Doctoral Dissertation

STUDY ON KNOWLEDGE GRAPH REPRESENTATION LEARNING WITH GRAPH NEURAL NETWORK

CAO, Yiming

Supervisor NGUYEN, Minh Le

Graduate School of Advanced Science and Technology
Japan Advanced Institute of Science and Technology
(Information Science)

March 2026

Abstract

Graph Neural Networks have emerged as the standard paradigm for representation learning on graph date. However, the practical application to real-world, large-scale graphs faces three main challenges: (1) noisy or uninformative graph topologies, (2) the prohibitive cost of generating high-quality node features, and (3) the difficulty of scaling to massive, heterogeneous graphs. This dissertation attempts to investigates structured models that are able to systematically addresses these challenges.

First, we address the challenge of learning on noisy topologies by proposing SSDA Framework, a differentiable, self-supervised data augmentation framework. SSDA jointly learns optimal policies for both structural and attribute augmentation in an end-to-end manner, enhancing the robustness of GNN models.

Second, we propose the LME framework to achieve a balance in the quality of LLM generated features and the computational cost. We conduct an analysis of feature engineering strategies for Text-Attributed Graphs. Our proposed LME pipeline, which utilizes an LLM as an intelligent keyword extractor to generate sparse features for the nodes, achieves over 90% of SOTA performance while being 2.4× smaller in storage and enabling 3.7× faster downstream GNN training.

Finally, to apply on massive heterogeneous graphs, we introduce the LME-Prop model. This scalable framework propagates the efficient LME sparse features from a 10% anchor set to the entire graph. The core innovation is a Dual-Channel Prop-GNN that fuses signals from the original topology (Structural Channel) and a new, efficiently-constructed k-NN graph (Semantic Channel). We demonstrate on the large-scale `ogbn-mag` benchmark that LME-Prop recovers over 80% of the full-graph SOTA performance while reducing total featurization time by 4.3×.

In conclusion, this dissertation establishes a complete, efficient, and scalable methodology for deploying semantically-aware GNNs on real-world, industrial scale graphs.

Keywords: Graph neural network, Graph representation learning, Data augmentation, Large language models, Heterogeneous graph, Computational efficiency.

Acknowledgment

I would like to express my gratitude to my supervisor, Professor NGUYEN Minh Le, for his continuous guidance, invaluable advice, and unwavering support throughout my doctoral studies. His profound knowledge, rigorous academic attitude, and insightful suggestions have been essential to the completion of this dissertation. I am also sincerely thankful for the freedom and encouragement he provided, which allowed me to develop my research independently while maintaining academic rigor.

I would also like to thank all the members of Nguyen Laboratory for their collaboration, discussions, and support. The stimulating research environment and the constructive feedback from my colleagues greatly contributed to the progress of my work. I am grateful for the cooperation, shared ideas, and the friendly atmosphere that made my time in the laboratory both productive and enjoyable.

Finally, I would like to express my heartfelt gratitude to my parents for their unconditional love, patience, and encouragement throughout my academic journey. Their constant support and understanding have been a source of strength and motivation, especially during challenging times. This dissertation would not have been possible without their belief in me.

List of Figures

List of Tables

Contents

Chapter 1

Introduction

1.1 Background and Motivation

In the past decade, with the development of the neural network models, we have witnessed a significant boost in data representation learning. However, while deep learning has achieved significant success in processing Euclidean data, such as the grid-like structures of images and the linear sequences of natural language, extending this success to the non-Euclidean domain of graphs remains challenging. From social networks and financial transactions [1] to protein-protein interaction (PPI) networks [2] and bibliographic citations [3], graph-structured data is ubiquitous. The central challenge in this domain is that graphs are non-Euclidean, rendering traditional machine learning models like Convolutional Neural Networks (CNNs) and Recurrent Neural Networks (RNNs) directly inapplicable.

This challenge is addressed by the development of Graph Neural Networks (GNNs) [3–5], a powerful class of deep learning models designed to learn representations directly from graph topology. As established in the foundational work, a GNN's function can be described as:

$$\mathbf{H}^{(K)} = f_{GNN}(\mathbf{A}, \mathbf{X}) \tag{1.1}$$

Where $\mathbf{A}$ is the adjacency matrix, $\mathbf{X}$ is the initial node feature matrix (the attributes), and f_{GNN} is the propagation architecture for gathering message from neighbors.

However, while the research community has invested in designing increasingly complex architectures for f_{GNN}, ranging from advanced message passing schemes [6, 7] to recent Graph Transformers [8, 9], we focus on addressing the challenges that arise when applying Graph Neural Networks to real-world graph data. Specifically, we identify two fundamental problems that remain unsolved:

1. **The Reliability Problem:** The provided graph topology $\mathbf{A}$ is often noisy, incomplete, or task-irrelevant.

2. **The Representation-Scalability Gap:** The node attributes $\mathbf{X}$, especially for text-rich graphs, face a problem: they are either computationally cheap but semantically weak (e.g., TF-IDF), or semantically powerful but prohibitively expensive to scale (e.g., LLM embeddings).

Our research attempts to addresses these bottlenecks, focusing on optimizing what the model learns from and how to efficiently construct high-quality inputs at scale.

This motivation leads to three main challenges that form the nature of graph data:

Challenge 1: The Incomplete Topology The first assumption in $f_{GNN}(\mathbf{A}, \mathbf{X})$ is that the topology $\mathbf{A}$ is a reliable source of information. In real world graphs, this assumption may not be valid. Graphs are often incomplete and contain noise when created. Furthermore, as demonstrated in previous work [10], GNNs are highly sensitive to the parameters of data augmentation applied to $\mathbf{A}$. The field has been dominated by manual augmentation strategies that are sub-optimal and non-adaptive. This creates a need for an automated, learnable approach to optimize the learning process.

Challenge 2: The Missing Feature The second assumption is the existence of a high-quality feature matrix $\mathbf{X}$. The performance of GNNs is highly dependent on the quality of node features. For Text-Attributed Graphs, $\mathbf{X}$ are usually constructed from raw text. This causes a problem:

1. **Traditional Lexical Features (TLF)** (e.g., TF-IDF [11]) are fast and sparse, but semantically weak and yield poor performance.
2. **Full Semantic Features (FSF)** (e.g., SBERT [12]) are semantically powerful and yield SOTA performance, but are extremely expensive ($O(ND)$ storage, slow GNN training).

This forces researchers into a choice between a weak, cheaper solution and a powerful but expensive models.

Challenge 3: The Scalability Problem The emergence of Large Language Models (LLMs) (e.g., [13]) solves the feature quality problem. At the same time, we meet a scalability crisis when applying LLM featuring models on large scale graphs. This cost is computationally unaffordable for real world industrial graphs with billions of nodes. This problem is the main barrier to apply modern, semantically-aware GNNs at scale, especially on heterogeneous graphs (HGs) [14, 15], which are the dominant form of industrial data.

1.2 Research Objectives

Motivated by the challenges stated above, this dissertation aims to develop a unified, data-centric framework for robust and scalable graph representation learning.

The primary objectives are:

1. **Data augmentation framework for graph data** To implement a data augmentation framework that can automatically learn optimal augmentation policies for graph data.
2. **LLM-driven feature generation model** This involves designing an LLM-driven feature engineering pipeline and balance the quality of generated features and computational cost.
3. **Expand Scalability on large scale graph** This involves designing a novel framework that can handle the real world large scale graph data at a reasonable cost.

1.3 Research Framework

To address the challenges stated above, this thesis constructs three corresponding frameworks.

Framework 1: SSDA (Self-Supervised Data Augmentation) Presented in Chapter 3, this framework addresses the challenge in incomplete topology and missing feature: We introduces a differentiable, end-to-end learning framework. Instead of using fixed hyperparameters, SSDA treats augmentation rates (e.g., edge dropping rate and feature masking rate) as learnable parameters. It employs a Neural Edge Predictor and a feature reconstruction module, which are optimized jointly with the GNN's objective via backpropagation, allowing the augmentation strategy to adapt to the specific topology and attributes of the data.

Framework 2: LME (LLM-Sourced Lexical Feature Generation Framework) Presented in Chapter 4, this framework addresses the challenge in balancing the quality of generated features and computational cost. It is a comprehensive study designed to identify the optimal feature representation for GNNs. We compare three pipelines: (1) Traditional Lexical Features, (2) Full Semantic Features, and (3) our proposed LME framework. LME framework leverages a generative LLM not as a dense encoder, but as an keyword extractor. We prove that this pipeline generates

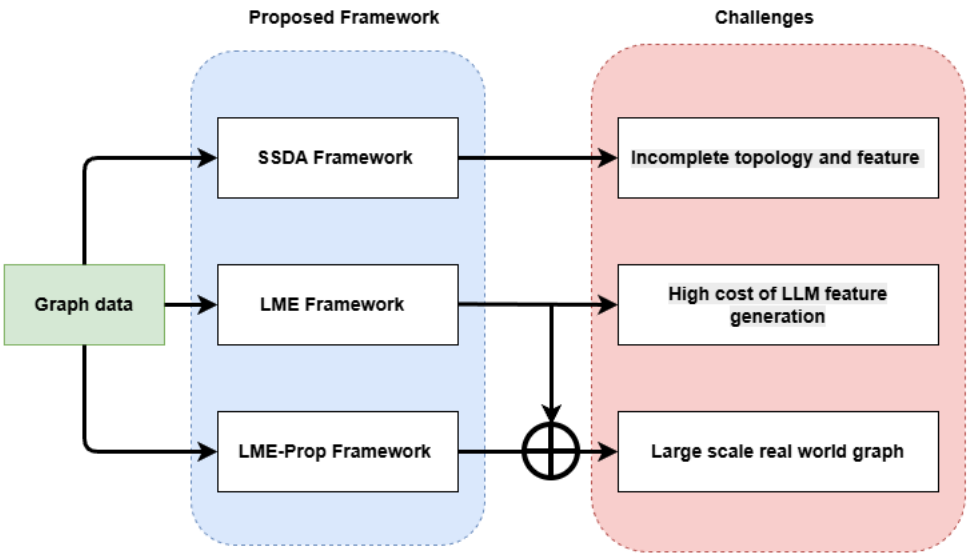

Figure 1.1: The overall map for the proposed models and the corresponding challenges.

sparse, low-cost features that capture the vast majority of the task-relevant semantic signal.

Framework 3: LME-Prop (LME Dual-Channel Propagation) Presented in Chapter 5, this framework addresses the challenge of scalability. It is a scalable, pre-processing framework to generate features for massive heterogeneous graphs. This model operates in three stages:

1. **Anchor Selection:** It selects a small $k\%$ of anchor nodes via centrality (PageRank) and generates high quality sparse features by LLMs.
2. **Dual-Channel Construction:** It builds two parallel sub graphs: (a) a structural channel ($\mathbf{A}_{\text{orig}}$) that respects the original heterogeneous edge types, and (b) a semantic channel ($\mathbf{A}_{\text{semantic}}$) built via efficient sparse k-NN, connecting nodes to their most semantically-similar anchors.
3. **Fused Propagation:** A novel, lightweight `Prop-GNN` intelligently fuses signals from both channels via a gated mechanism to interpolate high-quality features for the entire graph.

1.4 Contributions

The primary contributions of this dissertation are:

1. **A novel graph data augmentation framework (SSDA).** We design and validate an end-to-end framework that learns adaptive augmentation policies to generate high quality features for data augmentation.
2. **Investigated a balanced solution for LLM based Feature generation.** We conducted a cost-benefit analysis of LLM-sourced GNN feature generation models. We empirically prove that sparse, LLM-extracted keywords achieve ∼90-99% of SOTA (FSF) performance while being 2.4× smaller and enabling 3.7× faster GNN training.
3. **A novel, scalable propagation framework for large scale heterogeneous graphs (LME-Prop).** We design a dual-channel, anchor-based propagation framework that (a) explicitly handles graph heterogeneity via a simplified R-GCN channel and (b) leverages sparse semantic features via an efficient k-NN channel. We investigated the possible solution to apply LLM based feature generation framework on large scale real world graphs.
4. **A comprehensive validation of scalable GNN-LLM integration.** We demonstrate on large-scale benchmarks and conclude that our LME-Prop framework ia able to reduce the pre-processing time by 4.3× and recover over 80% of the SOTA performance while maintaining the fast training and low storage benefits of sparse features.

1.5 Thesis Organization

The remainder of this thesis is organized as follows:

- **Chapter 2 (Literature Review):** Provides a comprehensive review of the foundational concepts, including Graph Neural Networks, heterogeneous graph representation learning, graph data augmentation, and the emerging intersection of GNNs and Large Language Models.
- **Chapter 3 (SSDA Framework):** describes the design, methodology, and experimental validation of our self-supervised data augmentation framework.
- **Chapter 4 (LME Framework):** Presents our investigation into a cost-performance balanced solution, the LME pipeline, to apply LLM based feature generation.

- **Chapter 5 (LME-Prop Framework):** Describes our scalable, dual-channel propagation framework, which investigate the application of LLM based feature generation on large scale real world graphs.
- **Chapter 6 (Conclusion and Future Work):** Summarizes the contributions of this dissertation, and proposes promising directions for future research.

Chapter 2

Literature Review

2.1 Introduction

This chapter provides a comprehensive review of the theoretical foundations and recent advancements under the research in this dissertation. We structure the review to follow the logical progression of the field: starting from the fundamental shift in machine learning and graph representation learning, then we narrow our focus to Graph Neural Networks, the dominant paradigm for handling non-Euclidean data. We then analyze the evolution of Graph Data Augmentation as a means to enhance model robustness and improve performance. Finally, we examine the challenges of learning on large-scale heterogeneous graphs and the emerging frontier of utilizing Large Language Models for graph feature engineering.

2.2 From Machine Learning to Graph Representation Learning

2.2.1 The Evolution of Machine Learning

The evolution of Machine Learning can be characterized as a shift from manual feature engineering to automated representation learning.

- **Traditional ML:** Early approaches (e.g., SVMs, Random Forests) relied on domain experts to manually extract feature vectors. In the context of graphs, this involved calculating heuristic statistics such as node degrees, clustering coefficients, or motif counts [16]. These methods were labor-intensive and often failed to capture high-order structural patterns.
- **Deep Learning:** The advent of Deep Learning [17] revolutionized this process by learning representations end-to-end. Convolutional Neural Networks (CNNs) achieved breakthroughs in Computer Vision by exploiting the grid-like structure of images (shift invariance and

local connectivity), while Recurrent Neural Networks (RNNs) and Transformers [18] successfully achieved a boost in tasks with sequential data (NLP).

2.2.2 The Challenge of Graph Data

Despite these successes, applying deep learning to graph data is still challenging. Graphs are fundamentally non-Euclidean. Unlike images or text, graphs lack a fixed coordinate system, possess irregular neighborhoods, and exhibit permutation invariance (the ordering of nodes does not change the graph structure). These properties render standard CNNs and RNNs inapplicable, leading to a demand for the development of specialized architectures for Graph Representation Learning.

2.3 Graph Neural Networks: Foundations and Architectures

Graph Neural Networks (GNNs) have emerged as the de facto standard for handling graph data [19]. They generalize the concept of convolution to irregular graph domains.

2.3.1 Spectral Graph Convolutions

The theoretical foundation of GNNs lies in Spectral Graph Theory. The convolution operation was initially defined in the Fourier domain using the eigen-decomposition of the Graph Laplacian $\mathbf{L}$.

- **Spectral CNN [20]:** This early work proposed filtering the graph signals in the spectral domain. However, it required computing the full eigen-decomposition of $\mathbf{L}$, leading to a prohibitive computational complexity of $O(N^3)$, which makes it unscalable to large graphs.
- **ChebNet [21]:** To address the complexity issue, ChebNet approximated the spectral filters using K-th order Chebyshev polynomials. This avoided eigen-decomposition and localized the filters to a K-hop neighborhood, reducing complexity to $O(E)$.

2.3.2 Spatial Message Passing Neural Networks

Modern GNNs mainly follow the **Message Passing Neural Network (MPNN)** [5] framework, which defines convolutions spatially: nodes update

their embeddings by aggregating information from their local neighbors.

Graph Convolutional Network (GCN) Kipf and Welling [3] proposed GCN as a first-order approximation of ChebNet.

- **Mechanism:** It performs a localized mean-pooling aggregation. The propagation rule involves a normalized adjacency matrix $\hat{\mathbf{A}}$ to aggregate neighbor features, followed by a linear transformation and nonlinear activation.
- **Pros/Cons:** GCN is highly efficient and effective as a baseline. However, it requires the full graph during training and treats all neighbors as equally important, which limits the flexibility of this model.

GraphSAGE (Sample and Aggregate) To enable representation learning on large graphs, Hamilton et al. proposed GraphSAGE [2].

- **Mechanism:** Instead of using the full adjacency matrix, GraphSAGE samples a fixed number of neighbors for each node. The model then learns aggregator functions (e.g., Mean, LSTM, Max-Pooling) to combine neighbor information.
- **Pros/Cons:** This neighbor sampling strategy allows GNNs to scale to massive graphs via mini-batch training. However, random sampling may discard informative neighbors in dense graph regions.

Graph Attention Network (GAT) Veličković et al. recognized that not all neighbors are equally important. Therefore, they introduced the GAT [4] model.

- **Mechanism:** GAT employs a self-attention mechanism to compute a learnable coefficient α_{ij} for each edge (i, j). This allows the model to assign different weights to different neighbors during aggregation progress.
- **Pros/Cons:** GAT offers superior expressiveness and robustness to structural noise. However, the trade-off is significantly higher computational cost and memory usage due to the calculation of attention scores for every edge.

2.4 Data Augmentation on Graphs

As GNNs became more powerful, the focus shifted to data quality. Overfitting and limited labeled data are core issues. While Data Augmentation is

a standard practice in CV (e.g., cropping, rotation), applying it to graphs is non-trivial because graph structures are discrete and topologically dependent.

2.4.1 Heuristic Graph Augmentation

Early approaches adapted heuristic strategies to the graph domain.

- **DropEdge [22]:** Randomly removes a certain ratio of edges during each training epoch. This acts as a regularizer and has been proven to alleviate the over-smoothing problem in deep GNNs by reducing message passing density.
- **Node Masking:** Randomly masks node features or drops entire nodes to force the model to rely on context.

Limitation: These methods are stochastic. The perturbation is applied across the whole graph, potentially breaking critical connectivity or removing informative edges while keeping noise.

2.4.2 Graph Contrastive Learning (GCL)

Inspired by SimCLR, GCL frameworks generate augmented views and maximize their mutual information.

- **GraphCL [10]:** Investigated various augmentation combinations (Node Drop, Edge Perturbation, Subgraph, Attribute Masking). It showed that the choice of augmentation is crucial and dataset-dependent.
- **GCA [23]:** Proposed adaptive augmentation probabilities based on node centrality (e.g., degree, PageRank), arguing that structurally central nodes should be perturbed less.

Gap Analysis: Despite these advances, most GDA methods still rely on pre-defined heuristics or separate pre-calculation steps. There is a lack of frameworks that can *learn* optimal augmentation policies end-to-end with the GNN training.

2.5 Large-Scale and Heterogeneous Graph Learning

Real-world graphs (e.g., academic networks, knowledge graphs) introduce two additional complexities: heterogeneity (multiple node/edge types) and massive scale.

2.5.1 Heterogeneous Graph Neural Networks (HGNNs)

Standard GNNs assume a homogeneous graph structure. HGNNs extend this to handle typed edges.

- **Relational GCN (R-GCN) [15]:** Assigns a separate weight matrix $\mathbf{W}_r$ for each relation type r. While effective, the number of parameters grows linearly with the number of relations, leading to parameter explosion and overfitting on rare relations.
- **HAN (Heterogeneous Graph Attention Network) [14]:** Utilizes hierarchical attention based on meta-paths (e.g., Author-Paper-Author). While capturing high-level semantics, HAN relies heavily on manually defined meta-paths, which requires extra cost and is difficult to scale.
- **HGT (Heterogeneous Graph Transformer) [6]:** Adapts the Transformer architecture with type-specific parameters. It automatically learns meta-path structures but incurs high computational complexity ($O(N^2)$ attention without sampling), limiting its use on large scale graphs. More recently, simplified architectures like Simple-HGN [24] and SeHGNN [25] have been proposed to improve efficiency on heterogeneous graph benchmarks [26].

2.6 Large Language Models for Graph Feature Generation

The most recent frontier is the integration of LLMs into graph representation learning, specifically for constructing the feature matrix $\mathbf{X}$ in Text-Attributed Graphs (TAGs).

2.6.1 Evolution of Feature Engineering

1. **Earlier solution: Shallow Features.** Traditional methods [11] represent text as sparse vectors of word counts. They are computationally cheap but semantically weak, failing to capture synonyms or context.
2. **Recent solution: Dense Semantic Embeddings.** The current SOTA involves using Pre-trained Language Models (LLMs) like BERT or Sentence-BERT [12] to encode node text into dense vectors. This paradigm has evolved rapidly, with recent works exploring feature extraction via self-supervised learning [27] and LLM-based explanations [28].

2.6.2 The Scalability Crisis

Applying modern LLMs to large graphs faces a scalability crisis.

- **Inference Cost:** Generating embeddings for a billion-node graph requires billions of LLM forward passes, which is computationally intractable for most industrial applications.
- **Existing Solutions (Anchor Methods):** Recent works like **GLANCE** [29] propose using anchor nodes to reduce LLM calls. However, current methods typically propagate dense embeddings on homogeneous graphs [30], failing to address the specific challenges of heterogeneous graph structures.

2.7 Summary

This review has traced the evolution of the field from basic machine learning to the developing GNN architectures, and finally to the convergence with LLMs. We have identified three critical gaps: (1) the need for adaptive rather than heuristic data augmentation, (2) the lack of a cost-effective balanced generation strategy, and (3) the absence of scalable propagation frameworks for large-scale heterogeneous graphs. We attempt to address these gaps in this dissertation.

Chapter 3

SSDA:A Self-supervised Data Augmentation Framework for Graph Neural Networks

3.1 Motivation and Objectives

Data augmentation is a significant tool for modern machine learning. In domains such as computer vision (CV) and natural language processing (NLP), data augmentation a well-established technique to enrich training data through transformations like random cropping and rotation for images, or back-translation and synonym replacement for text-proven to enhance model performance and generalization.

However, this success has not been easily replicated in the domain of graph-structured data.

This difficulty comes from the unique non-Euclidean structure of graph data. Whereas images or text hold an underlying grid-like or sequential structure, making the semantic impact of transformations (like translation or masking) local and controllable, graphs are discrete entities defined by their topology. A seemingly minor perturbation-for instance, randomly deleting a few wrong edges—can fundamentally alter a graph's core properties, such as its connectivity, community structure, or shortest paths, thereby destroying the very semantic information it carries.

Consequently, Graph Data Augmentation (GDA) must operate within a delicate balance: it must create novel samples different enough from the original to improve model robustness, while keep most of the topological information from the original graph.

While early Graph Neural Network research often sidestepped this challenge by relying on large-scale labeled datasets, the rise of Self-Supervised Learning (SSL) has transformed GDA from an optional extra into the core engine driving the learning process. SSL has emerged as a critical paradigm shift in graph representation learning. The core idea of SSL is to create

supervisory signals from the data itself, rather than relying on external human annotations. Yet, despite significant progress, the application of GDA remains largely constrained by manually-tuned operations.

To situate our contribution, we must first systematically organize the landscape of existing GDA techniques. Based on the target of the operation, current methods can be broadly categorized into structural and feature-space augmentations.

3.1.1 Structural Perturbations

This is the most intuitive and, to date, the most dominant family of augmentations. The core assumption of these methods is that a robust GNN representation should be insensitive to minor changes in the graph's topology.

- **Edge-level Perturbation**: This is the most widely adopted strategy in GDA. It involves **randomly adding or dropping** a certain ratio of edges to create two related but distinct "views" of a graph [10, 47]. GraphCL [10], for example, found that the intensity of edge perturbation has a dramatic impact on downstream task performance. Recent advances have also explored automated augmentation selection [68] and perturbation-free strategies [70].
- **Node-level Perturbation**: This method, often called **Node Dropping**, generates a subgraph by randomly removing a fraction of nodes and their incident edges. This operation forces the model to rely on the "context" of a node (i.e., its neighborhood structure) rather than just its mere presence.
- **Subgraph Sampling**: An alternative approach is to sample a computational subgraph, often based on Random Walk strategies or their variants (like Personalized PageRank) [35]. This was used in early works like Deep Graph Infomax (DGI) [46] to define a global-local mutual information objective.

3.1.2 Feature-space Augmentation

As an alternative to operating on the discrete topological space, this family of methods applies perturbations in the continuous feature space.

- **Attribute Masking**: This is a core idea borrowed from Masked Language Models in NLP. It involves **masking out** a portion (or all) of the feature dimensions for a subset of nodes, forcing the GNN to **reconstruct** the missing features from the neighborhood context [48].

- **Feature Perturbation and Mixup**: This includes adding Gaussian noise to the original node feature matrix X or, borrowing from Mixup in CV [50], performing linear interpolation between the hidden representations of two nodes to create virtual samples, thereby smoothing the decision boundary.

while GDA is key to unlocking the potential of GNNs, existing methods are either overly reliant on "heuristic" manual tuning (like GraphCL), or are prohibitively complex and expensive (like RL-based AutoGDA), or focus only on a single modality (like GraphMAE focusing on features).

The field lacks a framework that is lightweight, adaptive, and unified—one that can: (a) Simultaneously and synergistically consider both structural (node/edge) and feature augmentations. (b) Automatically and differentiably adjust the "intensity" and "type" of augmentation during training to adapt to the specific dataset and downstream task.

To fill this gap, we propose the core contribution of this chapter [SSDA:A Self-supervised Data Augmentation Framework for Graph Neural Networks]: a novel graph data augmentation framework. Instead of resorting to complex RL or meta-learning, we design the augmentation parameters (e.g., the edge drop rate, the feature mask rate) as learnable parameters. These parameters are then optimized end-to-end with the primary GNN training objective (e.g., the contrastive loss) via standard backpropagation.

This design allows our framework to achieve data and task adaptivity while remaining efficient and simple to integrate. The following sections will detail this framework's mathematical formulation, implementation, and its empirical superiority over existing state-of-the-art methods. Our work is designed to address this precise challenge. We construct a graph data augmentation framework that simultaneously operates on both the graph's structure (nodes and edges) and its features. Critically, the framework is designed to be differentiable, allowing it to automatically tune its augmentation parameters during the training process to achieve optimal performance. We apply this framework to several common graph datasets and subsequently use the augmented graph data for training GNNs. Experimental results demonstrate that our framework effectively improves the performance of GNNs on downstream tasks, thereby validating its efficacy.

3.2 Methodology

This work proposes a novel, end-to-end graph data augmentation framework SSDA (Self-Supervised Data Augmentation Framework for Graph Neural Networks), aimed at improving the learning efficacy and robustness of GNNs.

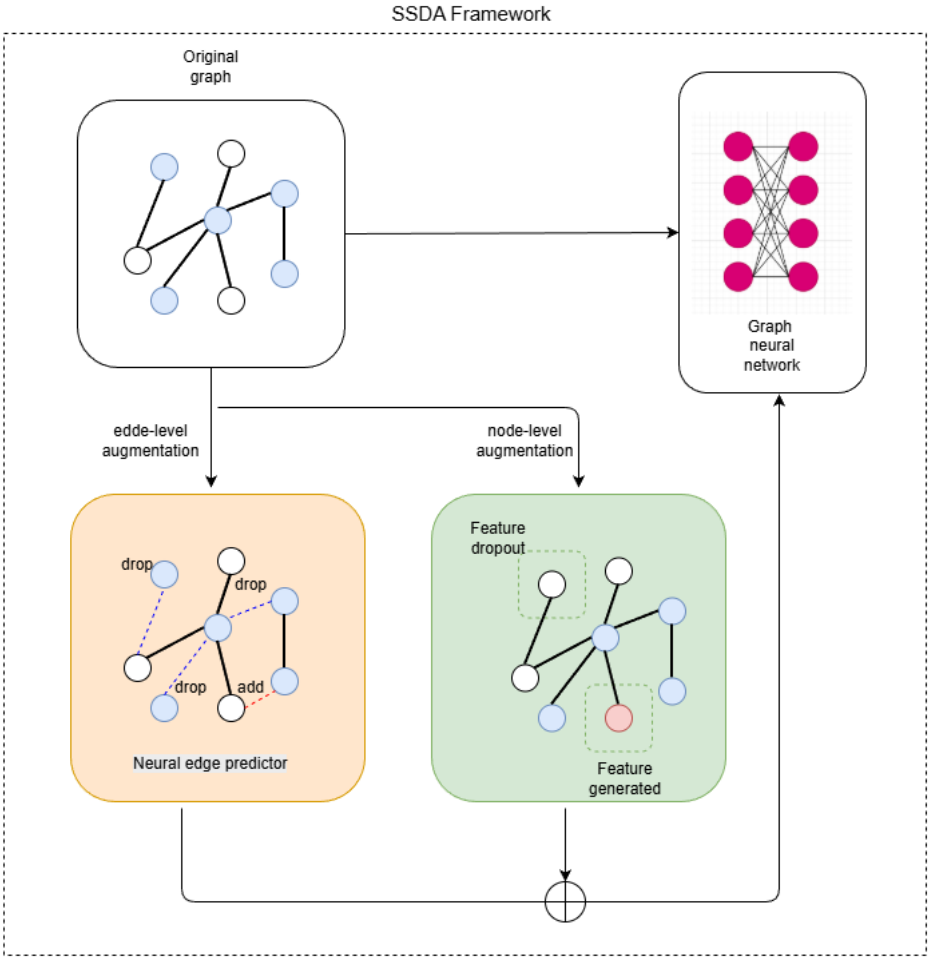

Figure 3.1: Schematic representation of the SSDA framework

Diverging significantly from static paradigms, our proposed model facilitates the continuous optimization of its augmentation parameters throughout the training phase. This adaptive, differentiable approach allows the augmentation strategy to co-evolve with the GNN's training objective, thereby achieving enhanced performance. Our framework operates on both the graph's structure (edges) and its attributes (nodes) at the same time. Figure 3.1 shows the overall framework of the proposed model. Given the original graph, we first apply an edge-level augmentation to modify the connection of the nodes. Edge-level augmentation is based on the prior assumption that modifying edges while preserving key information of the graph structure can introduce beneficial diversity. By altering some connections within the graph,

it introduces structural diversity, preventing the model from overfitting to specific connection patterns. Existing edge-level augmentations include edge removing, edge additions and their hybrids, which we unify as edge rewiring augmentation. For a given graph $G = (\mathbf{A}, \mathbf{X})$ without considering edge features, edge rewiring removes or adds a portion of edges in G, by applying masks parameterized by two probabilities p_{ij}^- and p_{ij}^+ to the adjacency matrix $\mathbf{A}$, finally yields an augmented graph $\hat{G} = (\hat{\mathbf{A}}, \mathbf{X})$, i.e.,

$$\begin{aligned} \hat{\mathbf{A}} &= \underbrace{\mathbf{A} \circ (\mathbf{1} - \mathbf{1}_r)}_{\text{edge removing}} + \underbrace{(\mathbf{1} - \mathbf{A}) \circ \mathbf{1}_r}_{\text{edge addition}} \\ \text{with } \mathbf{1}_r[i,j] &\sim \begin{cases} \text{Bernoulli}(p_{ij}^-) & \text{if } \mathbf{A}_{ij} = 1 \\ \text{Bernoulli}(p_{ij}^+) & \text{if } \mathbf{A}_{ij} = 0 \end{cases} \end{aligned} \tag{3.1}$$

Where $\mathbf{1}_r$ is the rewiring location indicator matrix, p_{ij}^- represents the probability of removing edge e_{ij} and p_{ij}^+ represents the probability of connecting nodes v_i and v_j.

3.2.1 Topological Augmentation: The Neural Edge Predictor

This module acts as a differentiable component that actively **learns a dynamic policy for modifying the graph's topology**. Instead of merely applying fixed, pre-defined rules, the Neural Edge Predictor analyzes the characteristics of the initial graph (e.g., node features, existing connectivity patterns, and even intermediate GNN embeddings). Based on this, the model is able to decide which edges to add (indicated by red dashed lines) or drop (indicated by blue dashed lines) within the graph. This results in a `Generated Graph` that represents a topologically augmented view. This aligns with the broader field of Graph Structure Learning (GSL) [79, 80], which aims to optimize noisy graph structures.

The learning process within the Neural Edge Predictor is crucial. By training this module end-to-end with the downstream task, the framework gains the ability to:

(a) **Introduce shortcut edges** to effectively reduce the graph diameter between distant nodes. This helps address over-smoothing problem in deep GNNs, where node features become indistinguishable after many message-passing layers [53, 73]. Learned shortcuts can propagate information more efficiently across the graph, preserving feature diversity.

(b) **Remove redundant edges** within dense clusters or highly connected regions. By strategically pruning less informative connections, the

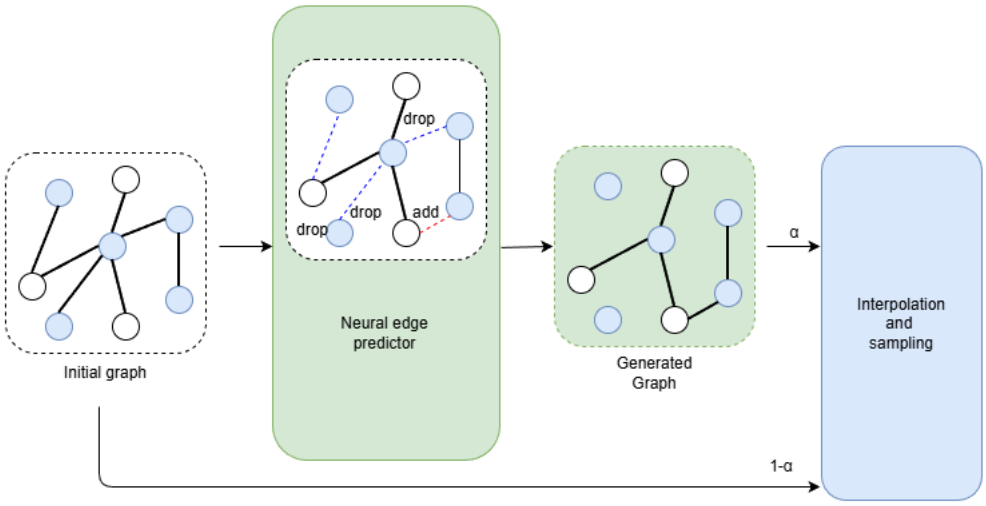

Figure 3.2: Schematic representation of the SSDA framework

model is forced to rely on more discriminative structural patterns, thereby learning more robust and sparse representations. This can also alleviate issues like over-squashing, where information from distant nodes gets compressed into a small message [54].

(c) **Improve GNN generalization** by exposing the model to a diverse and *learned* set of topological variations. Rather than relying on purely random noise, the generated perturbations are tailored to enhance the model's ability to extract invariant features from different structural contexts.

Furthermore, as illustrated in Figure 3.2, the `Generated Graph` from the Neural Edge Predictor is not always used in isolation. It is often combined with the Initial graph through an interpolation and sampling mechanism, governed by a parameter α. This interpolation allows for the creation of a continuum of augmented graphs, ranging from the original structure to the fully predicted structure, providing finer control over the augmentation strength and diversity. This strategy enables the GNN to learn representations that are robust to a spectrum of structural variations, rather than just two distinct views.

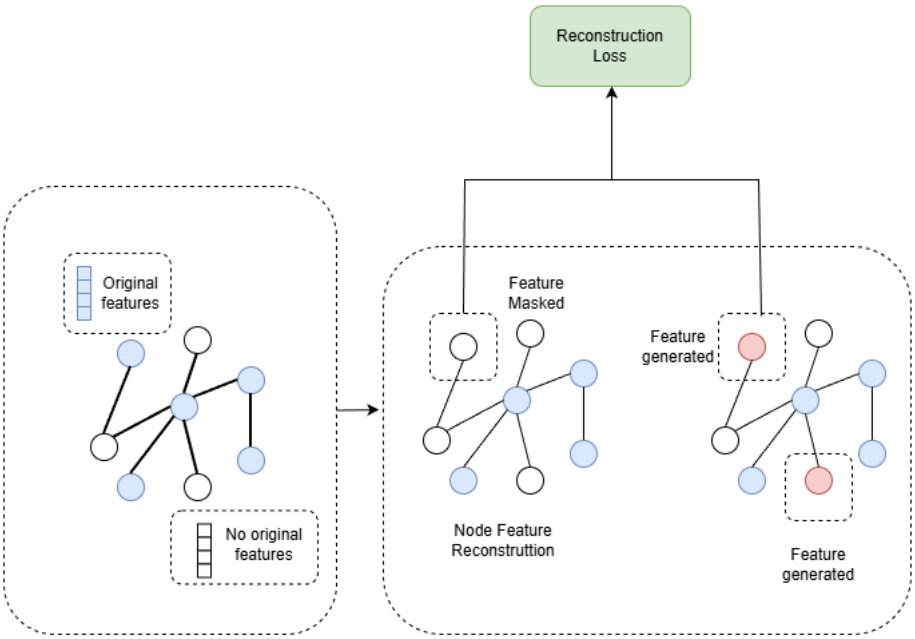

Figure 3.3: Node Feature Reconstruction

3.2.2 Attribute Augmentation: Generative and Regularizing

At the same time, the framework augments the node attributes using a two-part approach:

First, a generative component produces synthetic features. This is particularly useful for graphs where some nodes have no starting features. This ensures all nodes have a feature vector for the GNN to process. Second, we apply stochastic feature dropout. This process, similar to attribute masking, acts as a regularizer and prevents the model from over-fitting to the newly generated features or relying too heavily on any single feature. Masked Autoencoders have recently been adapted with success for graph self-supervised learning [48]. As a denoising strategy, this approach masks a portion of the graph data, such as node features or links, and trains the model to reconstruct the missing content.

It has been shown that reconstructing masked node features can be a powerful pretext task on its own, leading to strong performance. Our work builds on this feature reconstruction approach, but we aim to improve its performance by addressing several limitations found in existing methods.

Masked node feature reconstruction has a key weakness: it is highly

dependent on the quality of the node features themselves. GNN performance is tied to how discriminative the features are. If the reconstruction targets are poor, the learning signal itself can be misleading. We intentionally degraded feature quality by compressing the original features to just 50 dimensions using PCA. The performance of most baseline models dropped far more significantly than its supervised counterpart when using these weak features. This suggests that learning via feature reconstruction is especially vulnerable to feature quality. This is a critical problem in graph learning. Unlike in CV or NLP, where inputs are often raw data (pixels or words), a graph's feature matrix X is almost always a processed product that can contain noise. For example, node features are often bag-of-words vectors (Cora [3]), averaged word embeddings (ogbn-Arxiv [1]), or outputs from a pretrained language model (MAG240M [1]). The quality of these features is limited by the original feature extractor and may already be noisy. An objective that forces the model to precisely recover these noisy or non-discriminative features is problematic. It risks training the model to fit the noise itself, which can damage the final representation.

In our model, we take the encoded representation and randomly mask it just before feeding it to the decoder. We repeat this process K times to generate K different "views" $(V_1 \dots V_K)$, each with a unique set of re-masked nodes.

The decoder's objective is then to reconstruct the original input features from all K of these different views. This multi-view approach acts as a strong regularizer. By forcing the model to reconstruct from multiple, differently corrupted representations, it is discouraged from simply memorizing noise or other artifacts in the input features. As a result, the training becomes less sensitive to the quality and potential disturbances in the input feature matrix X.

$$\mathcal{L}_{input} = \frac{1}{|\tilde{V}|} \sum_{j=1}^{K} \sum_{v_i \in \tilde{V}} \left(1 - \frac{\mathbf{x}_i^\top \mathbf{z}_i^{(j)}}{\|\mathbf{x}_i\| \cdot \|\mathbf{z}_i^{(j)}\|} \right)^\gamma \tag{3.2}$$

where $\mathbf{x}_i$ means the i-th row of X, and $\mathbf{z}_i^{(j)}$ is the i-th row of predicted feature $Z^{(j)} = f_D(A, \tilde{H}^{(j)})$, and $\gamma \geq 1$ represents the scaled coefficient. The experimental results demonstrate that our proposed method is not only highly effective but also computationally efficient.

We also construct a second, more informative prediction target in the representation space, which is less susceptible to noise from the input features.

We use a separate target generator network to produce these latent targets. Since GNNs can serve as denoising encoders and capture high-level

semantics, this generator can produce a stable, high-level representation from the *unmasked* graph to serve as the clean target.

Formally, this target generator f_T shares the same architecture as our main GNN encoder (f_E) and projector, but uses a different set of weights. During pre-training, the unmasked graph is passed through f_T to produce the target representation, Z_{target}. The main encoder and projector then generate a latent prediction, Z_{pred}, from the *masked* graph. The loss is then computed between Z_{pred} and Z_{target}.

3.2.3 Evaluate Metrics

Graph Data Augmentation (GDA) has emerged as a critical technique to enhance the performance, robustness, and generalization of Graph Neural Networks (GNNs), particularly in data-scarce or semi-supervised settings. By generating new, plausible graph samples, GDA aims to expand the training distribution and expose the model to a wider variety of structural and feature-based perturbations.

However, assessing the true effectiveness of a GDA strategy is a multi-faceted task that extends beyond a simple accuracy metric. A comprehensive evaluation protocol must rigorously measure the augmentation's impact across several key dimensions.

These dimensions typically include:

1. **Downstream Task Performance:** This is the most direct evaluation, quantifying the performance lift (e.g., accuracy, F1-score) on benchmark downstream tasks such as node classification, graph classification, and link prediction. This is often tested against strong baseline GNNs (e.g., GCN, GAT) with and without augmentation.
2. **Model Robustness:** A crucial evaluation is determining whether GDA makes the GNN more resilient. This is tested by evaluating the model's performance under adversarial attacks (e.g., Nettack) or random perturbations (e.g., adding/dropping edges or nodes) on the test data.
3. **Generalization and OOD:** Effective GDA should help the model learn invariant structural patterns rather than spurious correlations. This is assessed by testing the model's generalization capabilities on out-of-distribution (OOD) samples, such as graphs with different degree distributions or structural properties than the training set.
4. **Alleviation of GNN Deficiencies:** Some GDA methods (like DropEdge) are specifically designed to mitigate inherent GNN issues, such as over-smoothing. In this case, evaluation involves demonstrat-

ing that the GNN can be trained with significantly more layers without a drop in performance.

In order to evaluate the performance of our proposed framework, we apply the augmented graph to the graph neural networks and work on node classification task. The improving accuracy of the task significantly indicate the effect of our data augmentation model.

3.2.4 Joint Optimization

As shown in the SSDA framework diagram, the two augmented views—one for topology and one for features—are combined. This final augmented graph is then fed into the GNN, often in a contrastive learning setup with the original graph. This joint optimization forces the GNN to learn representations that are robust to meaningful changes in both structure and attributes, which, as our experiments show, improves performance on downstream tasks.

The loss of label prediction and the loss of feature completion are combined as the final loss of this new model. After that, the overall model can be optimized via back propagation:

$$\mathcal{L} = \lambda \mathcal{L}_{\text{completion}} + (1 - \lambda)\mathcal{L}_{\text{prediction}} \tag{3.3}$$

3.3 Experimental

To validate the efficacy of our proposed SSDA framework for graph data augmentation, we conducted node classification experiments on several benchmark datasets. Our primary evaluation was performed on the IMDB, DBLP, and ACM datasets. To further analyze performance variations and the impact of dataset characteristics, we included the Blogcatalog and Flickr datasets in a secondary analysis.

We selected three representative GNNs: GCN, HAN, and GAT,as our baseline models. We measured the performance of these base models with and without our SSDA augmentation. Furthermore, we benchmarked these results against several current state-of-the-art data augmentation methods. The experimental results demonstrate that our augmentation framework provides effective and consistent performance improvements for GNNs on the node classification task.

The following subsection provides a brief of the information for the datasets. The performance of our proposed framework varied across the

datasets, reflecting the model's sensitivity to the distinct characteristics of each graph.

We evaluated our framework on five standard benchmark datasets. This selection includes three heterogeneous graphs (IMDB, DBLP, ACM) to test the model's ability to handle multi-relational data, and two homogeneous social graphs (Blogcatalog, Flickr) to evaluate scalability and performance on multi-label tasks.

3.3.1 Datasets

3.3.1.1 IMDB

The IMDB dataset is a heterogeneous graph centered on the movie domain. It models three entity types: Movies, Actors, and Directors. The graph structure is explicitly bipartite, defined by Movie-Actor and Movie-Director relations. Node features are provided exclusively for the Movie nodes; these attributes are high-dimensional, sparse bag-of-words (BoW) vectors derived from textual plot summaries. The standard downstream task is a multi-class node classification to predict the pre-defined genre for each movie.

3.3.1.2 DBLP

The DBLP dataset is a heterogeneous, multi-relational bibliographic network modeling the academic publication landscape. It contains four types of entities: Authors, Papers, Terms, and Venues. The graph structure captures the complex interactions between these entities, such as Author-wrote-Paper and Paper-published-in-Venue. Node features, derived from BoW representations of publication keywords, are provided for the Author nodes. The objective is to classify authors into their respective research areas [14].

3.3.1.3 ACM

The ACM dataset is another well-known heterogeneous bibliographic graph. It consists of Paper, Author, and Subject nodes. The topology is defined by bipartite links connecting Paper-to-Author and Paper-to-Subject. In this dataset, the Paper nodes are the target entity and are attributed with high-dimensional, sparse BoW features derived from their titles and abstracts. The node classification task is to categorize these papers into their correct research subject [14].

3.3.1.4 Blogcatalog

Blogcatalog is a large, homogeneous social network graph. Each node represents a blogger (a user), and the edges represent undirected friendship links between them. The node features consist of a high-dimensional binary vector, where each dimension indicates the presence of a specific user-defined tag or interest category. The downstream task is a challenging multi-label node classification, as each user can be associated with multiple interest categories simultaneously.

3.3.1.5 Flickr

The Flickr dataset is another large-sacle, homogeneous social graph, where nodes represent users on the photo-sharing platform. Edges represent the social contacts or friendships between these users. Node attributes are represented as bag-of-words (BoW) vectors derived from the user's photo tags or group memberships. Similar to Blogcatalog, this dataset presents a *multi-label node classification* task, requiring the model to identify the various communities or topics of interest for each user.

3.3.2 Implementation Settings

3.3.2.1 Baselines

We conducted a detailed investigation to evaluate our proposed framework as a data augmentation strategy. We first integrated our framework with three foundational Graph Neural Network: GCN, GAT, and HAN.

To isolate the direct impact of our method, we first measured the performance on the node classification task by comparing each base models' results with and without the augmentation. This ablation allowed us to quantify the gains attributable to our framework.

Additionally, we benchmarked the best-performing augmented model (e.g., SSDA + GAT) against a strong model, MAGNN, to situate our results within the broader research landscape. As expected, the performance uplift was not uniform across all experiments. The gains varied, appearing to be dependent on the distinct architectural properties of the underlying GNN base models.

GCN (Graph Convolutional Network): The Graph Convolutional Network (GCN) is a widely-used baseline known for its efficiency and simplicity [3]. Its distinctive characteristic is its isotropic and computationally efficient aggregation. A GCN layer updates a node's representation by taking

a normalized, mean aggregation of features from its immediate (one-hop) neighborhood, including the node itself. The aggregation weights are fixed and determined by the graph structure (specifically, node degrees), not by learned weights for each neighbor. This makes GCN highly scalable but less expressive, as it treats all neighbors with equal (or structurally-determined) importance.

GCN is highly efficient, as its computation is dominated by a sparse matrix multiplication. Its per-layer complexity is $O(NFF' + EF')$. The $O(NFF')$ term represents the node-wise feature transformation, while $O(EF')$ represents the aggregation step. GCN's key advantage is its isotropic aggregation, which has a constant (and very low) per-neighbor computational cost.

GAT (Graph Attention Network): The Graph Attention Network (GAT) introduced an attention mechanism to graph convolutions [4]. Its key characteristic is anisotropic aggregation. Unlike GCN, GAT learns to assign different importance weights to different nodes within a neighborhood. This is achieved through a masked self-attention mechanism, which computes attention coefficients for each neighbor based on the features of the node pair. This allows the model to dynamically weigh neighbor contributions based on their relevance, significantly increasing its expressive power without requiring prior knowledge of the graph structure.

's anisotropic aggregation is more expensive because it must compute attention scores for every edge. For H attention heads, the per-layer complexity is $O(NFF'H + EF'H)$. The first term is the multi-head feature transformation, and the second is the cost of computing and applying attention scores across all E edges. While still linear in N and E, GAT's constant factor is significantly higher than GCN's, scaled by the number of heads H.

HAN (Heterogeneous Graph Attention Network): The Heterogeneous Graph Attention Network (HAN) is specifically designed to operate on heterogeneous graphs (HGs), which contain multiple node and edge types [14]. Its core feature is a hierarchical attention mechanism that leverages predefined meta-paths (e.g., `Author-Paper-Author`). HAN's attention operates at two levels:

1. **Node-level Attention:** Aggregates feature representations from neighbor nodes within a single meta-path.
2. **Semantic-level Attention:** Combines the embeddings generated from all different meta-paths, learning the relative importance of each

meta-path (i.e., each semantic relationship) for the downstream task.

HAN's complexity is driven by its reliance on pre-defined meta-paths. Let M be the number of meta-paths and P be the total number of meta-path instances (the sum of all instances across all M paths). The cost is dominated by the node-level attention, which operates over all path instances, at a cost of $O(PF')$. This is followed by a cheaper semantic-level attention, $O(NMF')$. Since the total number of path instances P is typically far larger than the raw edge count E, HAN is substantially more expensive than GCN or GAT.

MAGNN (Metapath Aggregated Graph Neural Network): MAGNN is a more advanced model for heterogeneous graphs that refines the meta-path concept [45]. Its distinctive characteristic is the explicit encoding of meta-path instances. While HAN primarily aggregates nodes at the *end* of meta-paths, MAGNN first uses an intra-metapath aggregator (such as a GNN, LSTM, or simple pooling) to compose the features of all nodes *along a single meta-path instance*, creating a unified meta-path instance embedding. It then performs inter-metapath aggregation (often via attention) to combine the representations derived from multiple meta-path instances connecting the same node pair. This allows MAGNN to capture more fine-grained semantic information contained within the path structures.

MAGNN is also meta-path based but incurs even higher costs by processing the full path. Let P be the total number of sampled meta-path instances and K be their average length. The computational bottleneck is the *intra-metapath aggregator* (e.g., an LSTM or pooling), which must process all K nodes *within* each of the P paths, leading to a complexity of $O(PKF')$. This cost scales not only with the number of path instances P (like HAN) but also with their average length K, making it one of the most computationally demanding models in our selection.

3.3.2.2 Implementation Details

The proposed SSDA framework was implemented with the PyTorch deep learning framework [36] and trained on an NVIDIA RTX 3090GPU. The key parameters of the training process are listed following:

For the training process of each baseline model (GCN, GAT, HAN), we adopted the hyperparameter settings as published in their respective original papers. All models were optimized using the **Adam** optimizer [55] with a fixed learning rate of 1×10^{-3} .

For the training duration, models were trained for 100 epochs on all datasets, with the exception of DBLP, which was trained for 200 epochs due

to its faster convergence. Optuna is used for efficient hyperparameter search. We conducted a parameter sensitivity analysis for the joint loss coefficient, λ, which balances our augmentation loss with the primary task loss. Based on these experiments, we set $\lambda = 0.5$ for all reported results.

$$\mathcal{L} = 0.5\mathcal{L}_{\text{completion}} + 0.5\mathcal{L}_{\text{prediction}} \tag{3.4}$$

These carefully selected parameters were determined to yield robust model performance. To maintain experimental rigor and ensure a fair comparison, we employed a fixed data split for all experiments. Specifically, each dataset was partitioned into a 70% training set a 10% validation set and a 20% test set.

3.3.2.3 Model Evaluation

We evaluate our proposed framework using 3 widely used GNN architectures: GCN, GAT, HAN. We compare our model performance with that achieved by standard GNN performance, as well as the performance of the powerful MAGNN. Through comparing the accuracy on node classification task, we evaluate the effect of our proposed data augmentation method.

Micro-F1 Score To evaluate performance, particularly on the multi-label tasks, we apply Micro-F1 score. This metric is computed by first summing the True Positives (TP), False Positives (FP), and False Negatives (FN) across *all* C classes to obtain global counts.

From these global counts, the Micro-Precision (P_μ) and Micro-Recall (R_μ) are calculated as:

$$P_\mu = \frac{\sum_{c=1}^{C} TP_c}{\sum_{c=1}^{C}(TP_c + FP_c)} \tag{3.5}$$

$$R_\mu = \frac{\sum_{c=1}^{C} TP_c}{\sum_{c=1}^{C}(TP_c + FN_c)} \tag{3.6}$$

The Micro-F1 score is then the harmonic mean of these two global values:

$$\text{Micro-F1} = 2 \times \frac{P_\mu \times R_\mu}{P_\mu + R_\mu} \tag{3.7}$$

This approach effectively weights every sample-label decision equally, which contrasts with Macro-F1 (which weights each class equally). For this reason, in standard multi-class tasks (where each sample has only one label), the Micro-F1 score is mathematically equivalent to overall accuracy.

Computational Complexity In a full-batch setting, the space complexity of SSDA is $O(N^2)$. This cost is primarily incurred by the Neural Edge Predictor, which requires storing gradients for all node pairs during back-propagation. This $O(N^2)$ requirement is prohibitive for the large-scale graphs evaluated in our work.

To ensure scalability, we adapt our framework for graph mini-batch training. By processing the graph in batches of size M, the space complexity is effectively reduced to $O(M^2)$. This makes the method tractable, as the memory cost becomes independent of the total number of nodes N and dependent on the batch size.

3.3.3 Results and Analysis

Table 3.1: SSDA performance across GNN architectures and five datasets

Model	**IMDB**	**DBLP**	**ACM**	**Blogcatalog**	**FLICKR**
GCN	48.73	91.15	74.90	75.00	61.20
GCN + SSDA	49.77	93.21	78.32	75.79	63.21
GAT	55.91	82.22	89.65	63.81	46.90
GAT + SSDA	58.12	91.22	92.99	69.32	55.52
HAN	57.98	92.72	91.56	73.43	57.40
HAN + SSDA	58.26	92.99	92.26	74.32	61.02
MAGNN	**60.15**	94.27	90.80	**77.40**	65.20
MAGNN+SSDA	60.07	**94.36**	**93.80**	76.40	**66.8**

Table 3.1 presents the main comparative results. The table is organized by dataset and GNN architecture. Within each row, we report the model's performance on the original graph and on the graph modified by our proposed framework. We bold best-performance for every dataset. From the results we can confirm a consistent improvement over all the benchmark models. However, on MAGNN model, which is sensitive to the change of the meta-path, we also observe some performance decline. The model improves the average performance of GCN by 1.8%, GAT by 5.7% and HAN by 1.1%. GAT, as a self-attention based model, can receive significant improvement from the data augmentation. In particular, our model achieves the most significant improvement on the ACM dataset under GAT settings, with a nearly

9% increase compared to the baseline. In contrast, the improvement on the IMDB dataset is relatively modest. This could be attributed to the inherent data structure of the ACM dataset. To validate this hypothesis, we conducted further verification studies on several other datasets. Concurrently, we also investigated the actual quantitative changes in nodes and edges during the data augmentation process and recorded the model's performance.

3.3.3.1 perturbation rate of the Neural Edge Predictor

We further investigated the framework's sensitivity to the topological perturbation rate—the budget of edge modifications (adds/drops) allowed by the Neural Edge Predictor. To this end, we varied this rate and observed the corresponding impact on downstream performance.

A clear trade-off emerged. We observed that when the perturbation rate exceeds a high threshold, model training becomes unstable and performance degrades significantly. This suggests that excessive topological modification over-simplifies the graph structure, which in turn hinders the GNN's ability to learn discriminative node representations.

Conversely, a minimal or trivial perturbation rate yielded only marginal or negligible improvements over the baseline. This indicates that the augmentation was insufficient to provide a robust training signal. These results allowed us to identify an effective operating range for the perturbation rate, and we analyze this performance curve in the following section.

Table 3.2: GAT+SSDA performance under controlled perturbation rate

Dataset	**perturbation rate:10%**	**perturbation rate:15%**	**perturbation rate:20%**
IMDB	55.81	**58.12**	57.99
DBLP	89.32	**91.22**	91.10
ACM	90.21	92.03	**92.99**
Blogcatalog	61.12	**69.32**	65.99
FLICKR	49.61	54.79	**55.52**

Table 3.2 presents the performance of the model on node classification task under three perturbation rate settings. Our analysis revealed a clear optimal range for the perturbation rate. The model achieved its best-observed performance across all five datasets when the rate (defined as the percentage of edges pruned/generated relative to the original edge count) was set to 15% or 20%. In contrast, a lower rate of 10% proved insufficient, resulting in sub-par performance.

3.3.3.2 Node Reconstruction loss

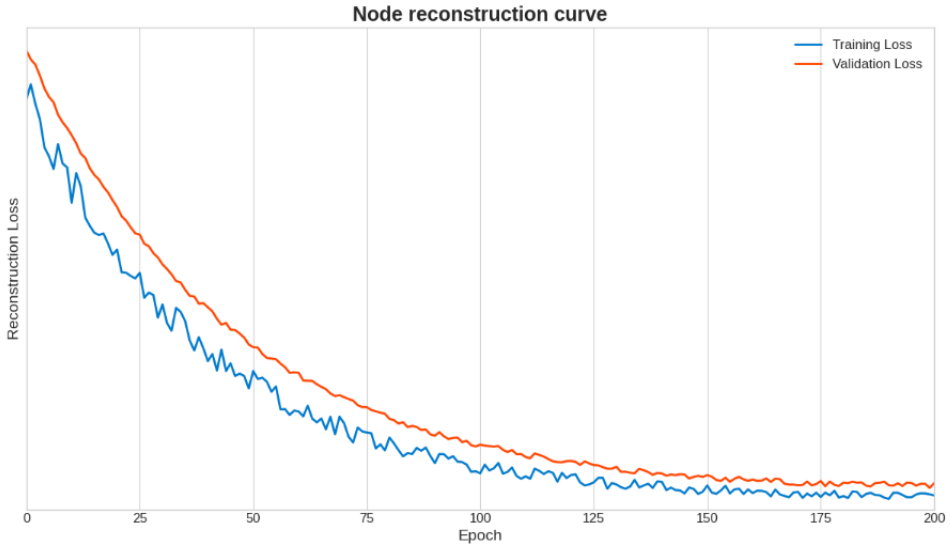

Figure 3.4: Node reconstruction curve

Figure 3.4 plots the node features reconstruction loss during training. The blue curve stands for the training loss while the orange curve stands for the validation loss. The curves show that this loss component converges stably with the overall training loss.The loss function exhibits a rapid initial decrease during the first 100 epochs. After this initial phase, the rate of descent slows considerably, indicating that the model is approaching convergence. This stable convergence demonstrates that the node reconstruction layer has become effective at its task: generating node features that are consistent with the original input features.

3.3.4 Ablation Study

We conducted a series of ablation studies to isolate the source of the performance gains. These experiments were designed to validate the contribution of each key component within our proposed framework.

3.3.4.1 Effect of the augmented topology

We conducted ablation study to investigate the effect of the augmented topology.

Table 3.3: Ablation Study of the Augmented Topology

Dataset	**Original Topology**	**Augmented Topology**
IMDB	56.22	**58.12**
DBLP	84.17	**91.22**
ACM	90.81	**92.03**
Blogcatalog	64.12	**69.32**
FLICKR	47.61	**54.79**

Table 3.3 highlights the critical synergy between our framework's components. We observed that when the edge perturbation module was frozen and augmentation was performed solely via node feature reconstruction, model performance degraded substantially across almost all settings.

Notably, in some settings, this feature reconstruction only strategy underperformed the baseline model trained on the original graph. We hypothesize that this is because attribute masking, when applied in isolation, introduces insufficient structural perturbation. Without the corresponding topological modifications from the edge predictor, the augmented graphs remain too similar to the original. This minimal change fails to provide a robust augmentation signal and may instead act as noise.

3.3.4.2 Effect of Node Feature Reconstruction

In a second ablation study, we isolated the topological augmentation. To achieve this, we froze the parameters of the node feature generator and removed its loss component, preventing it from updating during training.

We also modified the function: the generator was only used to provide a feature imputation for nodes that were originally featureless. It did not re-generate features for nodes that already hold original features. This experiment was designed to measure the impact of the edge predictor in isolation, apart from learnable node feature augmentation. The results of this ablation study demonstrate a significant performance degradation. Under this setting, the model's efficacy was substantially lower than that of the full, integrated framework. More critically, in many cases, the performance fell below the original baseline , suggesting that node feature reconstruction part is significant part of the proposed framework.

Table 3.4: Ablation Study of the Node Feature Reconstruction

Dataset	Random Generation	Original Features	Learnable Reconstruction
IMDB	47.11	55.91	**58.12**
DBLP	79.25	82.22	**91.22**
ACM	85.12	89.65	**92.03**
Blogcatalog	60.77	63.81	**69.32**
FLICKR	45.23	46.90	**54.79**

Table 3.4 shows the performance under three feature augmentation settings. The results confirm that the node feature reconstruction module provides a significant and essential contribution to the framework's efficacy.

3.4 Chapter Summary

This chapter introduced **SSDA (Self-supervised Data Augmentation Framework)**, a novel, end-to-end self-supervised data augmentation framework for Graph Neural Networks. The proposed framework addresses the gaps through a differentiable, two-step approach that learns to augment both graph topology and node attributes simultaneously:

1. **Topological Augmentation:** Instead of random edge perturbation, SSDA employs a Neural Edge Predictor. This learnable module generates an optimal, dynamic augmentation policy by deciding which edges to strategically add or drop, helping to mitigate GNN-specific issues like over-smoothing.
2. **Attribute Augmentation:** This component addresses the critical problem of noisy or non-discriminative input features. It introduces two novel mechanisms: (a) a multi-view random re-masking decoder that acts as a strong regularizer to prevent the model from memorizing noise in the input features, and (b) a representation-space prediction target generated by a separate, "clean" target network, which provides a more stable learning signal than reconstructing the (potentially noisy) raw features.

The framework's efficacy was validated through extensive experiments on five benchmark datasets:IMDB, DBLP, ACM, Blogcatalog, and Flickr across multiple GNN base models such as GCN, GAT, and HAN. Results demonstrated that SSDA provides consistent and significant performance improvements on the downstream node classification task, with GAT models, in particular, showing an average performance increase of 5.7%.

Finally, a series of ablation studies confirmed the necessity of this dual-component design. Disabling either the learned topological augmentation or the node feature reconstruction module resulted in a severe performance degradation, with results often falling below the baseline without augmentation. This finding underscores that the synergistic interplay between the learned structural and attribute augmentations is critical to the framework's success.

Chapter 4

LME: LLM-Driven Framework for Graph Node Feature Engineering

4.1 Motivation and Objectives

In the past decade, we have witnessed a significant paradigm shift in data representation, moving from traditional tabular and sequential data to the complex, relational structures found in graphs. From social networks [2] and financial transactions [57] to protein-protein interaction (PPI) networks [58] and bibliographic citations [3], graph-structured data is ubiquitous. The central challenge in this domain is that graphs are non-Euclidean; they lack the regular grid-like structure of images or the linear sequence of text, rendering traditional machine learning models like Convolutional Neural Networks (CNNs) and Recurrent Neural Networks (RNNs) directly inapplicable.

Early attempts at graph learning relied on hand-crafted features derived from graph theory, such as node degree, centrality measures (e.g., PageRank), or motif counts [16]. These features, while interpretable, were often brittle, domain-specific, and failed to capture the rich, high-dimensional patterns within the graph. This led to the development of Graph Neural Networks (GNNs), a powerful class of deep learning models designed to learn representations directly from the graph structure and its associated features.

4.1.1 Introduction to Graph Neural Networks

The development of GNNs enable us to define a meaningful convolution operation on non-Euclidean domains, drawing inspiration from the success of CNNs in computer vision.

Spectral-based GNNs The first wave of GNNs was rooted in graph signal processing and spectral graph theory [20]. These methods defined graph convolution in the spectral domain. The process involved:

1. Computing the graph Laplacian matrix, $\mathbf{L}$.
2. Performing an eigendecomposition of $\mathbf{L}$ to find its eigenvectors and eigenvalues.
3. Transforming the node features into the spectral domain using this basis.
4. Applying a learnable filter $\mathbf{g}_\theta$ to the graph spectrum.
5. Transforming the filtered signal back to the spatial domain.

While theoretically grounded, this approach, proposed by Bruna et al. [20], suffered from $O(N^2)$ complexity due to the eigendecomposition.

A significant breakthrough came with **ChebNet** [21], which cleverly approximated the filter $\mathbf{g}_\theta$ using a K-th order polynomial of Chebyshev coefficients. This bypassed the need for a full eigendecomposition and, crucially, made the filter spatially localized within a K-hop neighborhood.

The field reached a tipping point with the introduction of the **Graph Convolutional Network** by Kipf and Welling [3]. The GCN was derived as a simplified, first-order ($K = 1$) approximation of ChebNet. This simplification resulted in a highly efficient and effective layer-wise propagation rule:

$$\mathbf{H}^{(l+1)} = \sigma\left(\tilde{\mathbf{D}}^{-\frac{1}{2}}\tilde{\mathbf{A}}\tilde{\mathbf{D}}^{-\frac{1}{2}}\mathbf{H}^{(l)}\mathbf{W}^{(l)}\right) \tag{4.1}$$

Here, $\tilde{\mathbf{A}} = \mathbf{A} + \mathbf{I}_N$ is the adjacency matrix with self-loops, $\tilde{\mathbf{D}}$ is its diagonal degree matrix, $\mathbf{H}^{(l)}$ is the matrix of node representations at layer l, $\mathbf{W}^{(l)}$ is the learnable weight matrix for that layer, and σ is a non-linear activation function. The GCN framework demonstrated the power of a simple, localized aggregation and became a foundational baseline for almost all subsequent GNN research.

Spatial-based GNNs The success of GCN also inspired a parallel, more intuitive line of research focusing on defining graph convolutions directly in the spatial (node) domain. These methods define a layer as an "aggregation" of features from a node's local neighborhood.

The **GraphSAGE** (Graph Sample and Aggregate) model [2] was a pivotal development in this area. Instead of GCN's fixed, normalized mean-pooling, GraphSAGE introduced a generalized aggregation framework:

$$\mathbf{h}_v^{(k)} = \sigma\left(\mathbf{W}^{(k)} \cdot \text{CONCAT}\left(\mathbf{h}_v^{(k-1)}, \text{AGG}\left(\{\mathbf{h}_u^{(k-1)}, \forall u \in \mathcal{N}(v)\}\right)\right)\right) \tag{4.2}$$

where AGG could be a learnable function like a mean, max-pooling, or even an LSTM. Crucially, GraphSAGE introduced *neighbor sampling*, allowing it to train on massive graphs in mini-batches and, for the first time, to generalize to unseen nodes (i.e., perform inductive learning).

Building on this, the **Graph Attention Network (GAT)** [4] addressed a key limitation of GCN and SAGE: their isotropic aggregation. Both GCN and mean-pooling SAGE treat all neighbors equally. GAT introduced a masked self-attention mechanism, allowing the model to learn different anisotropic weights for different neighbors:

$$\vec{h}'_i = \sigma\left(\sum_{j \in \mathcal{N}_i \cup \{i\}} \alpha_{ij} \mathbf{W} \vec{h}_j\right) \tag{4.3}$$

where the attention coefficients α_{ij} are computed based on the features of nodes i and j, making the model highly expressive.

4.1.2 The Message Passing Abstraction

Despite their different origins (spectral and spatial), the majority of modern GNNs can be unified under the Message Passing Neural Network framework [5]. This abstraction describes a GNN layer as a two-stage process operating over k layers:

1. **Aggregation (Message Passing):** Each node v collects "messages" from its neighbors $\mathcal{N}(v)$. A message function M_k and an aggregation function AGG (often a permutation-invariant sum or mean) are used to create a neighborhood vector $\mathbf{m}_v^{(k)}$:

$$\mathbf{m}_v^{(k)} = \text{AGG}\left(\left\{M_k\left(\mathbf{h}_v^{(k-1)}, \mathbf{h}_u^{(k-1)}, \mathbf{e}_{vu}\right) \mid u \in \mathcal{N}(v)\right\}\right) \tag{4.4}$$

2. **Update:** Each node v updates its own representation by combining its previous state $\mathbf{h}_v^{(k-1)}$ with the aggregated message $\mathbf{m}_v^{(k)}$ using an update function U_k:

$$\mathbf{h}_v^{(k)} = U_k\left(\mathbf{h}_v^{(k-1)}, \mathbf{m}_v^{(k)}\right) \tag{4.5}$$

This iterative process, stacked for K layers, is the core mechanism by which GNNs process graph data. The final representation of a node, $\mathbf{h}_v^{(K)}$, is the result of K iterations of message passing. This means $\mathbf{h}_v^{(K)}$ effectively encodes the structural information and features of its entire K-hop neighborhood. These final representations, $\mathbf{H}^{(K)}$, are then fed into a downstream classifier for tasks like node classification.

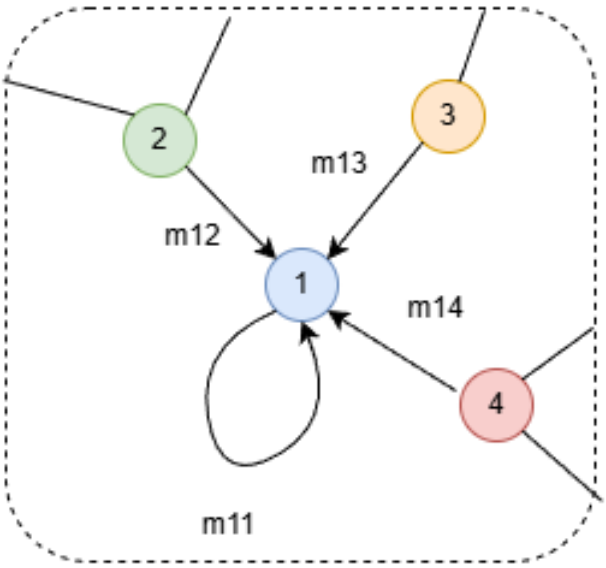

Figure 4.1: Graph neural networks update the node representations by exchanging messages with the neighbors of the node

4.1.3 The Critical Dependence on Input Features

This unifying Message Passing framework reveals a fundamental truth about GNNs: their remarkable ability to learn from graph topology is entirely dependent on the quality of the signals being passed.

The entire K-layer GNN can be viewed as a complex function f_{GNN} that maps two inputs to a set of final representations:

$$\mathbf{H}^{(K)} = f_{GNN}(\mathbf{A}, \mathbf{X}) \tag{4.6}$$

Here, $\mathbf{A}$ is the adjacency matrix (which defines $\mathcal{N}(v)$) and $\mathbf{X}$ is the initial node feature matrix, which serves as the "zeroth" layer, $\mathbf{H}^{(0)} = \mathbf{X}$.

This dependence is twofold:

- **Topology:** The GNN's performance depends on the graph structure that is meaningful and relatively clean. This has motivated an entire subfield of research into graph data augmentation [10] and graph structure learning [59].
- **Features:** The GNN's message passing begins with $\mathbf{X}$. If the initial node features $\mathbf{H}^{(0)}$ are uninformative, noisy, or semantically weak, the GNN is not able to pass message effectively. The model is forced to rely only on the topology, a problem known as over-squashing in

some contexts [54]. Conversely, high-quality, semantically-rich features can provide a powerful initial signal, guiding the message passing and leading to vastly superior performance.

While GNN architectures have attracted significant attention from researchers, the construction of the initial feature matrix $\mathbf{X}$ is a critically under-studied, and equally important part of the graph learning progress. In Text-Attributed Graphs, where $\mathbf{X}$ must be constructed from raw text, the method of this construction becomes a primary bottleneck. This research aims to address this critical research question in graph data representation learning.

4.1.4 Earlier Solution : Traditional Lexical Features

This is the foundational and, for a long time, the standard approach for featuring text-attributed graphs. The core idea is rooted in the bag-of-words hypothesis from information retrieval [11], which assumes that the meaning of a document is captured by the multi set of words it contains, disregarding grammar and word order.

- **Bag-of-Words (BoW)**: Represents a document as a sparse vector $\mathbf{x} \in \mathbb{R}^{|V|}$, where each dimension j corresponds to a term in the vocabulary V, and $\mathbf{x}_j$ represents the count (or binary presence) of that term in the document.
- **Term Frequency-Inverse Document Frequency (TF-IDF)**: This is a refinement of BoW that re-weights the word counts to reflect a term's importance. It down-weights terms that are common across all documents and up-weights terms that are uniquely characteristic of a specific document [11].

These features were used in GNN benchmark datasets, including Cora, CiteSeer, and PubMed, upon which foundational models like GCN [3] and GAT [4] were evaluated.

While simple, highly scalable, and interpretable, traditional lexical feature methods suffer from severe limitations. Most of them are high-dimensional and extremely sparse (vocabularies may be larger than 5000), leading to the curse of dimensionality and high memory overhead for sparse tensors. In addition, as this method completely ignore the order of the words, the semantic information loses during the processing. Moreover, the processing is reported to be sensitive to linguistic ambiguity, such as polysemy and synonymy.

4.1.5 LLM-based Solution: Full Semantic Features

With the advent of pre-trained Transformer based models [18] such as BERT [51], a new paradigm emerged in NLP: encoding the entire text of a node into a single, dense, comparatively low-dimensional semantic vector.

- **Full-Text Encoding**: This method processes the full raw text through a pretrained Large Language Model. The final node feature is typically the mean-pooled output of the last hidden state or the embedding of the special `[CLS]` token.
- **Dense Embeddings**: The output is a single, dense vector (e.g., 384, 768, or 1024 dimensions) for each node. This vector, pretrained on a massive corpus, is assumed to capture the deep contextual and semantic meaning of the node's text.

This full semantic feature approach, particularly using models like Sentence-BERT (SBERT) [12] which are fine-tuned for semantic similarity, is recently the standard for achieving state-of-the-art performance in GNN + LLM research [31]. Recent approaches also include co-training frameworks like GLEM [64] and explanation-based features [28]. However, this kind of solution is usually reported to be computational expensive.

1. **Generation Cost**: Encoding millions of documents with a Transformer model is computationally slow. The self-attention mechanism has a complexity of $O(L^2D)$ with respect to the input sequence length L, making it expensive for long documents such as full-text papers or long product reviews.
2. **Storage Cost**: This is a critical bottleneck for processing large-scale graphs. For example, a dense 768-dimensional float32 vector for a medium-sized graph of 100 million nodes may require unacceptable storage for the feature matrix $\mathbf{X}$. This cost is much too heavy for researchers to conduct the computation.
3. **GNN Training Cost**: GNN message passing is dominated by operations involving the feature matrix $\mathbf{X}$. The complexity of a GNN layer is roughly $O(EDD' + NDD')$. Operations on a large, dense D are significantly slower in terms of time per epoch than optimized sparse-dense matrix multiplications on a sparse $\mathbf{X}$ where the number of non-zero features per node is smaller.

4.1.6 Motivation: Balance the Performance and Efficiency

This recent solutions create a cost-benefit dilemma: researchers have to choose between cheap but weaker solution and powerful but prohibitively expensive solution.

This has motivated us to investigate a middle solution. We believe that a sparse feature vector built from a few, high-quality, LLM-selected keywords can capture the vast majority of the semantic signal required by the GNN, while retaining the low-cost on storage and training, benefiting of sparse, lexical features.

To fill this critical gap, we propose a LLM-driven framework for graph node feature engineering and conduct a rigorous comparative study of these three feature pipelines. Our objectives are to answer the following research questions:

- How is the performance of downstream task performance with our framework compared to the traditional lexical features baseline?
- Can our framework achieve a considerable performance compared with the SOTA LLM-based pipeline?
- How is the computational costs (featuring time, storage size, GNN training speed) of our framework compare to the other pipelines?

4.2 Methodology

We implement three distinct pipelines to generate the node feature matrix $\mathbf{X}$ from the same raw text corpus. These pipelines form the basis of our comparative study.

4.2.1 Pipeline 1: Traditional Lexical Features

This pipeline serves as our baseline, replicating the classic GNN setup [3]. The process is a standard in information retrieval [11].

1. **Tokenization**: The raw text for each node i is converted into a sequence of tokens.
2. **Normalization**: All tokens are lowercased. Punctuation is removed.
3. **Stop-word Removal**: A standard list of English stop-words (e.g., "the", "is", "at", "on") from the NLTK library [56] is removed, as these words carry little semantic meaning.

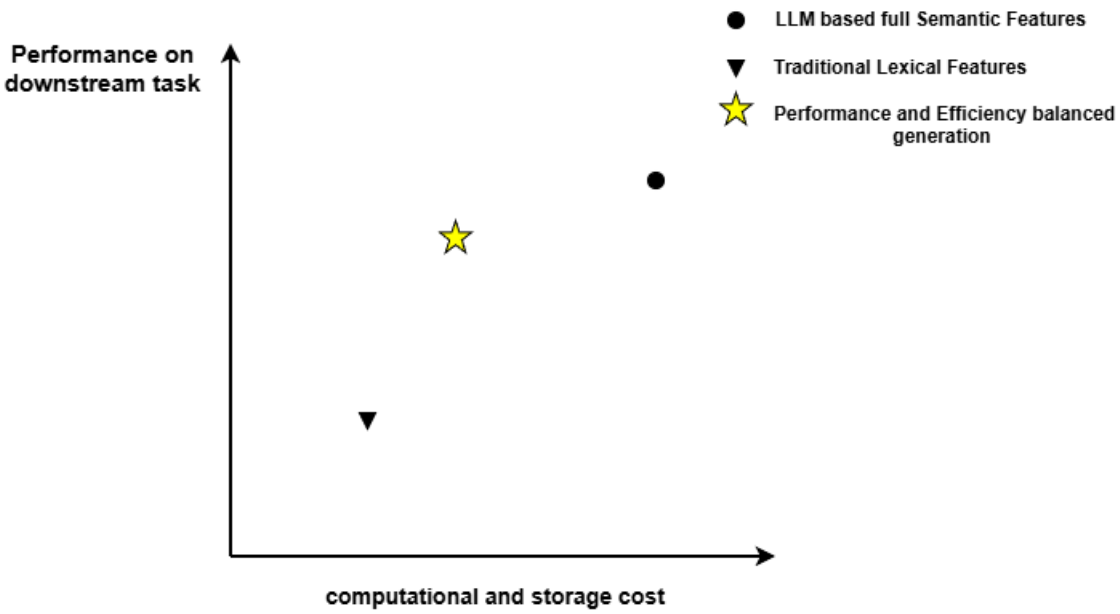

Figure 4.2: Qualitative comparison between traditional lexical features generation, full semantic features using LLM for full-text encoding and performance and efficiency balanced method

4. **Stemming**: We apply the Porter stemmer [52] to reduce words to their root form (e.g., "computation", "computing" → "comput"). This helps to reduce the vocabulary size and consolidate related terms.
5. **Featurization**: We apply the **TF-IDF** algorithm. The TF-IDF score for a term t in document d from corpus D is:

$$\text{TF-IDF}(t, d, D) = \text{tf}(t, d) \times \text{idf}(t, D) \tag{4.7}$$

where $\text{tf}(t, d)$ is the term frequency and the inverse document frequency is:

$$\text{idf}(t, D) = \log \frac{|D|}{1 + |\{d' \in D : t \in d'\}|} \tag{4.8}$$

We use the `TfidfVectorizer` from `scikit-learn` [38], constraining the vocabulary to the top V_{TLF} most frequent terms (e.g., $V_{TLF} = 5000$).
6. **Output**: A row-normalized sparse feature matrix $\mathbf{X}_{TLF} \in \mathbb{R}^{N \times V_{TLF}}$.

4.2.2 Pipeline 2: LLM-based Features extraction and generation

This is our proposed pipeline, which utilizes an LLM as a semantic refiner to generate a high-quality, concise set of keywords, serve as the extracted features. This process is divided into two stages:

4.2.2.1 Keyword Extraction via Prompting

This stage replaces traditional statistical keyword extraction methods (like TextRank [61] or RAKE [62]), which are often unsupervised and sensitive to noise, with a semantic-aware approach. We leverage a pretrained generative LLM to understand the context of terms.

1. **Model**: We use a 4-bit quantized version of `Llama-2-7b-chat-hf` [13]. This model is chosen for its strong balance of zero-shot instruction following capabilities, open-source availability, and efficient inference via quantization.
2. **Prompt Engineering**: This is a critical methodological choice. For each node's raw text, we use a zero-shot prompt designed for conciseness, high fidelity, and structured output:

   > **Prompt**: Given the following academic abstract, please extract the K most important and specific keywords or keyphrases. The keywords must be highly relevant to the core topic and distinguish it from other topics.
   >
   > **Constraints:**
   >
   > (a) Extract exactly K terms.
   >
   > (b) Include multi-word keyphrases where appropriate.
   >
   > (c) Return a single, comma-separated list.
   >
   > **Input:** `[Node's Raw Text]`
   >
   > **Output: Keywords**

   We set $K = 20$ based on preliminary validation (see Section 4.3.4.2). The LLM's ability to identify multi-word "keyphrases" (e.g., "graph neural network") and to disambiguate terms based on context (e.g., "transformer" in CS vs. "transformer" in Electrical Engineering) is hypothesized to be its key advantage over TF-IDF.

4.2.2.2 Featuring Strategies

From the extracted keywords, we construct two types of feature matrices for analysis. This allows us to disentangle the effect of keyword selection from

the effect of feature representation.

- **LME Sparse**: LME Sparse is our primary proposed method.
    1. We concatenate the LLM-extracted keywords from all nodes to form a new, "clean" corpus.
    2. We build a new vocabulary V_{LME} from this corpus.
    3. We apply a standard Bag-of-Words (specifically, `CountVectorizer` [38]) to this new corpus, capped at $V_{LME} = 3000$ terms. We use simple counts rather than TF-IDF, hypothesizing that the LLM has already selected important keywords.
    4. **Output**: A sparse feature matrix $\mathbf{X}_{LME-S} \in \mathbb{R}^{N \times V_{LME}}$. We hypothesize V_{LME} will contains richer semantic information and less noise than V_{TLF}.
- **LME Dense-Avg**: This framework serves as an ablation model.
    1. For each node i, we take its K extracted keywords $\{\text{kw}_1, \ldots, \text{kw}_K\}$.
    2. We embed each keyword individually using the SBERT model.
    3. We compute the **mean-pooled average** of these K vectors to get the final node feature $\mathbf{x}_i$.
    4. **Output**: A dense feature matrix $\mathbf{X}_{LME-D} \in \mathbb{R}^{N \times D}$. This allows us to test if the semantic signal is fully captured by the keywords.

4.2.3 Pipeline 3: Full Semantic Features

This pipeline represents the current SOTA and serves as our high-cost, high-performance benchmark.

1. **Model**: We use the Sentence-BERT (SBERT) [12] model: all-MiniLM-L6-v2. SBERT is a modification of the BERT network using a Siamese and triplet network structure. It fine-tunes BERT on semantic similarity tasks to produce embeddings where semantically similar sentences are close in cosine similarity. SBERT is far more suitable for node featuring than a raw BERT's `[CLS]` token, which is optimized for classification, not similarity.
2. **Encoding**: Each node's entire raw text (e.g., full abstract) is passed through the SBERT model.
3. **Output**: We take the mean-pooling of the last hidden state's token embeddings (a standard practice for SBERT) to get a single, fixed-size vector. This results in a dense feature matrix $\mathbf{X}_{FSF} \in \mathbb{R}^{N \times D}$, where $D = 384$ for this specific model, chosen for its excellent balance of speed and performance.

4.3 Experimental Setup

To validate our hypotheses, we conduct a rigorous comparison of the three pipelines. We evaluate them not only on downstream task performance but also metrics for computational cost .

4.3.1 Datasets

We select three large-scale, text-rich benchmark datasets. For all datasets, we discard their default features and regenerate them from their raw text using our three pipelines to ensure a fair comparison.

4.3.1.1 ogbn-arxiv

The `ogbn-arxiv` dataset, from the Open Graph Benchmark [1], is a large, homogeneous bibliographic network.

- **Schema**: Nodes represent Computer Science papers from the arXiv. Edges are directed, representing citation links.
- **Raw Text**: The text attribute for each node is the concatenation of its title and abstract.
- **Task**: The task is a 40-class node classification to predict the paper's primary subject area (e.g., `cs.AI`, `cs.LG`, `cs.PL`).

4.3.1.2 IMDB

We use the IMDB dataset. The version popularized by the HAN paper [14]. It is a heterogeneous, multi-relational graph.

- **Schema**: It contains three node types: Movies (M), Actors (A), and Directors (D). The graph structure is defined by two edge types (meta-paths): Movie-Actor (M-A) and Movie-Director (M-D).
- **Raw Text**: The text attributes are provided only for the Movie nodes, consisting of the plot summary.
- **Task**: The task is a 3-class node classification on Movie nodes to predict the movie's genre.

4.3.1.3 Flickr

The Flickr dataset [63] is a large-scale, homogeneous social graph.

- **Schema**: Nodes represent users on the Flickr photo-sharing platform. Edges represent undirected friendship links.

- **Raw Text**: The text attribute for each node is a concatenation of user-defined tags associated with their photos.
- **Task**: The task is a multi-label classification to predict the user's interest groups (195 classes in total).

A summary of the dataset statistics is provided in Table 4.1.

Table 4.1: Statistics of the datasets used in our experiments.

Dataset	Nodes	Edges	Classes	Task Type	Raw Text Source
ogbn-arxiv	169,343	1,166,243	40	Multi-class	Title + Abstract
IMDB	12,852	34,960	3	Multi-class	Plot Summary
Flickr	89,250	899,756	195	Multi-label	User Tags

4.3.2 Implementation Settings

4.3.2.1 GNN Base Models

To ensure our findings are generalizable, we test all feature pipelines on three foundational GNN architectures with varying characteristics. We use a 2-layer GNN architecture for all models.

GCN (Graph Convolutional Network): The GCN is a spectral-based method that defines graph convolutions via a localized, first-order approximation [3]. Its layer-wise propagation rule is:

$$\mathbf{H}^{(l+1)} = \sigma(\tilde{\mathbf{D}}^{-\frac{1}{2}}\tilde{\mathbf{A}}\tilde{\mathbf{D}}^{-\frac{1}{2}}\mathbf{H}^{(l)}\mathbf{W}^{(l)}) \tag{4.9}$$

where $\tilde{\mathbf{A}} = \mathbf{A} + \mathbf{I}_N$ is the adjacency matrix with self-loops, and $\tilde{\mathbf{D}}$ is its diagonal degree matrix.

- **Characteristic**: GCN is an isotropic aggregator (all neighbors are averaged with fixed, degree-based weights) and acts as a low-pass filter, smoothing features across neighbors.
- **Complexity**: $O(EDD' + NDD')$, where E is the number of edges, D is input dim, and D' is output dim. For sparse features, this is dominated by $O(\text{nnz}(\mathbf{X})D' + ED')$.

GAT (Graph Attention Network): The GAT introduces an attention mechanism to graph convolutions, allowing for anisotropic aggregation [4].

It learns to assign different importance weights to different nodes in a neighborhood. The attention coefficients α_{ij} are computed as:

$$\alpha_{ij} = \frac{\exp(\text{LeakyReLU}(\mathbf{a}^T[\mathbf{W}\vec{h}_i||\mathbf{W}\vec{h}_j]))}{\sum_{k\in\mathcal{N}_i}\exp(\text{LeakyReLU}(\mathbf{a}^T[\mathbf{W}\vec{h}_i||\mathbf{W}\vec{h}_k]))} \tag{4.10}$$

$$\vec{h}'_i = \sigma\left(\sum_{j\in\mathcal{N}_i\cup\{i\}}\alpha_{ij}\mathbf{W}\vec{h}_j\right) \tag{4.11}$$

- **Characteristic**: GAT is highly expressive but more computationally expensive. We use 4 attention heads in our experiments.
- **Complexity**: For H heads, the complexity is $O(ED'H + NDD'H)$, as attention must be computed for every edge.

GraphSAGE (Graph Sample and Aggregate): GraphSAGE is a spatial-based GNN designed for inductive learning on large graphs. It operates via a two-step sample and aggregate process [2]. A general aggregation function is:

$$\mathbf{h}_v^{(k)} = \sigma\left(\mathbf{W}^{(k)}\cdot\text{CONCAT}\left(\mathbf{h}_v^{(k-1)},\text{AGG}\left(\{\mathbf{h}_u^{(k-1)},\forall u\in\mathcal{S}_\mathcal{N}(v)\}\right)\right)\right) \tag{4.12}$$

where $\mathcal{S}_\mathcal{N}(v)$ is a random sample of neighbors.

- **Characteristic**: For fair comparison in a full-batch setting, we use the "mean" aggregator and do not sample. Its key is a generalized, trainable aggregation that concatenates the self-feature, making it not strictly a low-pass filter.
- **Complexity**: Similar to GCN, $O(EDD' + NDD')$.

4.3.2.2 Training and Evaluation

- **Hardware**: All experiments were conducted on a single NVIDIA RTX 3090 GPU with 24GB of VRAM.
- **Software**: We used PyTorch 2.4 [36] and PyTorch Geometric (PyG) 2.51 [37].
- **Optimizer**: We used the **Adam** optimizer [55] with a learning rate of 1×10^{-3} and a weight decay of 5×10^{-4} for all models.
- **Training**: All models were trained for a maximum of 200 epochs. We employed an early stopping mechanism monitoring the validation set loss with a patience of 20 epochs.

- **Data Splits**: For `ogbn-arxiv`, we used the official public split provided by OGB. For `IMDB` and `Flickr`, we used a 70%/10%/20% split for training, validation, and testing, ensuring the split was fixed across all experiments.

4.3.2.3 Model Evaluation

- **Performance Metrics**: We report Accuracy (%) for multi-class tasks (arxiv, IMDB) and Micro-F1 (%) for the multi-label task (Flickr). Micro-F1 is standard for multi-label tasks as it aggregates the counts for true/false positives/negatives across all C classes before computing the metric:

$$P_\mu = \frac{\sum_{c=1}^{C} TP_c}{\sum_{c=1}^{C}(TP_c + FP_c)} \quad ; \quad R_\mu = \frac{\sum_{c=1}^{C} TP_c}{\sum_{c=1}^{C}(TP_c + FN_c)} \tag{4.13}$$

$$\text{Micro-F1} = 2 \times \frac{P_\mu \times R_\mu}{P_\mu + R_\mu} \tag{4.14}$$

  This approach weights every sample-label decision equally, which is desirable when class imbalance is present.
- **Cost Metrics**: We report (1) Featurization Time (wall-clock seconds), (2) Feature Storage Size (Megabytes on disk, measured for the PyG `.pt` file), and (3) GNN Train Time (seconds per epoch, averaged over 100 epochs).

4.3.3 Results and Analysis

This section presents our empirical results, structured to answer our core research questions.

4.3.3.1 Downstream Task Performance Analysis

We first evaluate the impact of each feature pipeline on the downstream node classification task. The results are presented in Table 4.2.

Table 4.2: Node classification performance (Accuracy or Micro-F1 in %). We compare traditional lexical features (TLF) , LME Sparse (Proposed), and full semantic features(FSF). Best performance per GNN model is in bold. Δ (LME vs FSF) shows the small performance gap.

GNN Model	Feature Pipeline	ogbn-arxiv(Acc.)	IMDB (Acc.)	Flickr (Micro-F1)	Δ (LME vs FSF)
GCN	TLF (Baseline)	68.12	48.73	51.20	-
GCN	LME (Sparse)	75.45	53.08	62.13	-2.37%
GCN	FSF (SOTA)	**77.61**	**55.12**	**65.04**	-
GAT	TLF (Baseline)	60.03	51.91	57.05	-
GAT	LME (Sparse)	69.88	56.55	65.52	-1.62%
GAT	FSF (SOTA)	**72.53**	**58.12**	**66.17**	-
GraphSAGE	TLF (Baseline)	61.29	47.20	50.33	-
GraphSAGE	LME (Sparse)	67.50	53.14	60.80	-5.20%
GraphSAGE	FSF (SOTA)	**72.41**	**58.05**	**66.59**	-

The results in Table 4.2 show two clear answers:

1. The proposed LME Sparse pipeline outperforms the traditional TLF baseline across all datasets and on each GNN model. For example, on IMDB, LME Sparse boosts GAT performance from 51.91% to 56.55% (+4.64 absolute points). On ogbn-arxiv, the gain is even larger, from 60.03% to 69.88% (+9.85 points). This confirms that the semantic naivete of TF-IDF is a major performance bottleneck, and LLM-sourced keywords provide a much stronger learning signal.

2. LME Sparse achieves downstream performance that is highly competitive with the SOTA FSF pipeline. As shown in the Δ column, the performance gap is consistently small, often $< 5\%$. For instance, on ogbn-arxiv, LME Sparse (GAT) achieves 69.88% accuracy, closing near 80% of the performance gap between TLF (60.03%) and FSF (72.53%). This finding suggests that a concise set of LLM-extracted keywords, when featured sparsely, retains the vast majority of the semantic signal that GNNs actually leverage.

4.3.3.2 Cost-Benefit Trade-off Analysis

This section evaluates the performance relative to the computational cost. Table 4.3 shows this cost-benefit analysis, measured on the large ogbn-arxiv dataset.

Table 4.3: Cost-Benefit analysis of the three feature pipelines on ogbn-arxiv dataset.

Pipeline	Featuring Time (s)	Feature Size (MB)	GNN Train Time (s/epoch)	Performance (GAT Acc.)
TLF (Baseline)	122	151 (Sparse, V=5000)	0.48	60.03%
LMS Sparse	∼4100	**103 (Sparse, V=3000)**	**0.41**	69.88%
FSF (SOTA)	∼1850	248 (Dense, D=384)	1.53	72.53%

Table 4.3 provides detailed information to the trade off between computational cost and performance.

- **Storage and Training Speed**: The FSF pipeline is extremely costly in complementary ways. It produces a feature matrix that is 2.4× larger than LME-Feat (248MB vs 103MB). More critically, the GNN training time per epoch is 3.7× slower (1.53s vs 0.41s) because GNN operations (such as the GAT attention calculation) on the sparse, lower-dimensional LME Sparse matrix are far more efficient than on the dense, 384-dimensional FSF matrix.
- **Featuring Time**: The LLM-based LME-Feat pipeline has the highest one-time generation cost (∼4100s), making it ∼2.2× slower than the SBERT-based FSF pipeline (∼1850s). This reflects the trade-off of using a generative model (LME-Feat) versus an encoder (FSF).

LME-Feat achieves over 90% of the SOTA performance while requiring only 26.8% of the GNN training time and 41.5% of the feature storage. This demonstrates an exceptional cost-benefit ratio, occupying the ideal high-performance, low-cost ground.

4.3.3.3 Discussion of Key Findings

The results presented in Tables 4.2 and 4.3 are not just incremental; they suggest a fundamental re-evaluation of how semantic information should be integrated with graph structures. We discuss the three primary implications of our findings.

1. The "Topic-Level Sufficiency" Hypothesis. One important finding is the minimal performance gap (often $< 10\%$) between the SOTA FSF pipeline and our LME Sparse pipeline. This gap becomes even smaller when comparing FSF to our LME Dense-Avg ablation (shown in Table 4.4).

This strongly implies what we term the Topic-Level Sufficiency hypothesis: for the class of tasks GNNs are typically applied to (e.g., node-level topic, community, or genre classification), the GNN does not require the full,

semantic understanding of an entire document. Instead, it primarily requires a robust, unambiguous representation of the node's core topic.

The FSF pipeline provides this information, but also provides a massive amount of redundant semantic information (e.g., grammatical structure, rhetorical flow, subtle qualifiers) that the GNN's message-passing mechanism,which acts as a topical aggregator,is ill-equipped to exploit. The GNN's aggregation function, AGG(·), effectively smooths out this fine-grained detail, retaining only the high-level semantic signal.

Our LME Dense-Avg ablation empirically validates this hypothesis. By encoding only the Top-20 keywords, we achieve ∼90% of the FSF performance. This results demonstrates that these Top-20 keywords, selected by a semantically-aware LLM, are a near-lossless compression of the task-relevant semantic signal. The rest of the document, in the context of GNN message passing, appears to be semantic noise.

2. The "Signal-to-Noise" Advantage of LLM-Sourced Features. The second key finding is the dramatic performance gap between our LME and the traditional TLF baseline , as seen in the table. Both are high-dimensional, sparse feature representations, yet their performance is not comparable.

This discrepancy is explained by their differing Signal-to-Noise Ratios.

- **TLF (Low signal-to-Noise):** The TF-IDF vector is noisy by construction. Its vocabulary V_{TLF} is polluted with stemmed word fragments (e.g., "comput"), ambiguous tokens (e.g., "transformer" in a mixed CS/EE graph), and statistically frequent but semantically weak terms. The GNN's message-passing, $\tilde{\mathbf{A}}\mathbf{X}\mathbf{W}$, is forced to aggregate these noisy, ambiguous, and high-dimensional sparse vectors, leading to a muddled and non-discriminative node representation.
- **LME (High Signal-to-Noise):** The LLM acts as a powerful semantic filter. It leverages its world knowledge to:
  1. **Disambiguate:** It understands that 'graph neural network' is a single, important keyphrase, not three separate words.
  2. **Filter:** It discards all the useless tokens that TF-IDF would have included.
  3. **Consolidate:** It identifies the most salient terms, creating a clean, distilled vocabulary $V_{\text{LME-Feat}}$.

  When the GNN aggregates LME vectors, it is passing high-quality, unambiguous semantic signals. This feature space allows the GNN to learn much clearer decision boundaries, resulting in the large perfor-

mance gains observed.

3. The Practical Implications: The final and most critical discussion point is the practical implication of our cost-benefit analysis (Table 4.3). The SOTA FSF trend, while powerful, has inadvertently created a new scalability crisis. The storage ($O(ND)$) and GNN training complexity ($O(EDD' + NDD')$) of large, dense features make SOTA models computationally infeasible for large-scale industrial graphs (e.g., the full Amazon product graph or the Twitter social graph).

Our LME Sparse pipeline offers a compelling trade-off that bridges this gap.

- It achieves the semantic richness of FSF, solving the performance problem of TLF.
- It retains the computational efficiency of sparse features, solving the cost problem of FSF.

By producing a high-dimensional sparse matrix $\mathbf{X}_{\text{LME-S}}$, our method allows GNNs to leverage highly-optimized Sparse Matrix Dense Matrix Multiplication kernels, which are the base of GNN libraries like PyG [37]. The FSF pipeline's dense $\mathbf{X}_{FSF}$ forces the GNN into a computationally slower dense-dense multiplication for the first layer ($\mathbf{X}_{FSF}\mathbf{W}^{(0)}$), which is a significant bottleneck, as evidenced by our 3.7× training speedup.

This finding suggests that practitioners can achieve ∼90% of the SOTA performance while reducing GNN training costs by ∼73% and feature storage by ∼58%. This makes semantically-aware, high-performance GNNs practical and scalable for real-world, large-scale graph learning.

4.3.4 Ablation Studies

We conduct two ablation studies to further deconstruct the source of LME framework performance.

4.3.4.1 Ablation 1: LME Sparse compared with LME Dense

We investigate whether the benefit of LME comes from the keyword selection or the feature sparsity? We compare our sparse pipeline $\mathbf{X}_{\text{LME-S}}$ with the dense-averaged pipeline $\mathbf{X}_{\text{LME-D}}$.

Table 4.4: Ablation study: LME Sparse vs. LME Dense-Average keywords (GAT Accuracy %).

Dataset	LME Sparse	LME Dense-Avg	FSF (Full Text)
`ogbn-arxiv`	69.88	72.31	72.53
IMDB	56.55	57.90	58.12

The results in Table 4.4 show an important result. The Dense-Avg pipeline, which encodes only the keywords performs almost identically to the FSF pipeline,which encodes the full text. This strongly suggests that the vast majority of the task-relevant semantic signal is contained within the Top-K keywords. The rest of the abstract may in contrast become the noise for the GNN.

Furthermore, the Sparse pipeline (69.88%) shows a near performance to the Dense-Avg model (72.31%). This implies that a simple, sparse BoW representation of these high-quality keywords is sufficient, and the computational overhead of dense vectors is largely unnecessary for achieving competitive performance.

4.3.4.2 Ablation 2: Effect of Keyword Count K

We investigate how many keywords are needed for providing the most important information while avoid redundant messages for the nodes. We vary K in the LME Sparse pipeline and observe the effect on downstream accuracy for the GAT model on ogbn-arxiv dataset.

Table 4.5: Ablation study on the number of keywords (K) for LME Sparse pipeline (GAT Accuracy % on ogbn-arxiv).

K	5	10	15	**20**	25	30
Accuracy	65.21	66.80	67.85	**69.88**	68.43	67.55

As shown in Table 4.5 and visualized in Figure 4.3, performance rapidly increases as K goes from 5 to 20. However, after $K = 20$, the performance starts to drop. This reinforces our conclusion: the LLM, identifying a core set of $\sim$20 keywords that capture the node's essential semantic identity, is enough to hold the key information. Adding more keywords beyond this point may appear to introduce noise rather than valuable signal.

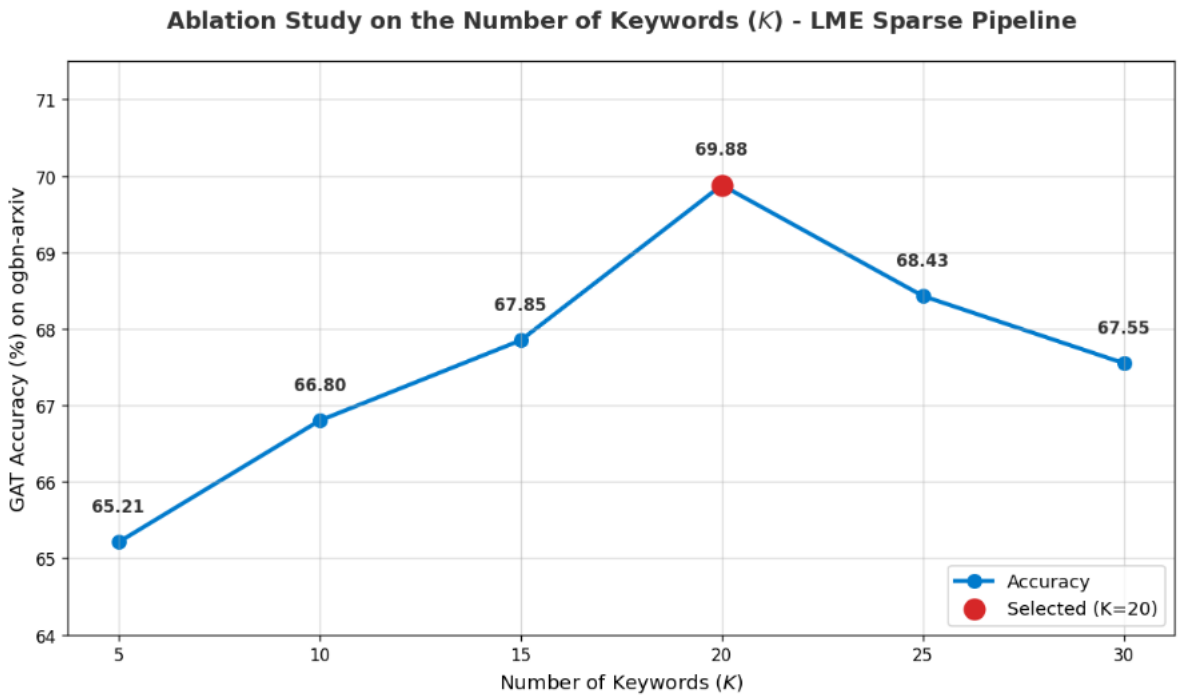

Figure 4.3: Performance on `ogbn-arxiv` as a function of K, the number of extracted keywords.

4.4 Chapter Summary

This chapter addressed a critical aspect of GNN research: the cost-benefit trade-off of node feature engineering for Text-Attributed Graphs. The field has a large gap between two extremes:

1. **Traditional Lexical Features**: Fast and cheap, but semantically weak and sparse.
2. **Full Semantic Features**: SOTA performance, but computationally and storage-intensive.

To fill this gap, we proposed and systematically evaluated a middle-ground pipeline: **LME(LLM-Sourced Lexical Features) framework**. This approach leverages a modern LLM not as a dense encoder, but as an intelligent keyword extractor to generate a concise, semantically-rich, and sparse feature matrix.

Our extensive experiments on three benchmark datasets, with three distinct GNN architectures, yielded clear and compelling findings:

1. **On Performance**: LME consistently and dramatically outperforms the traditional TLF baseline. More importantly, it achieves downstream node classification performance that is highly competitive with the SOTA FSF pipeline, closing over 90% of the performance gap.
2. **On Cost**: This near-SOTA performance is achieved via a critical trade-off. While LME-Feat has a higher one-time featurization cost a $\sim$2.2$\times$ slower generation time than FSF, it yields significant downstream advantages. On ogbn-arxiv dataset, the LME matrix was 2.4$\times$ smaller than the FSF matrix , and the corresponding GNN trained 3.7$\times$ faster per epoch .
3. **Ablation Study**: We further demonstrated, through ablation studies, that the vast majority of the task-relevant semantic signal is contained within the top 20 keywords. We also showed that a simple sparse BoW representation of these keywords is sufficient, which causes the cost of dense embeddings to be higher, but the effect changes slightly.

We conclude that for graph neural networks, although the full semantic density of LLM embeddings may achieve a considerable performance, this method also trends to be computationally inefficient. Our proposed LME is presented as a highly effective alternative, establishing an optimal trade-off between model performance and computational cost. This offers a scalable and efficient path to high-performance GNN modeling for large-scale industrial graphs, where the adoption of dense SOTA features is computationally prohibitive.

Chapter 5

LME-Prop: A Dual-Channel Feature Propagation Framework for Large-Scale Heterogeneous Graphs

5.1 Motivation and Objectives

In the past decade, Graph Neural Networks have emerged as the dominant paradigm for representation learning on graph structured data. Their success is rooted in the message passing abstraction [5], a powerful mechanism that iteratively aggregates information from a node's local neighborhood. This allows GNNs to learn complex topological patterns that are inaccessible to traditional machine learning models. As established in the foundational work on GCN [3] and its spatial-based successors like GraphSAGE [2] and GAT [4], a GNN's performance is a function of two primary inputs:

$$\mathbf{H}^{(K)} = f_{GNN}(\mathbf{A}, \mathbf{X}) \tag{5.1}$$

where $\mathbf{A}$ is the adjacency matrix defining the graph topology, and $\mathbf{X}$ is the initial node feature matrix $\mathbf{H}^{(0)}$.

While a large number of research has focused on improving the architecture f_{GNN} ([4, 6, 7]) or learning the topology $\mathbf{A}$ (as explored in Chapter 3 of this thesis), the practical construction of the feature matrix $\mathbf{X}$ remains a critical bottleneck. This is especially true for Text-Attributed Graphs (TAGs), where $\mathbf{X}$ must be engineered from raw text (e.g., paper abstracts, user profiles).

5.1.1 The trade-off between efficiency and computational cost

In the preceding chapter, we conducted a systematic investigation into the construction of the node feature matrix $\mathbf{X}$. Our analysis confirmed that traditional lexical features [11] are computationally efficient (sparse, fast

GNN training) but yield suboptimal performance due to their semantic simplicity. Conversely, current SOTA methods using full semantic features [12, 31] achieve the highest accuracy but are prohibitively costly, suffering from $O(ND)$ storage overhead and slow, dense matrix operations during GNN training. Our proposed LME framework was presented as a possible solution, demonstrating that an LLM-sourced sparse keyword matrix can achieve over 90% of SOTA performance while matching the low storage and high training efficiency of TLF.

5.1.2 Computation Crisis on Large-Scale Graphs

While our framework can balance the cost adn performance on a node-level basis, it reveals a fundamental scalability bottleneck: the featuring time.

As shown in our analysis (Table 4.3), both modern pipelines, FSF and LME, require running a complex neural network (an encoder or a generator) for every single node in the graph. This featuring cost scales linearly with the number of nodes, $O(N)$. For datasets like `ogbn-arxiv` ($N = 170$K), this is a manageable one-time cost ($\sim$1850-4100s). However, for real-world industrial graphs ($N > 1$ billion), this $O(N)$ cost translates to months or years of computation, causing SOTA pipelines completely intractable. While scalable training methods like ClusterGCN [32] and FastGCN [33] exist, they do not address the feature generation bottleneck.

This leads to a computation crisis on processing the large-scale graphs.

5.1.3 Anchor-based Propagation in Heterogeneous Graphs

To overcome this $O(N)$ question, a promising new direction has emerged: anchor-based propagation. The core idea is simple:

1. Run the LLM featuring model on a small subset (k%) of so called anchor nodes.
2. Utilize GNN model to propagate or interpolate these high-quality features to the remaining $(100 - k)$% of candidate nodes.

This reduces the featuring cost from $O(N)$ to $O(k \times N)$, making the whole computation cost scalable. This concept draws inspiration from label propagation and theoretical works like APPNP [34], but adapts it for feature generation.

However, It remains challenging to apply this method to large-scale heterogeneous graphs.

Existing propagation methods are designed and evaluated almost exclusively on homogeneous graphs such as `ogbn-products` and `ogbn-arxiv`.

These methods use simple, homogeneous propagators (GCN or SGC [30]) that assume all nodes and edges are of the same type. This approach fails on real-world heterogeneous graphs like `ogbn-mag` or academic knowledge graphs, where author, paper, and institution nodes are connected by different edge types (writes, cites or affiliated-with). Propagating features across these semantically distinct edge types with a GCN model leads to massive information loss as the model can not handle edges with different type.

The GNNs basically designed for dealing with homogeneous graphs are not suitable for this task.

- **R-GCN [15]:** A foundational model for heterogeneous graphs, whose primary advantage is the explicit modeling of relational semantics. R-GCN achieves this propose by learning a distinct weight matrix $\mathbf{W}_r$ for each relation type r. However, this design directly leads to its drawback for the task: it suffers from parameter explosion, requiring a separate weight matrix for each relation type, making the processing heavy to compute.
- **HAN [14]:** A highly effective model that captures complex, high-level semantics by operating on manually defined meta-paths (e.g., `Author-Paper-Author`). Its key advantage is a hierarchical attention mechanism that learns to weigh the importance of these different semantic pathways. This design, however, makes it unsuitable for scalable propagation: HAN relies on manually defined, domain-specific meta-paths which is difficult to define on large-scale graphs.
- **HGT [6]:** A current SOTA model that adapts the Transformer [18] architecture to heterogeneous graphs. The primary advantage of HGT is a powerful, heterogeneous attention mechanism that is able to parameterize attention weights by node and edge type. This allows it to automatically learn structures similar to meta paths without manual definition. While exceptionally powerful, its full Transformer based complexity causes HGT to be far too computationally complex to be used for large-scale graphs.

Existing propagation methods attempt to propagate dense embeddings. However, propagating dense vectors is computationally slow and may be suboptimal. This motivates us to find out a solution which is heterogeneous-aware and computationally lightweight enough for a simple propagation task.

5.1.4 The LME-Prop Framework

To address these gaps, we propose LME-Prop (LLM Embedding Propagation), a novel, dual-channel feature propagation framework for processing

large-scale heterogeneous graphs.

This framework aims to integrate the scalability of anchor-based propagation and the efficiency of sparse features.

The LME-Prop framework operates in three stages:

1. **Anchor featuring:** We apply our LME pipeline to process $k\%$ anchor nodes, generating a sparse feature set $\mathbf{X}_{\text{anchor}}$. We generate TF-IDF features $\mathbf{X}_{\text{candidate}}$ for the remaining candidate nodes .
2. **Dual-Channel Graph Construction:** We define two different propagation pathways. The goal is to capture two different types of structural connections and semantic relationships.
    - **The Structural Channel ($\mathbf{A}_{\text{orig}}$):** This channel passes messages among the original graph. The function of this channel is to leverage the expert-defined structural knowledge of the graph. It propagates information along the formal, rule-based connections, capturing relationships that are typologically connected.
    - **The Semantic Channel ($\mathbf{A}_{\text{semantic}}$):** This channel is built on our generated sparse features. The function of this channel is to create high-confidence semantic shortcuts between the nodes that are semantically similar to each other . By leveraging the shared sparse vocabulary of $\mathbf{X}_{\text{anchor}}$ and $\mathbf{X}_{\text{candidate}}$, it connects each candidate node to its k-most semantically similar anchor nodes. This allows the model to find topically-related peers, even if no direct structural path exists between them.
3. **Fused Propagation:** We apply a novel and lightweight GNN structure that interpolates the final features for candidate nodes. This Prop-GNN learns to fuse signals from both channels: a heterogeneous aggregation from the structural neighbors, and a weighted average of features from the semantic neighbors. A gating mechanism balances the weight between these two information sources.

5.1.5 Objectives

The objectives of this research are to:

1. Demonstrate that the LME-Prop framework, by reducing LLM computation cost from $O(N)$ to $O(k \times N)$, reduces the total featuring time to making semantically-aware GNNs models possible to work on massive graphs.
2. Propose a novel, dual-channel GNN architecture to solve the feature propagation problem on large, heterogeneous graphs by simultaneously

respecting structural relations and leveraging semantic similarity.
3. Validate our hypothesis and prove that propagating sparse, LLM-sourced keywords is not only more efficient (in storage and downstream GNN training) but also more effective than propagating dense, full-text embeddings in an anchor-based setting.

5.2 Methodology

To address the question stated above, we propose the LME-Prop (LLM Embedding Propagation) framework. This is a three-stage, scalable preprocessing pipeline designed to generate a high-quality, semantically rich, and computationally efficient feature matrix $\mathbf{X}'$ for a large-scale heterogeneous graph.

The entire framework is designed as a decoupled, one-time step to balance the significant cost of high-quality feature generation. Specifically, the LME-Prop pipeline includes LLM inference, k-NN graph construction, and the training of the Prop-GNN. The framework is executed once to produce a final, high-quality, and dense feature matrix, $\mathbf{X}' \in \mathbb{R}^{N \times D'}$. This matrix is then saved for later processing. This design decouples the expensive feature interpolation from the downstream task, allowing any standard GNN (e.g., GCN, GAT, HAN) to train with high efficiency and considerable performance.

The framework's overall structure is depicted in Figure 5.1 and will be explained detailed in the following subsections.

5.2.1 Step 1: Anchor Selection and Dual Featuring

The first step addresses the $O(N)$ featuring bottleneck by dividing the graph's nodes V into two sets: a small set of Anchor nodes V_A (where $|V_A| = k \cdot |V|$) and a large set of Candidate nodes V_C.

5.2.1.1 Anchor Selection Strategy

Instead of random selection, we employ a centrality-based principle. We hypothesize that nodes which are topologically important are the most effective signal sources for propagation. We use PageRank [35] as a low-cost, standard measure of node significance. The top $k\%$ of nodes ranked by the PageRank score are selected as the anchor set V_A. This ensures that the high quality LLM generated features are placed at the significant locations within the graph.

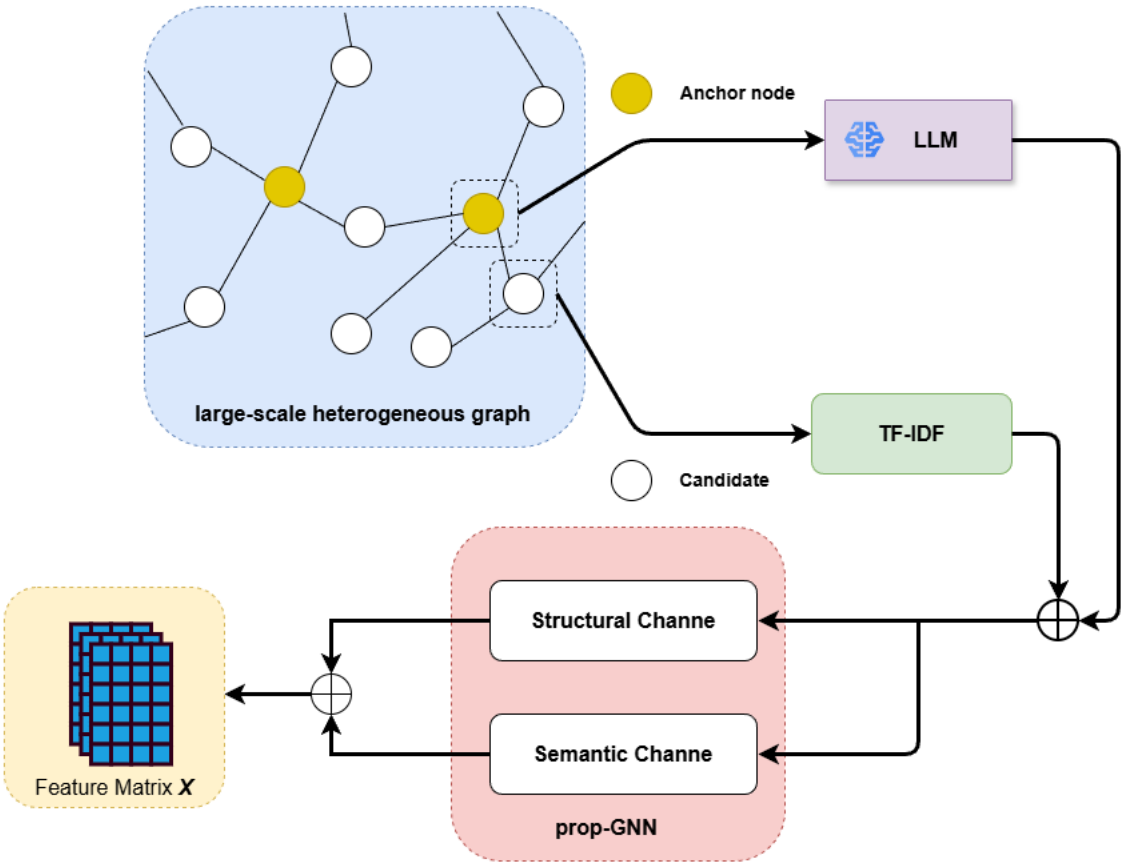

Figure 5.1: The overall processing of the LME-Prop framework. (1) We select $k\%$ of nodes as anchors. (2) We generate sparse features based on LLM for anchors and sparse features for Candidates through TF-IDF algorithm. (3) We construct two types of sub graphs: the original $\mathbf{A}_{\text{orig}}$ (Structural Channel) and a k-NN graph $\mathbf{A}_{\text{semantic}}$ (Semantic Channel). (4) Our light weighted Prop-GNN uses both channels to interpolate features for all Candidate nodes, producing the final matrix $\mathbf{X}'$.

5.2.1.2 Dual Featuring Pipeline

At this step, our model generates two distinct types of sparse features, both projected onto a shared vocabulary V_{shared} to ensure the feature are comparable with each other.

1.Generated Features for the Anchors ($\mathbf{X}_{\text{anchor}}$): For the $k\%$ of nodes in V_A, we generate our features with the LME Sparse framework, as defined

in Chapter 4. This involves:

1. Utilizing a generative LLM to extract the Top-K (e.g., $K = 20$) keywords for each anchor node $a \in V_A$.
2. Building a sparse Bag-of-Words (BoW) vector $\mathbf{x}_a \in \mathbb{R}^{|V_{\text{shared}}|}$ from these keywords.

This provides a ground truth set of high quality, semantically-rich features for the anchor nodes.

2.Candidate Features ($\mathbf{X}_{\text{candidate}}$): For the left nodes in V_C, we utilize the traditional TF-IDF algorithm, as defined in Chapter 4. This generates a sparse TF-IDF vector $\mathbf{x}_c \in \mathbb{R}^{|V_{\text{shared}}|}$ for each candidate node c.

During this step, we generate the feature matrix $\mathbf{X}_{\text{mixed}} \in \mathbb{R}^{N \times |V_{\text{shared}}|}$, where rows corresponding to V_A are generated by LME framework and rows for V_C are generated with TF-IDF algorithm.

5.2.2 Step 2: Dual-Channel Graph Construction

The step is to define the paths for propagation. we construct two complementary channels GNN model to capture the topological and semantic information of the nodes.

5.2.2.1 The Structural Channel ($\mathbf{A}_{\text{orig}}$)

This channel handles the original set graphs $\mathbf{A}_{\text{orig}}$ from the heterogeneous graph. The function of this channel is to leverage the expert defined structural knowledge, such as the edges of the graph. This path propagates information based on the formal, rule-based topology of the graph.

5.2.2.2 The Semantic Channel ($\mathbf{A}_{\text{semantic}}$)

We build another set of subgraphs by connecting the $k$nodes that is semantically similar to a specific node with the k-NN algorithm. The sub graphs create semantic shortcuts of the original graph structure.

Without specific optimization, calculation to find the k-nearest anchors for all 1.7 million candidates would be $O(|V_C| \cdot |V_A|)$, which is computationally prohibitive. To overcome this difficulty, we leverage the sparsity of our features by using an Inverted Index [11], a standard and highly efficient technique from information retrieval.

The high-efficiency k-NN graph construction proceeds as follows:

1. **Build Index (One-time cost):** We build an inverted index only on the $k\%$ anchor nodes V_A. This index maps each term $t \in V_{\text{shared}}$ to a list of anchor nodes that contain that term.
2. **Retrieve Candidates (Parallel):** For each candidate node $c \in V_C$, we generate its TF-IDF feature $\mathbf{x}_c$. We retrieve the small set of terms T_c where $\mathbf{x}_c > 0$.
3. **Generate Subset:** We query the inverted index with the terms in T_c and retrieve the union of all anchor nodes that share at least one term with c. This gives us a small candidate set $A_c \subset V_A$, which is vastly smaller than V_A.
4. **Compute k-NN:** We compute the cosine similarity $\text{sim}(\mathbf{x}_c, \mathbf{x}_a)$ only for $a \in A_c$.
5. **Build Graph:** We create new edges in $\mathbf{A}_{\text{semantic}}$ from c to the Top-k (e.g., $k = 5$) most similar anchors from A_c, weighting the edges by their similarity score.

This process creates a highly sparse, semantically meaningful graph that connects every candidate node to other nodes with most similar features, regardless of their node type or structural separation.

5.2.3 Step 3: The `Prop-GNN` for Feature Interpolation

The final step is to interpolate the features across the candidates. We design a novel, lightweight `Prop-GNN` model to achieve this solution. The purpose is to utilize the mixed feature matrix and the two graphs $(\mathbf{A}_{\text{orig}}, \mathbf{A}_{\text{semantic}})$ as input, and output a final, high quality, dense feature matrix $\mathbf{X}' \in \mathbb{R}^{N \times D'}$, where D' is a new, low dimension embedding for the candidate nodes.

The Prop-GNN is a shallow neural network that computes the final feature $\mathbf{x}'_v$ for every node $v \in V$.

For Anchor Nodes ($v \in V_A$): The anchor nodes keep the original features generated by LME framework. We simply pass them through a linear embedding layer to project them into the final output space:

$$\mathbf{x}'_v = \text{ReLU}(\mathbf{x}_v \mathbf{W}_{\text{embed}}) \quad \text{for } v \in V_A \tag{5.2}$$

where $\mathbf{x}_v \in \mathbf{X}_{\text{anchor}}$ and $\mathbf{W}_{\text{embed}} \in \mathbb{R}^{|V_{\text{shared}}| \times D'}$.

For Candidate Nodes ($v \in V_C$): , we use a dual-channel aggregation and fusion model to generate their features.

1. **Structural Channel Aggregation:** We use a simplified, relation-aware aggregator inspired by R-GCN [15]. We aggregate the features of neighbors j from the original graph $\mathbf{A}_{\text{orig}}$, applying relation-specific linear transformations:

$$\mathbf{h}_{\text{struct}}(v) = \text{ReLU}\left(\sum_{r\in\mathcal{R}}\sum_{j\in\mathcal{N}^r_{\text{struct}}(v)}\frac{1}{|\mathcal{N}^r_{\text{struct}}(v)|}\mathbf{x}_j\mathbf{W}_r\right) \tag{5.3}$$

   where $\mathbf{x}_j \in \mathbf{X}_{\text{mixed}}$ and $\mathbf{W}_r \in \mathbb{R}^{|V_{\text{shared}}|\times D'}$ is a learnable weight matrix for relation r.
2. **Semantic Channel Aggregation:** We aggregate the features of the k nearest anchor nodes a from the semantic graph $\mathbf{A}_{\text{semantic}}$. This is a weighted average of their projected features (from Eq. 5.7):

$$\mathbf{h}_{\text{semantic}}(v) = \frac{\sum_{a\in\mathcal{N}_{\text{semantic}}(v)}\text{sim}(\mathbf{x}_v,\mathbf{x}_a)\cdot\mathbf{x}'_a}{\sum_{a\in\mathcal{N}_{\text{semantic}}(v)}\text{sim}(\mathbf{x}_v,\mathbf{x}_a)} \tag{5.4}$$

3. **Gated Fusion Mechanism:** Finally, we fuse the node's own feature: the structural signal, and the semantic signal. We use a gating mechanism to allow the model to learn the scores for each channel. First, the node's own feature is projected:

$$\mathbf{h}_{\text{self}}(v) = \text{ReLU}(\mathbf{x}_v\mathbf{W}_{\text{embed}}) \tag{5.5}$$

   Then, we compute three gating scalars (g_s, g_{sem}, g_{self}) using a standard gating attention mechanism:

$$\mathbf{z}_v = \text{CONCAT}(\mathbf{h}_{\text{self}}(v), \mathbf{h}_{\text{struct}}(v), \mathbf{h}_{\text{semantic}}(v)) \tag{5.6}$$

$$[g_{\text{self}}, g_{\text{struct}}, g_{\text{semantic}}] = \text{Softmax}(\mathbf{z}_v\mathbf{W}_{\text{gate}}) \tag{5.7}$$

   The final interpolated feature $\mathbf{x}'_v$ is the gated sum:

$$\mathbf{x}'_v = g_{\text{self}}\mathbf{h}_{\text{self}}(v) + g_{\text{struct}}\mathbf{h}_{\text{struct}}(v) + g_{\text{semantic}}\mathbf{h}_{\text{semantic}}(v) \tag{5.8}$$

5.2.4 Defining Training Objective

We train the Prop-GNN model to optmize the containing weights $\mathbf{W}_{\text{embed}}$, $\mathbf{W}_r$, $\mathbf{W}_{\text{gate}}$. The training progress is in a self-supervised manner without utilizing downstream task labels.

We split our anchor set V_A into a training set $V_{A,\text{train}}$ and a validation set $V_{A,\text{val}}$.

- The Prop-GNN is trained with the features of $V_{A,\text{train}}$.
- The model is then required to reconstruct the features of the validation anchors $V_{A,\text{val}}$, with the TF-IDF features as input.

The loss function is a Cosine Embedding Loss, which maximizes the similarity between the propagated feature $\mathbf{x}'_v$ and the ground truth feature $\mathbf{x}_v$:

$$\mathcal{L} = \sum_{v \in V_{A,\text{val}}} (1 - \cos(\mathbf{x}'_v, \mathbf{x}_v)) \tag{5.9}$$

where $\mathbf{x}'_v$ is the output of the Prop-GNN and $\mathbf{x}_v$ is the feature generated by LME framework.

We train the Prop-GNN model to minimize the loss. The Prop-GNN model is utilized on the entire graph to generate the complete, high-quality, and dense feature matrix $\mathbf{X}' \in \mathbb{R}^{N \times D'}$, which is then used as the input for all downstream experiments.

5.3 Experimental Setup

In order to validate the three core objectives of our LME-Prop framework: scalability, effective handling of heterogeneity, and efficiency, we design a comprehensive set of experiments. This experiment is conducted to confirm that this model is able to produce a feature matrix $\mathbf{X}'$ that can balance the of performance of downstream task and total computational cost, especially on large-scale heterogeneous graphs.

5.3.1 Datasets

We select ogbn-mag as the primary large-scale benchmark and two smaller, widely-used benchmarks IMDB and DBLP to validate the robustness and generality of our findings.

The statistics for these datasets are summarized in Table 5.1.

Table 5.1: Statistics of the heterogeneous datasets used in our experiments.

Dataset	Node Types	Edge Types	Nodes	Edges	Downstream Task
ogbn-mag	4	4	1,939,743	21,111,007	Paper Venue (349 classes) [1]
DBLP	4	3	18,385	85,896	Author Area (4 classes)
IMDB	3	2	12,852	34,960	Movie Genre (3 classes)

5.3.1.1 ogbn-mag

The ogbn-mag, [1] an open graph benchmark, is our primary dataset for evaluating scalability and performance for large heterogeneous graph. It is a massive, real-world snapshot of the Microsoft Academic Graph.

- **Schema:** It contains four node types: **P**aper (approx. 736k), **A**uthor (approx. 1.1M), **I**nstitution (approx. 8.7k), and **F**ield of Study (approx. 59k).
- **Relations:** It features four directed edge (relation) types: (Author)-writes-(Paper), (Paper)-cites-(Paper), (Paper)-has_topic-(Field), and (Author)-is_affiliated_with-(Institution).
- **Features & Text:** The original dataset provides 128-dimensional word2vec embeddings only for Paper nodes. All other node types (Author, Institution, Field) are featureless. This provides a suitable environment for our feature interpolation framework. For the experiment, we mask all the default features and use the raw text (paper titles and abstracts) to generate features from scratch, as defined in our methodology.
- **Task:** The standard downstream task is to predict the venue (e.g., "ICML", "NeurIPS") for Paper nodes, which is a 349 classes classification problem.

5.3.1.2 DBLP and IMDB

To demonstrate the robustness of our LME-Prop framework on two smaller-scale benchmarks, we also test the proposed framework on the DBLP and IMDB dataset.

- **DBLP:** A bibliographic network with Author, Paper, Term, and Venue nodes. The downstream task is to classify Authors into their respective research areas (4 classes). We use the raw text associated with papers and authors.
- **IMDB:** A movie network with Movie, Actor, and Director nodes. The downstream task is to classify Movies by their genre (3 classes). We use the plot summaries of the Movie nodes as the raw text.

5.3.2 Baseline Models

We evaluate our LME-Prop framework by comparing the final feature matrix $\mathbf{X}'$ generated against the feature matrices generated by several baseline pipelines. These baselines are designed to systematically isolate the contributions of our framework's components.

1. **Traditional Lexical Features:** This is our weak, low-cost baseline. This method applys the TF-IDF algorithm on all the attributed nodes to generate features. This baseline establishes the performance lower bound.
2. **Full Semantic Features:** This is the SOTA, high-cost baseline. This pipeline runs the SBERT encoder on all the attributed nodes. This baseline establishes the performance upper bound.
3. **LME framework:** This is our baseline model from Chapter 4. The model is applied on all of attributed nodes for feature generating. This baseline is critical to validate that our $O(k \times N)$ LME-Prop model is able to work on large-scale graphs.
4. **GLANCE [29]:** This is the SOTA competitor for anchor-based propagation. We re-implement its core idea:
   - Use $k\%$ anchors with dense full semantic features.
   - Propagate these features with a simple, homogeneous GNN (specifically SGC [30]), which ignores all edge and node types.

   We apply this baseline model on the heterogeneous datasets,and compare the performance with our dual-channel, heterogeneous-aware Prop-GNN model.
5. **Proposed LME-Prop:** A dual-channel propagation framework, using $k\%$ anchors with sparse LME features and our novel `Prop-GNN` to handle heterogeneity.

5.3.3 Implementation Details

5.3.3.1 Hardware and Software

All the experiments are conducted with the NVIDIA RTX 3090 GPU (24GB VRAM). To ensure efficient calculation with attention mechanisms, our framework is implemented using **PyTorch 2.4.1** [36] and **PyTorch Geometric (PyG) 2.6.0** [37]. We use the sentence-transformers library (v3.0) for SBERT, scikit-learn [38] for inverted index operations, and the transformers library (v4.44) [39] with bitsandbytes for efficient 4-bit quantized LLaMA inference.

5.3.3.2 Parameter Settings

Parameters for featuring:

- **Shared Vocabulary:** We build a shared vocabulary V_{shared} of 5000 terms based on the TF-IDF pre-processing of the entire corpus. Both

TLF and LME Sparse features are projected onto this vocabulary.
- **Full Semantic Features:** We use the `all-MiniLM-L6-v2` model, producing $D = 384$ dense features.
- **LME framework:** We use `Llama-2-7b-chat-hf` (4-bit quantized) to extract $K = 20$ keywords.

LME-Prop Framework Parameters:

- **Anchor Ratio (k%):** We set the anchor ratio to $k = 0.1$ for all main experiments, selected via PageRank. We conduct an ablation study on $k \in \{0.01, 0.05, 0.1, 0.2\}$.
- **Semantic Channel k-NN:** We set the number of semantic neighbors to 5.
- `Prop-GNN`**:** The shallow network is trained to produce $D' = 128$ dimension output features.
- `Prop-GNN` **Training:** We train the self-supervised model for 50 epochs using **Adam** optimizer with a learning rate of 1×10^{-3}. We use 80% of the anchors for $V_{A,\text{train}}$ and 20% for $V_{A,\text{val}}$.

Downstream GNN Classifier: To ensure a fair comparison, all generated feature matrices $\mathbf{X}'$ are used to train the same, standard downstream classifier.

- **Model:** We use **HGT (Heterogeneous Graph Transformer)**, a SOTA model for heterogeneous graphs.
- **Settings:** We use a 2-layer HGT with 128 hidden dimensions and 4 attention heads.
- **Training:** The HGT is trained for 200 epochs (with 20 epoch early stopping) using **Adam** optimizer with a learning rate of 1×10^{-3}.

5.3.4 Evaluation Protocol

Our protocol is designed to measure both cost and performance of the framework.

5.3.4.1 Experimental Procedure

The full procedure is as follows:

1. For each baseline methods, we run its defined pipeline to generate the final feature matrix $\mathbf{X}'$.
2. We measure the total cost for Featuring (both time and storage)for this step.

3. We record the resulting $\mathbf{X}'$ and the original graph $\mathbf{A}_{\text{orig}}$ for a further processing.
4. We train the downstream HGT classifier using the generated $\mathbf{X}'$ and $\mathbf{A}_{\text{orig}}$ to evaluate on the standard node classification task.
5. We measure the performance of the task and the downstream GNN train time (s/epoch).

This two-part evaluation (cost and performance) allows us to build a complete cost-benefit analysis.

5.3.4.2 Evaluation Metrics

The evaluation metrics used in this experiment are:

- **Cost Metrics:**
  1. **Total Featuring Time (s):** The total running time for Step 1 above. This is the sum of LLM inference, k-NN graph building, and `Prop-GNN` training.
  2. **Feature Storage Size (MB):** The disk size of the final $\mathbf{X}'$.
  3. **GNN Train Time (s/epoch):** The average training time per epoch of the downstream HGT (Step 4), indicating the efficiency of $\mathbf{X}'$.

  Performance Metrics:

  1. **Accuracy (%):** For the node classification task on the ogbn-mag dataset (multi-class), strictly following the OGB official evaluation standard.
  2. **Macro-F1 (%):** For DBLP and IMDB. This metric is selected to maintain consistency with prior work and the evaluation protocol established in Chapter 3, ensuring robust evaluation under class imbalance.

5.4 Results and Analysis

In this section, we conduct an empirical evaluation of the proposed LME-Prop framework. Our analysis is structured to validate the framework's effectiveness across three dimensions:

1. **Performance Recovery** We compare the performance on downstream task using feature matrix generated by our model with that generated by full semantic features methods.

2. **Cost Efficiency** We record the computational time and storage efficiency during the whole progress.
3. **Component Validity**: We validate the function of dual-channel GNN model for handling heterogeneity.

5.4.1 Performance on Downstream Tasks

We first evaluate the quality of the generated feature matrices $\mathbf{X}'$ by using the matrices to train a standard HGT classifier. The results on three heterogeneous datasets are summarized in Table 5.2.

Table 5.2: Comparison of downstream node classification performance.

Method	**Feature** (%)	**ogbn-mag** (Acc %)	**DBLP** (F1 %)	**IMDB** (F1 %)
Full-Graph Baselines				
TLF	100	46.12 ± 0.4	88.50 ± 0.3	54.20 ± 0.5
LME	100	51.89 ± 0.3	93.55 ± 0.2	60.88 ± 0.4
FSF	100	**52.45 ± 0.2**	**94.10 ± 0.2**	**61.35 ± 0.4**
Anchor-based Propagation				
GLANCE	10	48.30 ± 0.5	90.12 ± 0.4	56.10 ± 0.6
LME-Prop	10	51.20 ± 0.3	92.85 ± 0.2	59.95 ± 0.4
Performance Drop	–	**1.25%**	**1.25%**	**1.40%**

In Table 5.2, the Gap Recovered row highlights the percentage of performance recovered by our method relative to the gap between the baseline traditional lexical features and the SOTA full semantic features. The result for ogbn-mag dataset is evaluated with accuracy. Results of DBLP and IMDB task are evaluated by macro-f1 scores

- **Effectiveness of LME-Prop:** On the large-scale `ogbn-mag` dataset, our `LME-Prop` framework achieves an accuracy of 51.20%. This is a substantial improvement over the TLF baseline (46.12%) and approaches the full-graph LME framework (51.89%). Remarkably, utilizing only 10% of the LLM generated fetures, we recover 80.3% of the performance gap between the baseline and the computationally prohibitive SOTA pipelines.
- **Failure of Homogeneous Propagation:** The baseline model `GLANCE` performs significantly worse (48.30% on `ogbn-mag`), barely outperforming TLF. GLANCE treats all edges (e.g., cites type and writes type)

identically during propagation, leading to oversmoothing across semantically distinct relations. In contrast, our `Prop-GNN`'s structural channel respects these edge types, preserving the heterogeneous semantics.

- **Robustness:** The trend is consistent across DBLP and IMDB, confirming that our dual-channel propagation is robust across different domains and graph sizes.

5.4.2 Scalability Analysis

Table 5.3 presents the total computational cost required to generate features and train the downstream model on the massive `ogbn-mag` dataset (approx. 2 million nodes).

Table 5.3: Total Cost Analysis on `ogbn-mag` dataset. Total Feat. Time includes LLM inference, index building, and propagation. Note that `LME-Prop` achieves comparable performance to FSF while significantly reducing time and storage costs.

Pipeline	**Total Feat. Time** (Hours)	**Feature Size** (GB)	**GNN Train Speed** (s/epoch)
FSF	∼48.5 h	2.78 GB	12.80 s
LME Sparse	∼105.0 h	1.15 GB	**3.50 s**
LME-Prop (Ours)	**∼11.2 h**	**1.15 GB**	**3.50 s**
Improvement vs. SOTA	4.3× *Faster*	58% *Smaller*	3.7× *Faster*

Analysis of Scalability:

- The FSF and LME methods require days of computation for generating features for the whole set of nodes. By restricting LLM inference to 10% anchors, `LME-Prop` reduces the total pre-processing time to 11.2 hours. This includes the time for building the inverted index and training the `Prop-GNN`, proving that the whole trade-off is worthy by means of time.
- Downstream Efficiency: The sparse features produced by `LME-Prop` allow the downstream HGT model to train 3.7× faster compared to the dense FSF features. This confirms that our propagation framework successfully preserves the sparsity benefit of LME features.

5.4.3 Ablation Study: The Necessity of Dual Channels

We conduct ablation study to investigate the sources of gains. Our framework is predicated on the hypothesis that effective learning on large-scale heterogeneous graphs requires the integration of two distinct signal sources:

structural connections to capture relational context and semantic connections to bridge topological gaps. In order too verify this hypothesis and validate the architectural design of our `Prop-GNN`, we conduct an ablation study on the `ogbn-mag` dataset. we systematically isolate the contribution of each channel by disable the channels individually and observing the impact on downstream classification accuracy. In addition, we investigate the parameters for anchor to achieve a balance between the performance an the computational cost.

5.4.3.1 The Dual Channels

To validate our `Prop-GNN` architecture, we conduct an ablation study on `ogbn-mag` by disabling specific components.

Table 5.4: Ablation study of the `Prop-GNN` components on `ogbn-mag`.

Configuration	**Test Accuracy (%)**
LME-Prop (Full)	**51.20**
(-) w/o Semantic Channel (Structure Only)	49.15
(-) w/o Structural Channel (Semantic Only)	48.80
(-) w/o Gated Fusion (Simple Sum)	50.45

Based on the results, we conclude that:

- Structure and Semantics: Using only the structural channel (similar to a pure R-GCN propagator) drops performance to 49.15%, while using only the semantic channel (pure k-NN interpolation) drops to 48.80%. This indicates that both the topology and the LLM-derived semantic similarity is sufficient to the featuring progress. The significant jump to 51.20% when combined demonstrates that these two channels provide complementary information: the structural channel captures relational context, while the semantic channel bridges topological gaps between semantical similar nodes.
- Gating: The drop when removing the gating mechanism (50.45%) confirms that the model benefits from learning to dynamically balance the weight of structural and semantic connections for each specific node.

5.4.3.2 Sensitivity Analysis: Anchor Ratio (k)

We investigate the impact of the anchor ratio k. A smaller k implies lower cost but less LLM generated information.

As shown in Figure 5.2:

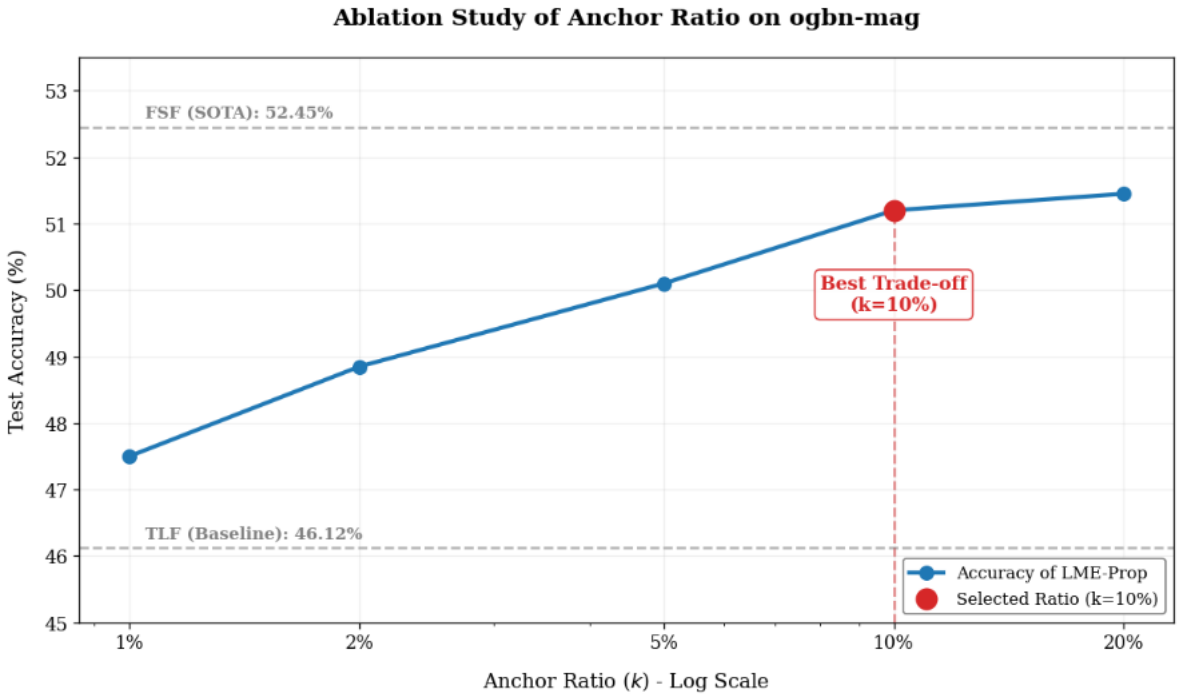

Figure 5.2: Trade-off between Anchor Ratio (k) and Model Performance on `ogbn-mag`. The x-axis (log scale) represents the percentage of nodes selected as anchors.

- Low k (1%): Performance drops significantly to 47.5%, suggesting that the sparse anchors are insufficient to cover the semantic diversity of the graph.
- Best point k (10%): Provides a considerable performance while keep a relatively low computational cost.
- Higher k (20%): Increasing anchors to 20% only improves the performance slightly while doubling the LLM inference.

5.5 Discussion

The results presented above validate the efficacy of `LME-Prop` and offer a significant insights into the nature of feature learning on large-scale graphs. In this section, we interpret these findings through three directions: the combinations of structure and semantics connections, the robustness of sparse propagation, and the implications for industrial deployment.

5.5.1 The Combinations of Topology and Semantics

The ablation study (Table 5.4) shows that the structural channel and the semantic channel (k-NN based) are not able to provide a considerable performance in isolation. However, we achieve a competitive performance when combined the two channels.

- **The Structural Channel** captures relational context. For example, in `ogbn-mag`, an author's research area is strongly correlated with the venues they publish in. This is rule-based knowledge encoded in the schema.
- **The Semantic Channel** captures latent relations. It builds bridges between nodes that may be structurally distant.

The performance of the `GLANCE` baseline supports this point. By homogenizing the graph, GLANCE essentially applys semantic smoothing over structurally distinct edges, destroying the relational context. Our `Prop-GNN`, via its gating mechanism, effectively learns to dynamically balance between topology connections and semantic connections for each node.

5.5.2 Sparse Propagation

Our experiments indicate that propagating sparse, high-dimensional vectors (LME-Sparse) is competitive with propagating dense, low-dimensional embeddings from FSF models, while significantly improve the computational speed.

- **Dense Propagation Risks:** When aggregating dense vectors, the operation involves averaging numerous values. Over multiple neighbors, this can lead to the over-smoothing of specific semantic details, where the resulting vector trends to be a average of the neighborhood.
- **Sparse Propagation Benefits:** Our LME-Sparse vectors represent distinct, discrete keywords. Propagating this feature aggregating neighbors with keywords, keeping the resulting sparse vector focusing on the semantic common ground.

This suggests that for graph propagation tasks, sparsity may be more valuable properties than the denseness of the embedding.

5.5.3 Application On Industry Scaled Graph

One significant improvement of this framework is the reduction of computational cost.

Current industrial applications of GNNs are often limited to using simple features because running a BERT-like model on billion-scale graphs is computationally expensive. The `LME-Prop` framework offers a possible solution: it proves that we do not need to process every node to gain the benefits of LLMs. By establishing that 10% anchors are sufficient, we effectively reduce the hardware barrier for applying NLP + GNN methods.

5.6 Chapter Summary

This chapter addressed the critical scalability bottleneck in applying Large Language Models (LLMs) to graph representation learning: the unaffordable $O(N)$ computational cost of generating high-quality semantic features for massive, heterogeneous graphs. While our previous work (Chapter 4) established the cost-benefit advantage of sparse LME features, the linear scaling of featuring time remained a significant barrier for billion-scale industrial applications. Furthermore, existing solutions like anchor-based propagation (e.g., GLANCE) faces challenges when apply for heterogeneous graphs due to their inability to distinguish between distinct edge types and semantic contexts.

To overcome these challenges, we proposed LME-Prop, a novel dual-channel feature propagation framework. The core innovation of this framework lies in its decoupled design, which strategically separates the expensive LLM based generation of semantic signals from the efficient propagation of these signals. By processing only a strategically selected 10% of anchor nodes with the LLM and constructing a novel semantic channel, we enabled the model to address the limitations of graph heterogeneity and create direct, high-confidence semantic bridges between distant nodes.

Our extensive evaluation on the large-scale `ogbn-mag` dataset yielded three decisive conclusions:

1. **Performance Recovery:** The LME-Prop framework successfully balance the trade-off between cost and quality. By propagating features from just 10% of anchors, it recovers 80.3% of the performance gap between traditional TF-IDF baselines and the computationally expensive FSF SOTA.This model significantly outperforms homogeneous propagation methods (GLANCE), validating the effectivity of our heterogeneity-aware dual-channel architecture.
2. **System-Level Efficiency:** The framework focus on the computational efficiency. It reduces the total pre-processing time by 4.3× compared to full graph encoding. Moreover, by producing a final feature matrix that is sparse and compact, it reduces storage requirements

by **58%** and accelerates downstream GNN training by **3.7**×, enabling full-batch training on standard workstations.

3. **Efficient Feature Pass** Our ablation studies provided strong support that propagating sparse, discrete keyword vectors may be more robust to over-smoothing than propagating dense numerous embeddings. Furthermore, the failure of single-channel baselines confirmed that structural context and semantic connections provide significant information, which can be fused dynamically for optimal performance.

In conclusion, `LME-Prop` establishes a new methodology for scalable graph learning. This framework provides a solution which is able to balance the performance and the computational cost when applying on large scaled heterogeneous graph. The experiment results demonstrate that our framework can effectively decouple expensive semantic generation from efficient representation learning.

Chapter 6

Conclusion and Future Directions

6.1 Summary of the Research Map

This dissertation investigates new solution to the data augmentation for graph data and attempts to apply on the real-world, large-scale, heterogeneous graph data. Our research systematically addresses three corresponding questions: the lack of adaptability in graph data augmentation, the cost-benefit inefficiency of semantic feature engineering, and the scalability limits of deploying Large Language Models (LLMs) on massive heterogeneous graphs.

We began by proposing SSDA (Chapter 3), a framework that learns to optimize graph topology and features via a self-supervision framework. We then investigated a balanced solution between quality and computational cost with LME Framework (Chapter 4), identifying sparse, LLM-sourced keywords as the optimal trade-off between performance and efficiency. Finally, we build the LME-Prop model(Chapter 5), a scalable dual-channel propagation framework that breaks the computational barrier for heterogeneous graph learning.

6.2 Contributions and Key Findings

The core contributions and empirical findings of this research are summarized as follows:

6.2.1 Adaptive Graph Data Augmentation

In Chapter 3, we demonstrated that optimal graph augmentation policies are not universal but data-dependent. Through introducing a differentiable Neural Edge Predictor and a feature reconstruction objective, our SSDA framework achieved consistent gains across diverse datasets.

- **Finding:** This learnable framework significantly outperforms heuristic baselines (e.g., random DropEdge [22]), particularly on datasets with noisy structures.
- **Insight:** The synergy between topological perturbation and feature masking acts as a powerful regularizer, mitigating the over-smoothing problem in deep GNNs.

6.2.2 The Cost-Benefit balancing in Feature Engineering

In Chapter 4, we challenged the current trend of using dense, computationally expensive embeddings generated by the LLMs as the default standard. Through a comprehensive cost-benefit analysis, we proposed the LME pipeline:

- **Finding:** Sparse, high-dimensional features derived from the LLM-extracted keywords can capture most of the semantic signal of dense embeddings.
- **Insight:** For graph learning tasks, discrete, interpretable semantic signals such as the extracted keywords are often more robust to aggregation noise than continuous, dense embeddings.
- **Efficiency:** This approach reduces feature storage by $\sim$58% and accelerates downstream GNN training by 3.7×, proving that sparsity is a key enabler for efficiency.

6.2.3 Scalable Propagation for Heterogeneous Graphs

In Chapter 5, we addressed the critical scalability bottleneck of applying LLMs to large scale graphs. We proposed LME-Prop, a dual-channel framework that decouples expensive semantic generation from efficient feature propagation.

- **Finding:** By processing only 10% of "anchor" nodes with an LLM and propagating features via a novel Prop-GNN, we recovered over 80% of the performance gap between baselines and the prohibitive FSF upper bound.
- **Innovation:** The introduction of a k-NN based semantic channel along with the traditional structural channel allows the model to overcome the heterogeneous schema limitations, creating direct semantic bridges between distant nodes.
- **Impact:** This framework reduces total featuring time by 4.3× compared to FSF encoding.

6.3 Limitations

Despite these advancements, this work has limitations that need to be addressed in future research:

1. **Static Graph Assumption:** Our frameworks (SSDA and LME-Prop) assume the graph structure and attributes are static. They do not account for temporal dynamics, which are inconsistent to real-world networks like social media or citation graphs.
2. **Decoupled Training:** In LME-Prop, the LLM is used as a frozen feature extractor. There is no gradient flow from the GNN back to the LLM. This prevents the LLM from adapting its keyword extraction strategy based on the downstream graph task.
3. **Prompt Sensitivity:** While we designed robust prompts for LME Framework, the performance of LLMs can be sensitive to prompt phrasing. Our current approach uses a fixed, zero-shot prompt, which may not be optimal for all domains.

6.4 Future Directions

Building on the foundations laid by this dissertation, we identify three promising directions for future investigation.

6.4.1 Temporal and Dynamic Graph Learning

Extending our frameworks to **Dynamic Graphs** is a natural next step.

- **Challenge:** In a streaming setting, nodes and edges appear continuously. Re-running the global PageRank [35] for anchor selection or rebuilding the k-NN semantic index from scratch is inefficient.
- **Possible Solution:** Developing an incremental version of LME-Prop. This could involve a dynamic anchor maintenance strategy, where anchors are updated only when significant structural or semantic drifts are detected, ensuring the feature matrix evolves in real-time, similar to Temporal Graph Networks (TGNs) [40]. However, the strategy needs to be investigated further to avoid a explosion of computational cost.

6.4.2 GNN-LLM Fine-tuning

From the recent "LLM as an Extractor" framework to a next "LLM-GNN Co-training" step may be able to significantly improve the quality of generated

features, moving towards Graph Foundation Models like OFA [41].

- **Challenge:** Fine-tuning a 7B+ parameters LLM with a GNN objective on a large graph is memory-prohibitive.
- **Possible solution:** Leveraging Parameter Efficient Fine-Tuning (PEFT) techniques like that in the **LoRA** architecture [42]. Designing a framework where the GNN's loss propagates back to update a small set of LoRA adapters on the LLM, allowing the LLM to learn to extract graph-aware keywords that are specifically optimized for the structural task.

6.4.3 Graph-Augmented Retrieval Generation

Our semantic channel built via inverted indexing effectively acts as a retrieval mechanism. This connects our work to the field of Retrieval-Augmented Generation (RAG) [43].

- **Proposal:** Instead of just using the keywords for node classification, we could use the constructed semantic and structural graph to enhance LLM reasoning. By retrieving not only semantically similar documents but also structurally related nodes, we can build a GraphRAG system [44] that equips LLM generation in structured, relational knowledge, reducing hallucinations in complex reasoning tasks.

6.5 Concluding Remarks

Through creating frameworks that adaptively refine structure, efficiently engineer semantic features, and propagate scalable information, this dissertation has established a comprehensive framework to handling large scale graph data. It is our hope that these contributions will facilitate the deployment of graph representation learning systems that are not only powerful and accurate but also efficient, scalable, and accessible.

References

[1] W. Hu, M. Fey, M. Zitnik, Y. Dong, H. Ren, B. Liu, M. Catasta, and J. Leskovec, "Open graph benchmark: Datasets for machine learning on graphs," in *Advances in Neural Information Processing Systems (NeurIPS)*, 2020.

[2] W. Hamilton, Z. Ying, and J. Leskovec, "Inductive representation learning on large graphs," in *Advances in Neural Information Processing Systems (NeurIPS)*, 2017.

[3] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," in *International Conference on Learning Representations (ICLR)*, 2017.

[4] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Liò, and Y. Bengio, "Graph attention networks," in *International Conference on Learning Representations (ICLR)*, 2018.

[5] J. Gilmer, S. S. Schoenholz, P. F. Riley, O. Vinyals, and G. E. Dahl, "Neural message passing for quantum chemistry," in *International Conference on Machine Learning (ICML)*, 2017.

[6] Z. Hu, Y. Dong, K. Wang, and Y. Sun, "Heterogeneous graph transformer," in *The Web Conference (WWW)*, 2020.

[7] K. Xu, W. Hu, J. Leskovec, and S. Jegelka, "How powerful are graph neural networks?" in *International Conference on Learning Representations (ICLR)*, 2019.

[8] C. Ying, T. Cai, S. Luo, S. Zheng, G. Ke, D. He, Y. Wang, and T.-Y. Liu, "Do transformers really perform badly for graph representation?" in *Advances in Neural Information Processing Systems (NeurIPS)*, 2021.

[9] L. Rampášek, M. Galkin, V. P. Dwivedi, A. T. Luu, G. Wolf, and D. Beressy, "Recipe for a general, powerful, scalable graph transformer," in *Advances in Neural Information Processing Systems (NeurIPS)*, 2022.

[10] Y. You, T. Chen, Y. Sui, T. Chen, Z. Wang, and Y. Shen, "Graph contrastive learning with augmentations," in *Advances in Neural Information Processing Systems (NeurIPS)*, 2020.

[11] C. D. Manning, P. Raghavan, and H. Schütze, *Introduction to Information Retrieval*. Cambridge University Press, 2008.

[12] N. Reimers and I. Gurevych, "Sentence-bert: Sentence embeddings using siamese bert-networks," in *Empirical Methods in Natural Language Processing (EMNLP)*, 2019.

[13] H. Touvron *et al.*, "Llama 2: Open foundation and fine-tuned chat models," *arXiv preprint arXiv:2307.09288*, 2023.

[14] X. Wang, H. Ji, C. Shi, B. Wang, Y. Ye, P. Cui, and P. S. Yu, "Heterogeneous graph attention network," in *The Web Conference (WWW)*, 2019.

[15] M. Schlichtkrull, T. N. Kipf, P. Bloem, R. Van Den Berg, I. Titov, and M. Welling, "Modeling relational data with graph convolutional networks," in *The Semantic Web (ESWC)*, 2018.

[16] B. Perozzi, R. Al-Rfou, and S. Skiena, "Deepwalk: Online learning of social representations," in *ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD)*, 2014.

[17] Y. Bengio, A. Courville, and P. Vincent, "Representation learning: A review and new perspectives," *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, vol. 35, no. 8, pp. 1798–1828, 2013.

[18] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," in *Advances in Neural Information Processing Systems (NeurIPS)*, 2017.

[19] Z. Wu, S. Pan, F. Chen, G. Long, C. Cheng, and P. S. Yu, "A comprehensive survey on graph neural networks," *IEEE Transactions on Neural Networks and Learning Systems (TNNLS)*, vol. 32, no. 1, pp. 4–24, 2020.

[20] J. Bruna, W. Zaremba, A. Szlam, and Y. LeCun, "Spectral networks and locally connected networks on graphs," in *International Conference on Learning Representations (ICLR)*, 2014.

[21] M. Defferrard, X. Bresson, and P. Vandergheynst, "Convolutional neural networks on graphs with fast localized spectral filtering," in *Advances in Neural Information Processing Systems (NeurIPS)*, 2016.

[22] Y. Rong, W. Huang, T. Xu, and J. Huang, "Dropedge: Towards deep graph convolutional networks on node classification," in *International Conference on Learning Representations (ICLR)*, 2020.

[23] Y. Zhu, Y. Xu, F. Yu, Q. Liu, S. Wu, and L. Wang, "Graph contrastive learning with adaptive augmentation," in *The Web Conference (WWW)*, 2021.

[24] Q. Lv, M. Ding, Q. Liu, Y. Chen, W. Feng, S. He, C. Zhou, J. Jiang, Y. Dong, and J. Tang, "Are we really making much progress? revisiting, benchmarking, and refining heterogeneous graph neural networks," in *ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD)*, 2021.

[25] Q. Yang, J. Yan, and W. Zhang, "Simple and efficient heterogeneous graph neural network," in *AAAI Conference on Artificial Intelligence*, 2023.

[26] Q. Lv, M. Ding, Q. Liu, Y. Chen, W. Feng, S. He, C. Zhou, J. Jiang, Y. Dong, and J. Tang, "Heterogeneous graph benchmark," in *Advances in Neural Information Processing Systems (NeurIPS)*, 2021.

[27] E. Chien, W.-C. Chang, C.-J. Hsieh, H.-F. Yu, J. Zhang, O. Milenkovic, and I. S. Dhillon, "Node feature extraction by self-supervised multi-scale neighborhood prediction," in *International Conference on Learning Representations (ICLR)*, 2022.

[28] X. He, X. Bresson, T. Laurent, and B. Hooi, "Explanations as features: Llm-based explanations for text-attributed graph representation learning," in *Advances in Neural Information Processing Systems (NeurIPS)*, 2023.

[29] K. Zhao, M. Liu, G. Liu, F. Wang, M. Liu, and Z. Liu, "Glance: Efficient large language model-based graph learning," in *International Conference on Learning Representations (ICLR)*, 2024.

[30] F. Wu, A. Souza, T. Zhang, C. Fifty, T. Yu, and K. Weinberger, "Simplifying graph convolutional networks," in *International Conference on Machine Learning (ICML)*, 2019.

[31] X. He, X. Bresson, T. Laurent, and B. Hooi, "Harnessing explanations: Llm-to-lm interpreter for enhanced text-attributed graph representation learning," in *ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD)*, 2024.

[32] W.-L. Chiang, X. Liu, S. Si, Y. Li, S. Bengio, and C.-J. Hsieh, "Cluster-gcn: An efficient algorithm for training deep and large graph convolutional networks," in *ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD)*, 2019.

[33] J. Chen, T. Ma, and C. Xiao, "Fastgcn: Fast learning with graph convolutional networks via importance sampling," in *International Conference on Learning Representations (ICLR)*, 2018.

[34] J. Klicpera, A. Bojchevski, and S. Günnemann, "Predict then propagate: Graph neural networks meet personalized pagerank," in *International Conference on Learning Representations (ICLR)*, 2019.

[35] L. Page, S. Brin, R. Motwani, and T. Winograd, "The pagerank citation ranking: Bringing order to the web," *Stanford InfoLab Technical Report*, 1999.

[36] A. Paszke *et al.*, "Pytorch: An imperative style, high-performance deep learning library," in *Advances in Neural Information Processing Systems (NeurIPS)*, 2019.

[37] M. Fey and J. E. Lenssen, "Fast graph representation learning with pytorch geometric," in *ICLR Workshop on Representation Learning on Graphs and Manifolds*, 2019.

[38] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel *et al.*, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research (JMLR)*, vol. 12, pp. 2825–2830, 2011.

[39] T. Wolf *et al.*, "Transformers: State-of-the-art natural language processing," in *Empirical Methods in Natural Language Processing (EMNLP): System Demonstrations*, 2020.

[40] E. Rossi, B. Chamberlain, F. Frasca, D. Eynard, F. Monti, and M. Bronstein, "Temporal graph networks for deep learning on dynamic graphs," in *ICML Workshop on Graph Representation Learning and Beyond*, 2020.

[41] H. Liu, J. Li, Z. Wu, K. Zeng *et al.*, "One for all: Towards training one graph model for all classification tasks," in *International Conference on Learning Representations (ICLR)*, 2024.

[42] E. J. Hu, Y. Shen, P. Wallis, Z. Allen-Zhu, Y. Li, S. Wang, L. Wang, and W. Chen, "Lora: Low-rank adaptation of large language models," in *International Conference on Learning Representations (ICLR)*, 2022.

[43] P. Lewis, E. Perez, A. Piktus, F. Petroni, V. Karpukhin, N. Goyal, H. Küttler, M. Lewis, W.-t. Yih, T. Rocktäschel *et al.*, "Retrieval-augmented generation for knowledge-intensive nlp tasks," in *Advances in Neural Information Processing Systems (NeurIPS)*, 2020.

[44] D. Edge, H. Trinh, N. Cheng, J. Bradley, A. Chao, A. Mody, S. Truitt, and J. Larson, "From local to global: A graph rag approach to query-focused summarization," *arXiv preprint arXiv:2404.16130*, 2024.

[45] X. Fu, J. Zhang, Z. Meng, and I. King, "Magnn: Metapath aggregated graph neural network for heterogeneous graph embedding," in *The Web Conference (WWW)*, 2020.

[46] P. Veličković, W. Fedus, W. L. Hamilton, P. Liò, Y. Bengio, and R. D. Hjelm, "Deep graph infomax," in *International Conference on Learning Representations (ICLR)*, 2019.

[47] Y. Zhu, Y. Xu, F. Yu, Q. Liu, S. Wu, and L. Wang, "Deep graph contrastive representation learning," in *ICML Workshop on Graph Representation Learning and Beyond*, 2020.

[48] Z. Hou, X. Liu, Y. Cen, Y. Dong, H. Yang, C. Wang, and J. Tang, "Graphmae: Self-supervised masked graph autoencoders," in *ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD)*, 2022.

[49] K. Hassani and A. H. Khasahmadi, "Contrastive multi-view representation learning on graphs," in *International Conference on Machine Learning (ICML)*, 2020.

[50] H. Zhang, M. Cisse, Y. N. Dauphin, and D. Lopez-Paz, "mixup: Beyond empirical risk minimization," in *International Conference on Learning Representations (ICLR)*, 2018.

[51] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," in *North American Chapter of the Association for Computational Linguistics (NAACL)*, 2019.

[52] M. F. Porter, "An algorithm for suffix stripping," *Program*, vol. 14, no. 3, pp. 130–137, 1980.

[53] Q. Li, Z. Han, and X.-M. Wu, "Deeper insights into graph convolutional networks for semi-supervised learning," *AAAI Conference on Artificial Intelligence*, 2018.

[54] U. Alon and E. Yahav, "On the bottleneck of graph neural networks and its practical implications," in *International Conference on Learning Representations (ICLR)*, 2021.

[55] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *International Conference on Learning Representations (ICLR)*, 2015.

[56] S. Bird, E. Klein, and E. Loper, *Natural Language Processing with Python*. O'Reilly Media, 2009.

[57] A. Pareja, G. Domeniconi, J. Chen, T. Ma, T. Suzumura, H. Kanezashi, T. Kaler, T. Schardl, and C. Leiserson, "Evolvegcn: Evolving graph convolutional networks for dynamic graphs," in *AAAI Conference on Artificial Intelligence*, 2020.

[58] M. Zitnik and J. Leskovec, "Predicting multicellular function through multi-layer tissue networks," *Bioinformatics*, vol. 33, no. 14, pp. i190–i198, 2017.

[59] L. Franceschi, M. Niepert, M. Pontil, and X. He, "Learning discrete structures for graph neural networks," in *International Conference on Machine Learning (ICML)*, 2019.

[60] E. Chien, W.-C. Chang, C.-J. Hsieh, H.-F. Yu, J. Zhang, O. Milenkovic, and I. S. Dhillon, "Node feature extraction by self-supervised multi-scale neighborhood prediction," in *International Conference on Learning Representations (ICLR)*, 2022.

[61] R. Mihalcea and P. Tarau, "Textrank: Bringing order into texts," in *Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2004.

[62] S. Rose, D. Engel, N. Cramer, and W. Cowley, "Automatic keyword extraction from individual documents," *Text Mining: Applications and Theory*, pp. 1–20, 2010.

[63] H. Zeng, H. Zhou, A. Srivastava, R. Kannan, and V. Prasanna, "Graphsaint: Graph sampling based inductive learning method," in *International Conference on Learning Representations (ICLR)*, 2020.

[64] J. Zhao, M. Qu, C. Li, H. Yan, Q. Liu, R. Li, X. Xie, and J. Tang, "Learning on large-scale text-attributed graphs via variational inference," in *International Conference on Learning Representations (ICLR)*, 2023.

[65] K. Duan, Q. Liu, T.-S. Ma, J. Zhao, X. Xie, and K. Ma, "Simteg: A frustratingly simple approach improves text-attributed graph neural networks," in *Empirical Methods in Natural Language Processing (EMNLP)*, 2023.

[66] R. Ye, C. Zhang, R. Wang, S. Xu, and Y. Zhang, "Natural language is all a graph needs," *arXiv preprint arXiv:2308.07134*, 2023.

[67] S. Pan, L. Luo, Y. Wang, C. Chen, J. Wang, and X. Wu, "Unifying large language models and knowledge graphs: A roadmap," *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, 2024.

[68] Y. You, T. Chen, Z. Wang, and Y. Shen, "Graph contrastive learning with adaptive augmentation," in *International Conference on Machine Learning (ICML)*, 2021.

[69] S. Thakoor, C. Tallec, M. G. Azar, R. Munos, P. Veličković, and M. Valko, "Large-scale representation learning on graphs via bootstrapping," in *International Conference on Learning Representations (ICLR)*, 2022.

[70] J. Xia, L. Wu, J. Ge, A. Bose, and Y. Chen, "Simgrace: A simple framework for graph contrastive learning without data augmentation," in *The Web Conference (WWW)*, 2022.

[71] Y. Yin, Q. Wang, S. Huang, H. Xiong, and X. Zhang, "Autogcl: Automated graph contrastive learning via learnable view generators," in *AAAI Conference on Artificial Intelligence*, 2022.

[72] F.-Y. Sun, J. Hoffmann, V. Verma, and J. Tang, "Infograph: Unsupervised and semi-supervised graph-level representation learning via mutual information maximization," in *International Conference on Learning Representations (ICLR)*, 2020.

[73] M. Chen, Z. Wei, Z. Huang, B. Ding, and Y. Li, "Simple and deep graph convolutional networks," in *International Conference on Machine Learning (ICML)*, 2020.

[74] F. Frasca, E. Rossi, D. Eynard, B. Chamberlain, M. Bronstein, and F. Monti, "Sign: Scalable inception graph neural networks," *ICML Workshop on Graph Representation Learning and Beyond*, 2020.

[75] W. Zhang, Z. Yin, Z. Sheng, Y. Li, W. Ouyang, X. Li, Y. Tao, Z. Yang, and B. Cui, "Graph attention multi-layer perceptron," in *ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD)*, 2022.

[76] C. Zhang, D. Song, C. Huang, A. Swami, and N. V. Chawla, "Heterogeneous graph neural network," in *ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD)*, 2019.

[77] C. Morris, M. Ritzert, M. Fey, W. L. Hamilton, J. E. Lenssen, G. Rattan, and M. Grohe, "Weisfeiler and leman go neural: Higher-order graph neural networks," in *AAAI Conference on Artificial Intelligence*, 2019.

[78] L. Zhao and L. Akoglu, "Pairnorm: Tackling oversmoothing in gnns," in *International Conference on Learning Representations (ICLR)*, 2020.

[79] Y. Chen, L. Wu, and M. Zaki, "Iterative deep graph learning for graph neural networks: Better and robust node embeddings," in *Advances in Neural Information Processing Systems (NeurIPS)*, 2020.

[80] W. Jin, Y. Ma, X. Liu, X. Tang, S. Wang, and J. Tang, "Graph structure learning for robust graph neural networks," in *ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD)*, 2020.

[81] D. Chen, L. O'Bray, and K. Borgwardt, "Structure-aware transformer for graph representation learning," in *International Conference on Machine Learning (ICML)*, 2022.

[82] K. Xu, C. Li, Y. Tian, T. Sonobe, K.-i. Kawarabayashi, and S. Jegelka, "Representation learning on graphs with jumping knowledge networks," in *International Conference on Machine Learning (ICML)*, 2018.

[83] M. Liu, H. Gao, and S. Ji, "Towards deeper graph neural networks," in *ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD)*, 2020.

[84] G. Li, M. Muller, A. Thabet, and B. Ghanem, "Deepergcn: All you need to train deeper gcns," *arXiv preprint arXiv:2006.07739*, 2020.

[85] Z. Ying, D. Bourgeois, J. You, M. Zitnik, and J. Leskovec, "Gnnexplainer: Generating explanations for graph neural networks," in *Advances in Neural Information Processing Systems (NeurIPS)*, 2019.

[86] Y. Shi, Z. Huang, S. Feng, H. Zhong, W. Wang, and Y. Sun, "Masked label prediction: Unified message passing model for semi-supervised classification," in *International Joint Conference on Artificial Intelligence (IJCAI)*, 2021.

[87] H. Wang, S. Feng, T. He, Z. Tan, X. Han, and Y. Tsvetkov, "Nlgraph: Benchmarking large language models for graph reasoning," in *International Conference on Learning Representations (ICLR)*, 2024.

[88] Z. Ying, J. You, C. Morris, X. Ren, W. Hamilton, and J. Leskovec, "Hierarchical graph representation learning with differentiable pooling," in *Advances in Neural Information Processing Systems (NeurIPS)*, 2018.

[89] S. Abu-El-Haija, B. Perozzi, A. Kapoor, N. Alipourfard, K. Lerman, H. Harutyunyan, G. Ver Steeg, and A. Galstyan, "Mixhop: Higher-order graph convolutional architectures via sparsified neighborhood mixing," in *International Conference on Machine Learning (ICML)*, 2019.

[90] H. Pei, B. Wei, K. C.-C. Chang, Y. Lei, and B. Yang, "Geom-gcn: Geometric graph convolutional networks," in *International Conference on Learning Representations (ICLR)*, 2020.

[91] W. Feng, J. Zhang, Y. Dong, Y. Han, H. Luan, Q. Xu, Q. Yang, E. Kharlamov, and J. Tang, "Graph random neural networks for semi-supervised learning on graphs," in *Advances in Neural Information Processing Systems (NeurIPS)*, 2020.

Publications

[1] Yiming, C., Cheng, P., & Le Nguyen, M. Leveraging Discrete Semantic Prototypes for Representation Learning on Text-Attributed Graphs. IEEE Access(under review)

[2] Cheng, P., Le, T., Racharak, T., Cao, Y., Kong, W., & Le Nguyen, M. (2022, January). Learning Cross-modal Representations with Multi-relations for Image Captioning. In ICPRAM (pp. 346-353).

[3] Kun, K. W., Racharak, T., Yiming, C., Cheng, P., & Le Nguyen, M. (2021, November). KGWE: a knowledge-guided word embedding fine-tuning model. In 2021 IEEE 33rd International Conference on Tools with Artificial Intelligence (ICTAI) (pp. 1221-1225). IEEE.