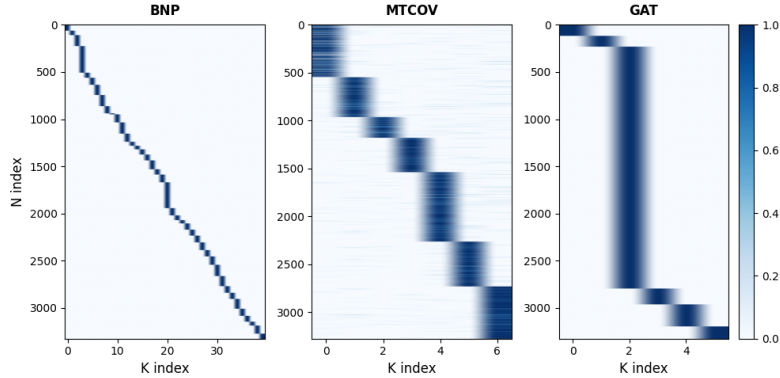


Figure 6.10: Inferred partitions on Citeseer dataset. For the PGMs we plot the membership vectors, mixed for MTCOV and hard for BNP. For GNNs, hard memberships are obtained by applying K-means clustering to the final-layer embeddings. The nodes are ordered depending on the community assignment K .

The results reveal distinct structural interpretations. BNP, which non-parametrically infers the number of communities K , converges on a trivial solution of numerous small, fragmented clusters. In contrast, MTCOV (representative of both MULTITENSOR and MTCOV, which yield similar results) identifies 7 coherent communities with minimal mixed membership when K is tuned within a range containing the number of classes (6 for Citeseer). Meanwhile, GAT produces one dominating larger cluster containing most nodes alongside several smaller groups. This demonstrates that equivalent link prediction

performance masks fundamentally different understandings of network organization.

6.9 Computational Complexity

Beyond interpretability, PGMs and GNNs also differ in their computational complexity, as summarized in Table 6.3. A key observation is that all models exhibit *linear or near-linear* scalability with respect to graph size, since their complexity grows proportionally with either the number of nodes N or edges E . This ensures that all models remain computationally tractable even for large networks. The main differences arise from the *multiplicative constants* attached to N or E , which reflect each model’s internal design.

Table 6.3: Computational complexities for the models under analysis. N the number of nodes, E the total number of edges, $E^{(2)}$ the number of edges between nodes that are connected via a 2-hop path, $d_{\max}$ the maximum degree across all nodes, K the number of communities, Z the number of attribute categories, F the input feature dimension, F' the hidden layer dimension, and L the number of GNN layers.

CLASS	MODEL	COMPUTATIONAL COMPLEXITY
PGM	MT	$O(E \cdot K^2)$
	MTCOV	$O(E \cdot K^2 + N \cdot K \cdot Z)$
	BNP	$O(N \ln(N^2))$
GNN	GAE	$O(L \cdot (E \cdot F' + N \cdot F \cdot F'))$
	VGAE	$O(L \cdot (E \cdot F' + N \cdot F \cdot F'))$
	GAT	$O(L \cdot (E \cdot F' + N \cdot F \cdot F'))$
	H ₂ GCN	$O(E \cdot d_{\max} + L \cdot ((E + E^{(2)}) \cdot F') + N \cdot F \cdot F')$

- For PGMs, the linear cost in graph size $O(E)$ or $O(N)$ is multiplied by factors related to the *latent community structure* and the *number of covariates*. For MULTITENSOR, the cost is $O(E \cdot K^2)$, where K is the number of latent communities. For MTCOV, this extends to $O(E \cdot K^2 + N \cdot K \cdot Z)$, where Z represents the number of categories in the node covariates (see Sec. 3.10). Since, in practice, K and Z have a similar order of magnitude and are typically much smaller than N or E , the overall scaling remains manageable. The BNP model explicitly avoids specifying K and achieves a near-linear scaling of $O(N \ln N^2)$.
- For GNNs, the computational cost scales with graph size as $O(L \cdot E \cdot F')$ and $O(L \cdot N \cdot F \cdot F')$, where the multiplicative factors are *architectural hyperparameters* and the *input feature dimension*. The number of layers L is usually kept small (e.g., 2-3) to prevent over-smoothing, while the hidden dimension F' is larger (e.g., 128-512), but still typically smaller than the number of nodes N or edges E . In

contrast, the input feature dimension F can be very high, even comparable to N in large graphs with rich node features, which directly increases computational cost. This analysis holds for more complex architectures like H₂GCN, which adds terms for higher-order neighborhoods ($E^{(2)}$) but still scales linearly in N and E .

Overall, while both PGMs and GNNs exhibit comparable asymptotic scaling with graph size, their effective computational cost differs in practice. PGMs are mainly affected by the quadratic dependence on K , which remains small in most applications, whereas GNNs are sensitive to the potentially large product $F \cdot F'$. As a result, in graphs with high-dimensional features, GNNs can be substantially more computationally expensive than PGMs.

An additional practical consideration is the burden of hyperparameter selection. For PGMs, model tuning is limited to a small number of interpretable hyperparameters: the number of communities K (MULTITENSOR, MTCOV); the parameter γ (MTCOV); and parameters related to Monte Carlo sampling, like burn-in period and number of samples (BNP). In contrast, GNNs require tuning a substantially larger combination of hyperparameters, including learning rate, weight decay, dropout, hidden dimension, and number of layers, all of which must be carefully tuned to ensure stable training and good convergence. Exploring this hyperparameter space introduces a significant computational overhead that may far exceed the cost of a single model evaluation. As a result, hyperparameter optimization becomes a non-negligible component of the overall computational cost for GNNs, further amplifying their computational cost relative to PGMs.

6.10 Discussion and Conclusions

In this work, we compare PGMs, specifically generative SBMs, and GNNs in how they handle network data, focusing on the link prediction task. Our results show that PGMs outperform GNNs when input node features are low-dimensional or noisy, a situation frequently encountered in real-world networks where node attributes are limited or unreliable. When high-quality, high-dimensional input features are available (e.g., rows of the adjacency matrix) and the graph is assortative, both methods can achieve comparable performance on average, but with GNNs outperforming PGMs on a fold-by-fold basis on real networks. Furthermore, PGMs display greater robustness to increasing graph heterophily, whereas the performance of GNNs deteriorates under the same conditions. Notably, we find that GNN performance can be improved on heterophilic graphs with a more careful selection of input features, echoing similar observations made for node classification in [232, 269]. Introducing noise into the edges amplifies these trends: the GNN advantage on real assortative graphs diminishes, while the regimes in which PGMs

outperform GNNs become even more pronounced.

It is important to contextualize these findings. A key factor underlying these results is the fundamental difference in how the two paradigms model edges. PGMs, through the SBM, *explicitly* parameterize edge probabilities, making link prediction a native task. In contrast, GNNs follow an *indirect* two-step procedure: they first learn node embeddings and then map these embeddings to edge probabilities. This architectural distinction may partly explain the relative advantage of PGMs in the link prediction setting. Although PGMs show stronger performance in our link prediction benchmark, GNNs may excel in downstream tasks that directly leverage their learned node-level representations, such as node classification or graph classification. Investigating how the insights derived from this work translate to other tasks is an interesting direction for future work. This would require selecting PGMs that are able to handle input features, but also to consider as baseline for comparison standard neural networks that only use input features. We expect that the ability to discard or attenuate uninformative features would play a similar role as the one we observed for link prediction.

Our goal was not to declare a universal winner, but to fill a gap in the literature by systematically comparing these paradigms on a core structural task. This comparison highlights that standard GNNs often act more as powerful *feature processors* rather than explicit *structure learners*. Such an understanding offers practitioners an evidence-based framework for selecting appropriate models based on data characteristics and task requirements.

However, an important advantage of GNNs lies in their flexibility in handling diverse input features. Unlike PGMs, which rely on ad-hoc design choices to incorporate heterogeneous covariates, GNNs seamlessly integrate arbitrary feature types, simplifying implementation and broadening applicability. Recent work [275] suggests, however, that similar black-box strategies could be employed to extend PGMs, such as SBMs, to handle arbitrary attributes more flexibly.

Finally, prior studies have shown that even simple linear models can outperform GNNs in link prediction tasks [276, 277]. This reinforces the view that GNNs struggle to truly learn from the underlying graph structure and depend on effective feature design. Given these limitations, GNNs architectures should ideally be designed to automatically select or discard features and incorporate structural signals (e.g., from the adjacency matrix) in a more principled way. In the absence of this additional learning step, practitioners should carefully consider which features to provide as input to maximize performance.

6.11 PGM vs. GNN scatterplots

To complement the results presented above, we analyze the performance of PGMs and GNNs in greater detail by comparing their per-fold AUC scores. This allows us to assess the relative performance of the models fold-by-fold, going beyond averages and standard deviations. Each point in the scatterplots corresponds to one of the 5 cross-validation folds, plotted as $(\text{AUC}_{\text{GNN}}, \text{AUC}_{\text{PGM}})$. Points above the diagonal indicate that the PGM performs better on that fold, while points below indicate that the GNN does. For each subplot, we also report the percentage of folds won by each model family.

We compare the per-fold performance for both real and synthetic datasets. For real datasets where features are available, we report either the *attribute-based* or *structure-based* GNN performance, selecting whichever achieves the highest score. All plots correspond to the case where the feature dimensionality is kept full and the feature vector entries are not randomized, i.e., no injected feature noise. We focus on both the original graphs and their corrupted versions, in which a fraction of the edges has been randomly perturbed (Section 6.6). This side-by-side comparison highlights whether corrupting the graph changes which model family dominates on each dataset.

In Figures 6.11 and 6.12, we see that edge noise generally shifts performance in favor of PGMs on real data, reducing the advantage of GNNs on assortative graphs and amplifying the advantage of PGMs on disassortative ones. In Figure 6.13, we observe the same shift for synthetic assortative graphs (with an amplified PGM advantage), whereas in Figure 6.14 GNN performance improves under noise for the synthetic disassortative ones, yet remains lower than that of PGMs.

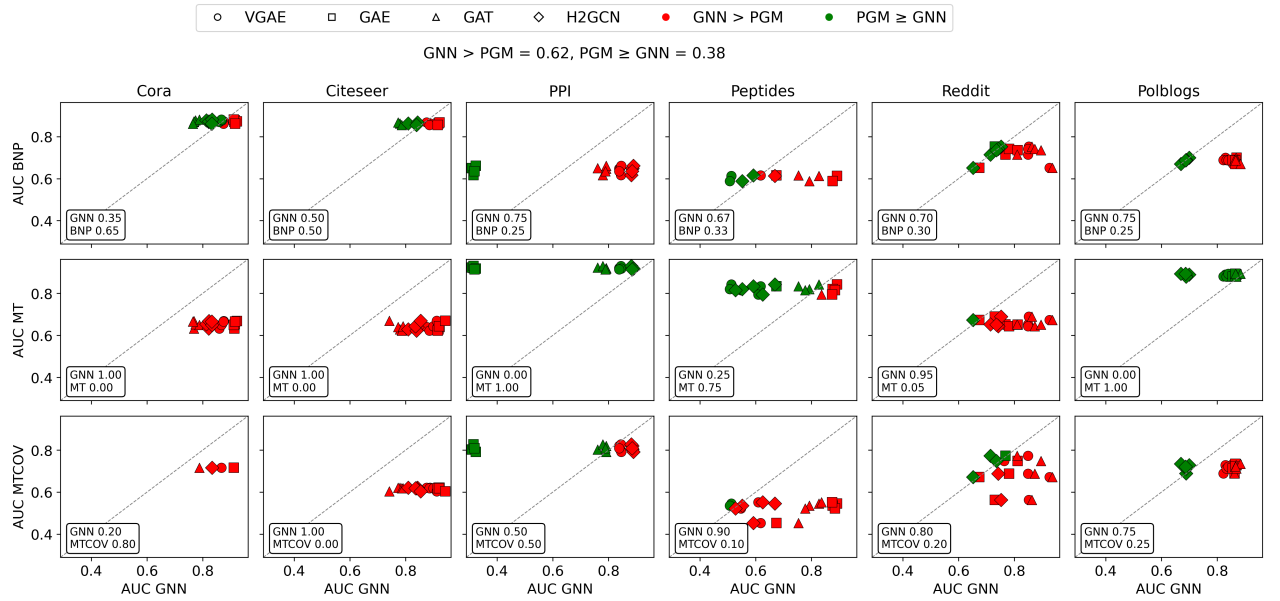
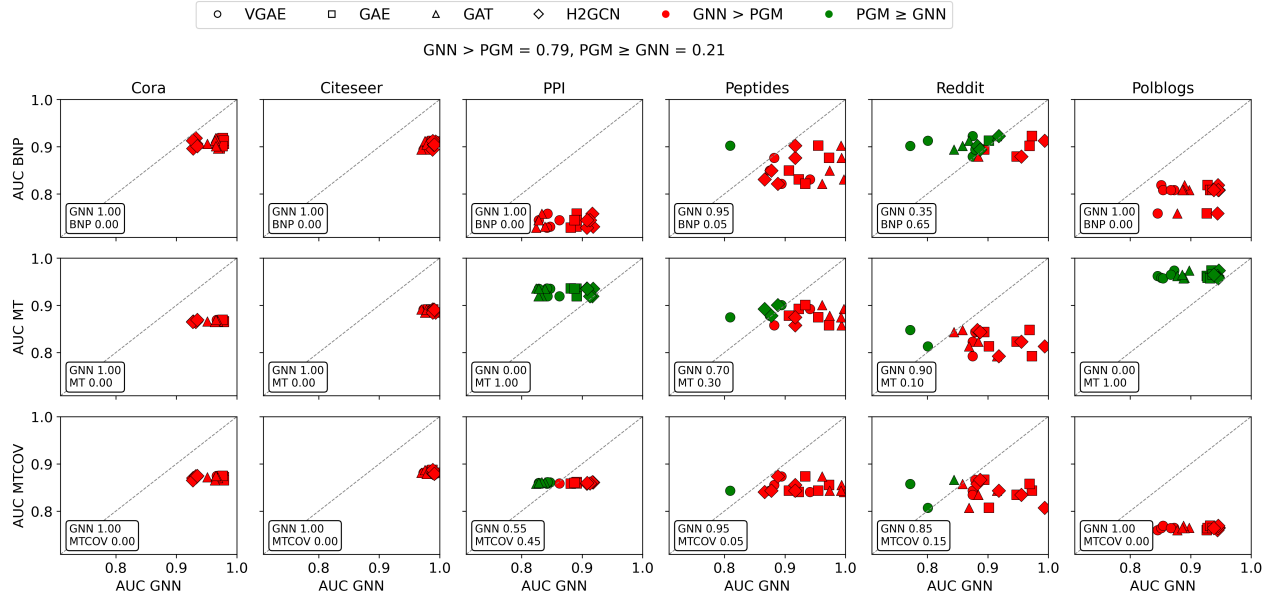
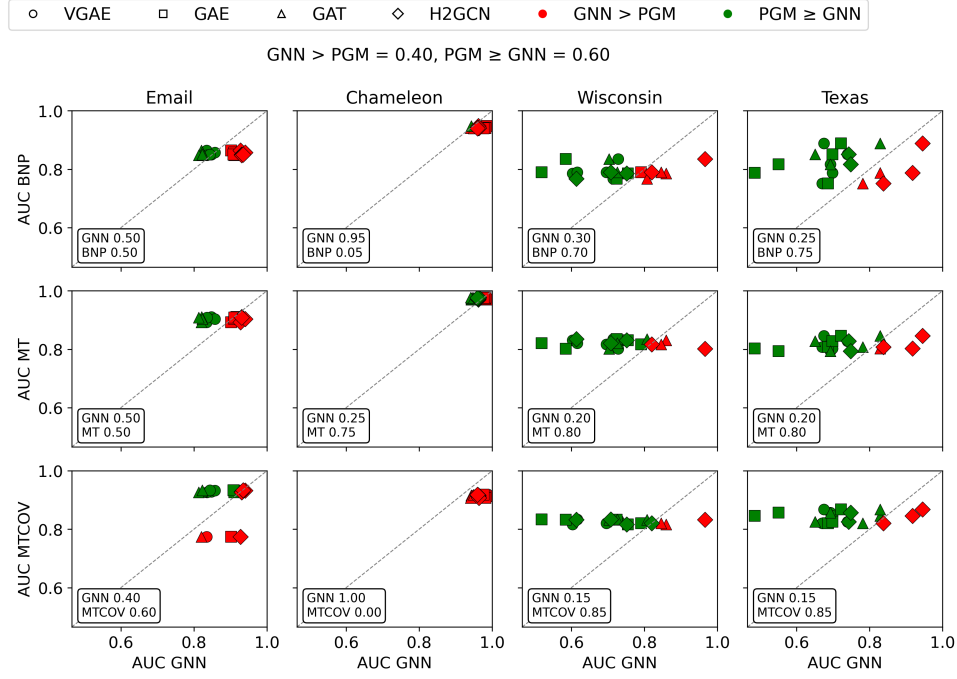
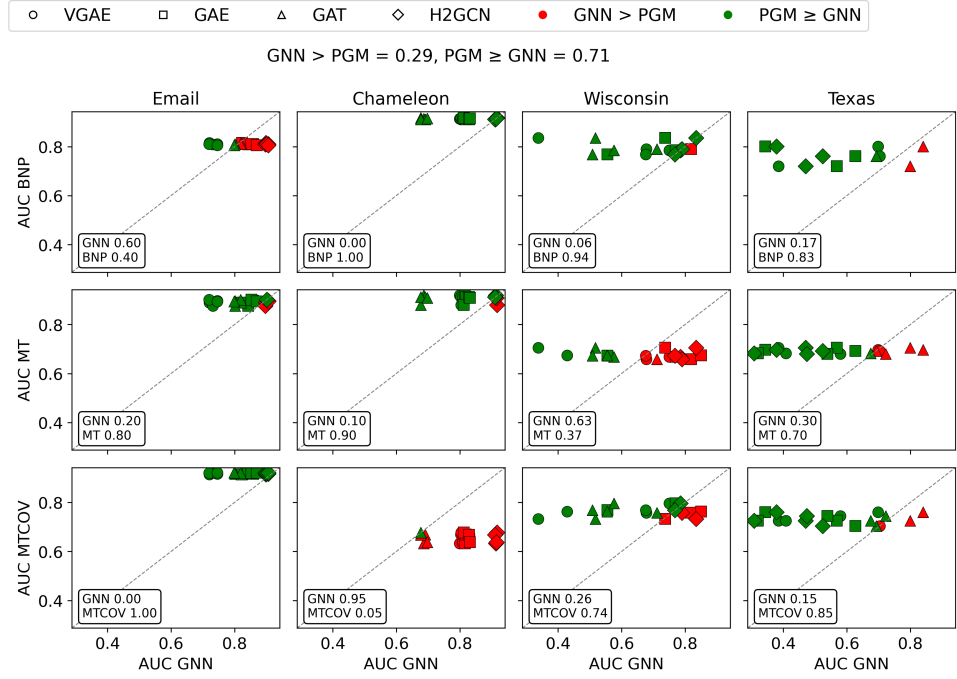


Figure 6.11: Real assortative datasets: per-fold comparison between PGMs and GNNs. For the PGMs, only MTCOV uses input features. Each point corresponds to one of the 5 cross-validation folds. Numbers inside the boxes are the fraction of folds where each model family outperforms the others.

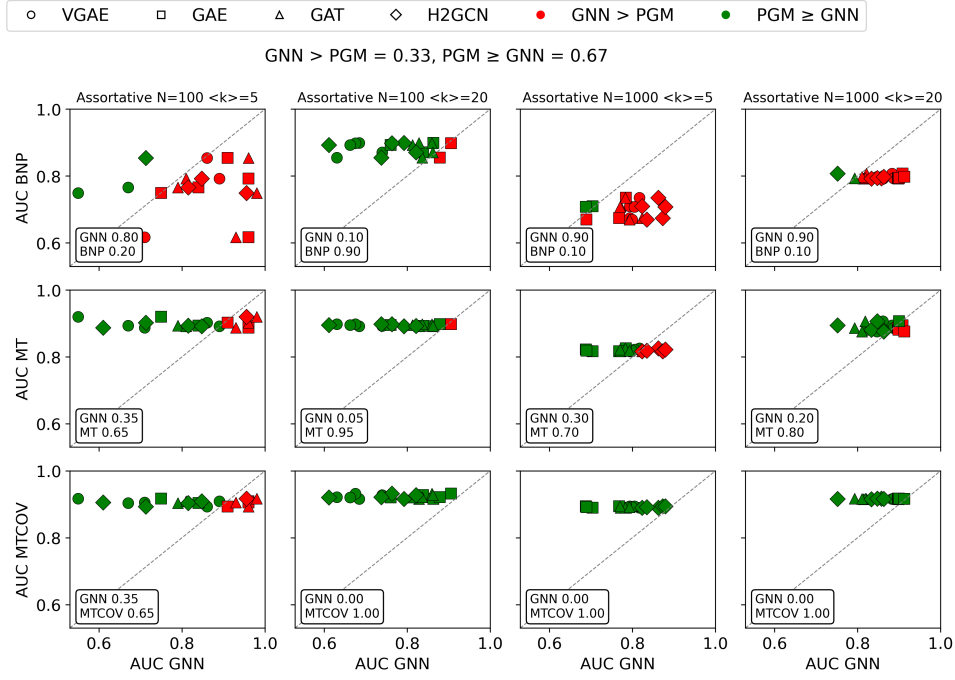


(a) Unperturbed graph (original edges).

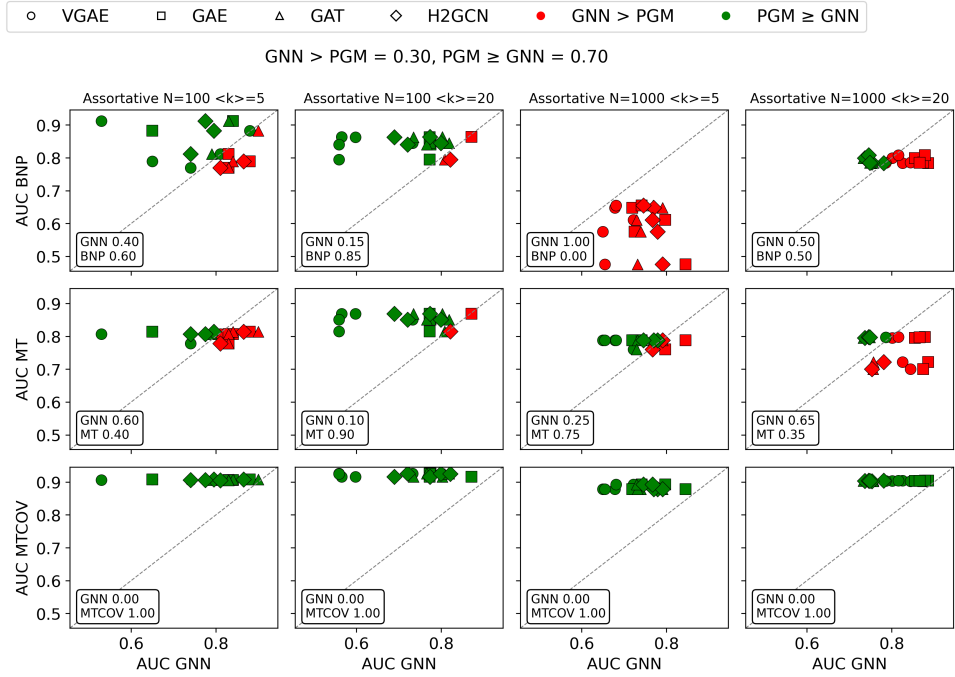


(b) Graph with 10% of edges randomly corrupted.

Figure 6.12: Real disassortative datasets: per-fold comparison between PGMs and GNNs.. For the PGMs, only MTCOV uses input features. Each point corresponds to one of the 5 cross-validation folds. Numbers inside the boxes are the fraction of folds where each model family outperforms the others.

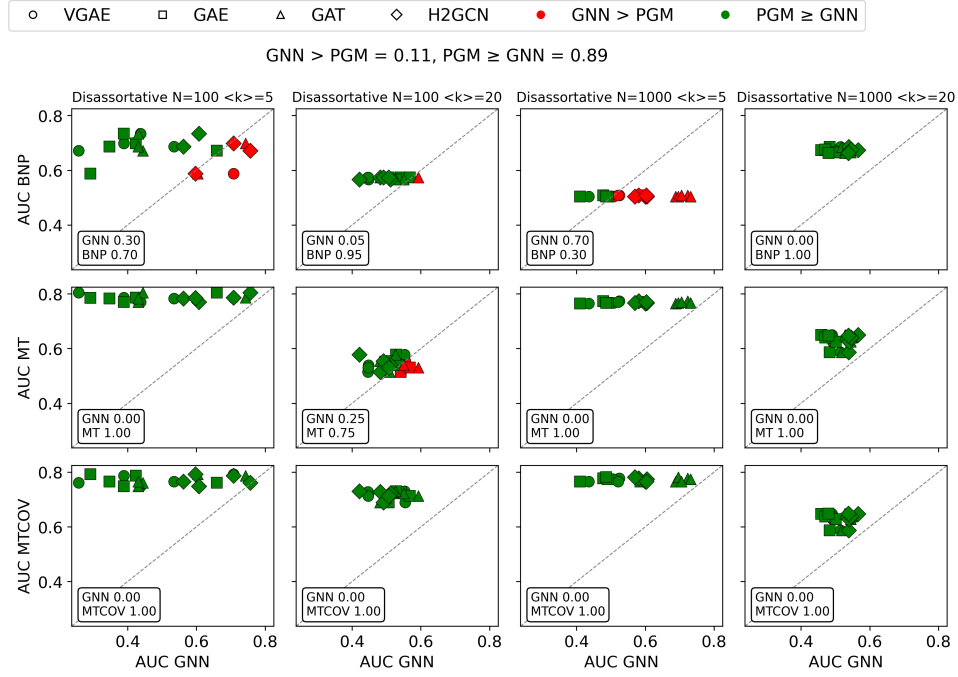


(a) Unperturbed graph (original edges).

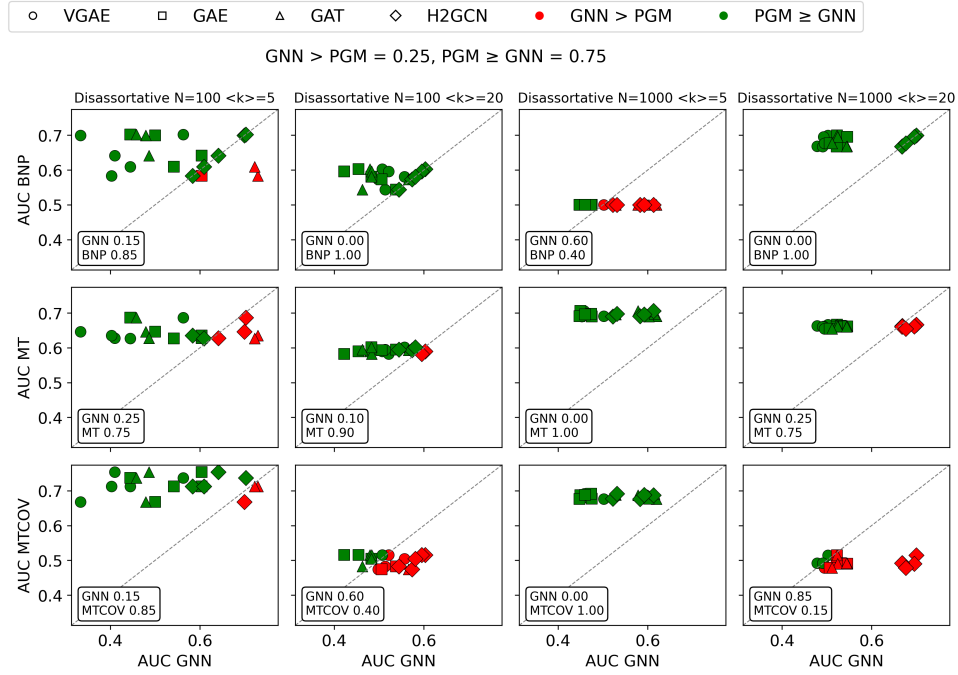


(b) Graph with 10% of edges randomly corrupted.

Figure 6.13: Synthetic assortative datasets: per-fold comparison between PGMs and GNNs. Results correspond to one of the 10 synthetic samples, allowing a direct comparison between the unperturbed and perturbed case. We note that the relative performance between PGMs and GNNs in the unperturbed setting is consistent across all 10 samples. For the PGMs, only MTCOV uses input features. Each point corresponds to one of the 5 cross-validation folds. Numbers inside the boxes are the fraction of folds where each model family outperforms the others.



(a) Unperturbed graph (original edges).



(b) Graph with 10% of edges randomly corrupted.

Figure 6.14: Synthetic disassortative datasets: per-fold comparison between PGMs and GNNs. Results correspond to one of the 10 synthetic samples, allowing a direct comparison between the unperturbed and perturbed case. We note that the relative performance between PGMs and GNNs in the unperturbed setting is consistent across all 10 samples. For the PGMs, only MTCOV uses input features. Each point corresponds to one of the 5 cross-validation folds. Numbers inside the boxes are the fraction of folds where each model family outperforms the others.

Chapter 7

Discussion and Conclusions

In this final chapter, we synthesize the core contributions of this thesis into a unified perspective on learning from graph-structured data. We first summarize the key findings emerging from the theoretical, practical, and comparative studies, and then weave them together to articulate a coherent understanding of Graph Neural Networks (GNNs). This synthesis highlights how GNNs process features, exploit graph topology, and differ from alternative paradigms such as Convolutional Neural Networks (CNNs) and Probabilistic Graphical Models (PGMs). In particular, we emphasize the potential of GNNs as powerful architectures for data fusion, capable of integrating heterogeneous information sources within a unified relational framework. Building on these insights, we discuss the broader implications and limitations of the presented work and outline promising directions for future research.

7.1 Summary of Contributions

This thesis presents a comprehensive investigation of GNNs, clarifying their operational mechanisms, fundamental limitations, and comparative advantages relative to other learning paradigms. Through theoretical, practical, and comparative analyses, we advance both the mechanistic understanding and methodological development of GNNs as models for graph-structured data.

From a *theoretical perspective*, the thermodynamic framework presented in Chapter 4 and published in [31] provides a physically grounded view of learning dynamics. By defining the SGD-temperature as the mean squared gradient activity during equilibrium training, we establish a dynamic and optimizer-agnostic criterion for structured pruning that surpasses magnitude-based methods. Applied to GNNs, this measure identifies the most informative input features as those inducing persistently high squared gradient on the weights (“hot” features) while “cold” features contribute little to learning. Furthermore, this analysis reveals that “hot” features often exhibit strong co-variation across nodes, indicating

redundancy among feature dimensions, and suggesting that GNNs rely on amplifying widely shared patterns rather than exploiting unique node-specific information. This finding reframes redundancy as a stabilizing factor in GNN learning rather than an inefficiency. We also uncover a critical dependency on feature dimensionality: GNNs prove robust when aggressively pruning high-dimensional input features, but performance deteriorates sharply once dimensionality drops below a threshold (of the order of a few dozen features), highlighting a fundamental trade-off between compression and representational richness.

From a *practical perspective*, Chapter 5 demonstrates that GNNs can successfully perform 3D microstructure segmentation when volumetric data is represented as graphs. The proposed Vision Graph Neural Network (ViG) [32] achieves segmentation accuracy comparable to state-of-the-art 3D U-Nets [28] on synthetic data while using significantly fewer parameters. Although the ViG performs competitively on synthetic datasets, its ability to generalize directly to out-of-domain experimental data, without fine-tuning, is less robust than that of the U-Net. We posit that GNN’s reliance on node features makes it more sensitive to changes in grayscale distributions between synthetic and experimental data, while the texture patterns learned by the U-Net’s convolutional filters provide resilience to domain shifts. Nonetheless, when fine-tuned on limited experimental data, the ViG consistently improves, while the U-Net often suffers from degradation. This behavior highlights the superior capacity of GNNs for data-efficient domain adaptation, stemming from their use of relational information, as seen in semi-supervised node classification [11]. However, GNN performance remains highly dependent on feature expressiveness, as using only scalar grayscale values instead of enriched CNN-derived embeddings leads to a substantial drop in segmentation accuracy.

From a *comparative perspective*, Chapter 6 systematically benchmarks GNNs against PGMs [33], focusing on Stochastic Block Models (SBMs). The analysis centers on link prediction, as it directly evaluates a model’s ability to capture pairwise relationships. Results show that PGMs outperform GNNs when node features are low-dimensional, noisy, or when graphs exhibit heterophily, conditions under which explicit probabilistic modeling of edge formation provides greater robustness. Conversely, GNNs excel when high-quality, high-dimensional features are available, achieving performance comparable to PGMs. Notably, GNN performance on heterophilic graphs improves substantially when topology is encoded directly in the node features, reinforcing the idea that their strength lies in processing feature-rich representations rather than inferring structure from connectivity alone. Interestingly, although GNNs lack the explicit community modeling of PGMs, they tend to organize their latent space in a way that implicitly mirrors the underlying community structure. However, the resulting embedding vectors are not directly interpretable, revealing a persistent interpretability gap between the two approaches. Finally, although both models scale linearly with graph size, GNNs’ need for expressive feature representations results in higher computational costs than PGMs.

7.2 Future Directions

Together, the findings presented in this thesis show that GNNs work as feature-driven models for deep representation learning, leveraging graph connectivity primarily as a computational backbone rather than as the principal source of learning signal. Their predictive strength arises from propagating and refining expressive node representations across the graph, rather than from explicitly modeling edges as in PGMs. GNNs naturally excel at tasks such as node classification, which directly leverage learned node representations, whereas link prediction poses a distinct challenge, requiring an additional mapping from node embeddings to edge-level predictions.

One of the most powerful strengths of GNNs, though unexplored in this thesis, lies in their ability to flexibly integrate heterogeneous input features within a unified representation space. Unlike PGMs, which rely on ad-hoc design choices to incorporate heterogeneous attributes, GNNs seamlessly integrate arbitrary feature types, simplifying implementation and broadening applicability. This capability positions them as natural *data-fusion engines*, capable of reasoning jointly across complementary sources of information.

For example, in traffic forecasting, recent work [278] demonstrates that GNNs enhance performance when combining multiple data within the node features, such as information collected from different sensors (e.g., GPS-enabled vehicles and traffic detectors) and heterogeneous attributes (e.g., vehicle speed, weather conditions, or road incidents). Analogously, in materials science, one could enrich node features with complementary techniques, such as X-ray tomography, infrared thermography, or spectroscopy. Extending the segmentation framework introduced in Chapter 5, this would allow the model to learn relationships informed not only by grayscale intensity but also by thermal, mechanical, or chemical properties, potentially enhancing the segmentation of phases that share similar grayscale values in XCT but differ in other physical attributes.

Beyond feature-level fusion, GNNs have also been employed to perform graph-level data fusion. The Evolution Temporal Graph Convolutional Network (ETGCN) [279] exemplifies this approach by combining three different kinds of graph adjacency matrices to identify spatial correlations. This fusion is achieved through a similarity-based cross-product attention mechanism that assigns learnable weights to each adjacency, producing a unified matrix which is then used for graph convolution. However, this approach ultimately collapse structural information into a single learned topology, limiting interpretability regarding how different relational layers interact. In contrast, SBMs provide an explicit probabilistic framework for modeling *multilayer graphs* [150], as in [121, 122], which preserve multiple adjacency matrices as distinct yet interdependent relational layers. Each layer represents a different type of interaction among the same entities, enabling both layer-specific and cross-layer reasoning, and offering a principled means to quantify interdependence between layers.

Combining the structured reasoning and interpretability of PGMs with the flexible handling of heterogeneous features and expressiveness of GNNs represents a promising direction for advancing deep learning on graph-structured data. Recent advances [275] suggest that black-box extensions of probabilistic models such as SBMs could achieve similar flexibility in handling arbitrary node attributes, further bridging the gap between PGMs and GNNs. Future research should therefore focus on developing learning models that not only accommodate heterogeneous features but also fully exploit graph connectivity in a data-driven and interpretable manner, potentially extending these capabilities to multilayer graph settings.

The potential of such hybrid approaches becomes particularly evident in neuroscience, where understanding the human brain requires integrating complementary forms of connectivity and diverse data sources [266]. Among the various neuroimaging modalities, MRI remains the most widely used for brain analysis. Different types of MRI data, such as functional MRI (fMRI) and Diffusion Tensor Imaging (DTI), enable the construction of *functional* and *structural* brain networks, respectively. In functional brain networks, connectivity reflects correlations between the time-series activity of brain regions, whereas in structural brain networks, it models the physical connections between gray matter regions. These modalities offer distinct yet interdependent perspectives on neural organization, but their integration poses a major challenge for conventional Euclidean deep learning models or shallow interpretable methods, such as graph kernels [280] and tensor factorization approaches [281], which are unable to capture the complex, multilayered graph structures underlying brain networks [282].

By pursuing these directions, GNNs can evolve from powerful yet opaque function approximators into structured and interpretable models for graph-based learning, capable of fusing heterogeneous information, reasoning over connectivity, and extracting knowledge from the complex, interconnected data that define modern science and the modern world. Neuroscience, in particular, represents perhaps the most promising frontier, offering a compelling proving ground for such advances, as it demands systems capable of integrating multimodal, multilayer data to produce interpretable, biologically grounded insights into the organization and dynamics of the human brain. Progress in this direction could unlock transformative insights into the brain and, ultimately, into the nature of intelligence itself, with far-reaching implications for humanity.

Bibliography

- [1] M. Gori, Gabriele Monfardini, and F. Scarselli. A new model for learning in graph domains. *Proceedings. 2005 IEEE International Joint Conference on Neural Networks, 2005.*, 2:729–734 vol. 2, 2005.
- [2] Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. The graph neural network model. *IEEE Transactions on Neural Networks*, 20(1):61–80, 2009.
- [3] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.
- [4] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *nature*, 521(7553):436–444, 2015.
- [5] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. *Commun. ACM*, 60(6):84–90, May 2017.
- [6] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Comput.*, 9(8):1735–1780, November 1997.
- [7] Alex Sherstinsky. Fundamentals of recurrent neural network (rnn) and long short-term memory (lstm) network. *Physica D: Nonlinear Phenomena*, 404:132306, 2020.
- [8] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Proceedings of the 31st International Conference on Neural Information Processing Systems, NIPS’17*, page 6000–6010, Red Hook, NY, USA, 2017. Curran Associates Inc.
- [9] Minyi Dai, Mehmet F. Demirel, Yingyu Liang, and Jia-Mian Hu. Graph neural networks for an accurate and interpretable prediction of the properties of polycrystalline materials. *npj Computational Materials*, 7(1):103, 2021.

- [10] Justin Gilmer, Samuel S. Schoenholz, Patrick F. Riley, Oriol Vinyals, and George E. Dahl. Neural message passing for quantum chemistry. In *Proceedings of the 34th International Conference on Machine Learning - Volume 70*, ICML'17, page 1263–1272. JMLR.org, 2017.
- [11] Thomas Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *ArXiv*, abs/1609.02907:1–14, 2016.
- [12] William L. Hamilton, Rex Ying, and Jure Leskovec. Inductive representation learning on large graphs. *Proceedings of the 31st International Conference on Neural Information Processing Systems*, pages 1–19, 2017.
- [13] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph attention networks. In *International Conference on Learning Representations (ICLR)*, 2018. <https://arxiv.org/abs/1710.10903>.
- [14] Chaitanya K. Joshi. Transformers are graph neural networks, 2025.
- [15] Xuhong Li, Haoyi Xiong, Xingjian Li, Xuanyu Wu, Xiao Zhang, Ji Liu, Jiang Bian, and Dejing Dou. Interpretable deep learning: Interpretation, interpretability, trustworthiness, and beyond. *Knowledge and Information Systems*, 64(12):3197–3234, 2022.
- [16] Qiang Huang, Makoto Yamada, Yuan Tian, Dinesh Singh, and Yi Chang. Graphlime: Local interpretable model explanations for graph neural networks. *IEEE Transactions on Knowledge and Data Engineering*, 35(7):6968–6972, 2022.
- [17] B.Yu. Weisfeiler and A.A. Leman. The reduction of a graph to a canonical form and the algebra arising during this reduction. *Nauchno-Technicheskaya Informatsia*, 9:1–11, 1968.
- [18] Van Leeuwen Erik Jan Shervashidze Nino, Schweitzer Pascal, Mehlhorn Kurt, and Borgwardt Karsten M. Weisfeiler-lehman graph kernels. *The Journal of Machine Learning Research*, 12:1–23, 2011.
- [19] Qimai Li, Zhichao Han, and Xiao-ming Wu. Deeper insights into graph convolutional networks for semi-supervised learning. *Proceedings of the AAAI Conference on Artificial Intelligence*, 32(1), Apr. 2018.
- [20] Uri Alon and Eran Yahav. On the bottleneck of graph neural networks and its practical implications. In *International Conference on Learning Representations (ICLR)*, 2021.

- [21] Sitao Luan, Chenqing Hua, Qincheng Lu, Jiaqi Zhu, Mingde Zhao, Shuyuan Zhang, Xiao-Wen Chang, and Doina Precup. Revisiting heterophily for graph neural networks. In *Proceedings of the 36th International Conference on Neural Information Processing Systems*, NIPS '22, Red Hook, NY, USA, 2022. Curran Associates Inc.
- [22] Francesco Di Giovanni, James Rowbottom, Benjamin Paul Chamberlain, Thomas Markovich, and Michael M. Bronstein. Understanding convolution on graphs via energies. *Transactions on Machine Learning Research*, 2023.
- [23] Sitao Luan, Chenqing Hua, Qincheng Lu, Liheng Ma, Lirong Wu, Xinyu Wang, Minkai Xu, Xiao-Wen Chang, Doina Precup, Rex Ying, et al. The heterophilic graph learning handbook: Benchmarks, models, theoretical analysis, applications and challenges. *arXiv preprint arXiv:2407.09618*, 2024.
- [24] Maria Chiara Angelini and Federico Ricci-Tersenghi. Modern graph neural networks do worse than classical greedy algorithms in solving combinatorial optimization problems like maximum independent set. *Nature Machine Intelligence*, 5(1):29–31, 2023.
- [25] Christopher Blöcker, Martin Rosvall, Ingo Scholtes, and Jevin D West. Insights from network science can advance deep graph learning. *arXiv preprint arXiv:2502.01177*, 2025.
- [26] Deepak Acharya and Huaming Zhang. Feature selection and extraction for graph neural networks. In *Proceedings of the 2020 ACM Southeast Conference*, 2020.
- [27] Sergei Evsevlev, Sidnei Paciornik, and Giovanni Bruno. Advanced deep learning-based 3d microstructural characterization of multiphase metal matrix composites. *Advanced Engineering Materials*, 22(4):1–6, 2020.
- [28] Athanasios Tsamos, Sergei Evsevlev, Rita Fioresi, Francesco Faglion, and Giovanni Bruno. Synthetic data generation for automatic segmentation of x-ray computed tomography reconstructions of complex microstructures. *Journal of Imaging*, 9(2):1–23, 2023.
- [29] Michael I Jordan. An introduction to probabilistic graphical models, 2003.
- [30] Jonathan S. Yedidia, William T. Freeman, and Yair Weiss. *Understanding belief propagation and its generalizations*, page 239–269. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2003.
- [31] Michela Lapenna, Francesco Faglion, Rita Fioresi, and Giovanni Bruno. Temperature-based pruning for input features in graph neural networks. *The European Physical Journal Plus*, 140:880, 2025.

- [32] M. Lapenna, A. Tsamos, F. Faglioni, R. Fioresi, F. Zanchetta, and G. Bruno. Vision gnn (vig) architecture for a fine-tuned segmentation of a complex al–si metal matrix composite xct volume. *Journal of Materials Science*, 60(16):6907–6921, 2025.
- [33] Michela Lapenna and Caterina De Bacco. How do probabilistic graphical models and graph neural networks look at network data? *Journal of Physics: Complexity*, 7(1):015008, feb 2026.
- [34] M. Lapenna, F. Faglioni, and R. Fioresi. Thermodynamics modeling of deep learning systems for a temperature based filter pruning technique. *Frontiers in Physics, Sec. Statistical and Computational Physics*, 11, 2023.
- [35] M. Lapenna, F. Faglioni, F. Zanchetta, and R. Fioresi. Geometric deep learning: A temperature based analysis of graph neural networks. In Frank Nielsen and Frédéric Barbaresco, editors, *Geometric Science of Information*, volume 14072 of *Lecture Notes in Computer Science*, pages 634–643. Springer, Cham, 2023.
- [36] M. Lapenna, A. Tsamos, F. Faglioni, R. Fioresi, F. Zanchetta, and G. Bruno. Geometric deep learning for enhanced quantitative analysis of microstructures in x-ray computed tomography data. *Discover Applied Sciences*, 6:1–9, 2024.
- [37] Paul W. Holland, Kathryn Blackmond Laskey, and Samuel Leinhardt. Stochastic blockmodels: First steps. *Social Networks*, 5(2):109–137, 1983.
- [38] Yuchung J. Wang and George Y. Wong. Stochastic blockmodels for directed graphs. *Journal of the American Statistical Association*, 82(397):8–19, 1987.
- [39] Tom Snijders and Krzysztof Nowicki. Estimation and prediction for stochastic blockmodels for graphs with latent block structure. *Journal of Classification*, 14:75–100, 01 1997.
- [40] Emmanuel Abbe. Community detection and stochastic block models: recent developments. *J. Mach. Learn. Res.*, 18(1):6446–6531, January 2017.
- [41] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation, 2015.
- [42] Leo Breiman, Jerome H. Friedman, Richard A. Olshen, and Charles J. Stone. *Classification and Regression Trees*. Chapman and Hall/CRC, 1st edition, 1984.
- [43] Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Mach. Learn.*, 20(3):273–297, September 1995.

- [44] Minshuo Chen, Yu Bai, Jason D. Lee, Tuo Zhao, Huan Wang, Caiming Xiong, and Richard Socher. Towards understanding hierarchical learning: benefits of neural representations. In *Proceedings of the 34th International Conference on Neural Information Processing Systems*, NIPS '20, Red Hook, NY, USA, 2020. Curran Associates Inc.
- [45] Francesco Cagnetta, Leonardo Petrini, Umberto M. Tomasini, Alessandro Favero, and Matthieu Wyart. How deep neural networks learn compositional data: The random hierarchy model. *Phys. Rev. X*, 14:031001, Jul 2024.
- [46] Frank Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65 6:386–408, 1958.
- [47] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. Learning representations by back-propagating errors. *Nature*, 323:533–536, 1986.
- [48] George V. Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals and Systems*, 2:303–314, 1989.
- [49] Abien Fred Agarap. Deep learning using rectified linear units (relu). *ArXiv*, abs/1803.08375, 2018.
- [50] Anqi Mao, Mehryar Mohri, and Yutao Zhong. Cross-entropy loss functions: theoretical analysis and applications. In *Proceedings of the 40th International Conference on Machine Learning*, ICML'23. JMLR.org, 2023.
- [51] Solomon Kullback and R. A. Leibler. On information and sufficiency. *Annals of Mathematical Statistics*, 22:79–86, 1951.
- [52] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. Learning internal representations by error propagation. In David E. Rumelhart, James L. McClelland, and the PDP Research Group, editors, *Parallel Distributed Processing: Explorations in the Microstructure of Cognition, Volume 1: Foundations*, chapter 8, pages 318–362. MIT Press, Cambridge, MA, USA, 1986.
- [53] Dor Bank, Noam Koenigstein, and Raja Giryes. Autoencoders. *Deep Learning in Science*, 2021.
- [54] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial networks. *Commun. ACM*, 63(11):139–144, October 2020.
- [55] Herbert E. Robbins. A stochastic approximation method. *Annals of Mathematical Statistics*, 22:400–407, 1951.

- [56] Léon Bottou. Large-scale machine learning with stochastic gradient descent. In *International Conference on Computational Statistics*, 2010.
- [57] Y. Nesterov. A method for solving the convex programming problem with convergence rate $o(1/k^2)$. *Proceedings of the USSR Academy of Sciences*, 269:543–547, 1983. <https://api.semanticscholar.org/CorpusID:145918791>.
- [58] D. P. Kingma and J. B. Ba. Adam: A method for stochastic optimization. In *3rd International Conference on Learning Representations (ICLR)*, 2015.
- [59] Max Welling and Yee Whye Teh. Bayesian learning via stochastic gradient langevin dynamics. In *International Conference on Machine Learning*, 2011.
- [60] W. K. Hastings. Monte carlo sampling methods using markov chains and their applications. *Biometrika*, 57(1):97–109, 1970.
- [61] M E J Newman and G T Barkema. *Monte Carlo Methods in Statistical Physics*. Oxford University Press, 02 1999.
- [62] Avrim Blum and Ronald L. Rivest. Training a 3-node neural network is np-complete. In *Proceedings of the 2nd International Conference on Neural Information Processing Systems*, NIPS’88, page 494–501, Cambridge, MA, USA, 1988. MIT Press.
- [63] Hao Li, Zheng Xu, Gavin Taylor, Christoph Studer, and Tom Goldstein. Visualizing the loss landscape of neural nets. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, NIPS’18, page 6391–6401, Red Hook, NY, USA, 2018. Curran Associates Inc.
- [64] Kaiming He, X. Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, 2015.
- [65] Yihe Dong, Jean-Baptiste Cordonnier, and Andreas Loukas. Attention is not all you need: Pure attention loses rank doubly exponentially with depth, 2023.
- [66] Kai Han, Yunhe Wang, Jianyuan Guo, Yehui Tang, and Enhua Wu. Vision gnn: an image is worth graph of nodes. *Proceedings of the 36th International Conference on Neural Information Processing Systems*, pages 1–15, 2022.
- [67] Giuseppe Carleo, Ignacio Cirac, Kyle Cranmer, Laurent Daudet, Maria Schuld, Naftali Tishby, Leslie Vogt-Maranto, and Lenka Zdeborová. Machine learning and the physical sciences. *Rev. Mod. Phys.*, 91:045002, Dec 2019.
- [68] J. Hopfield J. Neurons with graded response have collective computational properties like those of two-state neurons. *PNAS*, 81(10):3088, 3092.

- [69] J J Hopfield. Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the National Academy of Sciences*, 79(8):2554–2558, 1982.
- [70] E. Gardner. Maximum storage capacity in neural networks. *Europhysics Letters*, 4(4):481, aug 1987.
- [71] E Gardner. The space of interactions in neural network models. *Journal of Physics A: Mathematical and General*, 21(1):257, jan 1988.
- [72] Chiyuan Zhang, Samy Bengio, Moritz Hardt, Benjamin Recht, and Oriol Vinyals. Understanding deep learning requires rethinking generalization. *ArXiv*, abs/1611.03530, 2016.
- [73] Marco Baity-Jesi, Levent Sagun, Mario Geiger, Stefano Spigler, Gerard Ben Arous, Chiara Cammarota, Yann LeCun, Matthieu Wyart, and Giulio Biroli. Comparing dynamics: Deep neural networks versus glassy systems. In Jennifer Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 314–323. PMLR, 10–15 Jul 2018.
- [74] Qianxiao Li, Cheng Tai, and Weinan E. Stochastic modified equations and adaptive stochastic gradient algorithms. In *Proceedings of the 34th International Conference on Machine Learning*, 2017. <https://arxiv.org/abs/1511.06251>.
- [75] Pratik Chaudhari and Stefano Soatto. Stochastic gradient descent performs variational inference, converges to limit cycles for deep networks. *International Conference on Learning Representations ICLR 2018*, 2018.
- [76] Xiang Cheng, Dong Yin, Peter Bartlett, and Michael Jordan. Stochastic gradient and Langevin processes. In Hal Daumé III and Aarti Singh, editors, *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pages 1810–1819. PMLR, 13–18 Jul 2020.
- [77] Hannes Risken. *The Fokker-Planck Equation: Methods of Solution and Applications*. Springer, 1996.
- [78] Sho Yaida. Fluctuation-dissipation relations for stochastic gradient descent. *ArXiv*, abs/1810.00004, 2018.
- [79] Francesca Mignacco and Pierfrancesco Urbani. The effective noise of stochastic gradient descent. *Journal of Statistical Mechanics: Theory and Experiment*, 2022(8):083405, aug 2022.

- [80] Francesca Mignacco, Florent Krzakala, Pierfrancesco Urbani, and Lenka Zdeborová. Dynamical mean-field theory for stochastic gradient descent in gaussian mixture classification. In H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 9540–9550. Curran Associates, Inc., 2020.
- [81] Francesca Mignacco, Pierfrancesco Urbani, and Lenka Zdeborová. Stochasticity helps to navigate rough landscapes: comparing gradient-descent-based algorithms in the phase retrieval problem. *Machine Learning: Science and Technology*, 2(3):035029, jul 2021.
- [82] Levent Sagun, Utku Evci, V. Ugur Güney, Yann Dauphin, and Léon Bottou. Empirical analysis of the hessian of over-parametrized neural networks. *ArXiv*, abs/1706.04454, 2017.
- [83] Pratik Chaudhari, Anna Choromańska, Stefano Soatto, Yann LeCun, Carlo Baldassi, Christian Borgs, Jennifer T. Chayes, Levent Sagun, and Riccardo Zecchina. Entropy-sgd: biasing gradient descent into wide valleys. *Journal of Statistical Mechanics: Theory and Experiment*, 2019, 2016.
- [84] Carlo Baldassi, Alessandro Ingrosso, Carlo Lucibello, Luca Saglietti, and Riccardo Zecchina. Subdominant dense clusters allow for simple learning and high computational performance in neural networks with discrete synapses. *Physical Review Letter*, 115, 2015.
- [85] Jason Yosinski, Jeff Clune, Anh Totti Nguyen, Thomas J. Fuchs, and Hod Lipson. Understanding neural networks through deep visualization. *ArXiv*, abs/1506.06579, 2015.
- [86] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C.J. Burges, L. Bottou, and K.Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 25, pages 1–9. Curran Associates, Inc., 2012.
- [87] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pages 248–255, 2009.
- [88] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott E. Reed, Dragomir Anguelov, D. Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1–9, 2014.

- [89] C. Olah, A. Mordvintsev, and L. Schubert. Feature visualization. *Distill*, 2017.
- [90] A. Amidi and S. Amidi. Cs 230 - convolutional neural networks cheatsheet, 2023. Accessed: [Date].
- [91] Paul J. Werbos. Backpropagation through time: What it does and how to do it. *Proc. IEEE*, 78:1550–1560, 1990.
- [92] Ricky T. Q. Chen, Yulia Rubanova, Jesse Bettencourt, and David Duvenaud. Neural ordinary differential equations. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, NIPS’18, page 6572–6583, Red Hook, NY, USA, 2018. Curran Associates Inc.
- [93] Partha Pratim Ray. Chatgpt: A comprehensive review on background, applications, key challenges, bias, ethics, limitations and future scope. *Internet of Things and Cyber-Physical Systems*, 3:121–154, 2023.
- [94] DeepSeek-AI et al. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *CoRR*, abs/2501.12948, January 2025.
- [95] Michael M. Bronstein, Joan Bruna, Yann LeCun, Arthur Szlam, and Pierre Vandergheynst. Geometric deep learning: Going beyond euclidean data. *IEEE Signal Processing Magazine*, 34(4):18–42, 2017.
- [96] Michael M. Bronstein, Joan Bruna, Taco Cohen, and Petar Velivckovic. Geometric deep learning: Grids, groups, graphs, geodesics, and gauges. *ArXiv*, abs/2104.13478:1–160, 2021.
- [97] Petar Veličković. Message passing all the way up, 2022.
- [98] Peter W. Battaglia et al. Relational inductive biases, deep learning, and graph networks. *ArXiv*, abs/1806.01261, 2018.
- [99] R. Fioresi and F. Zanchetta. Deep learning and geometric deep learning: An introduction for mathematicians and physicists. *International Journal of Geometric Methods in Modern Physics*, 20(12):2330006, 2023.
- [100] Jure Leskovec Keyulu Xu, Weihua Hu and Stefanie Jegelka. How powerful are graph neural networks? *International Conference on Learning Representations*, pages 1–17, 2019.
- [101] Jiong Zhu, Yujun Yan, Lingxiao Zhao, Mark Heimann, Leman Akoglu, and Danai Koutra. Beyond homophily in graph neural networks: Current limitations and effective designs. In *Proceedings of the 34th International Conference on Neural*

- Information Processing Systems*, NIPS '20, Red Hook, NY, USA, 2020. Curran Associates Inc.
- [102] T. Konstantin Rusch, Michael M. Bronstein, and Siddhartha Mishra. A survey on oversmoothing in graph neural networks, 2023.
 - [103] Xinyi Wu, Amir Ajorlou, Zihui Wu, and Ali Jadbabaie. Demystifying oversmoothing in attention-based graph neural networks. In *Proceedings of the 37th International Conference on Neural Information Processing Systems*, NIPS '23, Red Hook, NY, USA, 2023. Curran Associates Inc.
 - [104] Kenta Oono and Taiji Suzuki. Graph neural networks exponentially lose expressive power for node classification. *arXiv: Learning*, 2019.
 - [105] Chen Cai and Yusu Wang. A note on over-smoothing for graph neural networks. *ArXiv*, abs/2006.13318, 2020.
 - [106] Miller McPherson, Lynn Smith-Lovin, and James M. Cook. Birds of a feather: Homophily in social networks. *Review of Sociology*, 27:415–444, 2001.
 - [107] A.K. McCallum, K. Nigam, J. Rennie, and K. Seymore. Automating the construction of internet portals with machine learning. *Information Retrieval*, 3(2):127–163, 2000.
 - [108] Giles C. Lee, Bollacker Kurt D., and Lawrence Steve. Citeseer: an automatic citation indexing system. In *Proceedings of the Third ACM Conference on Digital Libraries*, pages 89–98. ACM, 1998.
 - [109] Kshitij Karhadkar, Pratik K. Banerjee, and Guido Montúfar. Fosr: First-order spectral rewiring for addressing oversquashing in gnns. In *International Conference on Learning Representations (ICLR)*, 2023.
 - [110] Lingxiao Zhao and Leman Akoglu. Pairnorm: Tackling oversmoothing in gnns. In *International Conference on Learning Representations*, 2020.
 - [111] Khang Nguyen, Hieu Nong, Vinh Nguyen, Nhat Ho, Stanley Osher, and Tan Nguyen. Revisiting over-smoothing and over-squashing using ollivier-ricci curvature. In *Proceedings of the 40th International Conference on Machine Learning*, ICML'23. JMLR.org, 2023.
 - [112] T. K. Rusch, B. Chamberlain, J. Rowbottom, S. Mishra, and M. Bronstein. Graph-coupled oscillator networks. In K. Chaudhuri, S. Jegelka, L. Song, C. Szepesvári, G. Niu, and S. Sabato, editors, *Proceedings of the 39th International Conference on Machine Learning (ICML)*, volume 162, pages 18888–18909. PMLR, 2022.

- [113] G. Li, Matthias Müller, Ali K. Thabet, and Bernard Ghanem. Deepgcns: Can gcns go as deep as cnns? *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 9266–9275, 2019.
- [114] Jianfei Chen, Jun Zhu, and Le Song. Stochastic training of graph convolutional networks with variance reduction. In *International Conference on Machine Learning*, 2017.
- [115] Réka Albert and Albert-László Barabási. Statistical mechanics of complex networks. *Rev. Mod. Phys.*, 74:47–97, Jan 2002.
- [116] Martina Contisciani. *Probabilistic Generative Models for Inference on Complex Systems*. PhD thesis, University of Tübingen, Tübingen, Germany, 2023. Dissertation.
- [117] Paul L. Erdos and Alfréd Rényi. On the evolution of random graphs. *Transactions of the American Mathematical Society*, 286:257–257, 1984.
- [118] Mark Newman. Random graphs as models of networks. *arXiv: Statistical Mechanics*, pages 35–68, 2002.
- [119] David M. Blei. Build, compute, critique, repeat: Data analysis with latent variable models, 2014.
- [120] Leto Peel, Tiago P. Peixoto, and Manlio De Domenico. Statistical inference links data and theory in network science. *Nature Communications*, 13(1):6794, nov 2022.
- [121] Caterina De Bacco, Eleanor A. Power, Daniel B. Larremore, and Cristopher Moore. Community detection, link prediction, and layer interdependence in multilayer networks. *Phys. Rev. E*, 95:042317, Apr 2017.
- [122] Martina Contisciani, Eleanor Power, and Caterina De Bacco. Community detection with node attributes in multilayer networks. *Scientific reports*, 10:15736, 09 2020.
- [123] T.K. Moon. The expectation-maximization algorithm. *IEEE Signal Processing Magazine*, 13(6):47–60, 1996.
- [124] A. P. Dempster, N. M. Laird, and D. B. Rubin. Maximum likelihood from incomplete data via the em algorithm. *Journal of the Royal Statistical Society: Series B (Methodological)*, 39(1):1–22, 12 2018.
- [125] J. L. W. V. Jensen. Sur les fonctions convexes et les inégalités entre les valeurs moyennes. *Acta Mathematica*, 30(1):175–193, dec 1906.

- [126] Michael I. Jordan, Zoubin Ghahramani, Tommi S. Jaakkola, and Lawrence K. Saul. An introduction to variational methods for graphical models. *Machine Learning*, 37(2):183–233, nov 1999.
- [127] David M. Blei, Alp Kucukelbir, and Jon D. McAuliffe. Variational inference: A review for statisticians. *Journal of the American Statistical Association*, 112:859 – 877, 2016.
- [128] J. T. Ormerod and M. P. Wand. Explaining variational approximations. *The American Statistician*, 64(2):140–153, 2010.
- [129] Nicholas Metropolis, Arianna W. Rosenbluth, Marshall N. Rosenbluth, Augusta H. Teller, and Edward Teller. Equation of state calculations by fast computing machines. *The Journal of Chemical Physics*, 21(6):1087–1092, 1953.
- [130] Siddhartha Chib and Edward Greenberg. Understanding the metropolis-hastings algorithm. *The American Statistician*, 49(4):327–335, November 1995.
- [131] Santo Fortunato. Community detection in graphs. *Physics Reports*, 486(3–5):75–174, February 2010.
- [132] Michele Coscia, Fosca Giannotti, and Dino Pedreschi. A classification for community discovery methods in complex networks. *Stat. Anal. Data Min.*, 4(5):512–546, October 2011.
- [133] Mark E. J. Newman. Communities, modules and large-scale structure in networks. *Nature Physics*, 8:25 – 31, 2011.
- [134] Edoardo M. Airoldi, David M. Blei, Stephen E. Fienberg, and Eric P. Xing. Mixed membership stochastic blockmodels. In *Proceedings of the 22nd International Conference on Neural Information Processing Systems*, NIPS’08, page 33–40, Red Hook, NY, USA, 2008. Curran Associates Inc.
- [135] Santo Fortunato and Darko Hric. Community detection in networks: A user guide. *ArXiv*, abs/1608.00163, 2016.
- [136] Prem K. Gopalan, Sean Gerrish, Michael Freedman, David M. Blei, and David Mimno. Scalable inference of overlapping communities. In *Advances in Neural Information Processing Systems*, volume 25 of *NIPS’12*. Curran Associates, Inc., 2012.
- [137] M. E. J. Newman and Gesine Reinert. Estimating the number of communities in a network. *Phys. Rev. Lett.*, 117:078301, Aug 2016.

- [138] K Nowicki, A. Krysztof, and T.A.B. Snijders. Estimation and prediction for stochastic blockstructures. *Journal of the American Statistical Association*, 96(455):1077–1087, September 2001.
- [139] Tiago P. Peixoto. Entropy of stochastic blockmodel ensembles. *Phys. Rev. E*, 85:056122, May 2012.
- [140] Tiago P. Peixoto. Hierarchical block structures and high-resolution model selection in large networks. *Phys. Rev. X*, 4:011047, Mar 2014.
- [141] Kehui Chen and Jing Lei. Network cross-validation for determining the number of communities in network data, 2015.
- [142] Bowei Yan and Purnamrita Sarkar. Covariate regularized community detection in sparse graphs. *Journal of the American Statistical Association*, 116:734 – 745, 2016.
- [143] Yash Deshpande, Andrea Montanari, Elchanan Mossel, and Subhabrata Sen. Contextual stochastic block models. In *Neural Information Processing Systems*, 2018.
- [144] Jaewon Yang, Julian McAuley, and Jure Leskovec. Community detection in networks with node attributes. *2013 IEEE 13th International Conference on Data Mining*, pages 1151–1156, 2013.
- [145] David S. Choi, Patrick Wolfe, and Edoardo M. Airolidi. Confidence sets for network structure. In *Proceedings of the 25th International Conference on Neural Information Processing Systems*, NIPS’11, page 2097–2105, Red Hook, NY, USA, 2011. Curran Associates Inc.
- [146] Tracy M. Sweet. Incorporating covariates into stochastic blockmodels. *Journal of Educational and Behavioral Statistics*, 40(6):635–664, 2015.
- [147] Leto Peel, Daniel B Larremore, and Aaron Clauset. The ground truth about metadata and community detection in networks. *Science advances*, 3(5):e1602548, 2017.
- [148] Mark EJ Newman and Aaron Clauset. Structure and inference in annotated networks. *Nature communications*, 7(1):11863, 2016.
- [149] Anna Badalyan, Nicolò Ruggeri, and Caterina De Bacco. Structure and inference in hypergraphs with node attributes. *Nature Communications*, 15(1):7073, 2024.
- [150] Mikko Kivelä, Alex Arenas, Marc Barthélemy, James P. Gleeson, Yamir Moreno, and Mason A. Porter. Multilayer networks. *Journal of Complex Networks*, 2(3):203–271, 07 2014.

- [151] Brian Ball, Brian Karrer, and M. E. J. Newman. Efficient and principled method for detecting communities in networks. *Phys. Rev. E*, 84:036103, Sep 2011.
- [152] Prem Gopalan, Jake M. Hofman, and David M. Blei. Scalable recommendation with hierarchical poisson factorization. In *Proceedings of the Thirty-First Conference on Uncertainty in Artificial Intelligence*, UAI’15, page 326–335, Arlington, Virginia, USA, 2015. AUAI Press.
- [153] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 25:1097–1105, 2012.
- [154] W. James Murdoch, Chandan Singh, Karl Kumbier, Reza Abbasi-Asl, and Bin Yu. Interpretable machine learning: definitions, methods, and applications. *PNAS 2019*, 2019.
- [155] Vaishak Belle and Ioannis Papantonis. Principles and practice of explainable machine learning. *Frontiers in Big Data*, 2021.
- [156] Edwin T Jaynes. Information theory and statistical mechanics. *Physical review*, 106(4):620, 1957.
- [157] David H Ackley, Geoffrey E Hinton, and Terrence J Sejnowski. A learning algorithm for boltzmann machines. *Cognitive science*, 9(1):147–169, 1985.
- [158] Ruslan Salakhutdinov and Geoffrey Hinton. Deep boltzmann machines. In *Artificial intelligence and statistics*, pages 448–455. PMLR, 2009.
- [159] Rodrigo Veiga, Ludovic Stephan, Bruno Loureiro, Florent Krzakala, and Lenka Zdeborová. Phase diagram of stochastic gradient descent in high-dimensional two-layer neural networks. In *36th Conference on Neural Information Processing Systems (NeurIPS)*, 2022.
- [160] Dong Zhang. On the temperature of machine learning systems. <https://api.semanticscholar.org/CorpusID:269292921>, 2024.
- [161] R. Fioresi, P. Chaudhari, and S. Soatto. A geometric interpretation of stochastic gradient descent using diffusion metrics. *Entropy*, 2020.
- [162] Frédéric Barbaresco. Lie group statistics and lie group machine learning based on souriau lie groups thermodynamics and koszul-souriau-fisher metric. *Entropy*, 22(6):642, 2020.

- [163] Charles-Michel Marle. From tools in symplectic and poisson geometry to j. m. souriau’s theories of statistical mechanics and thermodynamics. *Entropy*, 18(10):370, 2016.
- [164] A. Anahory Simoes, M. De Leon, M. Lainz Valcazar, and D. M. De Diego. Contact geometry for simple thermodynamical systems with friction. *Proceedings of the Royal Society A*, 476(2241):20200244, 2020.
- [165] He Yang and Xiao Lingao. Structured pruning for deep convolutional neural networks: A survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2024.
- [166] H. Cheng, M. Zhang, and J. Q. Shi. A survey on deep neural network pruning: Taxonomy, comparison, analysis, and recommendations. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2024.
- [167] Jonathan Frankle and Michael Carbin. The lottery ticket hypothesis: Finding sparse, trainable neural networks. In *International Conference on Learning Representations (ICLR)*, 2019. <https://arxiv.org/abs/1803.03635>.
- [168] Manon Verbockhaven, Sylvain Chevallier, and Guillaume Charpiat. Growing tiny networks: Spotting expressivity bottlenecks and fixing them optimally, 2024. <https://arxiv.org/abs/2405.19816>.
- [169] Hao Zhou, Jose M. Alvarez, and Fatih Porikli. Less is more: Towards compact cnns. In *European Conference on Computer Vision (ECCV)*, 2016.
- [170] Wei Wen, Chunpeng Wu, Yandan Wang, Yiran Chen, and Hai Li. Learning structured sparsity in deep neural networks. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 29, 2016.
- [171] Hao Li, Asim Kadav, Igor Durdanovic, Hanan Samet, and Hans Peter Graf. Pruning filters for efficient convnets. *International Conference on Learning Representations ICLR 2017*, 2017.
- [172] Yann Le Cun, John S. Denker, and Sara A. Solla. Optimal brain damage. In *Advances in Neural Information Processing Systems*, volume 2, 1989.
- [173] Babak Hassibi and David G. Stork. Second order derivatives for network pruning: Optimal brain surgeon. In *Proceedings of the 5th International Conference on Neural Information Processing Systems (NIPS)*, 1992.
- [174] Pavlo Molchanov, Stephen Tyree, Tero Karras, Timo Aila, and Jan Kautz. Pruning convolutional neural networks for resource efficient inference. *ArXiv*, abs/1611.06440:1–12, 2017.

- [175] S Nose. Constant-temperature molecular dynamics. *Journal of Physics: Condensed Matter*, 2(S):SA115, dec 1990.
- [176] Shuichi Nosé. Constant temperature molecular dynamics methods. *Progress of Theoretical Physics Supplement*, 103:1–46, 1991.
- [177] Rita Fioresi, Francesco Faglioni, Francesco Morri, and Lorenzo Squadrani. On the thermodynamic interpretation of deep learning systems. *Geometric Science of Information: 5th International Conference*, 2021.
- [178] Lukas Balles and Philipp Hennig. Dissecting adam: The sign, magnitude and variance of stochastic gradients. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2017. <https://api.semanticscholar.org/CorpusID:46884491>.
- [179] Prithviraj Sen, Galileo Mark Namata, Mustafa Bilgic, Lise Getoor, Brian Gallagher, and Tina Eliassi-Rad. Collective classification in network data. *AI Magazine*, 29(3):93–106, 2008.
- [180] Marinka Zitnik, Monu Agrawal, and Jure Leskovec. Predicting multicellular function through multi-layer tissue networks. *Bioinformatics*, 33(14):1916–1923, 2017.
- [181] William L. Hamilton, Rex Ying, and Jure Leskovec. Inductive representation learning on large graphs. In *Proceedings of the 31st International Conference on Neural Information Processing Systems, NIPS’17*, page 1025–1035, Red Hook, NY, USA, 2017. Curran Associates Inc.
- [182] Juanhui Li, Harry Shomer, Haitao Mao, Shenglai Zeng, Yao Ma, Neil Shah, Jiliang Tang, and Dawei Yin. Evaluating graph neural networks for link prediction: Current pitfalls and new benchmarking. In *Proceedings of the 37th International Conference on Neural Information Processing Systems (NeurIPS)*, pages 169–182. Curran Associates Inc., 2023. <https://github.com/Juanhui28/HeaRT>.
- [183] S. Ratna, Sukhdeep Singh, and Anuj Sharma. An inclusive analysis for performance and efficiency of graph neural network models for node classification. *Computer Science Review*, 56:100722, 2025. <https://doi.org/10.1016/j.cosrev.2024.100722>.
- [184] Ryoma Sato. Graph neural networks can recover the hidden features solely from the graph structure. In *Proceedings of the 40th International Conference on Machine Learning, ICML’23*. JMLR.org, 2023.
- [185] Li Deng. The mnist database of handwritten digit images for machine learning research. *IEEE Signal Processing Magazine*, 29(6):141–142, 2012.

- [186] Sajid Anwar, Kyuyeon Hwang, and Wonyong Sung. Structured pruning of deep convolutional neural networks. *ACM Journal on Emerging Technologies in Computing Systems, Volume 13, Issue 3*, 2017.
- [187] Kamal Choudhary, Brian L. DeCost, Chi Chen, Anubhav Jain, Francesca Tavazza, Ryan Cohn, Cheol Woo Park, Alok N. Choudhary, Ankit Agrawal, Simon J. L. Billinge, Elizabeth A. Holm, Shyue Ping Ong, and Christopher M. Wolverton. Recent advances and applications of deep learning methods in materials science. *npj Computational Materials*, 8:1–26, 2021.
- [188] Prajjwal Acharya, T. P. Chu, K. R. Ahmed, and S. C. Kharel. A deep learning approach for defect detection and segmentation in x-ray computed tomography slices of additively manufactured components. *International Journal of Artificial Intelligence & Applications*, 13(4):1–15, 2022.
- [189] T. Strohmann, K. Bugelnig, E. Breitbarth, F. Wilde, T. Steffens, H. Germann, and G. Requena. Semantic segmentation of synchrotron tomography of multiphase al-si alloys using a convolutional neural network with a pixel-wise weighted loss function. *Sci. Rep.*, pages 1–10, 2019.
- [190] Athanasios Tsamos, Sergei Evsevlev, Rita Fioresi, Francesco Faglioni, and Giovanni Bruno. A complete strategy to achieve high precision automatic segmentation of challenging experimental x-ray computed tomography data using low-resemblance synthetic training data. *Advanced Engineering Materials*, 26:1–9, 2023.
- [191] Nobuyuki Otsu. A threshold selection method from gray-level histograms. *IEEE Transactions on Systems, Man, and Cybernetics*, 9(1):62–66, 1979.
- [192] Binbin Lin, Nima Emami, David A. Santos, Yuting Luo, Sarbajit Banerjee, and Bai-Xiang Xu. A deep learned nanowire segmentation model using synthetic data augmentation. *npj Computational Materials*, 8(1):88, Apr 2022.
- [193] Boyuan Ma, Xiaoyan Wei, Chuni Liu, Xiaojuan Ban, Haiyou Huang, Hao Wang, Weihua Xue, Stephen Wu, Mingfei Gao, Qing Shen, Michele Mukeshimana, Adnan Omer Abuassba, Haokai Shen, and Yanjing Su. Data augmentation in microscopic images for material data mining. *npj Computational Materials*, 6(1):125, Aug 2020.
- [194] Aleksei Boikov, Vladimir Payor, Roman Savelev, and Alexandr Kolesnikov. Synthetic data generation for steel defect detection and classification using deep learning. *Symmetry*, 13(7):1176, Jun 2021.

- [195] Tariq Alkhalifah, Hanchen Wang, and Oleg Ovcharenko. Mlreal: Bridging the gap between training on synthetic data and real data applications in machine learning. *Artificial Intelligence in Geosciences*, 3:101–114, 2022.
- [196] D. R. Sarvamangala and Raghavendra V. Kulkarni. Convolutional neural networks in medical image understanding: a survey. *Evolutionary Intelligence*, 15:1 – 22, 2021.
- [197] Vivian Wen Hui Wong, Max Ferguson, Kincho H. Law, Yung-Tsun Tina Lee, and Paul Witherell. Automatic volumetric segmentation of additive manufacturing defects with 3d u-net, 2021.
- [198] Wangzhe Du, Hongyao Shen, Jianzhong Fu, Ge Zhang, Xuanke Shi, and Quan He. Automated detection of defects with low semantic information in x-ray images based on deep learning. *Journal of Intelligent Manufacturing*, 32:141–156, 2020.
- [199] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.
- [200] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, 2016.
- [201] Nicolas Carion, Francisco Massa, Gabriel Synnaeve, Nicolas Usunier, Alexander Kirillov, and Sergey Zagoruyko. End-to-end object detection with transformers. *Computer Vision – ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part I*, page 213–229, 2020.
- [202] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale. *ArXiv*, abs/2010.11929:1–22, 2020.
- [203] Alexander Kirillov, Eric Mintun, Nikhila Ravi, Hanzi Mao, Chloé Rolland, Laura Gustafson, Tete Xiao, Spencer Whitehead, Alexander C. Berg, Wan-Yen Lo, Piotr Dollár, and Ross B. Girshick. Segment anything. *2023 IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 3992–4003, 2023.
- [204] Benyamin Ghojogh and Ali Ghodsi. Graph neural network, chebnet, graph convolutional network, and graph autoencoder: Tutorial and survey, 2024.
- [205] Loic Landrieu and Martin Simonovsky. Large-scale point cloud semantic segmentation with superpoint graphs. *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 4558–4567, 2017.

- [206] Yue Wang, Yongbin Sun, Ziwei Liu, Sanjay E. Sarma, Michael M. Bronstein, and Justin M. Solomon. Dynamic graph cnn for learning on point clouds. *ACM Trans. Graph.*, 38(5):1–13, oct 2019.
- [207] Yiding Yang, Jiayan Qiu, Mingli Song, Dacheng Tao, and Xinchao Wang. Distilling knowledge from graph convolutional networks. *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 7072–7081, 2020.
- [208] Victor Garcia Satorras, Emiel Hoogetboom, and Max Welling. E(n) equivariant graph neural networks. *ArXiv*, abs/2102.09844, 2021.
- [209] Laurens van der Maaten and Geoffrey Hinton. Visualizing data using t-sne. *Journal of Machine Learning Research*, 9:2579–2605, 11 2008.
- [210] Antoni Buades, Bartomeu Coll, and Jean-Michel Morel. A non-local algorithm for image denoising. *Computer Society Conference on Computer Vision and Pattern Recognition*, 2:60–65 vol. 2, 2005.
- [211] Athanasios Tsamos, Sergei Evsevlev, and Giovanni Bruno. Noise and blur removal from corrupted x-ray computed tomography scans: A multilevel and multiscale deep convolutional framework approach with synthetic training data (bam synthcond). *Tomography of Materials and Structures*, pages 1–16, 2023.
- [212] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph Attention Networks. *International Conference on Learning Representations*, pages 1–12, 2018.
- [213] T. Konstantin Rusch, Michael M. Bronstein, and Siddhartha Mishra. A survey on oversmoothing in graph neural networks. *ArXiv*, abs/2303.10993:1–10, 2023.
- [214] PengHui He, AiPing Qu, ShuoMin Xiao, and MeiDan Ding. A gnn-based network for tissue semantic segmentation in histopathology image. *Journal of Physics: Conference Series*, 2504(1):012047, may 2023.
- [215] Eiji Uchibe Stefan Elfving and Kenji Doya. Sigmoid-weighted linear units for neural network function approximation in reinforcement learning, 2017.
- [216] A. Genç, Libor Kovarik, and Hamish L. Fraser. A deep learning approach for semantic segmentation of unbalanced data in electron tomography of catalytic materials. *Scientific Reports*, 12:1–12, 2022.
- [217] Vajira Thambawita, Pegah Salehi, Sajad Amouei Sheshkal, Steven A. Hicks, Hugo L. Hammer, Sravanthi Parasa, Thomas de Lange, Pål Halvorsen, and Michael A. Riegler. Singan-seg: Synthetic training data generation for medical image segmentation. *PLOS ONE*, 17(5):1–24, May 2022.

- [218] Riccardo Corvi, Davide Cozzolino, Giovanni Poggi, Koki Nagano, and Luisa Verdoliva. Intriguing properties of synthetic images: from generative adversarial networks to diffusion models. *2023 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, pages 973–982, 2023.
- [219] Quang Nguyen, Truong Vu, Anh Tran, and Khoi Nguyen. Diffusion-based synthetic data generation for pixel-level semantic segmentation. *Advances in Neural Information Processing Systems*, 36:1–21, 2023.
- [220] Hongyang Gao and Shuiwang Ji. Graph u-nets. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 44(9):4948–4960, 2022.
- [221] Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. Convolutional neural networks on graphs with fast localized spectral filtering. In *Proceedings of the 30th International Conference on Neural Information Processing Systems*, NIPS’16, page 3844–3852, Red Hook, NY, USA, 2016. Curran Associates Inc.
- [222] Federico Monti, Davide Boscaini, Jonathan Masci, Emanuele Rodolà, Jan Svoboda, and Michael M. Bronstein. Geometric deep learning on graphs and manifolds using mixture model cnns. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 5425–5434, 2017.
- [223] Muhan Zhang and Yixin Chen. Link prediction based on graph neural networks. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, NIPS’18, page 5171–5181, Red Hook, NY, USA, 2018. Curran Associates Inc.
- [224] O. Duranthon and Lenka Zdeborov’a. Optimal inference in contextual stochastic block models. *Trans. Mach. Learn. Res.*, 2024, 2023.
- [225] Nikhil Mehta, Lawrence Carin, and Piyush Rai. Stochastic blockmodels meet graph neural networks. *ArXiv*, abs/1905.05738, 2019.
- [226] Rémi Boutin, Pierre Latouche, and Charles Bouveyron. The deep latent position block model for the block clustering and latent representation of networks, 2024.
- [227] Celia Rubio-Madrigal, Adarsh Jamadandi, and Rebekka Burkholz. GNNs getting comfy: Community and feature similarity guided rewiring. In *The Thirteenth International Conference on Learning Representations*, 2025.
- [228] Xiaoqi Wang and Han-Wei Shen. Gnninterpreter: A probabilistic generative model-level explanation for graph neural networks. In *International Conference on Learning Representations (ICLR)*, 2023.

- [229] KiJung Yoon, Renjie Liao, Yuwen Xiong, Lisa Zhang, Ethan Fetaya, Raquel Urtasun, Richard Zemel, and Xaq Pitkow. Inference in probabilistic graphical models by graph neural networks. In *2019 53rd Asilomar Conference on Signals, Systems, and Computers*, pages 868–875, 2019.
- [230] Hoang NT and Takanori Maehara. Revisiting graph neural networks: All we have is low-pass filters, 2019.
- [231] Uri Alon and Eran Yahav. On the bottleneck of graph neural networks and its practical implications. In *International Conference on Learning Representations (ICLR)*, 2021.
- [232] Anton Tsitsulin, John Palowitch, Bryan Perozzi, and Emmanuel Müller. Graph clustering with graph neural networks. *Journal of Machine Learning Research*, 24(127):1–21, 2023.
- [233] Aurelien Decelle, Florent Krzakala, Cristopher Moore, and Lenka Zdeborová. Asymptotic analysis of the stochastic block model for modular networks and its algorithmic applications. *Physical Review E—Statistical, Nonlinear, and Soft Matter Physics*, 84(6):066106, 2011.
- [234] Nicolò Ruggeri, Martina Contisciani, Federico Battiston, and Caterina De Bacco. Community detection in large hypergraphs. *Science Advances*, 9(28):eadg9159, 2023.
- [235] John Hood, Caterina De Bacco, and Aaron Schein. Broad spectrum structure discovery in large-scale higher-order networks, 2025.
- [236] Junfu Wang, Yuanfang Guo, Liang Yang, and Yunhong Wang. Understanding heterophily for graph neural networks. In Ruslan Salakhutdinov, Zico Kolter, Katherine Heller, Adrian Weller, Nuria Oliver, Jonathan Scarlett, and Felix Berkenkamp, editors, *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pages 50489–50529. PMLR, 21–27 Jul 2024.
- [237] Jiong Zhu, Yujun Yan, Lingxiao Zhao, Mark Heimann, Leman Akoglu, and Danai Koutra. Beyond homophily in graph neural networks: current limitations and effective designs. In *Proceedings of the 34th International Conference on Neural Information Processing Systems, NIPS ’20*, Red Hook, NY, USA, 2020. Curran Associates Inc.
- [238] Xiang Li, Renyu Zhu, Yao Cheng, Caihua Shan, Siqiang Luo, Dongsheng Li, and Weining Qian. Finding global homophily in graph neural networks when meeting heterophily. In *International Conference on Machine Learning*, pages 13242–13256. PMLR, 2022.

- [239] Jiong Zhu, Gaotang Li, Yao-An Yang, Jing Zhu, Xuehao Cui, and Danai Koutra. On the impact of feature heterophily on link prediction with graph neural networks. *arXiv preprint arXiv:2409.17475*, 2024.
- [240] Tiago P Peixoto. Efficient monte carlo and greedy heuristic for the inference of stochastic block models. *Physical Review E*, 89(1):012804, 2014.
- [241] Linlin Chao, Jianshan He, Taifeng Wang, and Wei Chu. Pairre: Knowledge graph embeddings via paired relation vectors. In *Annual Meeting of the Association for Computational Linguistics*, 2020.
- [242] Xinyu Wang, Yong Jiang, Nguyen Bach, Tao Wang, Zhongqiang Huang, Fei Huang, and Kewei Tu. Automated concatenation of embeddings for structured prediction. In *Annual Meeting of the Association for Computational Linguistics*, 2020.
- [243] J. MacQueen. Some methods for classification and analysis of multivariate observations. In *Proceedings of the 5th Berkeley Symposium on Mathematical Statistics and Probability*, pages 281–297, 1967.
- [244] S. Lloyd. Least squares quantization in pcm. *IEEE Transactions on Information Theory*, 28(2):129–137, 1982.
- [245] Emanuele Rossi, Bertrand Charpentier, Francesco Di Giovanni, Fabrizio Frasca, Stephan Günnemann, and Michael M. Bronstein. Edge directionality improves learning on heterophilic graphs. *ArXiv*, abs/2305.10498, 2023.
- [246] Michael Schlichtkrull, Thomas N. Kipf, Peter Bloem, Rianne van den Berg, Ivan Titov, and Max Welling. Modeling relational data with graph convolutional networks. In *The Semantic Web: 15th International Conference, ESWC 2018, Heraklion, Crete, Greece, June 3–7, 2018, Proceedings*, page 593–607, Berlin, Heidelberg, 2018. Springer-Verlag.
- [247] Zekun Tong, Yuxuan Liang, Changsheng Sun, David S. Rosenblum, and Andrew Lim. Directed graph convolutional network. *ArXiv*, abs/2004.13970, 2020.
- [248] G. Kollias, Vasileios Kalantzis, Tsuyoshi Id’e, Aurélie C. Lozano, and Naoki Abe. Directed graph auto-encoders. In *AAAI Conference on Artificial Intelligence*, 2022.
- [249] Liyu Gong and Qiang Cheng. Exploiting edge features for graph neural networks. In *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 9203–9211, 2019.
- [250] Matthias Fey and Jan Eric Lenssen. Fast graph representation learning with pytorch geometric. *ArXiv*, abs/1903.02428, 2019.

- [251] Thomas N. Kipf and Max Welling. Variational graph autoencoders. In *Proceedings of the NeurIPS Workshop on Bayesian Deep Learning*, 2016.
- [252] Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks, 2017.
- [253] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2014.
- [254] Zhilin Yang, William W. Cohen, and Ruslan Salakhutdinov. Revisiting semi-supervised learning with graph embeddings. In *Proceedings of the 33rd International Conference on International Conference on Machine Learning - Volume 48*, ICML’16, page 40–48. JMLR.org, 2016.
- [255] Vijay Prakash Dwivedi, Ladislav Rampásek, Mikhail Galkin, Ali Parviz, Guy Wolf, Anh Tuan Luu, and Dominique Beaini. Long range graph benchmark. In *Proceedings of the 36th International Conference on Neural Information Processing Systems*, NIPS ’22, Red Hook, NY, USA, 2022. Curran Associates Inc.
- [256] Pinar Yanardag and S.V.N. Vishwanathan. Deep graph kernels. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD ’15, page 1365–1374, New York, NY, USA, 2015. Association for Computing Machinery.
- [257] Lada A. Adamic and Natalie Glance. The political blogosphere and the 2004 u.s. election: divided they blog. In *Proceedings of the 3rd International Workshop on Link Discovery*, LinkKDD ’05, page 36–43, New York, NY, USA, 2005. Association for Computing Machinery.
- [258] Hao Yin, Austin R. Benson, Jure Leskovec, and David F. Gleich. Local higher-order graph clustering. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD ’17, page 555–564, New York, NY, USA, 2017. Association for Computing Machinery.
- [259] Hongbin Pei, Bingzhen Wei, Kevin Chen-Chuan Chang, Yu Lei, and Bo Yang. Geom-gcn: Geometric graph convolutional networks. *ArXiv*, abs/2002.05287, 2020.
- [260] Pan Li and Jure Leskovec. The expressive power of graph neural networks. *Graph Neural Networks: Foundations, Frontiers, and Applications*, pages 63–98, 2022.
- [261] Federico Errica, Marco Podda, Davide Bacciu, and Alessio Micheli. A fair comparison of graph neural networks for graph classification. *ArXiv*, abs/1912.09893, 2019.

- [262] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? *ArXiv*, abs/1810.00826, 2018.
- [263] Hejie Cui, Zijie Lu, Pan Li, and Carl Yang. On positional and structural node features for graph neural networks on non-attributed graphs. In *Proceedings of the 31st ACM International Conference on Information & Knowledge Management, CIKM '22*, page 3898–3902, New York, NY, USA, 2022. Association for Computing Machinery.
- [264] Victor M. Tenorio, Madeline Navarro, Santiago Segarra, and Antonio G. Marques. Recovering missing node features with local structure-based embeddings. In *ICASSP 2024 - 2024 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 9931–9935, 2024.
- [265] Hibiki Taguchi, Xin Liu, and Tsuyoshi Murata. Graph convolutional networks for graphs containing missing features. *Future Generation Computer Systems*, 117:155–168, 2021.
- [266] Hejie Cui, Wei Dai, Yanqiao Zhu, Xuan Kan, Antonio Aodong Chen Gu, Joshua Lukemire, Liang Zhan, Lifang He, Ying Guo, and Carl Yang. Braingb: A benchmark for brain network analysis with graph neural networks. *IEEE Transactions on Medical Imaging*, 42:493–506, 2022.
- [267] Emanuele Rossi, Henry Kenlay, Maria I Gorinova, Benjamin Paul Chamberlain, Xiaowen Dong, and Michael M Bronstein. On the unreasonable effectiveness of feature propagation in learning on graphs with missing node features. pages 11–1, 2022.
- [268] Andrea Giuseppe Di Francesco, Francesco Caso, Maria Sofia Bucarelli, and Fabrizio Silvestri. Link prediction under heterophily: A physics-inspired graph neural network approach. *arXiv*, 2024.
- [269] Yao Ma, Xiaorui Liu, Neil Shah, and Jiliang Tang. Is homophily a necessity for graph neural networks? *arXiv preprint arXiv:2106.06134*, 2021.
- [270] Song Yunchong, Chenghu Zhou, Xinbing Wang, and Zhouhan Lin. Ordered gnn: Ordering message passing to deal with heterophily and over-smoothing, 02 2023.
- [271] Bastian Epping, Alexandre René, Moritz Helias, and Michael T. Schaub. Graph neural networks do not always oversmooth. In *Proceedings of the 38th International Conference on Neural Information Processing Systems, NIPS '24*, Red Hook, NY, USA, 2025. Curran Associates Inc.

- [272] Fei Tian, Bin Gao, Qing Cui, Enhong Chen, and Tie-Yan Liu. Learning deep representations for graph clustering. In *Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence*, AAAI’14, page 1293–1299. AAAI Press, 2014.
- [273] Chun Wang, Shirui Pan, Ruiqi Hu, Guodong Long, Jing Jiang, and Chengqi Zhang. Attributed graph clustering: a deep attentional embedding approach. In *Proceedings of the 28th International Joint Conference on Artificial Intelligence*, IJCAI’19, page 3670–3676. AAAI Press, 2019.
- [274] Hyesoo Shin and Ki Yong Lee. Deep k-means clustering based on graph neural networks: Leveraging cohesion and separation in graph nodes. In *2024 IEEE International Conference on Big Data and Smart Computing (BigComp)*, pages 190–193, 2024.
- [275] Martina Contisciani, Marius Hobbhahn, Eleanor A Power, Philipp Hennig, and Caterina De Bacco. Flexible inference in heterogeneous and attributed multilayer networks. *PNAS Nexus*, 4(1):pgaf005, 01 2025.
- [276] Xiyuan Wang and Muhan Zhang. How powerful are spectral graph neural networks. In *International conference on machine learning*, pages 23341–23362. PMLR, 2022.
- [277] Balasubramaniam Srinivasan and Bruno Ribeiro. On the equivalence between positional node embeddings and structural graph representations. *arXiv preprint arXiv:1910.00452*, 2019.
- [278] Shams Forruque Ahmed, Sweetey Angela Kuldeep, Sabiha Jannat Rafa, Javeria Fazal, Mahfara Hoque, Gang Liu, and Amir H. Gandomi. Enhancement of traffic forecasting through graph neural network-based information fusion techniques. *Information Fusion*, 110:102466, 2024.
- [279] Zikai Zhang, Yidong Li, Haifeng Song, and Hairong Dong. Multiple dynamic graph based traffic speed prediction method. *Neurocomputing*, 461:109–117, 2021.
- [280] B. Jie, M. Liu, D. Zhang, and D. Shen. Sub-network kernels for measuring similarity of brain connectivity networks in disease diagnosis. *IEEE Transactions on Image Processing*, 27(5):2340–2353, 2018.
- [281] Lifang He, Kun Chen, Wanwan Xu, Jiayu Zhou, and Fei Wang. Boosted sparse and low-rank tensor regression. In *Neural Information Processing Systems*, 2018.
- [282] Joshua Faskowitz, Richard F. Betzel, and Olaf Sporns. Edges in brain networks: Contributions to models of structure and function. *Network Neuroscience*, 6(1):1–28, 02 2022.