National Cheng Kung University
MS Degree in Intelligent Computing
Master's Thesis
(Draft)

融合資料補全、模型剪枝與知識蒸餾聯合學習之輕量穩健圖神經網路

iPaDGNN: Joint Learning with Imputation, Pruning, and Distillation for Robust Lightweight Graph Neural Networks

學生： 李旻昊　　Student： Man-Ho Li
指導老師： 李政德 博士　　Advisor： Dr. Cheng-Te Li

July 2025

摘要

圖神經網路（Graph Neural Networks, GNNs）於多種圖結構資料分析任務中表現卓越。然而，其在實務應用上仍面臨三大環環相扣的挑戰：資料不完整、高昂的訓練成本，以及過高的推論開銷。儘管現有方法各自處理單一問題，卻未能提供一個整體的解決方案。

為解決此問題，本研究提出 **iPaD**（Joint Learning with **I**mputation, **P**runing, **A**nd **D**istillation for Robust Lightweight Graph Neural Networks），一個建立在「**協同優化**」核心原則之上的統一框架，旨在對 GNN 的完整流程進行聯合優化。本框架透過一套連貫的流程來實現此設計哲學：首先，以高效的特徵補全技術確保資料的完整性；接著，採用一創新的代理剪枝策略，打造出一個輕量且強健的 GNN 教師模型；最終，再將其知識蒸餾至一個更高效的 MLP 學生模型中，以達成快速推論的目的。

大量的實驗證明，iPaD 不僅能在高達 99.5% 特徵缺失的資料集上保持高準確性，其**推論速度更比基準 GNN 教師模型提升超過五倍**。本研究證明，此一整體性的設計方法，為建構能夠實際應用於真實世界、兼具魯棒性、高效率與可部署性的 GNN，提供了一條實用且強而有力的途徑。

關鍵字: 圖神經網路, 資料補全, 模型剪枝, 知識蒸餾, 協同學習

Abstract

Graph Neural Networks (GNNs) have demonstrated remarkable performance in various graph-structured data analysis tasks. However, their practical deployment is severely hindered by a trio of interconnected challenges: incomplete data, high training costs, and prohibitive inference overhead. While existing methods tackle these issues in isolation, they fail to provide a holistic solution.

To address this gap, we introduce **iPaD** (Joint Learning with **I**mputation, **P**runing, **A**nd **D**istillation for Robust Lightweight Graph Neural Networks), a unified framework built on the core principle of **synergistic optimization** across the entire GNN pipeline. Our framework actualizes this philosophy by first ensuring data integrity with an efficient feature imputation technique. It then employs a novel proxy-based pruning strategy to create a lightweight yet robust GNN teacher, whose knowledge is subsequently distilled into an even more efficient MLP student for fast inference.

Extensive experiments show that iPaD not only maintains high accuracy on datasets with up to 99.5% of features missing but also achieves over a **5x reduction in inference time** compared to its baseline GNN teacher. Our work demonstrates that this holistic approach provides a practical and powerful pathway for building robust, efficient, and deployable GNNs for real-world applications.

Keyword: Graph Neural Network, Data Imputation, Model Pruning, Knowledge Distillation, Collaborative Learning

Table of Contents / 目錄

List of Tables / 表格

List of Figures / 圖片

Chapter 1

Introduction

Graph Neural Networks (GNNs) have emerged as a fundamental paradigm for learning from graph-structured data [2, 3, 4], demonstrating state-of-the-art performance across diverse domains. By jointly leveraging both the topological structure of graphs and rich node-level features, GNNs are capable of capturing complex relational dependencies that are often inaccessible to traditional neural architectures [2, 3, 4]. Consider, for instance, a large-scale recommender system: GNNs can effectively model the intricate relationships between users and items to provide personalized recommendations. [5, 6]

However, deploying such powerful models in real-world scenarios is severely hindered by challenges. In the context of recommender systems, these challenges manifest as follows [5, 7, 8]: (1) many users have sparse interaction histories, leading to **incomplete data**; (2) training a GNN on billions of user-item interactions incurs **high training costs**; and (3) generating recommendations in real-time demands **low inference latency**, which often exceeds the capabilities of large GNNs. This thesis introduces a framework designed to holistically address these practical yet critical challenges.

1.1 Motivation

As the adoption of GNNs in high-stakes applications like recommender systems and social network analysis accelerates, the demand for robust and scalable graph learning becomes increasingly critical. In these practical settings, multiple challenges—including missing node features, incomplete graph structures, and limited computational resources—often co-occur and interact unpredictably [9, 10, 11].

Existing research often addresses these challenges separately, developing targeted solutions for either missing data, training efficiency, or inference cost independently. This siloed strategy, however, leads to a significant practical gap. A solution that excels at data imputation may still be too computationally expensive to train or deploy. Conversely, an efficient, pruned model might fail catastrophically when faced with the noisy and incomplete data typical of real-world environments. Consequently, a unified framework that addresses these challenges **simultaneously** is not merely a convenience, but a necessity for robust, practical

deployment. To fully appreciate the need for this integrated approach, we first dissect the distinct yet interrelated challenges limiting the practical use of GNNs.

1.2 Challenges

Despite recent advances in GNN research, several persistent challenges continue to impede the reliable and efficient deployment of GNNs in real-world scenarios:

1.2.1 Incomplete Data

Real-world graphs are often incomplete, containing absent node features or missing edges. In our recommender system example, this corresponds to new users with no purchase history. This missing information fundamentally disrupts the message-passing mechanism of GNNs, often leading to suboptimal representations and substantially degraded predictive performance [12, 13]. This data incompleteness not only impacts the final accuracy but also complicates the training of deeper, more expressive models.

1.2.2 High Training Cost

The recursive neighborhood aggregation and large parameter space inherent to many powerful GNNs cause training costs to scale rapidly with graph size and model depth [4, 14, 15]. Training a GNN on a graph with millions of users and items, for example, demands significant computational and memory resources, making it a costly and time-consuming endeavor, especially in resource-constrained settings. This high training cost is further exacerbated when models need to be large and deep to handle noisy, incomplete data. Ultimately, even if a powerful model can be trained, its practical utility is limited by its inference efficiency.

1.2.3 Inefficient Inference

At inference time, conventional GNNs require multi-hop neighborhood aggregation for each prediction. To recommend a single item to a user, the model might need to access information from thousands of neighboring nodes, introducing considerable computational overhead [5, 16, 17]. This inefficiency poses a significant obstacle for deployment in latency-sensitive applications like real-time recommendation. This often drives the use of smaller, simpler models, which in turn are more vulnerable to the data incompleteness issues discussed earlier, thus creating a vicious cycle of compromises.

1.2.4 Lack of Unified Solutions

These challenges reveal that most existing solutions focus on addressing individual issues in

isolation. There is a clear lack of a holistic solution capable of addressing the full spectrum of real-world constraints from data imperfection to deployment efficiency. This gap underscores the need for the unified and modular framework proposed in this thesis.

1.3 Research Goal

In direct response to the challenges of data incompleteness, high training costs, and inference inefficiency, the primary objective of this research is to design and develop a unified, modular framework. Our central hypothesis is that a framework **synergistically integrating** robust imputation, pre-training pruning, and architecture-aligned distillation could overcome these limitations in a holistic way.

Specifically, this thesis aims to develop **iPaD** (Joint Learning with **I**mputation, **P**runing, **A**nd **D**istillation for Robust Lightweight Graph Neural Networks), a framework that provides a comprehensive, end-to-end solution by:

1. Developing a feature imputation method **Approximate Pseudo-Confidence Feature Imputation (APCFI)** that is both robust to severe sparsity and computationally efficient enough for large-scale graphs.
2. Designing a novel proxy-based pruning strategy **Mirror Projection Pruning (MPP)** that reduces model complexity and accelerates training before the most expensive steps, by leveraging the isomorphism between GNNs and MLPs.
3. Creating a reliable knowledge distillation mechanism **Mirror Projection Knowledge-inspired Reliable Distillation (MP-KRD)** that efficiently transfers knowledge from a powerful GNN teacher to a lightweight, fast-inference MLP student, thus completing the pipeline from robust training to efficient deployment.

Through this unified approach, we aim to enable the practical deployment of GNNs across a wide range of real-world applications, ensuring strong performance even in challenging and imperfect environments.

1.4 Contributions

This thesis makes the following key contributions:

1. **A Unified, Synergistic Framework:** We introduce **iPaD**, a novel framework that systematically and synergistically addresses the three principal challenges of real-world GNN deployment. Its modules are designed to be complementary, where the output of one stage (e.g., the pruned student model from MPP) serves as the ideal input for the next (e.g., the distillation process in MP-KRD).

2. **A Highly Efficient Imputation Method (APCFI):** We develop **APCFI**, a feature imputation method that achieves comparable accuracy to state-of-the-art methods but with a significant **efficiency breakthrough**, reducing computation time by over 200x on benchmark datasets and making robust imputation practical for large graphs.
3. **A Novel Proxy Pruning Paradigm (MPP):** We propose **MPP**, a pre-training pruning strategy that introduces a new paradigm of proxy-based pruning. By projecting a GNN to an isomorphic MLP, it decouples the complex GNN pruning task from training and effectively leverages mature MLP pruning techniques, significantly reducing computational overhead.
4. **An End-to-End Distillation Pipeline (MP-KRD):** We design **MP-KRD**, an architecture-aligned knowledge distillation mechanism that completes the optimization pipeline. It uniquely uses the pruned MLP generated by MPP as a student, enabling highly efficient inference while preserving the relational reasoning capabilities of the original GNN teacher.
5. **Empirical Validation of Robust Architectures:** We are the first to systematically demonstrate that simple architectural modifications in a **tunedGNN**, such as adding LayerNorm and Dropout, not only improve accuracy on complete data but also serve as a critically effective backbone for enhancing GNN robustness against severe feature missingness.

1.5 Thesis Organization

This thesis is organized as follows:

- **Chapter 2: Related Work** reviews prior research on GNNs with incomplete data, feature imputation, model pruning, and knowledge distillation.
- **Chapter 3: Problem Statement** formally defines the key challenges addressed in this work and introduces the necessary notations and settings.
- **Chapter 4: Methodology** details the proposed iPaD framework and its core modules: APCFI, MPP, tunedGNN, and MP-KRD.
- **Chapter 5: Experiments** presents the experimental setup, benchmark datasets, baseline methods, and evaluation metrics, followed by a comprehensive analysis of results.
- **Chapter 6: Conclusion** summarizes the main findings of this study and discusses potential directions for future research on robust and efficient graph neural networks.

Chapter 2

Related Work

This chapter reviews prior research in areas crucial to our work. Methods for handling incomplete data in GNNs, particularly feature imputation, are first reviewed. The subsequent sections discuss the landscape of GNN model pruning and knowledge distillation. For each area, the limitations of existing approaches are analyzed to precisely situate the contributions of the proposed iPaD framework.

2.1 Feature Imputation in Graph Neural Networks

Feature imputation is a critical component for enhancing the robustness of Graph Neural Networks (GNNs) in the presence of missing node features—a common challenge in real-world graph-structured data [18, 19, 20, 21]. Existing approaches can be broadly classified into heuristic methods, learning-based models, and feature propagation techniques.

Heuristic methods, such as Zero, Mean, or Label Propagation (LP) Imputation, are computationally efficient but often fail to capture the complex feature distributions present in graphs [22], leading to suboptimal performance. Learning-based approaches, including GCNMF and PaGNN, employ deep neural network architectures to model and reconstruct missing features [12, 20]. Despite their theoretical expressiveness, these models can be limited by substantial computational overhead and may not outperform simpler methods in high-missingness regimes.

More recently, non-learning-based feature propagation methods have gained traction. Feature Propagation (FP) exploits the underlying graph structure to propagate known features, offering a strong balance of performance and efficiency [19]. Pseudo-Confidence Feature Imputation (PCFI) further advances this by introducing channel-wise pseudo-confidence to guide a two-stage diffusion process, achieving state-of-the-art imputation quality [21]. However, this high performance comes at a significant cost: PCFI's channel-by-channel calculation leads to extremely high computational complexity, hindering its scalability to large graphs. This reveals a critical trade-off between imputation performance and computational efficiency in the current literature.

To address this efficiency-accuracy trade-off, **APCFI** is introduced to retain the high imputation quality of confidence-aware methods such as PCFI, while drastically reducing compu-

tational burden via Approximate Shortest Distance Estimation (ASDE) and Dynamic Joint Channel Diffusion (DJCD).

Table 2.1 provides a clear comparison of these imputation strategies.

Method	**Imputation Performance**	**Computational Cost**	**Key Idea**	**Key Limitation**
Zero/Mean	Low	Very Low	Constant Filling	Unable to capture characteristic distribution
LP [22]	Low/Medium	Very Low	Feature Replacement	Ignore existing features
PaGNN [12]	Medium	Very High	Learnable Method	Sensitive to noise, poor scalability
GCNMF [20]	Medium	Very High	Matrix Factorization, Learnable Method	Sensitive to noise, poor scalability
FP [19]	Medium	Low	Feature Propagtion-base	Insensitive to differences between features
PCFI [21]	High	High	Confidence-aware Feature Propagtion	Channel-by-channel calculation, poor scalability
APCFI(Ours)	**High**	**Low**	Approximate Confidence & Joint Channel Diffusion	Not applicable to heterogeneous graphs

Table 2.1: Comparison of representative feature imputation methods.

2.2 Model Pruning in Graph Neural Networks

As GNNs scale to handle increasingly large and complex graphs, model pruning has emerged as a crucial strategy to reduce computational costs during both training and inference. Existing GNN pruning paradigms include channel pruning (e.g., GCNP [23]), hardware-aware optimization (e.g., PruneGNN [24]), and methods that simultaneously prune graph structures and model weights during training (e.g., UGS [25], ICPG [26]).

However, these methods often suffer from two major limitations. First, they typically entangle the pruning process with the GNN training pipeline, introducing significant computational overhead and complex, model-specific algorithmic designs that can substantially **increase the total training cost**. Second, they are often proposed as specific, monolithic algorithms,

lacking the **scalability and flexibility** to incorporate a wide array of mature pruning techniques from other domains, such as the MLP literature.

To overcome these challenges of high overhead and low flexibility, we introduce **MPP** (Mirror Projection Pruning). Instead of being another monolithic algorithm, MPP proposes a new **proxy-based pruning paradigm**. Its core idea is to decouple pruning from GNN training by projecting the GNN to an isomorphic MLP, where mature, off-the-shelf pruning algorithms can be applied efficiently *before* the main training commences. The advantages of this paradigm are summarized in Table 2.2.

Method	**Total Training Cost**	**Scalability**	**Key Idea**
GCNP [23]	↑	✗	Using a novel LASSO regression formulation to prune model channels
PruneGNN [24]	↑	✗	Introduces novel SIMD-aware kernels and accelerate structured GNN pruning on parallel hardware
UGS [25]	↑	✗	Prune graphs and weights simultaneously during training for specific ensemble algorithms
ICPG [26]	↑	✗	Iterative co-pruning during training for specific ensemble algorithms
MPP(Ours)	↓	✔	A GNN proxy pruning framework that off-the-shelf MLP-compatible algorithms

Table 2.2: Comparison of representative GNN pruning methods.

2.3 Knowledge Distillation for Graph Models

Knowledge distillation (KD) is a pivotal technique for compressing GNNs, enabling the deployment of lightweight yet high-performing models. In the context of graph data, KD approaches can be broadly categorized into GNN-to-GNN and GNN-to-MLP distillation.

GNN-to-GNN methods, such as LSP [27] and TinyGNN [28], focus on transferring knowledge from a large GNN teacher to a smaller GNN student. While effective, they do not eliminate the graph dependency at inference time. To achieve maximum inference speed, GNN-to-MLP distillation has emerged as a promising direction. Methods like GLNN [29] show

that an MLP student, once distilled, can achieve competitive performance without access to the graph structure. More advanced techniques like KRD [30] introduce reliability-aware curriculum learning to handle potentially noisy knowledge from the teacher.

Despite these advances, existing GNN-to-MLP methods exhibit several gaps. First, the student MLP is typically an **arbitrary MLP**, lacking any intrinsic structural alignment with the GNN teacher. Second, the distillation process is designed **independently of any model pruning stage**, missing a key opportunity for synergistic optimization. Finally, while some methods address teacher noise, a more robust, pipeline-aware approach is needed.

Our proposed **MP-KRD** (Mirror Projection Knowledge-inspired Reliable Distillation) is designed specifically to fill these gaps. It creates a powerful synergy by using the **structurally-aligned, isomorphic MLP** generated by our MPP stage as the student. This ensures architectural consistency and maximizes the effectiveness of the knowledge transfer, which is further enhanced by a reliability-aware curriculum. The unique characteristics of MP-KRD are summarized in Table 2.3.

Method	**Student Source**	**Feature Alignment**	**Handles Noisy Teacher**	**Key Idea**
LSP [27]	Small GNN	✗	✗	Transfer the topological structure
GNN-SD [31]	Teacher GNN	✔	✗	GNNs Self-Distillation framework
TinyGNN [28]	Small GNN	✗	✗	Distillation with fewer parameter GNNs
GLNN [29]	Arbitrary MLP	✗	✗	Learning predictions from GNNs
KRD [30]	Arbitrary MLP	✗	✔	Reliability-based learning
MP-KRD(Ours)	**Isomorphic MLP**	✔	✔	Structural Alignment + Reliability Course Learning

Table 2.3: Comparison of representative knowledge distillation methods for graph models.

2.4 High-Level Comparison of SOTA GNNs

Beyond the specific areas of imputation and compression, it is also instructive to compare our approach against the broader landscape of recent state-of-the-art (SOTA) GNNs, particularly concerning the trade-off between performance and efficiency.

Table 2.4 provides a qualitative comparison of prominent SOTA models. As the table illustrates, while most recent Transformer-based models like GraphGPS[32] and NAGphormer[33] achieve high predictive performance, they often do so at the cost of low computational efficiency. Our **MP-TunedGNN** (the teacher model in our framework), by integrating the MPP module, stands out by maintaining high performance while significantly boosting efficiency compared to both its "TunedGNN" backbone and other SOTA models. This highlights a key contribution of our work in addressing the prevalent performance-efficiency trade-off in modern GNN architectures.

Method	**Efficiency**	**Robustness**
GraphGPS [32]	Low	-
NAGphormer [33]	Medium	-
Exphormer [34]	Medium	-
GOAT [35]	Medium	-
NodeFormer [36]	Medium	-
SGFormer [37]	Medium	-
Polynormer [38]	Medium	-
TunedGNN [1]	High	✓*

Table 2.4: High-level comparison of recent SOTA GNN models on efficiency, and robustness. The asterisk (*) indicates that robustness is not the primary focus of the original paper but is evaluated in our study.

2.5 Chapter Summary

In this chapter, we have reviewed prior art across feature imputation, model pruning, and knowledge distillation, identifying the key limitations of existing methods in each domain. To synthesize these findings and crystallize the motivation for our work, we present a holistic comparison in Table 2.5. This table evaluates a wide spectrum of methods against the three core challenges identified in this thesis: handling data **Completeness**, achieving high **Training Efficiency**, and ensuring model **Scalability** for inference.

The table clearly illustrates the"siloed" nature of existing research. Imputation methods like PCFI excel at "Completeness" but do not address efficiency. High-performance SOTA models like GraphGPS [32] have low "Training Efficiency" and "Scalability". Distillation methods like KRD offer "Scalability" but do not handle data "Completeness".

Method	Completeness	Training Efficiency	Scalability
PaGNN [12]	✔	-	-
GCNMF [20]	✔	-	-
FP [19]	✔	-	-
PCFI [21]	✔	-	-
GraphGPS [32]	-	✔	-
NAGphormer [33]	-	✔	-
Exphormer [34]	-	✔	-
GOAT [35]	-	✔	-
NodeFormer [36]	-	✔	-
SGFormer [37]	-	✔	-
Polynormer [38]	-	✔	-
TunedGNN [1]	✔*	✔	-
LSP [27]	-	-	✔
GNN-SD [31]	-	-	-
TinyGNN [28]	-	-	✔
GLNN [29]	-	-	✔
KRD [30]	-	-	✔
iPaD(Ours)	✔	✔	✔

Table 2.5: Holistic comparison of representative methods across the core challenges of Completeness, Training Efficiency, and Scalability. Our proposed iPaD is the only framework designed to excel in all three aspects.

Chapter 3

Problem Statement

3.1 Problem Background

Graph Neural Networks (GNNs) have achieved remarkable success in a wide range of real-world applications, such as social network analysis, recommender systems, and bioinformatics [16, 39, 40]. However, their practical performance is highly contingent on the completeness of the input graph data [41, 42]. In realistic scenarios, graphs are frequently afflicted by missing node features or absent edges, stemming from factors like privacy constraints, sensor failures, or data collection limitations [43, 44]. Such missing information fundamentally disrupts the message-passing mechanisms central to GNN architectures, often resulting in suboptimal node representations and diminished performance on downstream tasks.

Missingness in graph data typically manifests in two primary forms:

- Uniform missingness refers to randomly distributed missing features throughout the graph, often caused by unpredictable device malfunctions or sporadic data loss.
- Structural missingness arises when missing data is concentrated within specific subgraphs or clusters of nodes, commonly due to systematic issues such as privacy restrictions or selective sampling policies.

In addition to data incompleteness, the scaling of GNNs to large and complex graphs introduces substantial challenges in terms of training cost (computational resources and time) and inference cost (the need for extensive neighborhood aggregation at test time) [45, 46]. Existing approaches tend to focus on addressing a single aspect—such as feature imputation, model pruning, or architectural tuning—without providing a unified and modular solution that jointly addresses the intertwined challenges of robustness, efficiency, and scalability under incomplete data.

To overcome these practical challenges, a comprehensive framework capable of robust feature imputation, resource-efficient training, and cost-effective inference is essential for deploying GNNs in real-world settings characterized by high levels of missingness and limited computational resources [47].

3.2 Notations

Notation	Description
$\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathbf{A})$	An undirected graph, where $\mathcal{V}$ is the set of nodes, $\mathcal{E}$ is the set of edges, and $\mathbf{A}$ is the adjacency matrix.
$\mathcal{V} = \{v_i\}_{i=1}^{N}$	The set of N nodes in the graph.
$\mathbf{A} \in \{0,1\}^{N \times N}$	The adjacency matrix; $\mathbf{A}_{ij} = 1$ if there is an edge between v_i and v_j, otherwise 0.
$\mathbf{X} = [x_{i,d}] \in \mathbb{R}^{N \times F}$	Node feature matrix; $x_{i,d}$ is the d-th feature of node v_i.
$\mathcal{N}(v_i)$	The set of neighbors of node v_i.
$\mathbf{M} \in \{0,1\}^{N \times F}$	Mask matrix: 1 for observed features, 0 for missing.
$V_k^{(d)}$	Set of nodes with known d-th feature (source nodes).
$V_u^{(d)}$	Set of nodes with unknown d-th feature (missing nodes).
$x^{(d)} = \begin{bmatrix} x_k^{(d)} \\ x_u^{(d)} \end{bmatrix}$	Graph signal for d-th channel, partitioned into known and unknown.
$\mathbf{A}^{(d)} = \begin{bmatrix} \mathbf{A}_{kk}^{(d)} & \mathbf{A}_{ku}^{(d)} \\ \mathbf{A}_{uk}^{(d)} & \mathbf{A}_{uu}^{(d)} \end{bmatrix}$	Adjacency matrix reordered for known/unknown nodes for channel d.
$\hat{\mathbf{X}} = [\hat{x}_{i,d}]$	The recovered (imputed) feature matrix.

Table 3.1: Summary of main notations used throughout the paper. This table lists the symbols, definitions, and descriptions for graph structure, feature matrices, and related variables.

3.3 Formal Statement

Given a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathbf{A}, \mathbf{X})$ with node feature matrix $\mathbf{X} \in \mathbb{R}^{N \times F}$ and a missingness mask $\mathbf{M} \in \{0,1\}^{N \times F}$, the objective of this work is to develop a unified framework that addresses the following intertwined challenges in graph representation learning under incomplete data:

1. Robust Feature Imputation:

   Given that a subset of features $x_{i,d}$ is missing (i.e., $\mathbf{M}_{i,d} = 0$), the goal is to recover the missing values $\hat{x}_{i,d}$ such that the imputed feature matrix $\hat{\mathbf{X}}$ maximizes the downstream task performance (e.g., node classification accuracy) under varying missingness patterns (uniform/structural).

2. Resource-Efficient Training:

   Let $C_{\text{train}}(f)$ denote the computational cost (e.g. Time or FLOPs) of training a GNN-based model f on $\mathcal{G}$. The challenge is to minimize $C_{\text{train}}(f)$ without sacrificing the performance on incomplete graphs, ideally through effective pruning or architectural optimization.

3. Cost-Effective Inference:

   Define $C_{\text{infer}}(f)$ as the inference cost for deploying f on new, potentially incomplete graphs. The target is to minimize $C_{\text{infer}}(f)$ (e.g., reducing message passing or memory overhead) while ensuring high accuracy and reliability, possibly by leveraging knowledge distillation or graph-to-MLP transformation [29, 30, 48].

The joint optimization problem underlying our framework can be formalized as follows:

$$
\begin{aligned}
&\max_{f,\hat{\mathbf{X}},\hat{\mathcal{A}}} \quad \text{TaskAcc}\big(f(\hat{\mathbf{X}},\hat{\mathcal{A}})\big) \\
&\min_{f} \quad C_{\text{train}}(f) \\
&\min_{f} \quad C_{\text{infer}}(f) \\
&\text{s.t.} \begin{cases} \mathbf{M}_{i,d}=0 \implies \hat{x}_{i,d} \text{ is imputed} \\ \text{MissingRate}(\mathbf{M}) \le \rho_X \\ \text{MissingRate}(\mathbf{A}) \le \rho_A \end{cases}
\end{aligned}
$$

This joint optimization objective seeks to maximize the downstream task accuracy, $\text{TaskAcc}\big(f(\hat{\mathbf{X}},\hat{\mathcal{A}})\big)$, by simultaneously optimizing the model f, the imputed feature matrix $\hat{\mathbf{X}}$, and the imputed adjacency matrix $\hat{\mathcal{A}}$. The framework also aims to minimize both the training cost $C_{\text{train}}(f)$ and the inference cost $C_{\text{infer}}(f)$, ensuring efficiency during both model development and deployment.

Subject to the following constraints: - Every missing feature entry (i,d), indicated by $\mathbf{M}_{i,d}=0$, must be properly imputed in $\hat{x}_{i,d}$. - The missing rate of the feature matrix, $\text{MissingRate}(\mathbf{M})$, must not exceed the predefined threshold ρ_X. - The missing rate of the adjacency matrix, $\text{MissingRate}(\mathcal{A})$, must not exceed the threshold ρ_A.

Here, $\mathbf{M}$ denotes the feature mask indicating missing entries, while $\mathcal{A}$ and $\hat{\mathcal{A}}$ represent the original and imputed adjacency matrices, respectively. The parameters ρ_X and ρ_A define the maximum tolerable missing rates for features and edges, guaranteeing robust model performance under severe data incompleteness. In our experiments, we set $\rho_X = 99.5\%$ for

feature imputation in APCFI, and evaluate iPaD under edge missing rates up to $\rho_A = 90\%$, demonstrating the framework's strong resilience to extremely high levels of missing data.

3.4 Summary

The central research question addressed in this work is as follows: **How can a modular and unified learning framework be designed to achieve robust prediction performance under severe data incompleteness, while simultaneously reducing training and inference costs for practical deployment?**

This framework tackles the joint objective by seamlessly integrating feature imputation, pruning, architectural tuning, and knowledge distillation into a single pipeline. Such integration not only enhances robustness and efficiency, but also provides a principled foundation for the subsequent methodological developments presented in this paper.

Chapter 4

Methodology

This chapter details the architecture and workflow of our proposed **iPaD** framework. As illustrated in Figure 4.1, iPaD is an end-to-end pipeline designed to address the challenges of incomplete data, high training costs, and inefficient inference in a unified and synergistic manner. The entire framework can be understood as a three-stage process:
(1) Robust Data Imputation
(2) Parallel Teacher-Student Preparation
(3) Teacher Training
(4) Architecture-Aligned Knowledge Reliable Distillation.
The subsequent sections will delve into the technical details of each core module.

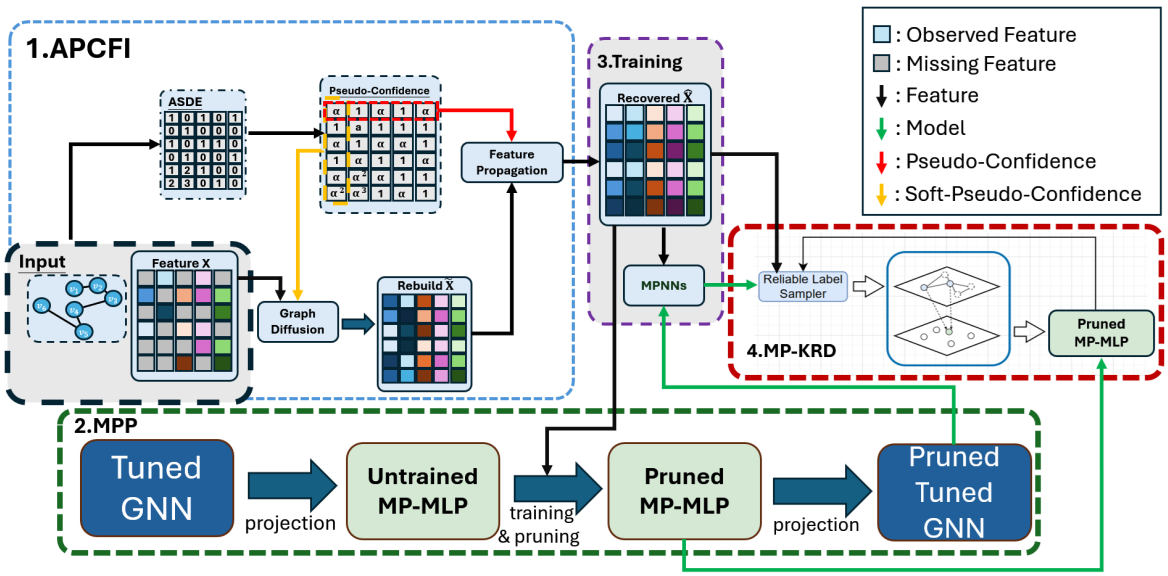


Figure 4.1: The overall architecture of the iPaD framework. The process begins with imputing the incomplete feature matrix via **APCFI**. In parallel, a powerful GNN teacher is trained on the recovered data, while the **MPP** module generates a pruned, lightweight MLP student. Finally, the **MP-KRD** module distills knowledge from the teacher to the student, producing a final model that is both robust and efficient for inference.

4.1 Framework Overview

The workflow of our proposed iPaD framework, as depicted in Figure 4.1, is a systematic

pipeline that transforms a graph with incomplete features into a lightweight, robust, and fast inference model. The process unfolds across three distinct yet interconnected stages:

1. **Stage 1: Robust Data Imputation with APCFI.** The process begins with an input graph whose feature matrix, $\mathbf{X}$, contains missing values. To ensure a high-quality foundation for subsequent learning, the graph is first passed through our **APCFI** (Approximate Pseudo-Confidence Feature Imputation) module. APCFI leverages graph diffusion and an **ASDE** to generate pseudo-confidence scores, which guide a feature propagation process. The output of this stage is a fully recovered feature matrix, $\hat{\mathbf{X}}$, where missing entries have been robustly imputed.

2. **Stage 2: Synergistic Teacher and Student Generation via MPP.** The second stage masterfully prepares both the teacher and student models in a sequential, highly efficient process driven by our **MPP** module. The workflow is as follows:

   (a) **Proxy Pre-training and Pruning:**
   We begin with our powerful GNN architecture (e.g., **tunedGNN**) and project it to its isomorphic MP-MLP. This proxy MLP is then pre-trained and pruned on the target dataset. This efficient step determines the optimal sparse structure and directly yields our lightweight **Student Model** (the "Pruned MP-MLP").

   (b) **Teacher Architecture Definition:**
   The pruned structure from the student model is then inversely projected back to the original GNN architecture. This creates the final, optimized architecture for our teacher: the "Pruned GNN".

   This synergistic process is a core innovation of our framework: the lightweight student's architecture is defined in the same efficient step that determines the optimized architecture for the powerful teacher, all before the main, computationally expensive GNN training commences.

3. **Stage 3: Teacher Training.**
   The "Pruned GNN" architecture, defined in the previous stage, is fully trained on the imputed dataset $(\hat{\mathbf{X}}, \mathcal{G})$ produced by the APCFI module. This dedicated training phase enables the pruned model to achieve strong predictive performance while maintaining high computational efficiency. The resulting model, referred to as the **Teacher Model**, serves as a robust and lightweight source of knowledge for the subsequent distillation process. By leveraging both the optimized sparse architecture and the high-quality imputed features, the teacher model is well-suited to guide the training of the student model in the final stage.

4. **Stage 4: Knowledge Distillation with MP-KRD.** In the final stage, knowledge transfer is orchestrated. The **MP-KRD** module takes the high-performance "Pruned GNN" Teacher and the lightweight "Pruned MP-MLP" Student. It employs a reliability-aware sampling strategy to distill the rich, relational knowledge from the teacher into the student. The final output of the entire iPaD pipeline is this pruned, distilled MLP student —a model that is highly efficient, robust against incomplete data, and ready for fast deployment in real-world applications.

4.2 APCFI: Approximate Pseudo-Confidence Feature Imputation

As the first stage of the iPaD pipeline, the foundational challenge of data incompleteness is addressed. To this end, **APCFI** is proposed as a robust and scalable feature completion strategy, advancing beyond prior art by introducing a more sophisticated, two-stage normalization process to enhance imputation quality. Figure 4.2 illustrates the overall workflow of the APCFI module.

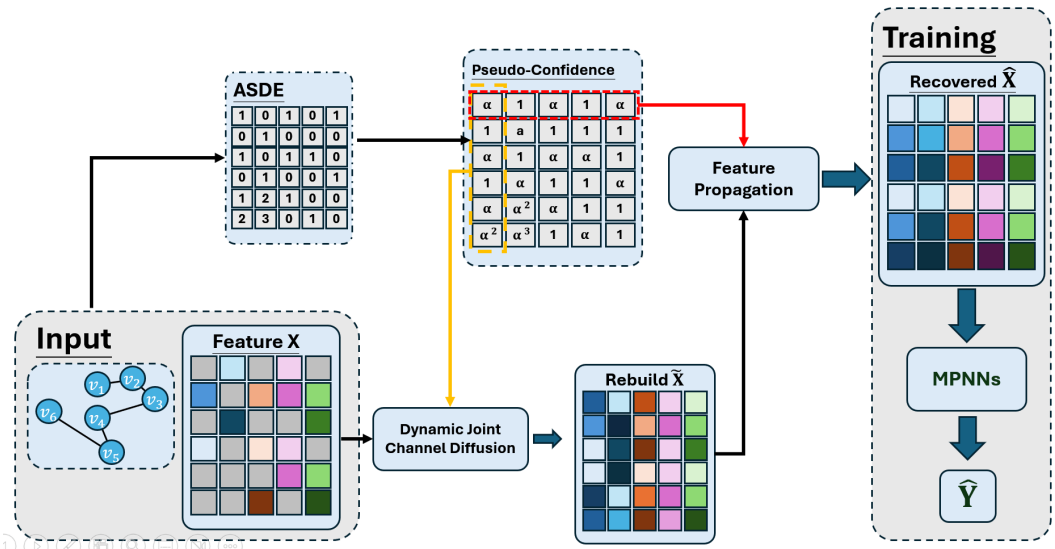


Figure 4.2: The overall workflow of the APCFI module. It takes a graph with an incomplete feature matrix as input and leverages ASDE and Dynamic Joint Channel Diffusion (DJCD) to output a robustly recovered feature matrix for downstream tasks.

4.2.1 Motivation and Core Idea

Previous high-performance imputation methods like PCFI [21] successfully utilize channel-wise confidence to guide feature diffusion. However, this approach faces two key limitations: the calculation of exact shortest paths is computationally expensive, and the raw confidence scores may not optimally reflect the relative importance of nodes.

APCFI is designed to overcome both limitations. Its core philosophy is to improve both the

efficiency of confidence calculation and the quality of the diffusion weights. This is achieved through three key innovations:

1. **Approximate Shortest Distance Estimation (ASDE):** Replaces PCFI's costly k-hop traversal with a highly parallelizable message-passing scheme to efficiently approximate shortest-path distances.
2. **Soft-Pseudo Confidence (Soft-PC):** Introduces a "softmax" layer before the standard row-stochastic normalization to transform raw confidence scores into context-aware, relative reliability weights.
3. **Dynamic Joint Channel Diffusion:** Replaces PCFI's sequential diffusion with a parallel process, reducing complexity from $O(d \times k)$ to $O(k)$.

4.2.2 Technical Workflow

The APCFI workflow consists of three main steps to transform an incomplete feature matrix $\mathbf{X}$ into a recovered matrix $\hat{\mathbf{X}}$.

Step 1: Approximate Shortest Distance Estimation (ASDE)

To efficiently quantify the reliability of feature imputation, the raw pseudo-confidence (PC) is computed for each node-feature pair based on the shortest-path distance (SPD) from a missing-feature node to its nearest observed-feature node in the same channel. The ASDE algorithm, illustrated in Figure 4.3, iteratively propagates distance information through the graph, enabling rapid approximation of these SPD values.

Formally, the shortest-path distance for node v_i and feature channel d is defined as:

$$\text{SPD}_{i,S_d} = \begin{cases} 1, & \text{if } x_{i,d} \text{ is observed} \\ \min\left\{ t \mid \hat{x}_{i,d}^{(t)} \neq 0 \right\}, & \text{if } x_{i,d} \text{ is missing} \end{cases} \tag{4.1}$$

The raw pseudo-confidence is then given by:

$$pc_i^{(d)} = \alpha^{\text{SPD}_{i,S_d}} \tag{4.2}$$

where $\alpha \in (0, 1)$ is a decay factor that controls the influence of distance on confidence.

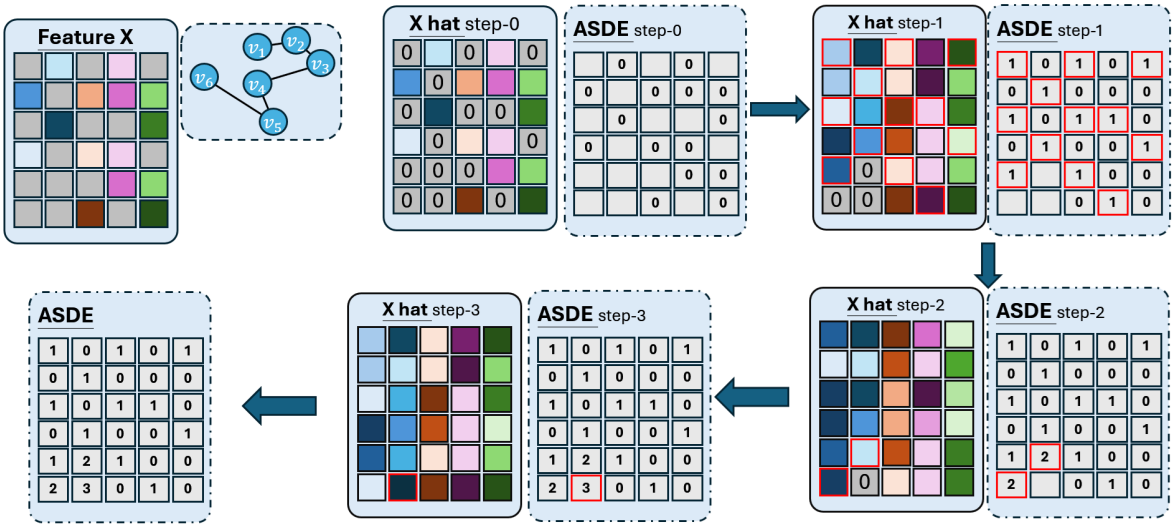


Figure 4.3: Illustration of the Approximate Shortest Distance Estimation (ASDE) process. The algorithm begins with observed features (step-0) and propagates shortest-path information through the graph. Each entry in the ASDE matrix reflects the shortest hop-distance to an observed feature within the same channel.

Step 2: Two-Stage Normalization

This step is the core of our imputation mechanism, involving a two-stage normalization process to generate the final diffusion weights.

Stage 1: Soft-Pseudo Confidence

The raw pseudo-confidence scores $c_i^{(d)}$ are first passed through a "softmax" operation to produce the "soft confidence matrix". The goal is to generate context-aware, relative reliability scores, which is a key innovation over PCFI. The resulting weights $w_{i,d}$ are calculated as:

$$w_{i,d} = \frac{\exp(c_i^{(d)})}{\sum_{j=1}^{N} \exp(c_j^{(d)})} \tag{4.3}$$

To illustrate the effectiveness of this mechanism, consider the following example. Given a raw pseudo-confidence matrix **PC**:

$$\mathbf{PC} = \begin{bmatrix} 0.5 & 0.5 \\ 1.0 & 0.7 \\ 0.25 & 1.0 \end{bmatrix}$$

The resulting soft-pseudo confidence weights **soft_PC** are:

$$\textbf{soft\_PC} = \begin{bmatrix} 0.292 & 0.258 \\ 0.481 & 0.316 \\ 0.227 & 0.426 \end{bmatrix}$$

Notably, even though the first node has the same pseudo-confidence value (0.5) for both feature channels, its final soft-pseudo confidence weights differ (0.292 vs. 0.258). This is because softmax normalization takes into account the entire pseudo-confidence distribution for each channel, allowing identical values to have different relative importance depending on their context. This makes the weighting process more flexible and robust for the subsequent diffusion step.

Stage 2: Row-Stochastic Transition

To ensure a stable and mathematically sound diffusion process, the "soft confidence matrix" (denoted as $\mathbf{W}'$) undergoes a final row-stochastic normalization. This step is crucial for preserving the feature scale during propagation [21]. It is achieved by normalizing each weight by its corresponding row sum:

$$\bar{\mathbf{W}} = \mathbf{D}^{-1}\mathbf{W}' \tag{4.4}$$

where $\mathbf{D}$ is the diagonal degree matrix with $D_{ii} = \sum_j W'_{ij}$. The resulting matrix $\bar{\mathbf{W}}$ is row-stochastic, meaning each row sums to 1.

Step 3: Dynamic Joint Channel Diffusion (DJCD)

In this step, joint diffusion is performed based on the normalized weights, enabling efficient feature propagation across multiple channels.

After obtaining the normalized, row-stochastic weight matrix $\bar{\mathbf{W}}$, the final step involves performing the actual feature diffusion. This iterative process, conducted for a total of K steps, propagates information from observed nodes to missing nodes across the graph.

The update rule for each node v_i and feature channel d at iteration $t+1$ is defined as:

$$\hat{x}_{i,d}^{(t+1)} = \begin{cases} x_{i,d}, & \text{if } (i,d) \text{ is observed} \\ \frac{\sum_{j \in \mathcal{N}(i)} \bar{w}_{ij} \cdot \hat{x}_{j,d}^{(t)}}{\sum_{j \in \mathcal{N}(i)} \bar{w}_{ij}}, & \text{otherwise} \end{cases} \tag{4.5}$$

For any feature that is already **observed**, its value is kept fixed. This is a crucial step to prevent the loss of ground-truth information and mitigate the "over-smoothing" problem, a

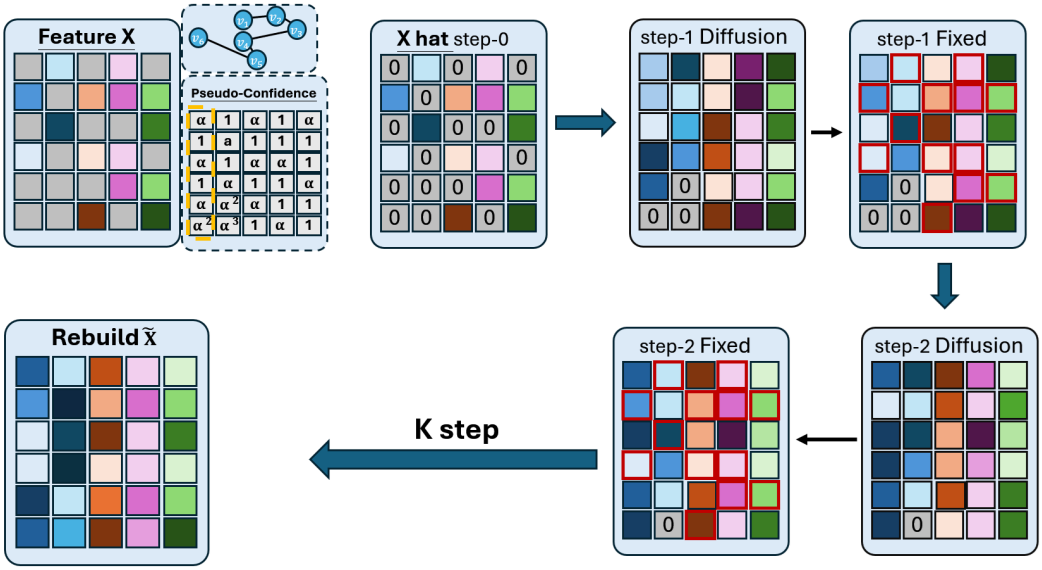


Figure 4.4: The iterative process of Dynamic Joint Channel Diffusion. In each of the K steps, a diffusion operation updates the missing values based on neighbors, followed by a "Fixed" step where the original, observed feature values are reset to prevent oversmoothing and preserve data fidelity.

technique also validated in the Feature Propagation (FP) [19] paper.

For any **missing** feature, its value is updated by computing a **weighted average** of the features of its neighboring nodes from the previous iteration. The weights $\bar{w}_{ij}$ are precisely the ones we derived from our two-stage normalization process, ensuring that more reliable neighbors contribute more significantly to the imputation.

Notably, due to this design, the diffusion process can be conducted jointly and in parallel across all feature channels, which constitutes a key efficiency advantage of APCFI over PCFI. The entire iterative process is illustrated in Figure 4.4.

Step 4: Node-wise Inter-Channel Propagation

As a final refinement, following the primary feature diffusion in the DJCD step, a node-wise inter-channel propagation mechanism is introduced. This step is designed to leverage latent correlations between different feature channels, further fine-tuning the imputed values for each node. The core design is guided by the **Humility Principle**: nodes with lower confidence in a certain feature value should be more receptive to "suggestions" from other highly correlated features.

The process consists of the following computational steps. Let $\hat{\mathbf{X}}_{\text{djcd}}$ denote the feature matrix recovered from the DJCD step, and let $\mathbf{PC} \in \mathbb{R}^{N \times F}$ be the matrix of pseudo-confidence scores, where each entry $pc_{i,d}$ represents the confidence in the d-th feature of node v_i.

1. **Inter-Channel Correlation:** The feature correlation matrix $\mathbf{C} \in \mathbb{R}^{F \times F}$ is computed to capture the global linear relationships between all feature channels across the graph:

$$\mathbf{C} = \text{Corrcoef}(\widetilde{\mathbf{X}}_{\text{djcd}}^{T})$$

2. **Confidence-Weighted Signal:** For each node, an initial "suggestion" signal is generated by taking the deviation of each feature from its global mean, weighted by its pseudo-confidence score. This ensures that more reliable features contribute more strongly to the suggestion signal. Let $\widetilde{\overline{\mathbf{X}}}_{\text{djcd}}$ be the feature mean vector:

$$\mathbf{M}_{\alpha} = \mathbf{PC} \odot (\widetilde{\mathbf{X}}_{\text{djcd}} - \widetilde{\overline{\mathbf{X}}}_{\text{djcd}})$$

   where $\odot$ denotes element-wise multiplication.

3. **Suggestion Aggregation:** The signal $\mathbf{M}_{\alpha}$ is propagated through the correlation matrix $\mathbf{C}$, aggregating weighted suggestions from all other channels and enabling features to "advise" each other based on global correlation:

$$\mathbf{M}_{\text{conf}} = \mathbf{M}_{\alpha} \cdot \mathbf{C}$$

4. **Humility-based Update:** The update term $\Delta\mathbf{X}$ is computed by weighting the aggregated suggestions $\mathbf{M}_{\text{conf}}$ with a "humility" factor $(1 - \mathbf{PC})$ and a hyperparameter β. The humility factor ensures that features with low initial confidence are updated more significantly:

$$\Delta\mathbf{X} = \beta \cdot (1 - \mathbf{PC}) \odot \mathbf{M}_{\text{conf}}$$

   The final imputed feature matrix is obtained by applying this update to the result of the DJCD step:

$$\hat{\mathbf{X}} = \widetilde{\mathbf{X}}_{\text{djcd}} + \Delta\mathbf{X}$$

This two-tiered process, combining broad graph-based diffusion with a fine-grained, correlation-aware adjustment, enables APCFI to produce a more robust and nuanced feature representation, providing a solid foundation for downstream GNN models.

4.2.3 Implementation and Pseudocode

The complete APCFI workflow is detailed in the algorithms below (Algorithms 1 to 4).

In summary, by introducing a sophisticated two-stage normalization process and a highly efficient joint diffusion mechanism, APCFI provides a robust and scalable solution for data

incomplete. The recovered feature matrix, $\hat{\mathbf{X}}$, serves as a high-quality data foundation for subsequent model training. Specifically, the enhanced completeness and reliability of $\hat{\mathbf{X}}$ enable the downstream GNN backbone to better exploit graph structure and node attributes, thereby improving overall learning stability and predictive performance, even in challenging real-world scenarios with severe missingness. The next section introduces the robust GNN backbone, which is designed to fully leverage this improved feature representation.

Algorithm 1 PyTorch-style Pseudo-code of APCFI

```
function apcfi_fill(feature, edge_index)
    mask = feature.isnan()

    Step 1: ASDE and Dynamic Joint Channel Diffusion
    diffusion_x, asde_matrix = diffusion(feature, edge_index, mask)

    Step 2: Node-wise Inter-Channel Propagation
    output = node_propagation(diffusion_x, asde_matrix)

    Return the imputed feature matrix
    return output
```

Algorithm 2 PyTorch-style Pseudo-code of Function `diffusion`

```
function diffusion(feature, edge_index, mask)

    Step 1: Calculate ASDE matrix
    asde_matrix = ASDE(edge_index, mask)

    Step 2: Compute pseudo-confidence
    confidence_matrix = pseudo_confidence(edge_index, asde_matrix)
    soft_confidence_matrix = F.softmax(confidence_matrix, dim=1)

    Step 3: Normalize by row transition
    normalize_confidence = row_transition(soft_confidence_matrix, edge_index[0])

    Step 4: Perform channel-wise diffusion
    diffusion_x = approximate_channel_diffusion(feature, normalize_confidence,
                                                edge_index, mask)

    Return: Channel-Diffusion(imputed) feature matrix
    return diffusion_x, asde_matrix
```

Algorithm 3 PyTorch-style Pseudo-code of Function `row_transition`

```
function row_transition(soft_confidence_matrix, edge_index[0])

    Step 1: row-wise sum
    deg_w = torch_scatter.scatter_add(confidence_matrix, row_index, dim=0)

    Step 2: Inverse degree normalization
    deg_w_inv = deg_w.pow(-1.0)

    Step 3: Fill diagonal with zeros
    deg_w_inv.masked_fill(deg_w_inv == float(nf), 0)

    Step 4: Normalize confidence matrix
    norm_confidence_matrix = confidence_matrix * deg_w_inv[row_index]

    Return: Normalized confidence matrix
    return norm_confidence_matrix
```

Algorithm 4 PyTorch-style Pseudo-code of Function `node_propagation`

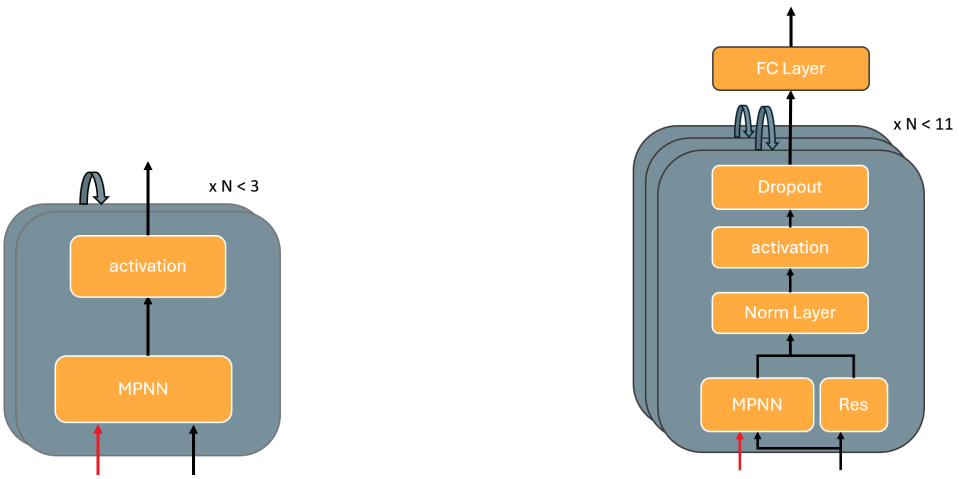
```
function node_propagation(diffusion_x, asde_matrix)

    Step 1: Compute inter-channel correlation (conf)
    conf = torch.corrcoef(diffusion_x.T).nan_to_num().fill_diagonal_(0)

    Step 2: Compute channel mean difference
    alpha_mean = (alpha ** asde_matrix) * (diffusion_x - torch.mean(diffusion_x,
dim=0))

    Step 3: Aggregate with channel correlation
    alpha_mean_conf = torch.matmul(alpha_mean, conf)

    Step 4: Update node features
    update_x = beta * (1 - (alpha ** asde_matrix)) * alpha_mean_conf
    updated_x = diffusion_x + update_x

    Return: Node-updated feature matrix
    return updated_x
```

4.3 tunedGNN: Architecture Optimization for Robustness

Having established a method to obtain a complete and robust feature matrix, the next logical step is to define a GNN architecture capable of effectively learning from this data. While standard GNNs like GCN or GraphSAGE are widely used, they are often architecturally shallow and can be sensitive to the subtle noise and imperfections that may remain even in imputed data. To build a powerful and reliable "teacher" model for our framework, a more resilient foundational architecture is required.

To this end, the **tunedGNN** is introduced as the backbone model. It is important to note that **tunedGNN** is not a novel architecture proposed in this thesis. Rather, it is a systematic application of several well-established, state-of-the-art architectural best practices from recent GNN research [1]. The goal is to construct a model with enhanced stability, expressiveness, and, most critically, robustness against data perturbations. As illustrated in Figure 4.5b, the **tunedGNN** structure is significantly deeper and more complex than classic GNN layers.

4.3.1 Key Design Components



(a) Classic GNN layer structure **(b) Tuned GNN layer structure**

Figure 4.5: Comparison of Tuned and Classic GNN layer structures. (a) The classic GNN layer stacks message-passing and activation modules, typically with fewer than 3 layers. (b) The tuned GNN layer incorporates residual connections, normalization, activation, and dropout, allowing for deeper architectures (up to 11 layers) and improved robustness to incomplete data.

The key design components systematically integrated into our **tunedGNN** backbone are as follows:

- **Deeper Layer Stacking:**
  A deeper architecture (e.g., up to 11 layers in the experiments) is employed compared to traditional shallow GNNs. This allows for a larger receptive field, enabling the model to capture more complex, higher-order relational patterns within the graph, while carefully managing the risk of over-smoothing through the components below.

- **Normalization (e.g., LayerNorm):**
  Each GNN layer is followed by a normalization layer, typically LayerNorm, to stabilize the distribution of intermediate feature representations. This practice is crucial for promoting faster, more stable convergence and is particularly effective in mitigating the impact of noise in high-missingness scenarios.

- **Residual Connections:**
  To enable effective training of such a deep architecture, skip (or residual) connections are added between layers. As shown in Figure 4.5b, the output of the message-passing block is added to the original input before subsequent transformations. This mitigates the vanishing gradient problem and ensures a smoother gradient flow throughout the network.

- **Dropout:**
  Dropout layers are applied after each non-linearity or normalization layer. This acts as a powerful regularization technique, enhancing the model's generalization capabilities and improving its robustness to perturbations in both node and edge features.

The overall layer structure for **tunedGNN** can be formally summarized by the following sequence of operations for each layer, transforming the input representation h_i to the output h_{i+1}:

$$h'_i = \text{Aggregate}(h_i, \{h_j \mid j \in \mathcal{N}(i)\}) \tag{4.6}$$

$$h''_i = h'_i + \text{Linear}(h_i) \tag{4.7}$$

$$h'''_i = \text{Norm}(\sigma(h''_i)) \tag{4.8}$$

$$h_{i+1} = \text{Dropout}(h'''_i) \tag{4.9}$$

where Equation 4.6 represents the core message-passing step, and Equation 4.7 represents the residual connection.

In conclusion, the **tunedGNN** architecture provides a highly performant and robust foundation for our framework. However, this power and resilience come at the cost of a large parameter count and significant computational complexity. This directly motivates the need for the next module in our pipeline: an effective and efficient pruning strategy to make this

powerful model practical for real-world training and deployment.

4.4 MPP: Mirror Projection Pruning

The tunedGNN [1] architecture, as described in the previous section, provides a powerful and robust foundation for our framework. However, its depth and complexity lead to high computational costs during training. To address this challenge, we introduce **MPP**, a novel paradigm designed to efficiently reduce model complexity without sacrificing representational capacity.

Existing GNN pruning techniques are often tightly intertwined with the training process, resulting in significant computational overhead and a lack of flexibility. In contrast, MPP introduces a **proxy-based, pre-training pruning** strategy. Its core idea is to decouple the complex pruning task by projecting the GNN to an isomorphic MLP, applying mature pruning techniques in the simpler MLP space, and then mapping the pruned structure back. This design, inspired by studies like MLPinit [49] that demonstrate GNN-MLP parameter alignment, allows us to leverage a rich ecosystem of existing pruning tools, such as DepGraph (torch-pruning) [50]. Crucially, MPP serves a dual purpose in our pipeline: (1) it produces an efficient, pruned GNN to act as our **Teacher model**, and (2) it simultaneously generates a lightweight, structurally-aligned MLP to serve as our **Student model** for the final distillation stage.

4.4.1 Mirror Projection Pruning (MPP): Theory and Workflow

Mirror Projection Pruning (MPP) (See Figure 4.6) is built on the structural isomorphism between common GNN layers and their MLP counterparts, extending the insight of MLPInit [49] that an MLP with the same architecture as a GNN can serve as a parameter proxy.

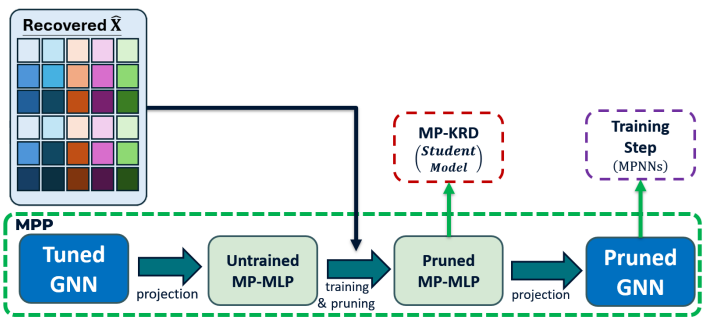


Figure 4.6: The high-level workflow of Mirror Projection Pruning (MPP). A large GNN architecture is projected to a proxy MP-MLP, which is then trained and pruned. The resulting sparse structure is projected back to define the final, efficient 'Pruned GNN'.

Our approach generalizes this concept: an isomorphic MLP can act as an efficient surrogate for a GNN, enabling all training, pruning, and architectural search to be performed in the lightweight MLP domain. Afterward, the resulting sparse structure and weights are seamlessly mapped back to the original GNN, yielding an efficient, pruned GNN with minimal engineering effort.

The Principle of GNN–MLP Isomorphism

The isomorphism between GNN and MLP layers is achieved by removing the neighborhood aggregation (message-passing) operation from the GNN, resulting in a pure node-level transformation (See Figure 4.7).

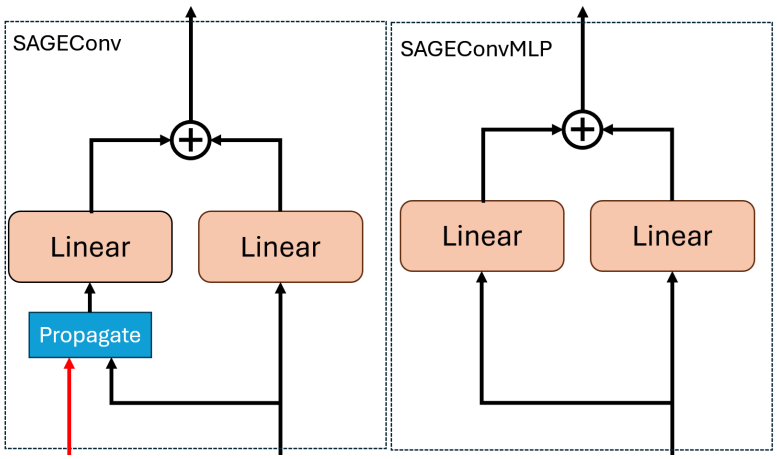


Figure 4.7: Structural illustration of SAGEConv and its MP-MLP projection. Left: The SAGEConv layer consists of two linear branches, one operating on neighborhood-aggregated features (Propagate). Right: The SAGEConvMLP projection replaces neighborhood aggregation with a second self-branch, yielding a pure local MLP structure with identical parameterization. This one-to-one mapping is fundamental to efficient pruning and parameter transfer in MPP.

For example, a SAGEConv layer comprises two linear branches—one operating on the node's own features, the other on the aggregated features from its neighbors. By eliminating the propagation step, we obtain an MLP layer (SAGEConvMLP) with two self-branches, preserving both parameterization and output dimension.

This theoretical foundation ensures that all layers, hidden dimensions, and normalization structures in the original GNN are exactly matched by the MP-MLP, making direct parameter and mask transfer possible in both directions.

Step 1: Mirror Projection—GNN to MP-MLP

Given a tuned GNN backbone (e.g., TunedGNN [1]), the first step is to construct a mirror-projected MLP (MP-MLP) by replacing each GNN-layer with its isomorphic ConvMLP layer. All message-passing operations are removed, while linear transformations, normalization, and nonlinearity are preserved. This results in an untrained MP-MLP network that is structurally identical to the original GNN, as shown in Figure 4.8.

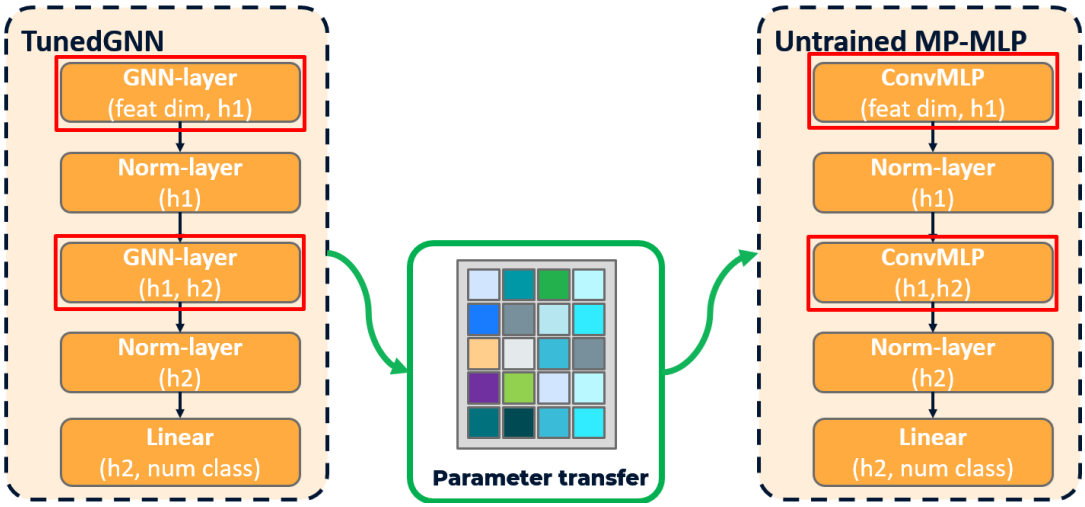


Figure 4.8: Mirror Projection from GNN to MP-MLP. Each GNN-layer is replaced by a corresponding ConvMLP layer, yielding an untrained MP-MLP that mirrors the GNN architecture.

Step 2: Training and Pruning in MP-MLP Space

All subsequent training and pruning are performed in the MP-MLP domain. The MP-MLP is first trained on the completed feature set for the target task, then pruned using state-of-the-art MLP pruning algorithms (e.g., LAMP [51], DepGraph [50]), which efficiently remove redundant weights or channels based on importance. This step is illustrated in Figure 4.9.

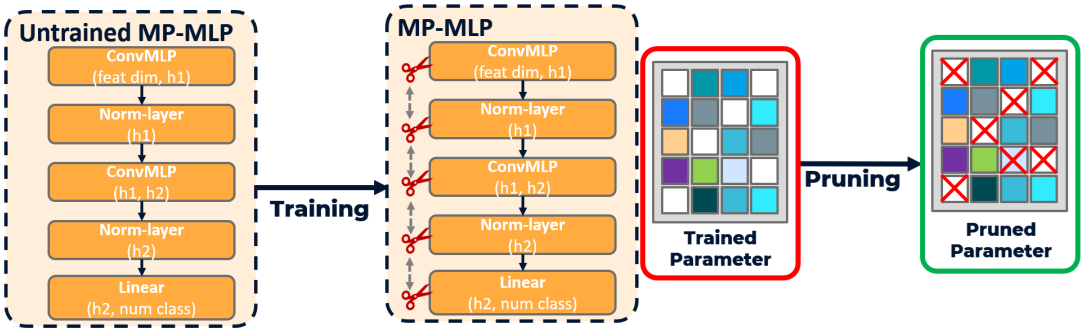


Figure 4.9: Proxy Training and Pruning in MP-MLP. The MP-MLP is trained and pruned to yield a sparse, efficient student model.

Step 3: Inverse Projection—Parameter and Mask Transfer Back to GNN

Once pruning is completed, the sparse weights and binary pruning mask from the MP-MLP are mapped directly back to the original GNN, thanks to the strict structural alignment. Each pruned ConvMLP layer's parameters and mask are copied to the corresponding GNN-layer, transforming the large, dense GNN into a compact, sparse network without modifying the original codebase. See Figure 4.10.

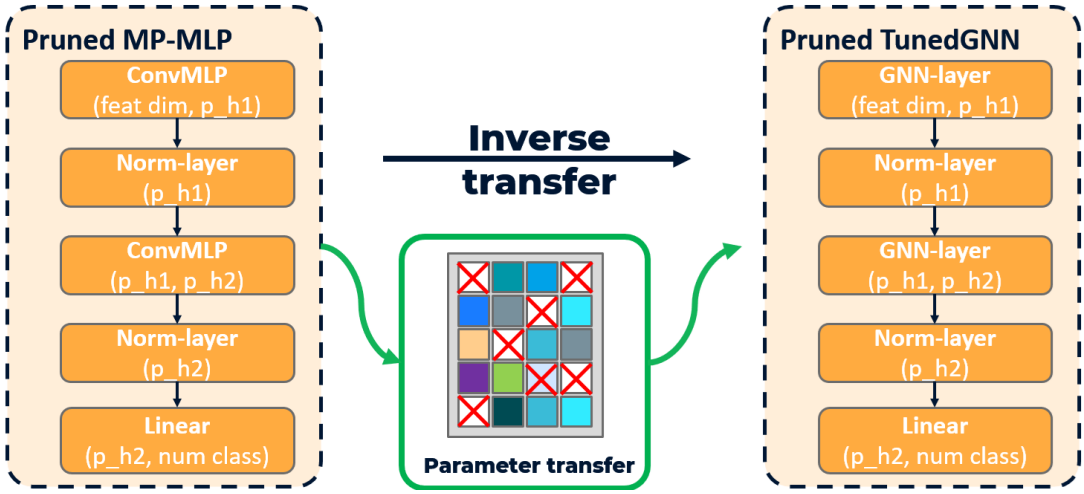


Figure 4.10: Inverse Projection from Pruned MP-MLP to Pruned GNN. The pruning mask and weights are transferred directly, instantly producing a sparse, efficient GNN.

Advantages of MPP

- **Computational efficiency:** Training and pruning are much faster in the MLP domain, as no neighborhood aggregation is needed.
- **Structural flexibility:** Any MLP pruning or search technique can be applied to the proxy MP-MLP and directly benefit the GNN.
- **Seamless transfer:** Strict one-to-one mapping enables direct synchronization of parameters and masks between MLP and GNN.
- **Minimal engineering overhead:** No modification to the GNN implementation is required; all changes are applied through parameter transfer.

This mirror projection–pruning–inverse projection workflow offers a highly flexible, efficient, and practical GNN compression paradigm, making advanced pruning and architectural optimization accessible for real-world graph learning tasks.

Implementation and Pseudocode

The entire process is outlined in Algorithm 5. This workflow efficiently produces a pruned teacher GNN and a structurally-aligned student MLP, perfectly setting the stage for our final knowledge transfer module.

Algorithm 5 MPP Proxy Pruning Pseudocode

```
function mpp_prune(gnn_model, feature, label)

    Step 1: Remove message-passing and project to MLP proxy
    mp_mlp = project_to_mlp(gnn_model)

    Step 2: Pre-train the proxy model
    mp_mlp.train(feature, label)

    Step 3: Prune the proxy MLP
    pruning_mask = prune_mlp(mp_mlp, method="LAMP")

    Step 4: Generate the student model
    pruned_mlp = apply_mask(mp_mlp, pruning_mask)

    Step 5: Parameter and Mask Transfer back to original GNN (teacher model)
    pruned_gnn = apply_mask(gnn_model, pruning_mask)

    Return the pruned GNN and pruned MLP
    return pruned_gnn, pruned_mlp
```

4.5 Teacher Training

After obtaining the pruned GNN from the MPP workflow , a dedicated training phase is conducted to optimize its parameters on the imputed dataset generated by the APCFI module. Specifically, the pruned GNN is trained using the recovered feature matrix $\widehat{\mathbf{X}}$ and the original graph structure, following standard supervised learning protocols for node classification.

Throughout this stage (see Fig. 4.11), the model benefits from both the lightweight architecture achieved via pruning and the improved data quality resulting from feature imputation. The cross-entropy loss is adopted as the objective function to maximize classification accuracy. Upon completion of training, the resulting high-performance and efficient GNN is designated as the **Teacher Model** for the subsequent knowledge distillation phase.

This process ensures that the teacher model not only possesses strong predictive capabilities but also maintains computational efficiency, thereby serving as an ideal source of knowledge for transferring to a student model.

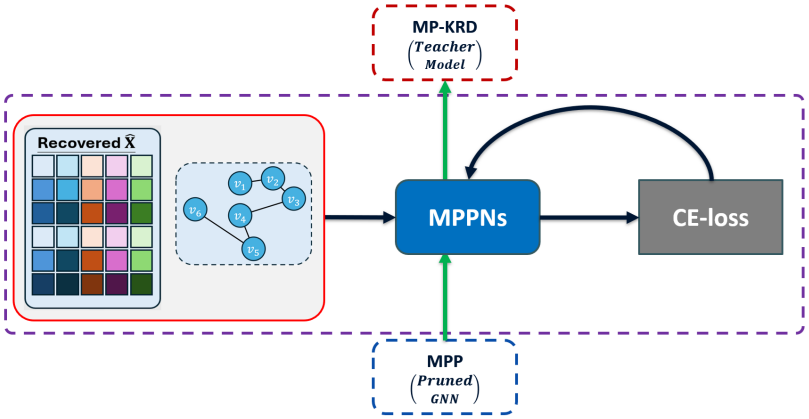


Figure 4.11: Teacher Training Workflow. The pruned GNN obtained from the MPP module is fully trained on the imputed feature matrix $\widehat{\mathbf{X}}$ generated by APCFI. The resulting efficient model is used as the Teacher for knowledge distillation.

4.6 MP-KRD: Mirror Projection Knowledge-inspired Reliable Distillation

The final stage of our iPaD pipeline, **MP-KRD**, orchestrates the knowledge transfer from the powerful teacher model to the lightweight student model, both of which were prepared by the preceding MPP module. Specifically, it takes the trained "Pruned tunedGNN" as the Teacher and the "Pruned MP-MLP" as the Student. As depicted in Figure 4.12, the goal is to create a final inference model that is both highly efficient and robust.

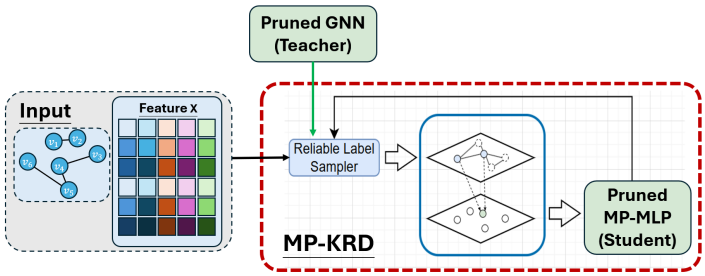


Figure 4.12: The high-level view of the MP-KRD module. It takes the trained GNN Teacher and the Pruned MP-MLP Student as input. A Reliable Label Sampler, guided by a curriculum, selects high-quality knowledge to distill, producing the final, efficient inference model.

4.6.1 Motivation: The Need for Reliable Distillation

A naive distillation process that simply forces the student to mimic all of the teacher's outputs can be problematic. A teacher trained on imputed or noisy data may exhibit low confidence or even make mistakes on certain samples. Forcing the student to learn this "unreliable knowledge" can harm its performance. To motivate our approach, consider two samples (A and B) with the same ground-truth label $[0, 1, 0]$, where the teacher GNN produces the following output logits:

- **Sample A:** GNN output logits $[0.3,\ 0.5,\ 0.2]$
- **Sample B:** GNN output logits $[0.075,\ 0.9,\ 0.025]$

Although both samples are correctly classified, the cross-entropy loss for Sample B is noticeably lower than for Sample A. Intuitively, this means the model is more confident in its prediction for Sample B. Prediction reliability is systematically quantified using a robust mechanism rather than relying solely on raw logit values.

Therefore, instead of treating all teacher knowledge equally, MP-KRD employs a **reliability-aware curriculum**. The core idea is to quantify sample "difficulty" or "unreliability" by the **stability of the teacher's prediction under noise perturbation**, measured via entropy change. This allows the student to learn from the most stable and reliable knowledge first.

4.6.2 The MP-KRD Workflow

The MP-KRD process is an iterative workflow that adaptively transfers knowledge from the teacher to the student (see Fig. 4.13).

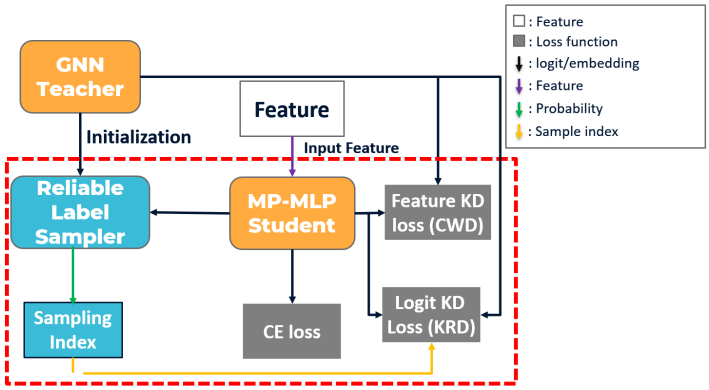


Figure 4.13: The workflow of the MP-KRD module. The pruned GNN teacher provides sample reliability signals for curriculum-based distillation, guiding the training of the MP-MLP student via cross-entropy, logit KD (KRD), and feature KD (CWD) losses.

Step 1: Sample Difficulty Calculation.

The workflow begins by quantifying the reliability of the teacher's prediction for each training sample i. This is done by measuring the change in the teacher's output entropy when a small Gaussian noise, $\epsilon \sim \mathcal{N}(0, \delta)$, is added to the input features $\mathbf{X}$.

$$\Delta H_i = |H(f_{\text{GNN}}(\mathbf{X}', i)) - H(f_{\text{GNN}}(\mathbf{X}, i))| \tag{4.10}$$

$$\Delta e_i = \frac{\Delta H_i - \min_j \Delta H_j}{\max_j \Delta H_j - \min_j \Delta H_j} \tag{4.11}$$

The normalized entropy change, $\Delta e_i \in [0, 1]$, serves as our metric for sample difficulty, where a higher value indicates a less reliable or more difficult sample.

Step 2: Curriculum-based Sampling.

In each training epoch, we use a curriculum to select a subset of reliable samples. The probability of selecting sample i is governed by a dynamic power function:

$$\mathrm{p_i} = 1 - \Delta e_i^{\text{power}}$$

Based on this probability, a mask is generated via Bernoulli sampling, and only the selected samples (S_{epoch}) are used for training in that epoch. This ensures that easier samples (low Δe_i) are prioritized in the early stages of training.

Step 3: Composite Distillation Objective.

The student model is trained on the sampled subset S_{epoch} using a composite loss function designed to capture knowledge from multiple perspectives:

$$L_{\text{MP-KRD}} = L_{\text{CE}} + \lambda L_{\text{KRD}} + \alpha L_{\text{CWD}} \tag{4.12}$$

The hyperparameters λ and α balance the contributions of these different forms of knowledge.

- **Cross-entropy loss** (L_{CE}): The standard supervised cross-entropy loss against the ground-truth labels,

$$L_{\text{CE}} = -\sum_{i \in N} y_i \log\left(\hat{y}_i^S\right) \tag{4.13}$$

  where y_i is the true label and $\hat{y}_i^S$ denotes the student model's softmax output.

- **Logit-based distillation loss** (L_{KRD}): Encourages the student's output distribution to match the teacher's,

$$L_{\text{KRD}} = \sum_{i \in S_{\text{epoch}}} \text{KL}(\text{Student}_{\text{logit}}(x_i),\ \text{Teacher}_{\text{logit}}(x_i)) \tag{4.14}$$

  where KL denotes the Kullback-Leibler divergence, and S_{epoch} is the set of training samples selected for distillation at the current epoch (e.g., the sampled reliable points under the curriculum policy).

- **Feature-based distillation loss** (L_{CWD}): Pushes the student's hidden representations (z_i^{student}) to align with the teacher's (z_i^{teacher})[52],

$$L^l_{\text{node-wise}} = \text{KL}\left(\text{Softmax}(\mathbf{Z}^{S,l}),\ \text{Softmax}(\mathbf{Z}^{T,l})\right) \tag{4.15}$$

$$L^l_{\text{channel-wise}} = \text{KL}\left(\text{Softmax}\left((\mathbf{Z}^{S,l})^\top\right),\ \text{Softmax}\left((\mathbf{Z}^{T,l})^\top\right)\right) \tag{4.16}$$

$$L^l_{\text{CWD}} = L^l_{\text{node-wise}} + L^l_{\text{channel-wise}} \tag{4.17}$$

$$L^{\text{total}}_{\text{CWD}} = \sum_{l \in L} L^l_{\text{CWD}} \tag{4.18}$$

  where l indexes the selected layers for distillation, and L is the set of such layers.

Step 4: Adaptive Curriculum Update.

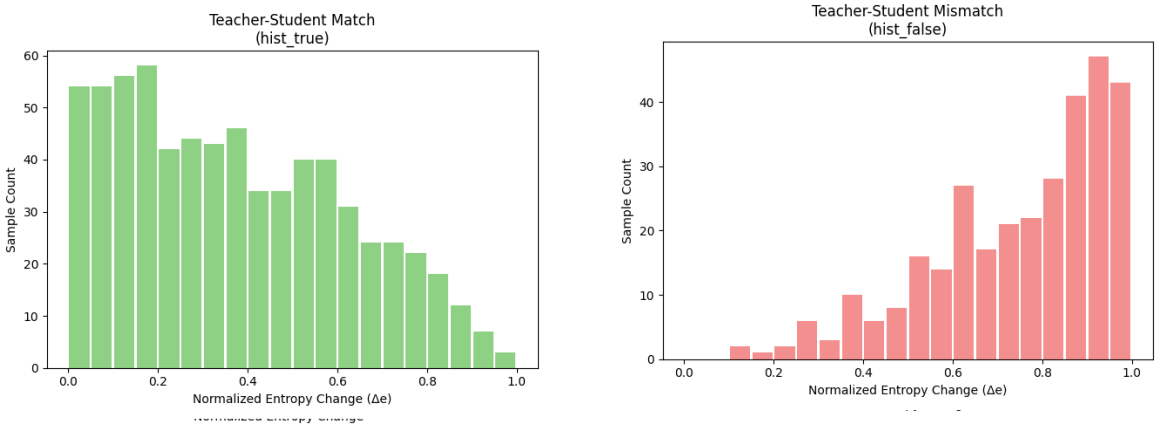
To make the curriculum dynamic, the 'power' parameter is updated at the end of each epoch. This is done by analyzing the student's current performance against the teacher's.

In order to investigate how sample difficulty affects the consistency between teacher and student predictions, all training samples are partitioned into discrete bins based on their normalized entropy change (Δe). For each bin, the frequencies of teacher-student matches and mismatches are computed, and their distributions are illustrated in Figure 4.14. This provides an empirical foundation for the subsequent curriculum adjustment process.

To quantitatively measure the agreement probability within each entropy bin, the following metric is defined:

$$hist = \frac{hist_{true}}{hist_{true} + hist_{false} + \varepsilon} \tag{4.19}$$

where $hist_{true}$ and $hist_{false}$ denote the number of agreement and disagreement samples in ε is a small constant for numerical stability.



(a) Teacher-student agreement histogram **(b) Teacher-student disagreement histogram**

Figure 4.14: Histograms of teacher-student agreement (a) and disagreement (b) plotted against the normalized entropy change. Easier samples (low entropy change) show high agreement, while harder samples show high disagreement.

Next, as shown in Figure 4.15, we fit the empirical probabilities hist_{fin} using the function:

$$p = 1 - \Delta e^{\text{power}_{fit}} \tag{4.20}$$

$$\text{power}_{fit} = \frac{\log(1-p)}{\log(\Delta e)} \tag{4.21}$$

where Δe is the normalized entropy change and power_{fit} is determined by least-squares fitting. This process dynamically determines the optimal curriculum power parameter for the current epoch.

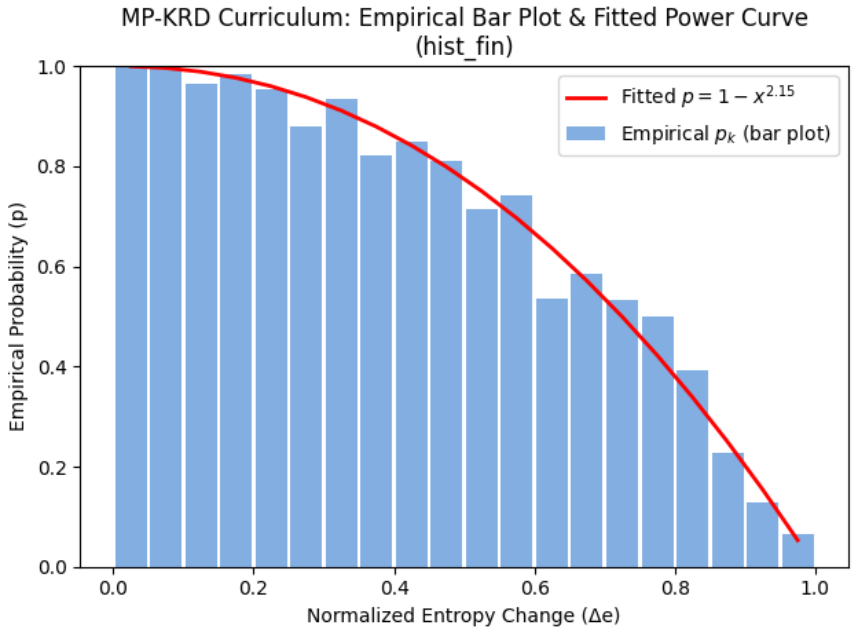


Figure 4.15: The empirical agreement probability (p_k) is fitted with the power curve (red line) to dynamically determine the optimal curriculum "power$_{fit}$" parameter.

Finally, the "power" for the next epoch is updated using an **Exponential Moving Average (EMA)** to ensure a smooth curriculum progression:

$$\text{power}(t) = \beta \cdot \text{power}(t-1) + (1-\beta) \cdot \text{power}_{fit}(t) \tag{4.22}$$

where β is a smoothing coefficient(momentum) and power$_{fit}(t)$ is the fitted value from the previous step.

4.6.3 Deployment

After MP-KRD training, only the sparse pruned mirror-projected MLP student is deployed, achieving both high efficiency and robustness through distilled GNN knowledge.

Chapter 5

Experiment

5.1 Experimental Setup

To ensure a thorough, fair, and reproducible evaluation of the proposed iPaD framework, this section details the datasets, compared methods, training configurations, and evaluation protocols used throughout this chapter.

5.1.1 Datasets

Experiments on eight widely-used node classification benchmark dataset are conducted. These include three classic citation networks: **Cora**, **CiteSeer**, and **PubMed** [53]. Additionally, five medium-to-large-scale graphs are utilized: the **Computers** and **Photo** co-purchase networks, the **CS** and **Physics** co-authorship networks [54], and the **WikiCS** dataset [55]. Key statistics for each dataset are summarized in Table 5.1. All tasks are formulated as semi-supervised node classification, with accuracy as the primary evaluation metric.

Datasets	nodes	edges	Features	Classes	Metric
Cora [53]	2,708	5,278	1,433	7	Accuracy
CiteSeer [53]	3,327	4,522	3,703	6	Accuracy
PubMed [53]	19,717	44,324	500	3	Accuracy
Computers	13,752	245,861	767	1	Accuracy
Photo	7,650	119,081	745	8	Accuracy
WikiCS [55]	11,701	216,123	300	6	Accuracy
CS	18,333	81,894	6,805	15	Accuracy
Physics	34,493	247,962	8,415	5	Accuracy

Table 5.1: Statistics of Benchmark Datasets Used in Node Classification Tasks. The table summarizes the number of nodes, edges, features, classes, and the evaluation metric for each dataset.

5.1.2 Compared Methods

Experiments involve a diverse range of baseline and state-of-the-art models. For imputation-specific analysis (Section II), The proposed **APCFI** is compared against **Zero**, **LP** [22], **sRMGCNN** [56], **GCNMF** [20], **PaGNN** [12], **FP** [19], and **PCFI** [21]. For end-to-end model performance (Section III), **iPaD** is benchmarked against a strong **tunedGNN** [1], baseline and recent state-of-the-art models, including **GraphGPS** [32], **NAGphormer** [33],, **Exphormer** [34], **GOAT** [35], **NodeFormer** [36], **SGFormer** [37], and **Polynormer** [38]. Additionally, for knowledge distillation, we directly compare our method with state-of-the-art KD baselines, including **GLNN**[29] and **KRD**[30], to demonstrate the effectiveness of our approach under distillation scenarios.

5.1.3 Training Details

All models are trained using the Adam optimizer. To ensure a fair and robust comparison, critical hyperparameters, including the learning rate, are systematically tuned for each model and dataset using the **Optuna** [57]. Unless otherwise specified, the number of training epochs follows the settings from the TunedGNN benchmark [1].

To account for stochasticity, **all reported experimental results are the average of 10 independent runs, each with a different but fixed random seed**. All experiments are conducted on a workstation equipped with an NVIDIA RTX 4090 GPU, an AMD Ryzen 9 7900 CPU, and 64 GB of DDR5 RAM.

5.1.4 Missing Data Settings

To evaluate robustness, robustness is evaluated by systematically introducing missing data under two primary settings: **feature missing** and **edge missing**.. Feature missing includes both uniform (random) and structural (patterned) masking, with missing rates set to **0.0**, **0.3**, **0.6**, **0.9**, and the extreme **0.995**. Edge missing uses random masking with rates of **0.0**, **0.3**, **0.6**, and **0.9**.

5.1.5 Dataset Split Strategy

To ensure fair comparisons with prior work, the specific data split protocols established in the benchmark literature are adopted to ensure fair comparisons with prior work. The experiments use two distinct splitting strategies based on the evaluation's focus:

Imputation Performance Analysis (Section II).

The protocol from the PCFI paper [21] is strictly followed. Specifically, for each dataset, 10 different random splits are generated. In each split, 20 nodes per class are allocated for training (N_{train}). The validation set size is then set to $1500 - N_{\text{train}}$. All remaining nodes are used for testing.

All Other Experiments (SOTA, Robustness, Ablations).

For all other experiments (Sections III to VII), the widely-accepted splits from the TunedGNN benchmark [1] are used. This involves several settings:

- For **Cora, CiteSeer, and PubMed**, we use the standard semi-supervised splits as defined in [53].
- For **Computers, Photo, CS, and Physics**, we use a standard 60%/20%/20% split for training, validation, and testing, respectively.
- For **WikiCS**, we use the official public split provided by its original authors [55].

Adhering to these established protocols ensures the reliability and comparability of our results with the broader academic community.

5.2 Validating APCFI: A Superior Balance of Accuracy and Efficiency

This section validates our first claim: that **APCFI** effectively resolves the performance-efficiency trade-off present in existing feature imputation methods. Specifically, we evaluate whether APCFI achieves greater efficiency than its high-performance predecessor, PCFI, while maintaining comparable imputation accuracy. The analysis focuses on verifying that the observed efficiency improvements do not compromise the quality of feature imputation.

5.2.1 Efficiency Comparison

As shown in Table 5.2, APCFI achieves significant improvements in efficiency. Across all benchmark datasets, APCFI is only marginally slower than the simple FP baseline. Crucially, compared to the high-performance PCFI, APCFI demonstrates a massive efficiency improvement, running on average **over 200 times faster**. This result confirms that APCFI is a practically viable imputation solution for large-scale graphs where PCFI's complexity would be prohibitive.

Method	Cora	CiteSeer	PubMed	Photo	Computers	Runtime(s)
FP	0.05 ± 0.01	0.06 ± 0.00	0.06 ± 0.00	0.06 ± 0.00	0.09 ± 0.00	0.06
PCFI	73.40 ± 0.42	252.8 ± 2.29	31.18 ± 0.43	32.45 ± 0.16	39.99 ± 0.17	85.97
APCFI(our)	**0.19 ± 0.04**	**0.39 ± 0.02**	**0.19 ± 0.01**	**0.46 ± 0.00**	**0.72 ± 0.00**	**0.39**

Table 5.2: Runtime comparison of different feature imputation methods across datasets. The table reports the runtime (in seconds) for FP, PCFI, and APCFI methods on several benchmark datasets. APCFI achieves comparable efficiency to FP and significantly outperforms PCFI in computational cost.

5.2.2 Accuracy under High Missing Rates

Table 5.3 addresses the second question, showing that this immense efficiency gain does not compromise performance. Under both structural and uniform missingness scenarios with up to 99.5% of features absent, APCFI's accuracy is statistically on par with the much slower PCFI. It consistently and significantly outperforms other baselines, especially in high-missingness regimes.

These results confirm our hypothesis: **APCFI successfully combines the high accuracy of PCFI with the high efficiency of FP**, establishing it as a powerful and practical foundation for the data imputation stage of our framework.

Missing Type	Method	Cora	CiteSeer	PubMed	Photo	Computers	Average↑	Avg.rank↓
Structural	Zero*	44.15 ± 8.44	31.68 ± 4.50	48.20 ± 3.65	79.68 ± 2.17	72.03 ± 1.91	55.15	8.2
	LP*	**74.52 ± 1.60**	**65.89 ± 2.29**	72.25 ± 3.78	82.42 ± 2.57	76.28 ± 1.43	74.27	5
	sRMGCNN*	29.31 ± 0.71	24.21 ± 1.35	OOM	26.10 ± 1.89	37.15 ± 0.12	⋆	10.6
	GCNMF*	29.20 ± 1.13	24.50 ± 1.52	40.19 ± 0.95	26.82 ± 6.33	30.59 ± 9.81	30.26	10.4
	PaGNN*	30.55 ± 8.85	25.69 ± 3.98	50.82 ± 4.61	66.91 ± 3.99	56.50 ± 3.29	46.10	8.8
	FP*	72.84 ± 2.85	59.76 ± 2.47	72.69 ± 2.66	86.57 ± 1.50	77.42 ± 1.59	73.86	5.6
	FP	72.69 ± 2.84	59.65 ± 2.61	72.67 ± 2.88	86.46 ± 1.50	77.58 ± 1.55	73.81	6.2
	PCFI*	**75.49 ± 2.10**	**66.18 ± 2.75**	**74.66 ± 2.26**	**87.70 ± 1.29**	**79.25 ± 1.19**	**76.66**	**1**
	PCFI	73.42 ± 2.74	**62.85 ± 2.83**	**73.44 ± 2.32**	**87.44 ± 1.37**	**78.76 ± 1.45**	**75.18**	**2.6**
	APCFI(our)	**73.44 ± 2.51**	62.02 ± 2.79	**73.39 ± 2.29**	**87.24 ± 1.39**	**77.77 ± 1.22**	**74.77**	**3.2**
Uniform	Zero*	62.63 ± 2.64	63.19 ± 1.83	54.70 ± 3.03	85.40 ± 1.33	79.49 ± 1.21	69.08	7.8
	LP*	74.52 ± 1.60	65.78 ± 2.29	72.25 ± 3.78	82.42 ± 2.57	76.28 ± 1.43	74.25	7.8
	sRMGCNN*	29.32 ± 0.74	24.66 ± 1.90	OOM	26.58 ± 1.68	37.16 ± 0.12	⋆	10.2
	GCNMF*	27.85 ± 2.27	24.29 ± 1.47	39.47 ± 0.76	25.98 ± 3.90	34.78 ± 4.69	30.47	10.8
	PaGNN*	53.75 ± 2.03	44.95 ± 2.59	60.24 ± 3.78	85.30 ± 1.05	78.04 ± 1.18	64.45	8.4
	FP*	77.55 ± 2.01	68.00 ± 2.16	73.88 ± 2.35	87.75 ± 1.07	81.47 ± 0.91	77.73	5.1
	FP	77.55 ± 2.01	67.47 ± 2.28	73.42 ± 2.33	88.05 ± 1.16	81.25 ± 1.05	77.55	5.5
	PCFI*	**78.53 ± 1.39**	**69.40 ± 1.85**	**76.44 ± 1.64**	**88.60 ± 1.30**	**81.79 ± 0.70**	**78.95**	**1.2**
	PCFI	**78.57 ± 1.55**	**68.60 ± 1.71**	**75.88 ± 1.74**	**88.54 ± 1.18**	**81.76 ± 0.9**	**78.67**	**1.8**
	APCFI(our)	**78.26 ± 1.79**	**68.31 ± 2.01**	**75.22 ± 2.24**	**88.08 ± 1.25**	**81.63 ± 0.98**	**78.30**	**3.0**

Table 5.3: Performance comparison of various feature imputation methods under structural and uniform missing scenarios. The table reports the accuracy (%) on five datasets for each method. Methods marked with ∗ are directly introduced from the PCFI paper. APCFI achieves the best or near-best performance and average rank under both missing types. The best, second-best, and third-best results are highlighted in **green**, **cyan**, and **red**

5.3 Baseline Performance: Establishing Competitiveness with Complete Data

Before stress-testing our framework's robustness against incomplete data, we must first establish its credibility as a high-performing model in a standard setting. To establish a strong baseline, this section evaluates the fundamental soundness of the iPaD framework and compares its performance with state-of-the-art (SOTA) models under standard, complete-data conditions.

The results, summarized in Tables 5.4 and 5.5, demonstrate a consistently strong and competitive performance, validating our overall design. The nature of this performance, however, shows interesting nuances depending on the GNN backbone.

Model	Cora	CiteSeer	PubMed	Avg Accuracy
GraphGPS (NerurIPS 2022)*	82.84 ± 1.03	72.73 ± 1.23	79.94 ± 0.26	78.50
NAGphormer (ICLR 2023)*	82.12 ± 1.18	71.47 ± 1.30	79.73 ± 0.28	77.77
Exphormer (ICML2023)*	82.77 ± 1.38	71.63 ± 1.19	79.46 ± 0.35	77.95
GOAT (ICML 2023)*	83.18 ± 1.27	71.99 ± 1.26	79.13 ± 0.38	78.10
NodeFormer (NeurIPS 2022)*	82.20 ± 0.90	72.0 ± 1.10	79.90 ± 1.00	78.20
SGFormer (NeurIPS 2023)*	**84.50 ± 0.80**	72.60 ± 0.20	80.30 ± 0.60	79.13
Polymormer (ICLR 2024)*	83.25 ± 0.89	72.31 ± 0.83	79.24 ± 0.73	78.27
GCN	83.77 ± 0.57	71.83 ± 0.28	79.89 ± 0.18	78.50
GLNN-GCN (ICLR 2022)	83.94 ± 0.77	**73.93 ± 0.62**	**81.40 ± 0.69**	**79.77** **1.27** ↑
KRD-GCN (ICML 2023)	83.96 ± 1.02	**74.26 ± 0.88**	**83.08 ± 0.43**	**80.43** **1.93** ↑
TunedGCN(NeurIPS 2024)	**84.52 ± 0.62**	72.41 ± 0.71	80.88 ± 1.20	79.27 **0.77** ↑
iPaD-GCN(our)	**85.08 ± 0.77**	**75.37 ± 0.58**	**82.19 ± 2.27**	**80.88** **2.38** ↑
GraphSAGE	80.60 ± 0.82	68.82 ± 0.92	73.45 ± 0.77	74.29
GLNN-SAGE (ICLR 2022)	80.89 ± 0.55	70.36 ± 0.46	76.23 ± 1.15	75.83 **1.54** ↑
KRD-SAGE (ICML 2023)	83.79 ± 0.55	72.00 ± 0.85	79.33 ± 0.90	77.78 **4.08** ↑
TunedGCN(NeurIPS 2024)	83.34 ± 0.58	70.15 ± 0.52	76.96 ± 0.78	76.82 **2.53** ↑
iPaD-SAGE(our)	83.16 ± 1.43	72.98 ± 0.99	78.59 ± 3.05	78.24 **3.95** ↑

Table 5.4: SOTA comparison on complete classic datasets. The best, second-best, and third-best results are highlighted in **green**, **cyan**, and **red**, respectively. Results marked with * are from tunedGNN's paper.

Model	Computer	Photo	WikiCS	CS	Physics	Avg Accuracy
GraphGPS (NerurIPS 2022)*	91.19± 0.54	95.06 ± 0.13	78.66 ± 0.49	93.93 ± 0.12	97.12 ± 0.19	91.19
NAGphormer (ICLR 2023)*	91.22± 0.14	95.49 ± 0.11	77.16 ± 0.72	95.75 ± 0.09	**97.34 ± 0.03**	91.39
Exphormer (ICML2023)*	91.47± 0.17	95.35 ± 0.22	78.54 ± 0.49	94.93 ± 0.01	96.89 ± 0.09	91.44
GOAT (ICML 2023)*	90.96± 0.90	92.96 ± 1.48	77.00 ± 0.77	94.21 ± 0.38	96.24 ± 0.24	90.27
NodeFormer (NeurIPS 2022)*	86.98± 0.62	93.46 ± 0.35	74.73 ± 0.94	95.64 ± 0.22	96.45 ± 0.28	89.45
SGFormer (NeurIPS 2023)*	91.99± 0.76	95.10 ± 0.47	73.46 ± 0.56	94.78 ± 0.20	96.60 ± 0.18	90.39
Polymormer (ICLR 2024)*	**93.68 ± 0.21**	**96.46 ± 0.26**	80.10 ± 0.67	95.53 ± 0.16	97.27 ± 0.08	**92.61**
GCN	71.77 ± 0.12	91.43 ± 0.11	79.60 ± 0.12	90.28 ± 0.12	94.50 ± 0.09	85.52
GLNN-GCN (ICLR 2022)	86.67 ± 0.45	94.72 ± 0.31	80.18 ± 0.19	94.71 ± 0.13	96.15 ± 0.17	90.49 **4.97** ↑
KRD-GCN (ICML 2023)	84.29 ± 0.47	93.31 ± 0.31	79.76 ± 0.38	94.85 ± 0.12	96.60 ± 0.08	89.76 **4.12** ↑
TunedGCN(NeurIPS 2024)	**93.83 ± 0.25**	95.97 ± 0.30	**80.52 ± 0.25**	95.89 ± 0.14	**97.37 ± 0.06**	**92.72** **7.20** ↑
iPaD-GCN(our)	**93.00 ± 0.34**	**96.23 ± 0.22**	**81.11 ± 0.40**	**96.14 ± 0.11**	**97.31 ± 0.10**	**92.76** **7.24** ↑
GraphSAGE	91.71 ± 0.24	95.75 ± 0.26	80.21 ± 0.32	95.70 ± 0.10	96.75 ± 0.12	92.02
GLNN-SAGE (ICLR 2022)	86.43 ± 0.56	95.18 ± 0.42	78.19 ± 0.41	92.93 ± 0.23	96.88 ± 0.10	89.92 **2.76** ↓
KRD-SAGE (ICML 2023)	90.21 ± 0.41	95.22 ± 0.24	79.88 ± 0.43	94.74 ± 0.30	97.14 ± 0.09	91.44 **1.22** ↓
TunedSAGE(NeurIPS 2024)	92.90 ± 0.26	**96.04 ± 0.24**	**80.33 ± 0.37**	**95.94 ± 0.11**	96.96 ± 0.08	92.43 **0.41** ↑
iPaD-SAGE(our)	91.07 ± 0.66	95.37 ± 1.29	79.67 ± 0.14	**96.46 ± 0.16**	96.70 ± 0.47	91.85 **0.17** ↓

Table 5.5: SOTA comparison on complete medium datasets. The best, second-best, and third-best results are highlighted in **green**, **cyan**, and **red**, respectively. Results marked with * are from tunedGNN's paper.

For the GCN backbone, the impact of our framework is remarkably positive. "iPaD-GCN" not only achieves state-of-the-art results on several datasets, such as **CiteSeer (75.37%)** and **PubMed (82.09%)**, but also shows a massive average performance uplift on larger datasets compared to its baseline (e.g., an average accuracy improvement from **85.64%** to **92.72%** on medium datasets (See Tab 5.5). This confirms that the architectural principles of our framework can significantly enhance the performance of standard GNN models.

For the SAGE variant, "iPaD-SAGE" demonstrates a highly competitive performance that is statistically on par with its powerful "GraphSAGE" baseline. On the medium datasets, its average accuracy is **91.68%**, a negligible difference of only 0.18% compared to the baseline's 91.86% (See Tab 5.5). This result is highly significant in its own right: it proves that incorporating our framework's complex machinery for robustness and efficiency introduces **no meaningful performance penalty** on complete data, maintaining the high performance of an already strong SOTA model.

In conclusion, by demonstrating a significantly improved GCN variant and a competitively on-par SAGE variant, iPaD establishes its credibility as a top-tier framework. Its ability to either substantially boost or maintain SOTA performance validates its fundamental design. Having confirmed this strong baseline, we can now proceed to evaluate the framework on its primary and most critical task: **maintaining robustness in the face of extreme data incompleteness.**

5.4 Validating MPP: Quantifying Gains in Training and Inference Efficiency

To rigorously validate the effectiveness of the Mirror Projection Pruning (MPP) mechanism, extensive experiments were conducted to quantify its impact on both training and inference efficiency. The key metrics considered include total training time, per-epoch time, average inference latency, and model parameter count.

5.4.1 Impact on Computational Costs

Our results show that MPP leads to dramatic savings across multiple efficiency metrics. As shown in Table 5.6, increasing the prune rate from 0.0 to 0.9 drastically reduces the computational cost (measured in GFLOPs) and inference time. Specifically, GFLOPs decrease by over **10x** from **275.01** to **25.67**, while the inference time drops by more than **5x** from **15.76 ms** to just **2.79 ms**. This demonstrates the significant resource-saving benefits brought by the MPP pruning mechanism for model deployment.

Prune Rate	GFLOPs↓	Inference Time (ms)↓
0.0	275.01	15.76
0.5	132.70	6.54
0.6	104.96	6.46
0.7	78.15	5.69
0.8	51.72	3.69
0.9	25.67	2.79

Table 5.6: Impact of MPP pruning on FLOPs and inference time of the GCN model. Increasing the prune rate substantially reduces both computational cost (GFLOPs) and inference time, demonstrating the efficiency gains enabled by the MPP pruning strategy.

Furthermore, Figure 5.1 demonstrates that this efficiency gain extends to the entire training pipeline. As the prune rate increases, the total training time—which includes the overhead of the MPP process itself—decreases substantially from **60.04s** to just **18.48s**, a more than three-fold reduction. This confirms that MPP is a highly effective strategy for not only accelerating deployment but also shortening the model development cycle.

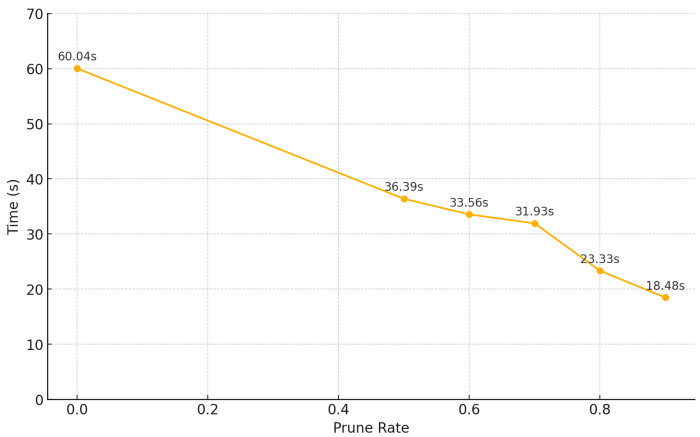


Figure 5.1: Total training time (including the MPP step) of the MPP-pruned GCN model on the CS dataset under different prune rates. Higher prune rates lead to a significant reduction in total training time.

5.4.2 Accuracy-Efficiency Trade-off and the "Sweet Spot"

Having established the significant efficiency gains, we now analyze the trade-off with predictive performance. Figure 5.2 plots the test accuracy of both GCN and SAGE models against the prune rate on the Physics dataset.

The results reveal a crucial finding: the existence of a wide **"efficiency sweet spot"**. For both

models, the accuracy remains remarkably stable with negligible degradation up to a prune rate of 0.7. For instance, the GCN model's accuracy only drops from **97.46%** to **97.23%** at a 70% prune rate, while achieving the substantial computational savings described above. This indicates that a large portion of the model's parameters can be removed via MPP without harming its core expressive power. While higher prune rates (0.8-0.9) offer further efficiency gains, they come at the cost of a more noticeable, yet still graceful, decline in accuracy, making them suitable for scenarios where efficiency is the absolute priority.

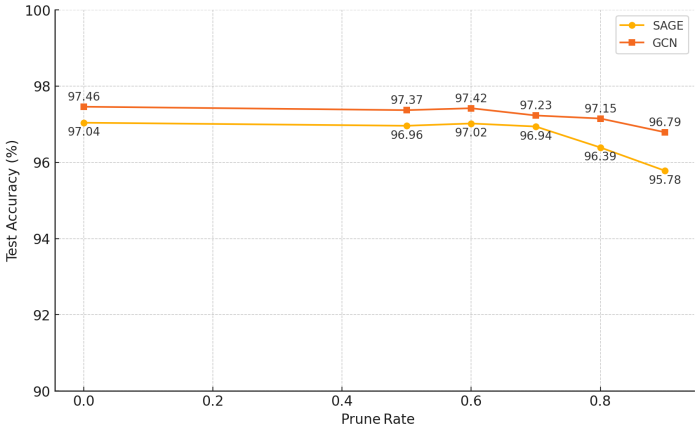


Figure 5.2: Test accuracy of SAGE and GCN models on the Physics dataset under different MPP prune rates. The accuracy remains highly stable up to a prune rate of 0.7, revealing a wide "sweet spot" for efficiency gains without a significant performance penalty.

In conclusion, our experiments provide strong evidence for the efficacy of the MPP module. It enables significant reductions in both training and inference costs while maintaining high accuracy up to aggressive pruning rates, confirming it as a practical and powerful component of the **iPaD** pipeline.

5.5 Stress Test: Evaluating Robustness under Extreme Data Corruption

The central promise of **iPaD** is its robustness. To rigorously evaluate this property, the framework is tested under extreme levels of data corruption, specifically targeting both the graph structure (edge missing) and node attributes (feature missing).

5.5.1 Performance under Edge Missing

The framework' s resilience to structural corruption is first evaluated. The results, presented in Tables 5.7 and 5.8, reveal a remarkable finding: the distilled **iPaD**' student model demonstrates superior resilience, often **outperforming its own GNN teacher** when the graph structure is heavily damaged.

This is particularly evident under a high 90% edge missing rate. For example, on the Photo dataset, while the "TunedGCN" teacher's accuracy drops sharply to 76.42%, the "iPaD-GCN" student retains a significantly higher accuracy of 80.39%, surpassing its teacher by nearly 4%. This pattern of superior tolerance to structural corruption is observed across multiple datasets, confirming **iPaD**'s clear advantage in these scenarios.

Dataset	**Full edge**	**30% Missing**	**60% Missing**	**90% Missing**
Cora	83.77	79.38	72.98	59.32
CiteSeer	72.33	68.29	64.29	59.51
PubMed	80.43	79.64	75.58	70.57
Computers	93.54	92.79	92.06	87.47
Photo	95.86	95.62	95.10	93.18
WikiCS	80.52	79.61	77.21	72.79
CS	80.52	79.61	77.21	72.79
Physic	97.36	97.04	96.35	95.74
Average	**85.54**	**84.00 1.54** ↓	**81.35 4.19** ↓	**76.42 9.12** ↓

Table 5.7: Test accuracy of TunedGCN [1] on various datasets under different edge missing ratios. The table reports accuracy under full edge, 30%, 60%, and 90% edge missing conditions. Average accuracy and relative decreases are shown at the bottom.

Dataset	**Full edge**	**30% Missing**	**60% Missing**	**90% Missing**
Cora	85.02	82.39	79.62	67.66
CiteSeer	75.37	72.48	70.44	66.37
PubMed	82.09	82.05	78.87	74.11
Computers	92.90	92.60	92.18	90.11
Photo	96.16	95.97	95.94	94.44
WikiCS	81.11	80.30	79.72	77.19
CS	81.11	80.30	79.72	77.19
Physic	97.31	96.43	96.63	96.04
Average	**86.38**	**85.32 1.07** ↓	**84.14 2.24** ↓	**80.39 5.99**↓

Table 5.8: Test accuracy of iPaD-GCN on various datasets under different edge missing ratios. The table reports accuracy under full edge, 30%, 60%, and 90% edge missing conditions. Average accuracy and relative decreases are provided at the bottom.

5.5.2 Performance under Feature Missing

The most challenging scenario involves evaluating the framework' s resilience when node features are severely corrupted. This is investigated under two distinct patterns—uniform and structural missingness—at an extreme rate of 99.5%.

Across all four testbeds, a clear and consistent narrative emerges: the performance of the final distilled student model (**iPaD-GCN**) is fundamentally governed by the performance of its teacher (the **MPP-GCN(tuned)** model). This demonstrates a highly successful and faithful knowledge transfer, while also highlighting the inherent limits of distillation when input information is scarce.

Under Uniform Feature Missing.

As shown in Table 5.9 (classic datasets) and Table 5.10 (medium datasets), our framework shows remarkable stability. For instance, on the medium datasets, the **MPP-SAGE(tuned)** teacher achieves an average accuracy of 86.69%. **iPaD-SAGE** student closely follows at 86.53%, indicating a near-perfect transfer of knowledge with almost no performance degradation during distillation. This pattern holds true for the GCN backbone as well.

Model	Cora	CiteSeer	PubMed	Accuracy	Avg Rank
GCN w Zero	23.29 ± 0.78	16.70 ± 1.99	44.04 ± 0.94	23.29	9.8
GCN w FP	68.55 ± 0.21	44.72 ± 0.46	70.97 ± 0.22	68.55 **45.3**↑	8.0
TunedGCN w FP	74.46 ± 0.47	51.61 ± 1.29	72.68 ± 0.74	74.46 **51.2**↑	5.3
TunedGCN w APCFI(our)	**75.55 ± 0.80**	51.20 ± 0.73	**75.22 ± 0.68**	**75.22 51.9**↑	**2.7**
iPaD-GCN(our)	**75.21 ± 0.36**	**53.96 ± 1.27**	72.10 ± 4.90	**75.21 51.9**↑	**3.3**
GraphSAGE w Zero	34.14 ± 1.34	23.98 ± 1.05	39.92 ± 2.66	34.14	9.3
GraphSAGE w FP	71.24 ± 0.47	50.77 ± 0.85	74.87 ± 0.68	71.24 **37.1**↑	5.3
TunedSAGE w FP	74.10 ± 1.56	53.30 ± 0.85	73.26 ± 1.05	74.10 **40.0**↑	5.3
TunedSAGE w APCFI(our)	**74.90 ± 1.22**	**54.72 ± 0.51**	**76.47 ± 0.62**	**74.90 40.8**↑	**2.0**
iPaD-SAGE(our)	74.52 ± 0.56	**53.94 ± 0.82**	**76.19 ± 0.75**	74.52 **40.4**↑	**3.3**

Table 5.9: Comparison under 99.5% uniform feature missing: vanilla GNN, MPP-Tuned, and iPaD with common imputation methods on classic datasets. Model names indicate the imputation strategy. The best results are highlighted in **green**, the second-best in **cyan**, and the third-best in **red**.

Model	Computer	Photo	WikiCS	CS	Physics	Accuracy	Avg Rank
GCN w Zero	38.50 ± 0.04	28.09 ± 0.19	51.03 ± 0.26	22.74 ± 0.00	50.41 ± 0.00	38.15	9.8
GCN w FP	38.51 ± 0.38	65.25 ± 4.78	71.36 ± 0.31	23.13 ± 1.17	79.25 ± 3.33	55.50 **17.4**↑	8.8
TunedGCN w FP	**88.81 ± 0.34**	71.48 ± 1.38	74.51 ± 0.35	84.00 ± 0.51	**94.60 ± 0.21**	82.68 **44.5**↑	5.0
TunedGCN w APCFI(our)	**88.19 ± 0.29**	**86.47 ± 1.13**	**76.64 ± 0.34**	**91.09 ± 0.15**	94.52 ± 0.16	**87.38** **49.2**↑	**2.0**
iPaD-GCN(our)	86.72 ± 0.36	86.25 ± 0.29	74.49 ± 0.32	90.00 ± 0.15	**94.87** ± 0.08	**86.47** **48.3**↑	4.5
GraphSAGE w Zero	55.26 ± 3.42	72.07 ± 1.56	42.94 ± 0.63	60.72 ± 3.67	50.41 ± 0.00	56.28	8.3
GraphSAGE w FP	79.74 ± 0.45	**86.48 ± 0.41**	71.47 ± 0.48	89.77 ± 0.35	86.67 ± 1.73	82.83 **26.6**↑	5.5
TunedSAGE w FP	**86.05 ± 0.35**	74.72 ± 3.51	**90.23 ± 0.55**	87.08 ± 0.50	85.18 ± 2.54	85.18 **28.9**↑	4.0
TunedSAGE w APCFI(our)	86.69 ± 0.35	86.20 ± 3.39	**75.12 ± 0.22**	**90.80 ± 0.16**	92.69 ± 1.42	86.30 **30.0**↑	**3.8**
iPaD-SAGE(our)	84.57 ± 0.68	**87.73 ± 2.15**	73.63 ± 0.41	**91.69 ± 0.27**	**94.59 ± 0.25**	**86.44** **30.2**↑	**3.5**

Table 5.10: Comparison under 99.5% uniform feature missing: vanilla GNN, MPP-Tuned, and iPaD with common imputation methods on medium datasets. Model names indicate the imputation strategy. The best results are highlighted in **green**, the second-best in **cyan**, and the third-best in **red**.

Under Structural Feature Missing.

This same trend is observed under the challenging structural missingness scenario, as shown in Table 5.11 and Table 5.12. The **MPP-Tuned** models, equipped with our APCFI imputer, consistently set the performance benchmark. The corresponding **iPaD** student models again track this performance closely. For example, on the medium datasets, the **MPP-GCN(tuned)** teacher scores 84.25%, while the **iPaD-GCN** student achieves a comparable 83.24%.

Model	Cora	CiteSeer	PubMed	Accuracy	Avg Rank
GCN w Zero	23.29 ± 1.00	16.70 ± 0.34	44.04 ± 0.08	28.01	9.7
GCN w FP	68.55 ± 0.90	44.72 ± 0.24	70.97 ± 0.36	61.41 **33.4**↑	6.0
TunedGCN w FP	**70.84 ± 1.06**	46.80 ± 0.70	**73.13 ± 1.18**	**63.59** **35.6**↑	4.0
TunedGCN w APCFI(our)	**70.46 ± 0.75**	50.62 ± 0.57	71.77 ± 0.53	**64.28** **36.3**↑	4.0
iPaD-GCN(our)	60.94 ± 2.51	**52.22 ± 0.53**	72.35 ± 0.35	61.84 **33.8**↑	**3.7**
GraphSAGE w Zero	34.14 ± 0.16	23.98 ± 0.74	41.28 ± 0.36	32.53	9.3
GraphSAGE w FP	**71.24 ± 0.89**	**50.77 ± 0.75**	**74.97 ± 0.48**	**65.66** **32.5**↑	**1.7**
TunedSAGE w FP	51.72 ± 5.41	48.39 ± 1.41	**74.52 ± 0.97**	58.21 **25.1**↑	5.3
TunedSAGE w APCFI(our)	65.96 ± 0.95	50.57 ± 2.00	69.50 ± 3.91	62.01 **28.9**↑	5.7
iPaD-SAGE(our)	59.56 ± 4.11	**52.08 ± 1.60**	64.31 ± 4.63	58.65 **25.5**↑	5.7

Table 5.11: Comparison under 99.5% structural feature missing: vanilla GNN, MPP-Tuned, and iPaD with common imputation methods on classic datasets. Model names indicate the imputation strategy. The best results are highlighted in **green**, the second-best in **cyan**, and the third-best in **red**.

Model	Computer	Photo	WikiCS	CS	Physics	Avg Accuracy	Avg Rank
GCN w Zero	38.54 ± 0.02	28.27 ± 0.20	39.32 ± 0.63	22.74 ± 0.00	50.41 ± 0.00	35.86	9.8
GCN w FP	38.64 ± 0.69	59.77 ± 0.39	63.12 ± 0.32	25.83 ± 0.07	83.35 ± 3.18	54.14 **18.3**↑	8.4
TunedGCN w FP	84.88 ± 0.31	65.65 ± 4.22	**75.46 ± 0.37**	71.24 ± 2.05	92.74 ± 0.14	77.99 **42.1**↑	4.6
TunedGCN w APCFI(our)	**87.97 ± 0.26**	77.92 ± 4.17	**74.13 ± 0.39**	**87.57 ± 0.20**	**93.68 ± 0.13**	**84.25** **48.4**↑	**2.6**
iPaD-GCN(our)	**86.47 ± 0.11**	79.55 ± 4.53	71.63 ± 0.39	85.20 ± 0.26	**93.33 ± 0.06**	83.24 **47.4**↑	**3.6**
GraphSAGE w Zero	47.75 ± 1.38	45.49 ± 4.53	52.71 ± 0.36	33.22 ± 0.64	50.41 ± 0.03	45.92	8.6
GraphSAGE w FP	79.73 ± 1.00	**86.05 ± 0.60**	70.71 ± 0.26	83.79 ± 0.24	84.66 ± 2.51	80.99 **35.1**↑	5.8
TunedSAGE w FP	83.46 ± 1.04	80.93 ± 3.96	72.50 ± 0.65	83.79 ± 0.30	90.88 ± 0.27	82.31 **36.4**↑	4.6
TunedSAGE w APCFI(our)	**85.32 ± 0.51**	**87.35 ± 0.49**	**73.35 ± 0.34**	**88.71 ± 0.24**	89.31 ± 1.44	**84.81** **38.9**↑	**2.8**
iPaD-SAGE(our)	83.36 ± 0.39	**86.46 ± 0.44**	74.74 ± 0.34	**87.93 ± 0.18**	**93.05 ± 0.08**	**84.31** **38.4**↑	3.8

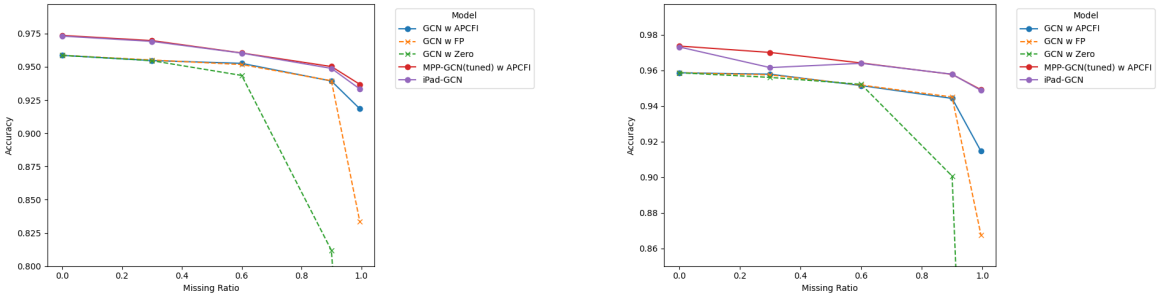
Table 5.12: Comparison under 99.5% structural feature missing: vanilla GNN, MPP-Tuned, and iPaD with common imputation methods on medium datasets. Model names indicate the imputation strategy. The best results are highlighted in **green**, the second-best in **cyan**, and the third-best in **red**.

Interpretation of Results.

Figure 5.3 presents the robustness of various GCN configurations under increasing feature missing ratios on the Physics dataset. Across both structural and uniform missing settings, the **iPaD-GCN** (student) and **MPP-GCN(tuned)** (teacher) models consistently achieve high accuracy, even at an extreme missing rate of 90%. This demonstrates the effectiveness and robustness of the proposed framework in handling severe feature incompleteness.

Notably, the performance of the distilled student model is always on par with, but never surpasses, that of its teacher. This observation highlights a key boundary of knowledge distillation in the context of feature missing: while the student can faithfully inherit the relational knowledge captured by the teacher, it cannot recover information that was never present in the input. Consequently, the quality of feature imputation and the upper bound of the teacher's performance together define a hard ceiling for the student. In contrast, as discussed in the

following section, in the case of edge (structural) missing, the student occasionally surpasses the teacher by filtering out noise through distillation. This distinction underlines the fundamental differences in the limits of distillation under feature versus structural incompleteness.



(a) Performance under Structural Missing.

(b) Performance under Uniform Missing.

Figure 5.3: Robustness comparison of different GCN configurations under varying feature missing ratios on the Physics dataset. Both the 'MPP-GCN(tuned)' teacher and the final 'iPaD-GCN' student model demonstrate superior robustness, maintaining high accuracy even at a 90% missing ratio. In contrast, the baseline 'GCN w/ Zero' imputation suffers a catastrophic performance collapse.

5.6 Dissecting the Framework: An Ablation Study on Core Modules

To validate the central claim that the strength of iPaD arises from the synergistic interaction of its components, a comprehensive ablation study is conducted. Each of the three core modules—the **tuned** backbone, **Mirror Projection Pruning (MPP)**, and **Mirror Projection Knowledge-inspired Reliable Distillation (MP-KRD)**—is systematically enabled to analyze their individual and combined contributions. Experiments are performed on the Photo and CS datasets using both GCN and SAGE backbones. This analysis assesses whether all modules are essential for achieving the optimal balance of accuracy, training efficiency, and inference speed.

5.6.1 Ablation Results with GCN Backbone

Table 5.13 presents the ablation results of the **Tuned**, **MPP**, and **MP-KRD** modules on the Photo and CS datasets (GCN backbone), evaluating node classification accuracy, training speed, and inference latency under various module combinations.

The baseline model (row 1, all modules disabled) demonstrates the standard performance-efficiency trade-off for GCNs. Enabling the Tuned module leads to a substantial increase in accuracy but incurs higher inference time and reduced training speed, highlighting the cost of improved expressiveness. The introduction of the MPP module markedly improves computational efficiency, both in training and inference, while the integration of MP-KRD yields

Tuned	MPP	MP-KRD	Photo			CS		
			Accuracy↑	Training Speed (epoch/s)↑	Inference Time (ms)↓	Accuracy↑	Training Speed (epoch/s)↑	Inference Time (ms)↓
✗	✗	✗	91.43 ± 0.10	118.2 ± 6.33	3.09 ± 0.04	90.28 ± 0.12	45.1 ± 0.18	8.45 ± 0.41
✓	✗	✗	**95.97 ± 0.30**	**28.8 ± 2.24**	**12.69 ± 1.37**	**95.89 ± 0.14**	**23.3 ± 0.42**	**16.55 ± 1.00**
✗	✓	✗	90.69 ± 0.17	166.1 ± 0.69	2.47 ± 0.20	90.04 ± 0.06	77.5 ± 0.27	5.50 ± 0.17
✗	✗	✓	93.94 ± 0.39	165.5 ± 6.33*	1.27 ± 0.25	93.89 ± 0.13	52.1 ± 0.18*	6.03 ± 0.61
✓	✓	✗	95.92 ± 0.12	47.2 ± 1.15	7.52 ± 0.54	96.01 ± 0.08	47.7 ± 0.20	8.16 ± 1.01
✓	✗	✓	**96.21 ± 0.23**	**89.0 ± 2.24***	**2.59 ± 0.34**	**96.34 ± 0.12**	**27.2 ± 0.42***	**10.36 ± 0.58**
✗	✓	✓	93.46 ± 0.31	165.9 ± 0.69*	1.20 ± 0.07	93.79 ± 0.15	75.9 ± 0.27*	3.95 ± 0.34
✓	✓	✓	**96.23 ± 0.22**	**118.4 ± 1.15***	**1.65 ± 0.07**	**96.14 ± 0.10**	**48.9 ± 0.20***	**6.34 ± 0.47**

Table 5.13: Ablation study of tuned, MPP, and MP-KRD modules on the Photo and CS datasets (GCN Model). The table reports accuracy, training speed (epoch/s), and inference time (ms) for different combinations of modules. When MP-KRD is used, (*) indicates the speed measured during knowledge distillation.

further gains in model compactness and inference speed through knowledge distillation.

Of particular note, the configuration that combines all three modules (Tuned, MPP, and MP-KRD) achieves the optimal balance: it consistently yields the highest or near-highest accuracy, fastest inference, and robust training efficiency across both datasets. Intermediate configurations confirm that each module addresses the limitations introduced by the others, and their joint integration delivers the best overall trade-off.

5.6.2 Ablation Results with SAGE Backbone

Table 5.14 presents the ablation results for the combination of Tuned, MPP, and MP-KRD modules on the Photo and CS datasets, using the SAGE backbone. Each configuration is evaluated in terms of accuracy, training speed (epoch/s), and inference time (ms).

The baseline SAGE model, with all modules disabled, establishes a reference for both performance and computational efficiency. Introducing the Tuned module significantly enhances accuracy but results in increased inference latency and reduced training speed, illustrating the classic accuracy-efficiency trade-off. The application of the MPP module effectively recovers computational efficiency, improving both training speed and inference time, although the accuracy does not surpass that of the tuned backbone.

Tuned	MPP	MP-KRD	Photo			CS		
			Accuracy↑	Training Speed (epoch/s)↑	Inference Time (ms)↓	Accuracy↑	Training Speed (epoch/s)↑	Inference Time (ms)↓
✗	✗	✗	95.14 ± 0.20	103.9 ± 0.46	4.31 ± 0.27	95.67 ± 0.10	20.1 ± 0.16	20.60 ± 0.51
✓	✗	✗	**96.04 ± 0.24**	**73.4 ± 2.25**	**5.00 ± 0.41**	**95.94 ± 0.11**	**15.7 ± 0.08**	**25.08 ± 1.56**
✗	✓	✗	95.14 ± 0.32	118.8 ± 1.15	3.53 ± 0.24	95.59 ± 0.10	25.3 ± 0.75	18.20 ± 0.41
✗	✗	✓	94.70 ± 0.39	139.6 ± 0.46*	1.51 ± 0.33	95.75 ± 0.14	32.0 ± 0.16*	9.32 ± 0.50
✓	✓	✗	95.42 ± 1.05	74.7 ± 3.02	4.63 ± 0.10	95.87 ± 0.15	21.3 ± 0.52	20.58 ± 2.10
✓	✗	✓	**96.01 ± 0.21**	**81.2 ± 2.25***	**2.13 ± 0.14**	**96.45 ± 0.12**	**20.2 ± 0.08***	**14.3 ± 1.08**
✗	✓	✓	93.82 ± 1.07	150.5 ± 1.15*	1.31 ± 0.18	95.67 ± 0.16	53.8 ± 0.75*	5.77 ± 0.53
✓	✓	✓	**95.37 ± 1.29**	**78.6 ± 3.02***	**2.12 ± 0.06**	**96.46 ± 0.16**	**35.8 ± 0.52***	**8.35 ± 0.61**

Table 5.14: Ablation study of tuned, MPP, and MP-KRD modules on the Photo and CS datasets (GraphSAGE Model). The table reports accuracy, training speed (epoch/s), and inference time (ms) for different combinations of modules. When MP-KRD is used, (*) indicates the speed measured during knowledge distillation.

Incorporating the MP-KRD module, either alone or in combination with other modules, results in a lightweight student model, characterized by improved inference speed and competitive accuracy. Notably, the complete pipeline—integrating **Tuned**, **MPP**, and **MP-KRD**—achieves the best overall trade-off. This configuration delivers the highest or near-highest accuracy and the fastest inference speed across both datasets. For example, on the CS dataset, this combination yields an accuracy of **96.46%** with an inference time of only **8.35 ms**.

These findings further validate the necessity of a holistic integration of all modules. The joint design not only addresses the individual limitations of each component but also provides a synergistic effect, resulting in a lightweight, accurate, and efficient GNN model suitable for real-world deployment.

5.6.3 Conclusion of Ablation Study

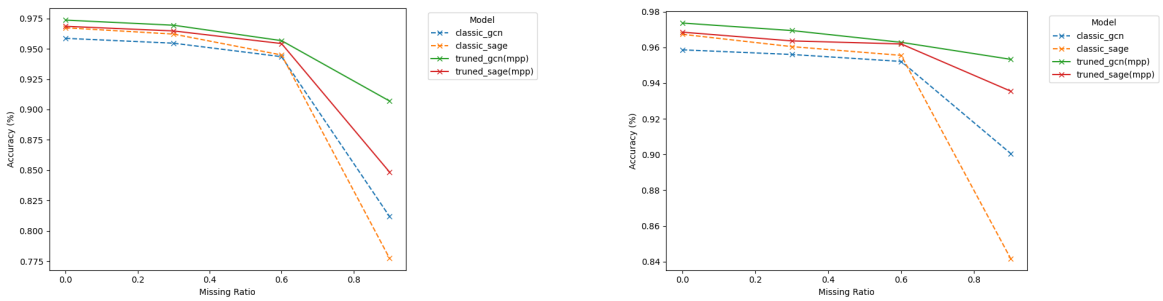
These ablation studies provide compelling evidence that the joint design of the **tuned backbone**, **MPP**, and **MP-KRD** is critical. Each module systematically addresses the limitations introduced by the previous one, demonstrating a powerful synergistic effect. The results confirm that it is the holistic integration of these components, rather than any single module alone, that enables iPaD to achieve its final, optimal balance of accuracy, training efficiency, and inference speed.

5.7 Component-Level Validation: Internal Ablation Studies

Having demonstrated the synergistic success of the overall iPaD pipeline, we now zoom in to validate the specific design choices within our key modules. Targeted ablation studies are conducted to assess the robustness of the tunedGNN architecture, the contribution of the Soft-PC mechanism to APCFI, and the necessity of all components within the MP-KRD module.

5.7.1 Validating the Robustness of the tunedGNN Backbone

This experiment validates our choice of **tunedGNN** as the foundational teacher architecture. We compare its performance against classic GNN counterparts under varying degrees of feature missingness. As illustrated in Figures 5.4, the **tunedGNN** variants consistently and substantially outperform the classic GNNs, especially at high missing rates. The accuracy of classic GNNs often collapses, while the **tunedGNN** models exhibit a much more graceful degradation. This confirms that our chosen architecture provides the essential resilience needed to serve as a reliable teacher in imperfect data conditions.



(a) Comparison under Structural missing.

(b) Comparison under Uniform missing.

Figure 5.4: Robustness comparison of tuned and classic GCN/SAGE architectures on the Physics dataset. All models are evaluated using Zero imputation for missing features, providing a uniform baseline for comparison. Both tuned GCN (green) and tuned SAGE (red) with MPP maintain higher accuracy than their classic counterparts as the missing ratio increases, demonstrating the impact of model tuning and pruning on robustness.

5.7.2 Validating the APCFI Design: A Component-wise Ablation Study

To justify the specific design of the APCFI module, a detailed component-wise ablation study is conducted. This analysis isolates the individual contributions of the three key innovations —**ASDE**, **DJCD**, and **Soft-PC**—in terms of both computational efficiency and final imputation accuracy.

Analysis of Efficiency Contributions

Table 5.15 presents a runtime breakdown of the APCFI components, demonstrating how each innovation contributes to the overall efficiency breakthrough.

The baseline configuration (row 1), which mimics the slow, sequential nature of PCFI, suffers from extremely high runtimes. The results clearly show that:

- Enabling our proposed **ASDE** module (row 2) reduces the SPD runtime catastrophically, for instance, from 148.89s down to a mere 0.021s on the CiteSeer dataset. This validates ASDE as the key to solving the confidence calculation bottleneck.
- Subsequently, enabling **DJCD** (row 3) provides a similar dramatic speedup for the diffusion step, reducing its runtime from 101.7s to just 0.057s. This confirms that our joint channel diffusion is the key to efficient feature propagation.

ASDE	DJCD	Soft-PC	CiteSeer		Photo	
			SPD-RunTime(s)↓	Diffusion-RunTime(s)↓	SPD-RunTime(s)↓	Diffusion-RunTime(s)↓
✗	✗	✗	148.89	101.75	10.18	23.39
✔	✗	✗	0.021	101.7	0.021	23.39
✔	✔	✗	0.021	0.057	0.021	0.23
✔	✔	✔	**0.021**	**0.061**	**0.021**	**0.24**

Table 5.15: Ablation study of APCFI components on runtime performance. The table reports the runtime (in seconds) for the two main stages of imputation (SPD calculation and Diffusion). Each row corresponds to enabling a key innovation, demonstrating how each component contributes to the final efficiency.

Analysis of Accuracy Contribution

Having established the efficiency gains from ASDE and DJCD, we now investigate the role of the final component, **Soft-PC**. Notably, Table 5.15 (comparing rows 3 and 4) shows that adding Soft-PC introduces **negligible computational overhead**. Its contribution lies in enhancing accuracy.

The results of this targeted ablation study, as presented in Table 5.16, reveal a nuanced impact of the Soft-PC mechanism. While the inclusion of Soft-PC results in a marginal performance decrease on three out of five datasets (Cora, CiteSeer, and Photo), it yields a substantial accuracy improvement on the Computers dataset (from 80.65% to 81.63%). This notable gain is sufficient to increase the overall average accuracy from 78.23% to 78.40%. These findings

Method	Cora	CiteSeer	PubMed	Photo	Computers	Average↑
APCFI wo soft-PC	78.35	68.37	75.69	88.09	80.65	78.23
APCFI	78.29	68.31	75.69	88.08	81.63	**78.40**

Table 5.16: Ablation study of soft-PC in APCFI. The table reports test accuracy on Cora, CiteSeer, PubMed, Photo, and Computers datasets for the APCFI method with and without soft-PC. Avg Accuracy indicates the average accuracy across all datasets.

indicate that the adaptive, context-aware weighting provided by Soft-PC is particularly advantageous for specific graph structures, such as the co-purchase network in the Computers dataset. Although the effect is not uniformly positive across all benchmarks, the potential for significant improvement on certain graph topologies, combined with a slight overall performance increase, supports the inclusion of Soft-PC as a valuable component to enhance the general robustness of the **APCFI** model.

Conclusion

In conclusion, this component-wise analysis validates our design. **ASDE** and **DJCD** are responsible for the massive efficiency breakthroughs that make APCFI practical for large-scale graphs, while **Soft-PC** provides a final layer of performance refinement at virtually no additional cost. This justifies the inclusion of all three innovations in our final APCFI model.

5.7.3 Validating the MP-KRD Design

Finally, the complex MP-KRD module is dissected to clarify the contribution of its core components: the reliability-aware curriculum (**KRD**), the structurally-aligned student (**MP-MLP**), and the feature-based loss (**CWD-Loss**).

For classic datasets

Tables 5.17 and 5.18 demonstrate a clear synergistic effect among the core components. For both GCN and SAGE backbones, the combination of all three modules consistently achieves the highest average accuracy. These findings indicate that, even on smaller and more homogeneous graphs, each component makes a distinct and beneficial contribution to the overall distillation process.

KRD	MP-MLP	CWD-Loss	Cora	CiteSeer	PubMed	
			Accuracy (Inference(ms))	Accuracy (Inference(ms))	Accuracy (Inference(ms))	Average Accuracy
✗	✗	✗	79.22 (1.49)	70.13 (1.63)	78.78 (1.46)	76.04
✔	✗	✗	84.65 (1.49)	74.17 (1.63)	**83.02** (1.46)	80.61 **4.57**↑
✔	✔	✗	**84.93** (1.29)	**75.18** (1.37)	**82.67** (1.32)	**80.93** **4.89**↑
✔	✔	✔	**85.02** (1.29)	**75.37** (1.37)	82.09 (1.32)	**80.83** **4.79**↑

Table 5.17: Ablation study of KRD, MP-MLP, and CWD-loss components in the MP-KRD module on classic datasets using the GCN model. The table reports test accuracy on Cora, CiteSeer, and PubMed for different component combinations. Avg Accuracy indicates the average accuracy and relative improvement. The best, and second-best results are highlighted in **green**, and **red**

KRD	MP-MLP	CWD-Loss	Cora	CiteSeer	PubMed	
			Accuracy (Inference(ms))	Accuracy (Inference(ms))	Accuracy (Inference(ms))	Average Accuracy
✗	✗	✗	73.74 (1.57)	62.05 (1.73)	73.58 (2.47)	69.79
✔	✗	✗	80.96 (1.57)	70.39 (1.73)	77.59 (2.47)	76.31 **6.52**↑
✔	✔	✗	**81.27** (1.38)	**72.64** (1.82)	**78.37** (2.30)	**77.43** **7.64**↑
✔	✔	✔	**83.16** (1.38)	**72.78** (1.82)	**78.59** (2.30)	**78.18** **8.39**↑

Table 5.18: Ablation study of KRD, MP-MLP, and CWD-loss components in the MP-KRD module on classic datasets using the SAGE model. The table reports test accuracy on Cora, CiteSeer, and PubMed for different component combinations. Avg Accuracy indicates the average accuracy and relative improvement. The best, and second-best results are highlighted in **green**, and **red**

For Medium-scale datasets

Tables 5.19 and 5.20 present comprehensive ablation results for the MP-KRD module and its key components, evaluated respectively on the GCN and SAGE backbones across multiple benchmark datasets. In both cases, the full configuration—including KRD, MP-MLP, and CWD-loss—consistently delivers the highest or near-highest average accuracy, demonstrating the clear benefit of integrating all three components.

KRD	MP-MLP	CWD-Loss	Computers	Photo	WikiCS	CS	Physics	Average Accuracy
			Accuracy (Inference(ms))	Accuracy (Inference(ms))	Accuracy (Inference(ms))	Accuracy (Inference(ms))	Accuracy (Inference(ms))	
✗	✗	✗	91.30 (1.88)	95.21 (2.24)	79.62 (1.67)	95.97 (6.17)	**97.25** (7.55)	91.82
✔	✗	✗	**91.78** (1.88)	95.64 (2.24)	**80.58** (1.67)	96.12 (6.17)	96.95 (7.55)	**92.21** 0.39↑
✔	✔	✗	91.72 (1.65)	**95.71** (2.43)	79.74 (1.55)	**96.32** (6.34)	97.16 (7.29)	92.13 0.31↑
✔	✔	✔	**93.00** (1.65)	**96.23** (2.43)	**81.11** (1.55)	**96.14** (6.34)	**97.31** (7.29)	**92.76** 0.90↑

Table 5.19: Ablation study of KRD, MP-MLP, and CWD-loss components in the MP-KRD module on medium-scale datasets using the GCN model. The table reports test accuracy on Computers, Photo, WikiCS, CS, and Physics datasets for different component combinations. Avg Accuracy indicates the average accuracy and relative change. The best, and second-best results are highlighted in **green**, and **red**

For the GCN backbone, the complete MP-KRD setup achieves an average accuracy of **92.76%**, outperforming all ablated variants, with notable improvements observed in datasets such as Photo and CS. Similarly, on the SAGE backbone, the full model attains an average accuracy of 91.68%, marginally surpassing the KRD teacher baseline **(91.51%)**. This outcome highlights the remarkable efficacy of MP-KRD: the distilled student not only maintains the teacher' s predictive quality, but in some cases even achieves a slight accuracy gain.

These results confirm that each individual component—knowledge distillation, MLP-based message passing, and curriculum-weighted loss—contributes positively to model performance, while their combination yields synergistic improvements in robustness and accuracy. More importantly, the final MP-KRD distilled student provides efficient, graph-agnostic inference without sacrificing accuracy, reaffirming the framework's practical value for real-world deployment where both efficiency and reliability are critical.

KRD	MP-MLP	CWD-Loss	Computers	Photo	WikiCS	CS	Physics	Average Accuracy
			Accuracy (Inference(ms))	Accuracy (Inference(ms))	Accuracy (Inference(ms))	Accuracy (Inference(ms))	Accuracy (Inference(ms))	
✗	✗	✗	85.18 (2.39)	93.37 (2.42)	77.28 (1.42)	93.68 (6.17)	96.50(10.15)	89.20
✔	✗	✗	**91.37** (2.39)	**95.10**(2.42)	**80.18** (1.42)	93.75 (6.17)	**97.14**(10.15)	**91.51** 2.31↑
✔	✔	✗	87.07 (2.39)	**94.27** (2.12)	**80.52** (1.69)	**96.46** (6.34)	**96.99** (8.49)	91.06 1.86↑
✔	✔	✔	**90.89** (2.39)	94.92 (2.12)	79.60 (1.69)	**96.44** (6.34)	96.50 (8.49)	**91.68** 2.48↑

Table 5.20: Ablation study of KRD, MP-MLP, and CWD-loss components in the MP-KRD module on medium-scale datasets using the SAGE model. The table reports test accuracy on Computers, Photo, WikiCS, CS, and Physics datasets for different component combinations. Avg Accuracy indicates the average accuracy and relative change. The best, and second-best results are highlighted in **green**, and **red**

Chapter 6

Discussion and Conclusion

6.1 Summary of Findings and Contributions

This thesis addressed three interconnected challenges that hinder the practical deployment of Graph Neural Networks: incomplete data, high training costs, and inefficient inference. We argued that isolated solutions are insufficient and proposed that a **holistic, synergistic framework** is necessary to overcome these obstacles simultaneously.

To this end, we presented **iPaD**, a unified framework that seamlessly integrates the entire GNN pipeline. The journey begins with **APCFI**, which robustly and efficiently imputes missing features. It then proceeds to **MPP**, a novel proxy-pruning paradigm that creates a lightweight yet powerful **tunedGNN** teacher and a structurally-aligned MLP student before expensive training commences. Finally, **MP-KRD** completes the pipeline by efficiently distilling the teacher's knowledge into the student, producing a final model optimized for real-world deployment.

Our extensive experiments have validated the effectiveness of this synergistic design, leading to several key findings:

- **APCFI successfully resolves the accuracy-efficiency trade-off**, achieving imputation quality comparable to state-of-the-art methods like PCFI but with a more than 200x speedup in computation.
- **MPP provides a clear "efficiency sweet spot"**, capable of reducing training and inference costs by over 3x and 5x respectively, with negligible impact on model accuracy at moderate prune rates.
- Most significantly, our framework exhibits nuanced robustness properties. The distilled student model can **outperform its GNN teacher under severe edge missingness**, showcasing resilience to structural corruption. Conversely, its performance is **capped by the teacher under feature missingness**, revealing a fundamental boundary condition of knowledge distillation for GNNs.

These contributions collectively demonstrate that a holistic design philosophy can lead to GNN systems that are at once robust, efficient, and high-performing.

6.2 Limitations

Despite its demonstrated strengths, this work has several limitations that open avenues for future research:

- **Applicability to Graph Types:**
  The current designs of APCFI and MPP are primarily tailored for homogeneous graphs. Their effectiveness and generalization capability on highly heterogeneous graphs or graphs with diverse node and edge types have not yet been comprehensively validated.
- **Model and Task Generalization:**
  While iPaD shows strong results on node classification, its applicability to other critical graph tasks, such as graph classification and link prediction, remains to be explored. Furthermore, its performance on industrial-scale graph datasets with billions of nodes and edges has not been tested.
- **Automation and Hyperparameters:**
  The framework, particularly the MP-KRD module, involves several hyperparameters that currently require manual tuning. A fully adaptive, automated mechanism for hyperparameter selection is still lacking.
- **"tunedGNN" Cost:**
  Although MPP effectively mitigates its cost, the underlying **tunedGNN** architecture is inherently more computationally expensive than simpler GNNs, which presents a baseline overhead.

6.3 Future Work

The limitations of this study naturally point toward several exciting directions for future research:

- **Unlocking Heterogeneous Graphs:**
  A primary goal is to extend the principles of APCFI and MPP to be applicable to heterogeneous graphs, which are common in real-world industrial scenarios.
- **Towards a Fully Automated Pipeline:**
  Integrating AutoML techniques, especially for automated hyperparameter optimization (e.g., for the distillation loss weights λ_1, λ_2), would significantly lower the barrier to entry and enhance the framework's practical usability.
- **Expanding to New Tasks and Scales:**
  A crucial next step is to adapt and validate the iPaD pipeline for other graph learning tasks, such as link prediction and graph classification, and to benchmark its perfor-

mance on industrial-scale datasets.

- **Deepening the Understanding of Distillation Robustness:**
  Building upon our key finding regarding edge vs. feature missingness, a promising theoretical direction is to further investigate the boundary conditions of knowledge distillation for GNNs, formally characterizing when and why a distilled student can be more robust than its teacher.

6.4 Concluding Remarks

In conclusion, this thesis presented iPaD, a comprehensive solution to enhance the practicality of Graph Neural Networks. More than just a collection of algorithms, iPaD embodies a design philosophy centered on synergy and end-to-end optimization. By demonstrating that it is possible to achieve robustness, training efficiency, and inference speed simultaneously without compromising performance, this work provides a valuable blueprint and a solid foundation for the next generation of GNN systems designed for the complexities of the real world.

References

[1] Y. Luo, L. Shi, and X.-M. Wu, "Classic gnns are strong baselines: Reassessing gnns for node classification," *arXiv preprint arXiv:2406.08993*, 2024.

[2] L. Waikhom and R. Patgiri, "A survey of graph neural networks in various learning paradigms: methods, applications, and challenges," *Artificial Intelligence Review*, vol. 56, no. 7, pp. 6295–6364, 2023.

[3] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, and P. S. Yu, "A comprehensive survey on graph neural networks," *IEEE transactions on neural networks and learning systems*, vol. 32, no. 1, pp. 4–24, 2020.

[4] B. Khemani, S. Patil, K. Kotecha, and S. Tanwar, "A review of graph neural networks: concepts, architectures, techniques, challenges, datasets, applications, and future directions," *Journal of Big Data*, vol. 11, no. 1, p. 18, 2024.

[5] S. Wu, F. Sun, W. Zhang, X. Xie, and B. Cui, "Graph neural networks in recommender systems: a survey," *ACM Computing Surveys*, vol. 55, no. 5, pp. 1–37, 2022.

[6] R. Ying, R. He, K. Chen, P. Eksombatchai, W. L. Hamilton, and J. Leskovec, "Graph convolutional neural networks for web-scale recommender systems," in *Proceedings of the 24th ACM SIGKDD international conference on knowledge discovery & data mining*, pp. 974–983, 2018.

[7] J. Jiang, C. Wilson, X. Wang, W. Sha, P. Huang, Y. Dai, and B. Y. Zhao, "Understanding latent interactions in online social networks," *ACM Transactions on the Web (TWEB)*, vol. 7, no. 4, pp. 1–39, 2013.

[8] H. Zhang, B. Wu, X. Yuan, S. Pan, H. Tong, and J. Pei, "Trustworthy graph neural networks: Aspects, methods, and trends," *Proceedings of the IEEE*, 2024.

[9] K. Huang, J. Zhai, Z. Zheng, Y. Yi, and X. Shen, "Understanding and bridging the gaps in current gnn performance optimizations," in *Proceedings of the 26th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, pp. 119–132, 2021.

[10] A. Mohi ud din and S. Qureshi, "A review of challenges and solutions in the design and implementation of deep graph neural networks," *International Journal of Computers and Applications*, vol. 45, no. 3, pp. 221–230, 2023.

[11] X. Chen, S. Chen, J. Yao, H. Zheng, Y. Zhang, and I. W. Tsang, "Learning on attribute-missing graphs," *IEEE transactions on pattern analysis and machine intelligence*, vol. 44, no. 2, pp. 740–757, 2020.

[12] B. Jiang and Z. Zhang, "Incomplete graph representation and learning via partial graph neural networks," *arXiv preprint arXiv:2003.10130*, 2020.

[13] J. Zhou, G. Cui, S. Hu, Z. Zhang, C. Yang, Z. Liu, L. Wang, C. Li, and M. Sun, "Graph neural networks: A review of methods and applications," *AI open*, vol. 1, pp. 57–81, 2020.

[14] Z. Huan, Y. Quanming, and T. Weiwei, "Search to aggregate neighborhood for graph neural network," in *2021 IEEE 37th International Conference on Data Engineering (ICDE)*, pp. 552–563, IEEE, 2021.

[15] H. Ma, Y. Rong, and J. Huang, "Graph neural networks: Scalability," *Graph Neural Networks: Foundations, Frontiers, and Applications*, pp. 99–119, 2022.

[16] C. Gao, Y. Zheng, N. Li, Y. Li, Y. Qin, J. Piao, Y. Quan, J. Chang, D. Jin, X. He, *et al.*, "A survey of graph neural networks for recommender systems: Challenges, methods, and directions," *ACM Transactions on Recommender Systems*, vol. 1, no. 1, pp. 1–51, 2023.

[17] X. Zheng, Y. Wang, Y. Liu, M. Li, M. Zhang, D. Jin, P. S. Yu, and S. Pan, "Graph neural networks for graphs with heterophily: A survey," *arXiv preprint arXiv:2202.07082*, 2022.

[18] J. You, X. Ma, Y. Ding, M. J. Kochenderfer, and J. Leskovec, "Handling missing data with graph representation learning," *Advances in Neural Information Processing Systems*, vol. 33, pp. 19075–19087, 2020.

[19] E. Rossi, H. Kenlay, M. I. Gorinova, B. P. Chamberlain, X. Dong, and M. M. Bronstein, "On the unreasonable effectiveness of feature propagation in learning on graphs with missing node features," in *Learning on graphs conference*, pp. 11–1, PMLR, 2022.

[20] H. Taguchi, X. Liu, and T. Murata, "Graph convolutional networks for graphs containing missing features," *Future Generation Computer Systems*, vol. 117, pp. 155–168, 2021.

[21] D. Um, J. Park, S. Park, and J. young Choi, "Confidence-based feature imputation for graphs with partially known features," in *The Eleventh International Conference on Learning Representations*, 2023.

[22] X. ZhuЃ and Z. GhahramaniЃн, "Learning from labeled and unlabeled data with label propagation," *ProQuest number: information to all users*, 2002.

[23] H. Zhou, A. Srivastava, H. Zeng, R. Kannan, and V. Prasanna, "Accelerating large scale real-time gnn inference using channel pruning," *arXiv preprint arXiv:2105.04528*, 2021.

[24] D. Gurevin, M. Shan, S. Huang, M. A. Hasan, C. Ding, and O. Khan, "Prunegnn: Algorithm-architecture pruning framework for graph neural network acceleration," in *2024 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, pp. 108–123, IEEE, 2024.

[25] T. Chen, Y. Sui, X. Chen, A. Zhang, and Z. Wang, "A unified lottery ticket hypothesis for graph neural networks," in *International conference on machine learning*, pp. 1695–1706, PMLR, 2021.

[26] Y.-D. Sui, X. Wang, T. Chen, M. Wang, X.-N. He, and T.-S. Chua, "Inductive lottery ticket learning for graph neural networks," *Journal of Computer Science and Technology*, vol. 39, no. 6, pp. 1223–1237, 2024.

[27] Y. Yang, J. Qiu, M. Song, D. Tao, and X. Wang, "Distilling knowledge from graph convolutional networks," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 7074–7083, 2020.

[28] B. Yan, C. Wang, G. Guo, and Y. Lou, "Tinygnn: Learning efficient graph neural networks," in *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pp. 1848–1856, 2020.

[29] S. Zhang, Y. Liu, Y. Sun, and N. Shah, "Graph-less neural networks: Teaching old mlps new tricks via distillation," *arXiv preprint arXiv:2110.08727*, 2021.

[30] L. Wu, H. Lin, Y. Huang, , and S. Z. Li, "Quantifying the knowledge in gnns for reliable distillation into mlps," in *International Conference on Machine Learning*, PMLR, 2023.

[31] Y. Chen, Y. Bian, X. Xiao, Y. Rong, T. Xu, and J. Huang, "On self-distilling graph neural network," *arXiv preprint arXiv:2011.02255*, 2020.

[32] L. Rampášek, M. Galkin, V. P. Dwivedi, A. T. Luu, G. Wolf, and D. Beaini, "Recipe for a general, powerful, scalable graph transformer," *Advances in Neural Information Processing Systems*, vol. 35, pp. 14501–14515, 2022.

[33] J. Chen, K. Gao, G. Li, and K. He, "Nagphormer: A tokenized graph transformer for node classification in large graphs," *arXiv preprint arXiv:2206.04910*, 2022.

[34] H. Shirzad, A. Velingker, B. Venkatachalam, D. J. Sutherland, and A. K. Sinop, "Exphormer: Sparse transformers for graphs," in *International Conference on Machine Learning*, pp. 31613–31632, PMLR, 2023.

[35] K. Kong, J. Chen, J. Kirchenbauer, R. Ni, C. B. Bruss, and T. Goldstein, "Goat: A global transformer on large-scale graphs," in *International Conference on Machine Learning*, pp. 17375–17390, PMLR, 2023.

[36] Q. Wu, W. Zhao, Z. Li, D. P. Wipf, and J. Yan, "Nodeformer: A scalable graph structure learning transformer for node classification," *Advances in Neural Information Processing Systems*, vol. 35, pp. 27387–27401, 2022.

[37] Q. Wu, W. Zhao, C. Yang, H. Zhang, F. Nie, H. Jiang, Y. Bian, and J. Yan, "Sgformer: Simplifying and empowering transformers for large-graph representations," *Advances in Neural Information Processing Systems*, vol. 36, pp. 64753–64773, 2023.

[38] C. Deng, Z. Yue, and Z. Zhang, "Polynormer: Polynomial-expressive graph transformer in linear time," *arXiv preprint arXiv:2403.01232*, 2024.

[39] K. Sharma, Y.-C. Lee, S. Nambi, A. Salian, S. Shah, S.-W. Kim, and S. Kumar, "A survey of graph neural networks for social recommender systems," *ACM Computing Surveys*, vol. 56, no. 10, pp. 1–34, 2024.

[40] X.-M. Zhang, L. Liang, L. Liu, and M.-J. Tang, "Graph neural networks and their current applications in bioinformatics," *Frontiers in genetics*, vol. 12, p. 690049, 2021.

[41] H. Zhang, B. Wu, X. Yuan, S. Pan, H. Tong, and J. Pei, "Trustworthy graph neural networks: Aspects, methods, and trends," *Proceedings of the IEEE*, vol. 112, no. 2, pp. 97–139, 2024.

[42] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, and P. S. Yu, "A comprehensive survey on graph neural networks," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 32, no. 1, pp. 4–24, 2021.

[43] X. Jiang, Z. Tian, and K. Li, "A graph-based approach for missing sensor data imputation," *IEEE Sensors Journal*, vol. 21, no. 20, pp. 23133–23144, 2021.

[44] D. Adhikari, W. Jiang, J. Zhan, Z. He, D. B. Rawat, U. Aickelin, and H. A. Khorshidi, "A comprehensive survey on imputation of missing data in internet of things," *ACM Computing Surveys*, vol. 55, no. 7, pp. 1–38, 2022.

[45] Z. Lin, C. Li, Y. Miao, Y. Liu, and Y. Xu, "Pagraph: Scaling gnn training on large graphs via computation-aware caching," in *Proceedings of the 11th ACM Symposium on Cloud Computing*, pp. 401–415, 2020.

[46] D. Zheng, C. Ma, M. Wang, J. Zhou, Q. Su, X. Song, Q. Gan, Z. Zhang, and G. Karypis, "Distdgl: Distributed graph neural network training for billion-scale graphs," in *2020 IEEE/ACM 10th Workshop on Irregular Applications: Architectures and Algorithms (IA3)*, pp. 36–44, IEEE, 2020.

[47] Z. Xue, Y. Yang, and R. Marculescu, "Sugar: Efficient subgraph-level training via resource-aware graph partitioning," *IEEE Transactions on Computers*, vol. 72, no. 11, pp. 3167–3177, 2023.

[48] W. Lu, Z. Guan, W. Zhao, and Y. Yang, "Adagmlp: Adaboosting gnn-to-mlp knowledge distillation," in *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pp. 2060–2071, 2024.

[49] X. Han, T. Zhao, Y. Liu, X. Hu, and N. Shah, "MLPInit: Embarrassingly simple GNN training acceleration with MLP initialization," in *International Conference on Learning Representations*, 2023.

[50] G. Fang, X. Ma, M. Song, M. B. Mi, and X. Wang, "Depgraph: Towards any structural pruning," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 16091–16101, 2023.

[51] J. Lee, S. Park, S. Mo, S. Ahn, and J. Shin, "Layer-adaptive sparsity for the magnitude-based pruning," *arXiv preprint arXiv:2010.07611*, 2020.

[52] C. Shu, Y. Liu, J. Gao, Y. Zheng, and C. Shen, "Channel-wise knowledge distillation for dense prediction," in *ICCV*, 2021.

[53] P. Sen, G. Namata, M. Bilgic, L. Getoor, B. Galligher, and T. Eliassi-Rad, "Collective classification in network data," *AI magazine*, vol. 29, no. 3, pp. 93–93, 2008.

[54] O. Shchur, M. Mumme, A. Bojchevski, and S. Günnemann, "Pitfalls of graph neural network evaluation," *arXiv preprint arXiv:1811.05868*, 2018.

[55] P. Mernyei and C. Cangea, "Wiki-cs: A wikipedia-based benchmark for graph neural networks," *arXiv preprint arXiv:2007.02901*, 2020.

[56] F. Monti, M. Bronstein, and X. Bresson, "Geometric matrix completion with recurrent multi-graph neural networks," *Advances in neural information processing systems*, vol. 30, 2017.

[57] T. Akiba, S. Sano, T. Yanase, T. Ohta, and M. Koyama, "Optuna: A next-generation hyperparameter optimization framework," in *The 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pp. 2623–2631, 2019.

Appendix A

hyperparameters

This appendix provides a detailed analysis of the key hyperparameters for our proposed modules, justifying the settings used in our main experiments. To ensure a systematic and unbiased selection, we utilized the **Optuna** [57] to perform automated hyperparameter optimization (HPO). For each major experiment, we ran over 100 trials, optimizing for the validation set accuracy. The following sections present the insights gained from this process.

A.1 Analysis of APCFI Hyperparameters

For our **APCFI** module, we analyzed the two most sensitive hyperparameters: the confidence decay factor alpha and the inter-channel propagation strength beta. The contour plot in Figure A.1 visualizes their complex interaction.

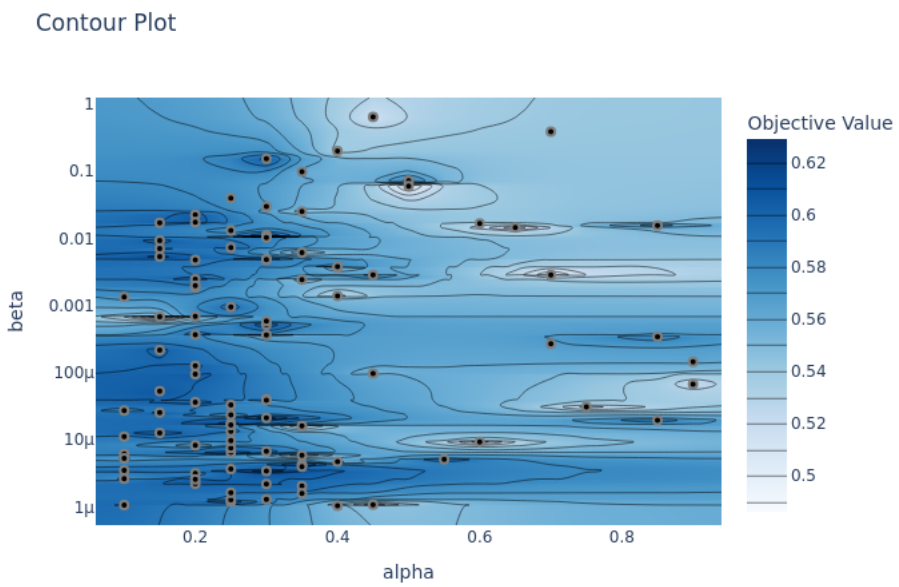


Figure A.1: Contour plot for the key hyperparameters (alpha and beta) of the APCFI module on the Cora dataset.

The key insight from this plot is the **strong interaction between alpha and beta**. The optimal value for one parameter is highly dependent on the value of the other, leading to several

islands of high performance. For instance, when **beta** is very low (around 10^{-5}), a lower **alpha** (around 0.2-0.4) can achieve good results. However, the largest and most stable high-performance region occurs when **alpha is in a mid-to-high range (0.5 to 0.8)** and **beta is in a low-to-mid range (around 10^{-3})**. This analysis demonstrates that while the module is sensitive to this parameter combination, its behavior is predictable and optimizable, and it provides a clear guideline for selecting a robust set of default parameters.

A.2 Analysis of MP-KRD Hyperparameters

For our **MP-KRD** module, we identified **alpha**, **lamb**, and **momentum** as three critical hyperparameters influencing the distillation curriculum and training dynamics. Figure A.2 shows the contour plot illustrating the pairwise interactions of these parameters on the final model accuracy.

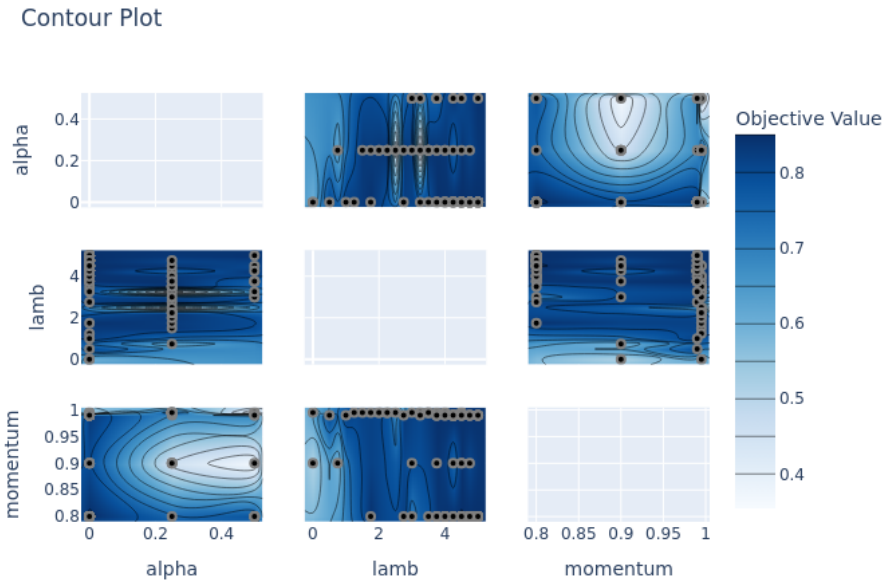


Figure A.2: Contour plot for the key hyperparameters of the MP-KRD module. Darker blue regions indicate higher accuracy. The plot reveals strong dependencies between parameters.

The analysis reveals several clear trends. The most decisive parameter is **momentum**, where higher values (approximately 0.85 to 1.0) consistently correlate with superior performance. Conversely, the **lamb** parameter performs best in a lower range (approximately 0 to 2.5). The impact of **alpha** is more subtle and interacts with the other two. Overall, the plot indicates that the optimal performance **sweet spot** is achieved with a combination of **high momentum**, **low lamb**, and **mid-to-low alpha**. This data-driven insight guided our final parameter selection for the MP-KRD experiments.