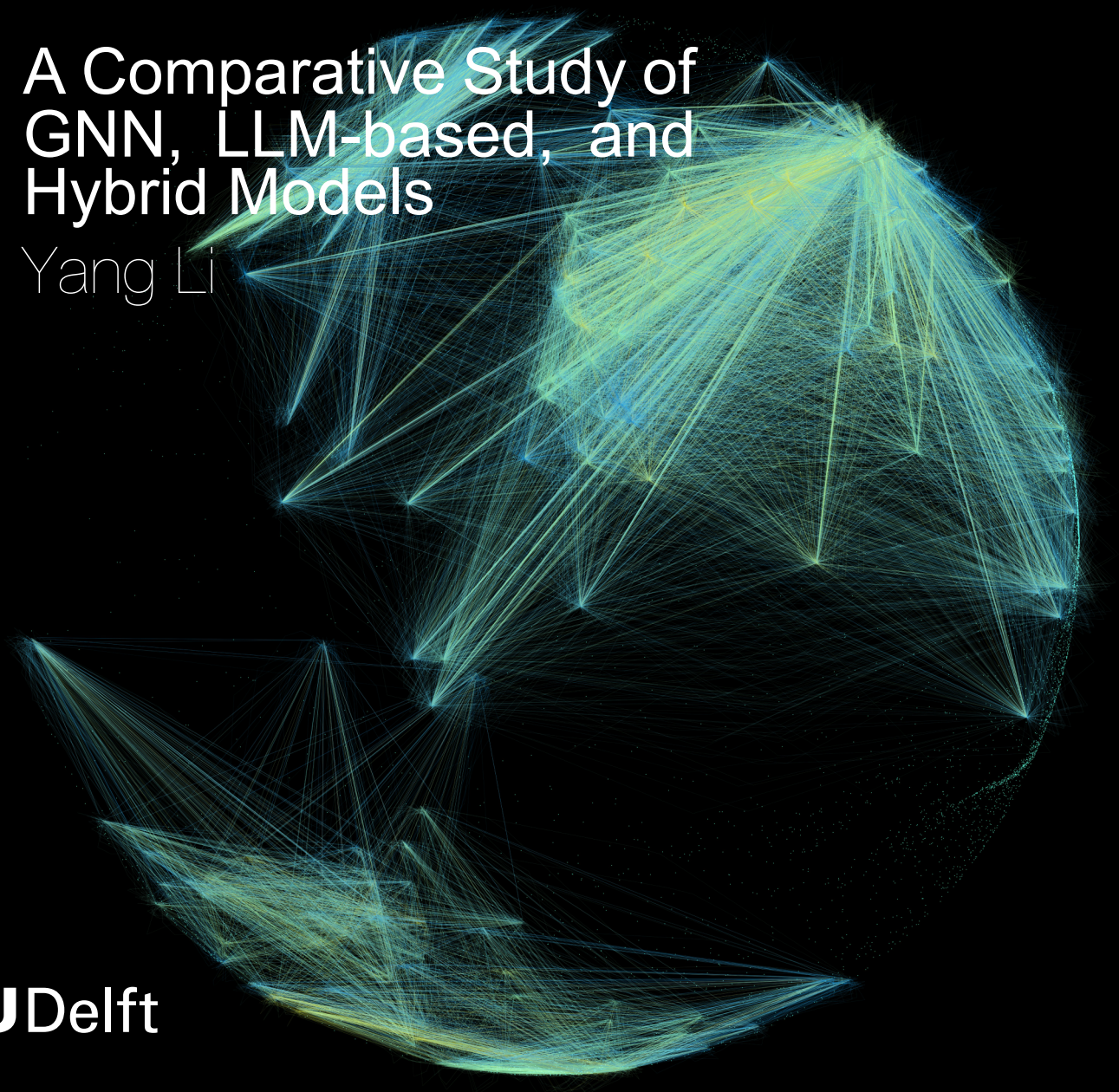
GNN–LLM Hybrids for Node Classification

A Comparative Study of GNN, LLM-based, and Hybrid Models

Yang Li

Delft University of Technology



TU Delft

GNN–LLM Hybrids for Node Classification

A Comparative Study of GNN, LLM-based, and Hybrid Models

by

Yang Li

to obtain the degree of Master of Science
at the Delft University of Technology,
to be defended publicly on Wednesday February 25, 2026 at 13:00 PM.

Student number:	5922674	
Project duration:	May 20, 2025 – February 25, 2026	
Committee Members:	Dr. E. Isufi,	TU Delft, Thesis Advisor
	Dr. M. Khosla,	TU Delft, Daily Supervisor
	T. Zhao,	TU Delft, Daily Co-Supervisor
	Dr. R. Hai,	TU Delft

TU Delft

Abstract

Node classification on text-attributed graphs requires both structural reasoning and rich semantic understanding. While Graph Neural Networks (GNNs) have become the dominant solution by leveraging graph topology, they often rely on limited textual representations. Recent advances, therefore, explore integrating large language models (LLMs) to provide stronger semantic encoding for graph learning. However, existing work often evaluates different LLM integration paradigms under individually designed experimental settings, making it difficult to assess their relative strengths for node classification tasks. In this work, we present a controlled empirical comparison between classical message-passing graph neural networks, parameter-efficient LLM-GNN integration (ENGINE), prompt-based generative reasoning (LLaGA), and a lightweight hybrid model that combines structural message passing with label-aware semantic alignment. We evaluate all models on two widely used textual graph benchmarks, Cora and WikiCS, under a unified transductive evaluation protocol and varying levels of training supervision. Our results show that the performance of ENGINE consistently outperforms strong GNN baselines, whereas LLaGA is more sensitive to inference constraints and evaluation protocols. The additional hybrid model we propose in this thesis further demonstrates complementary benefits, particularly in low-supervision regimes. These findings clarify practical trade-offs between discriminative and generative LLM-based graph models and highlight hybrid designs as a promising direction for efficient and robust node classification on textual graphs.

Contents

Abstract ii

1 Introduction 1

2 Related Work 3
2.1 Graph Neural Networks (GNNs) . . . 3
2.1.1 Graph Convolutional Networks (GCNs) . . . 3
2.1.2 Graph Attention Networks (GATs) . . . 4
2.1.3 GraphSAGE . . . 5
2.1.4 Summary . . . 5
2.2 LLM-Based Models . . . 6
2.2.1 ENGINE . . . 6
2.2.2 LLaGA . . . 8
2.2.3 Use case of ENGINE and LLaGA . . . 10

3 Methodology 13
3.1 A Lightweight Language-guided Hybrid Model . . . 13
3.1.1 Token Construction . . . 13
3.1.2 Prompt-Guided Cross-Attention . . . 14
3.1.3 Node-wise Dynamic Gating and Prediction . . . 16
3.2 Implementation and Adaptation Details . . . 16
3.2.1 Learning Setting . . . 16
3.2.2 GNN Models . . . 17
3.2.3 ENGINE . . . 18
3.2.4 LLaGA . . . 19

4 Experimental Results 23
4.1 Experimental Setup . . . 23
4.1.1 Datasets . . . 23
4.1.2 Evaluation Protocol . . . 25
4.2 Results Analysis . . . 25
4.2.1 Horizontal Comparison: Different Models on the Same Dataset . . . 26
4.2.2 Vertical Comparison: Model Behavior Across Training Regimes . . . 26
4.2.3 Overall Discussion . . . 27
4.3 Answers to the Research Questions . . . 28
4.3.1 Parameter-Efficient Integration vs. Prompt-Based Reasoning(RQ1) . . . 28
4.3.2 Do LLM Representations Help?(RQ2) . . . 29
4.3.3 Complementary Effects of Our Lightweight model.(RQ3) . . . 30
4.3.4 Model Robustness Under Limited Supervision.(RQ4) . . . 30
4.4 Implications . . . 31

5 Conclusion 33
5.1 Overview . . . 33
5.2 Limitations . . . 33
5.3 Future Directions . . . 34

Bibliography 36

1 Introduction

Graph-structured data appear in numerous applications, such as citation networks [1, 2], knowledge graphs [3, 4], and online information systems [5, 6]. In recent years, graph neural networks (GNNs) [7, 8, 9] have become essential for learning from this type of data because they can share and combine information across graph edges. Classic GNN models, such as Graph Convolutional Networks [7], Graph Attention Networks [9], and GraphSAGE [8], perform reliably on standard node classification tasks and are considered strong benchmarks in graph representation learning.

Large Language Models (LLMs), such as LLaMa [10], GPT-4 [11], and Falcon [12], have achieved remarkable success in multiple tasks that require natural language understanding, such as open-domain question answering (OD-QA) [13], text-aware recommendation systems [14] and solving quantitative reasoning problems [15]. Motivated by the rich semantic knowledge encoded in LLMs, multiple researchers [16, 17, 18, 19] have explored the incorporation of LLMs into graph learning frameworks, particularly for text-attributed graphs (TAGs). TAGs are graphs in which each node is associated with free-form natural language text, and edges represent structural relations between nodes. The need to jointly model rich textual semantics and graph topology makes TAGs a natural setting for integrating LLMs into graph learning.

LLM-based representations can provide more detailed semantic information than traditional sparse text features, which may help with tasks such as node classification. Hence, several frameworks have been developed. One such framework, ENGINE [17], adopts a parameter-efficient approach by adding a lightweight GNN module alongside a frozen LLM, so structural information is included without changing the language model itself. LLaGA [16], another one of these frameworks, on the other hand, turns graph reasoning into a generative task by converting graph structures into word sequences and using prompts to make predictions. In the original ENGINE and LLaGA experiments, both methods performed well and outperformed classical GNNs, demonstrating the strong potential of combining graph reasoning with large language models.

Despite these achievements, the practical advantages of LLM-based graph models over strong GNN baselines remain insufficiently characterized under realistic evaluation constraints. In particular, existing studies often assess LLM-based methods under heterogeneous settings, involving different feature assumptions and computational budgets, which makes direct comparison difficult. Moreover, many LLM-based graph frameworks rely on generative inference or prompt-based decoding. So the effectiveness can be sensitive to decoding strategies and evaluation protocols. In closed-set node classification, where predictions must match a fixed set of labels and reproducibility is essential, it remains unclear whether such generative reasoning consistently gains higher classification accuracy. Finally, the computational cost and engineering complexity of LLM-based models raise questions about their robustness and efficiency when deployed under constrained environments, such as limited decoding options or restricted computing resources.

As a result, it is still an open question whether LLM-based graph models can reliably and consistently outperform well-established GNN benchmarks when evaluated under controlled and reproducible experimental conditions. This work aims to address these gaps through a systematic and controlled empirical comparison of classical GNNs, including GCN [7], GAT [9], and GraphSAGE [8], LLM-based graph reasoning frameworks, including ENGINE [17] and LLaGA [16], and a lightweight hybrid model. The datasets employed are Cora [20] and WikiCS [21]. All models are evaluated under identical data splits, consistent preprocessing, and a unified multi-seed evaluation protocol to ensure fair comparison.

Our experimental results reveal several important observations. Under the standard training setting using 60% of the full dataset, classical GNNs already provide strong performance, whereas ENGINE achieves the best accuracy across both datasets. LLaGA shows competitive performance on WikiCS but lags on Cora, indicating that pure generative LLM-based reasoning may be sensitive to the characteristics of the dataset. Notably, our hybrid model achieves a performance comparable to strong GNN baselines and remains competitive with LLM-based methods while using a significantly lighter architecture.

We further investigate data efficiency by reducing the training set to 10%, 20%, and 30% of the full dataset. The performance of classical GNNs decreases gradually as the training data shrink, reflecting their stable structural inductive bias. ENGINE remains robust even under limited supervision, whereas LLaGA exhibits substantial performance degradation, particularly in low-data regimes. Our hybrid model shows stable accuracy under reduced training data, outperforming LLaGA in low-data settings and remaining competitive with GNN baselines.

Beyond benchmarking existing approaches, our proposed lightweight hybrid architecture merges structure-aware node representations with semantic label tokens extracted from language models and connects them via a prompt-guided cross-attention mechanism. Moreover, unlike LLM-based hybrid models such as ENGINE and LLaGA, which depend on large generative or deep transformer backbones, we use a frozen sentence-level language encoder to obtain semantic embeddings. We introduce an alignment mechanism that uses discriminative attention to bridge node structural labels and label semantics. This design preserves the complementary strengths of GNNs in efficient structural modelling and LLMs in providing rich semantic prior knowledge, while significantly reducing the computational overhead.

Our main contributions are as follows:

- We provide a controlled comparative study of classical GNNs, recent LLM-based graph frameworks, and our lightweight hybrid model on node classification, under identical experimental settings.
- We propose a lightweight language-guided hybrid model that integrates structure-aware node encoders with prompt-based semantic label tokens via cross-attention, achieving competitive performance with significantly lower computational cost.
- We analyze the performance of the same model structure across multiple training ratios, revealing the robustness differences between GNNs, LLM-based models, and our hybrid approach.

2

Related Work

2.1. Graph Neural Networks (GNNs)

Graphs provide a natural representation for relational data, where nodes correspond to entities and edges encode relationships between them. Unlike independent and identically distributed (i.i.d.) data, graph-structured data exhibits complex dependency patterns, as the properties of a node are often influenced by its local neighborhood and the broader graph structure. A fundamental learning problem on graphs is node classification, which aims to predict a categorical label for each node by jointly using node attributes and graph connectivity.

Node classification on graphs is challenging for several reasons. First, node features alone are often insufficient, as semantically related nodes may not share similar attributes but are connected through the graph structure. Second, graph dependencies are inherently non-local, meaning that information associated with node labels may propagate through multiple hops within the graph. Traditional embedding methods based on random walks, such as DeepWalk [22] and Node2Vec [23], capture certain structural patterns, but they decouple representation learning from downstream tasks and cannot fully exploit label supervision.

GNNs address the above-mentioned challenges by directly operating on graph-structured data through neighborhood-based message passing. Some studies [24] indicate that GNNs can generate more effective node representations for downstream graph learning tasks, particularly for supervised node classification tasks.

In a typical GNN, each node iteratively aggregates information from its neighbors and updates its representation via learnable transformations. This process enables nodes to encode both local structural context and higher-order dependencies as the number of layers increases. The learned node embeddings serve as task-specific representations that can be optimized end-to-end for supervised objectives, making GNNs particularly effective for node classification tasks.

Building on message passing, GCN [7], GAT [9], and GraphSAGE [8] differ mainly in their neighborhood aggregation strategies, ranging from fixed normalization to attention and sampling-based aggregation. They form standard baselines for supervised node classification in this work.

2.1.1. Graph Convolutional Networks (GCNs)

Graph Convolutional Networks (GCNs) [7] are derived from spectral graph theory. GCNs define graph convolutions as localized spectral filtering operations on graph signals, where node features are smoothed according to the graph structure. Instead of learning arbitrary spectral filters, Kipf and Welling propose a simplified first-order approximation, which enables efficient message propagation without explicit eigen-decomposition of the graph Laplacian.

Due to the linear aggregation and uniform weighting mechanism, GCNs are sensitive to noisy

neighbors and may suffer from over-smoothing when multiple graph convolutional layers are stacked. Consequently, employing appropriate normalization and shallow architectures proves essential in practice.

The message propagation at layer l is formulated as follows:

$$H^{(l+1)} = \sigma\big(\tilde{D}^{-1/2}\tilde{A}\,\tilde{D}^{-1/2}H^{(l)}W^{(l)}\big), \tag{2.1}$$

where $\tilde{A} = A + I$ denotes the addition of self-loops, $\tilde{D}$ is the degree matrix, and $W^{(l)}$ stores the learnable weights. The normalization

$$\tilde{D}^{-1/2}\tilde{A}\,\tilde{D}^{-1/2}$$

prevents numerical problems when the operation is applied multiple times. Without this normalization step, feature magnitudes may either explode or vanish as more layers are stacked, leading to unstable representations.

This normalization is therefore central to the success of GCNs, as it ensures numerically stable message propagation across layers. By applying symmetric normalization to the adjacency matrix, each node aggregates information from its neighbors with weights inversely proportional to their degrees, such that high-degree nodes contribute less to each individual neighbor, which aligns with intuitive expectations. When a node is connected to many other nodes, no single connection should dominate the information aggregation, as each neighbor is likely to carry only a limited amount of unique signals. Therefore, degree-based normalization balances the influence of nodes with varying connectivity, preventing high-degree nodes from dominating the representation learning process.

However, GCNs have a trade-off in that they treat all neighbors equally, except for degree-based normalization. This uniform weighting assumes that all neighbors contribute equally to the representation of the target node, even though real graphs often contain a mixture of informative and noisy neighbors. Because the aggregation operation is linear and occurs before nonlinear activation, the model lacks the capacity to distinguish among neighbors based on their semantic relevance. As a result, highly informative neighbors receive the same weight as noisy neighbors. When representations from all neighbors are averaged, strong and discriminative signals are diluted by numerous weaker neighbors. This causes node representations to drift toward the average semantics of their neighborhoods.

In short, in graphs with heterogeneous neighborhood structures, this inability to emphasize reliable neighbors while suppressing noisy neighbors becomes a major limitation of uniform aggregation.

2.1.2. Graph Attention Networks (GATs)

Graph Attention Networks (GATs) [9] are introduced to address one of the key limitations of GCNs. The main idea of GATs is that not all neighbors should influence a node equally, meaning that some connections carry more relevant information than others. Instead of using fixed normalization weights, GATs learn attention coefficients that highlight the most informative neighbors. For each edge (i, j), the model calculates a raw attention score e_{ij} as follows:

$$e_{ij} = \mathrm{LeakyReLU}\big(a^{\top}[Wh_i \,\|\, Wh_j]\big), \tag{2.2}$$

where $\|$ denotes feature concatenation, W transforms node features, and a is a trainable attention vector. The model then normalizes these scores with a softmax over the local neighborhood to obtain the attention weights α_{ij}. This makes each neighbor's importance depend on the context and not just on degree statistics. The final aggregation is as follows:

$$h_i' = \sigma\left(\sum_{j\in\mathcal{N}(i)} \alpha_{ij} W h_j\right), \tag{2.3}$$

This approach allows the model to focus more on useful edges and ignore those that add noise. Through this process, GATs can capture heterogeneous neighborhood structures far more effectively than simple averaging in GCNs.

Multi-head attention (MHA) further strengthens this capability because each attention head operates independently and can focus on a different relational signal. For instance, one head may find semantic similarities, whereas another may identify structural differences. However, this expressiveness comes at the cost of additional computation, since the model must calculate the attention scores for every edge. In addition, having more parameters can make the model more likely to overfit small datasets.

2.1.3. GraphSAGE

GraphSAGE [8] approaches the neighborhood aggregation problem from a different perspective. Instead of increasing expressiveness through attention, it focuses on inductive generalization and large-scale training. It learns aggregation functions over sampled neighborhoods. Therefore, GraphSAGE can work with graphs that are too large for full-batch methods. The propagation rule consists of two steps. First, for each node, it samples a fixed-size set of neighboring nodes and aggregates the features of the sampled neighbors using an aggregator, such as mean or pooling:

$$h_{\mathcal{N}(v)}^{(k)} = \mathrm{AGGREGATE}_k\left(\{\, h_u^{(k-1)} : u \in \mathcal{N}(v) \,\}\right). \tag{2.4}$$

Second, this neighborhood summary is combined with the node's previous representation by concatenating them and applying a learned transformation:

$$h_v^{(k)} = \sigma\left(W^{(k)} \cdot [\, h_v^{(k-1)} \,\|\, h_{\mathcal{N}(v)}^{(k)} \,]\right). \tag{2.5}$$

This concatenation is important because it creates a skip connection between a node's previous representation and the aggregated neighborhood information. This design improves the gradient flow and helps the model retain information across layers.

Among the different aggregation strategies, the pooling aggregator achieves the best performance. It applies a learnable nonlinear transformation to each neighbor and aggregates the transformed features using an element-wise max operation. This step enables the model to capture the most distinctive features among neighboring nodes. In contrast, the mean aggregator behaves similarly to a simplified GCN without degree normalization. Although this aggregation is computationally efficient, it has limited expressive power compared with the pooling-based approach.

Despite these differences, GraphSAGE benefits from its sampling-based training procedure and flexible aggregator design, which together provide strong scalability and enable natural generalization to unseen nodes. This is an advantage over the transductive graph convolutional models.

Although sampling introduces variance and may discard useful structural information when the sample size is small, the efficiency and inductive generalization capabilities usually make this trade-off worthwhile for large-scale or dynamically evolving graph structures.

2.1.4. Summary

Despite the robust performance and widespread adoption of classical GNN architectures, they still exhibit limitations in node classification tasks on textual graphs. First, GNNs primarily rely on numerical input features and local neighborhood aggregation. This characteristic makes them highly dependent on the quality of initial node features. In the early stages, GNNs are combined with static embedding representations [25, 26] which provide only shallow semantic information. However, in rich textual graphs, such embedding methods struggle to capture semantic information into their initial node embeddings, limiting their ability to capture subtle relationships between nodes. Second, although message passing effectively captures local structural dependencies, in practice, GNNs used for node classification are commonly limited to only two layers due to over-smoothing and noise amplification. As a result, such shallow GNN architectures have limited capacity to model cross-neighborhood semantic relations, especially when these relations are expressed through rich textual attributes.

Finally, while GNNs can be effective in semi-supervised settings, their performance may degrade when the amount of labelled data becomes limited.

2.2. LLM-Based Models

To overcome the reliance of classical GNNs on shallow textual features and local structural signals, recent studies have explored integrating LLMs into graph representation learning. Unlike traditional embedding methods, LLMs are pretrained on massive text corpora and can encode rich semantic knowledge and contextualized linguistic representations. This capability is especially valuable for TAGs, where node semantics play a central role and cannot be fully captured through message passing alone [27].

An early and widely adopted idea is to treat LLMs as *semantic enhancers* for GNNs. In this setting, LLMs are used to generate enriched node representations from raw text, which are then consumed by downstream GNNs for structural aggregation and prediction. Typical examples include approaches where LLMs or pretrained text encoders provide contextualized embeddings that replace or augment traditional Bag-of-Words or TF-IDF [28] features [29, 30, 31]. These methods retain the discriminative training paradigm of GNNs while significantly improving the semantic expressiveness of node features.

Beyond static feature enhancement, recent work explores deeper integration between GNNs and language models through joint or collaborative optimization. A first category focuses on *symmetric alignment*, where GNN and LLM representations are jointly learned in a shared latent space. Representative approaches employ contrastive objectives, such as ConGraT [32], to explicitly align structural and textual embeddings on TAGs. Another category adopts *asymmetric integration*, in which one model provides auxiliary supervision to the other. Examples include graph-nested transformer architectures such as GraphFormers [33], graph-aware distillation methods like GRAD [34], and iterative update frameworks such as THLM [35], where GNNs and LLMs exchange information across training stages. Related collaborative paradigms, including GLEM [36], also iteratively refine structural and semantic representations. While these methods enable rich interaction between graph structure and textual semantics, they typically require repeated LLM forward passes or fine-tuning. This substantially increases memory consumption and training cost, limiting scalability in large-scale graphs or resource-constrained settings.

Another emerging research direction treats LLMs not only as feature extractors but as the *primary reasoning engine (predictors)* for graph tasks. Instead of relying on GNNs as the primary inference backbone, these approaches convert graph structures into textual or token sequences that can be directly processed by LLMs [37, 38]. By encoding nodes, neighborhoods, and structural relations into sequential representations, LLMs are prompted to perform graph reasoning through language generation. While this paradigm enables flexible reasoning and leverages the strong generalization abilities of LLMs, representing complex graph topologies in natural language or token sequences remains challenging and often leads to unstable or inefficient inference [39, 40].

In this work, we focus on two representative LLM-based graph learning frameworks that exemplify distinct integration paradigms. ENGINE [17] follows the LLM as an *enhancer* paradigm, where a frozen language model is used as a semantic encoder and lightweight graph-aware modules are injected alongside the LLM to incorporate structural information in a parameter-efficient and discriminative manner. In contrast, LLaGA [16] belongs to the LLM as a *predictor* paradigm, formulating node classification as a prompt-based generative reasoning task by translating graph neighborhoods into continuous token embeddings that are directly processed by a large language model. Both models represent state-of-the-art approaches within their respective paradigms and together capture the key design trade-offs between discriminative semantic enhancement and generative graph reasoning in LLM-based graph learning.

2.2.1. ENGINE

Pointing out a key trade-off in applying large language models to textual graphs, Zhu et al. [17] observe that jointly optimizing language models and graph neural networks can substantially improve performance, but often incurs prohibitive computational cost. Related studies [41, 42] further note that cascaded or tightly coupled architectures, which sequentially encode textual and structural features,

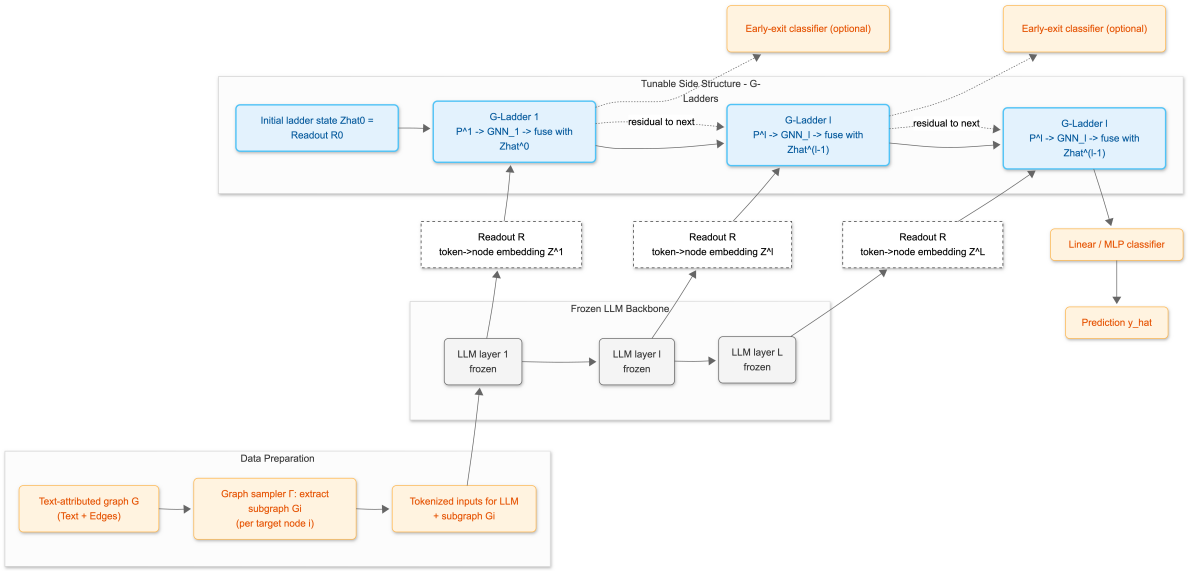


Figure 2.1: Illustration of the ENGINE architecture. A frozen language model backbone encodes textual subgraphs, whereas a parallel trainable GNN-based side structure (*G-Ladder*) injects graph structural information at multiple transformer layers. The readout functions map token representations to node embeddings, which are then used for downstream node classification.

may be suboptimal because of their reliance on repeated cross-module interactions. In particular, straightforward joint optimization requires backpropagation through the full language model at every training step, leading to heavy memory consumption and high training latency when scaling to large models.

ENGINE addresses these challenges by adopting a parameter-efficient integration strategy. Rather than fine-tuning the language model or inserting trainable adapters into its transformer layers, ENGINE keeps the LLM entirely frozen and attaches lightweight graph-aware side modules, termed *G-Ladders*, alongside each transformer layer. During the forward pass, each *G-Ladder* operates in parallel with the corresponding LLM layer. It first applies a readout operation (e.g., mean pooling) to the token-level hidden states produced by the frozen LLM, yielding node-level representations. These representations are then projected to a lower-dimensional space and refined through GNN-based message passing over the sampled subgraph, allowing structural information to be injected in a layer-wise manner. Importantly, *G-Ladders* are the only trainable components in ENGINE. All gradient updates are confined to the *G-Ladder* parameters and do not propagate through the LLM backbone, ensuring that the transformer weights remain fixed throughout training. This design enables efficient joint modeling of textual semantics and graph structure while significantly reducing computational and memory overhead. The architecture of the ENGINE model is illustrated in Figure 2.1.

This method differs from adapter-based fine-tuning methods, such as LoRA [43], which introduce a small number of trainable parameters but still require backpropagation through the entire transformer network to compute the adapter gradients. By decoupling structural learning from language model optimization and avoiding backpropagation through the language model, ENGINE substantially reduces memory usage and training cost. Simultaneously, it preserves the benefits of pretrained semantic representations for closed-set node classification.

For clarity, we follow the original formulation and omit lower-level implementation details, as they do not affect the optimization behavior or inference mechanism of the model.

Formally, the model considers a textual graph:

$$G = (V, \{t_n\}, A, Y),$$

where each node $n \in V$ is associated with a text attribute t_n. To improve the scalability of the model, ENGINE applies a sampling function to extract a local subgraph around each target node. Within each

subgraph, the model processes the text of these nodes in batches and feeds them into a frozen LLM.

At the l-th LLM layer, token-level representations:

$$\mathbf{H}^l \in \mathbb{R}^{B\times Q\times D}$$

are obtained, where B denotes the batch size, Q the sequence length, and D the hidden dimensions. Each LLM layer performs the following standard transformer operations:

$$\mathbf{H}^l = \mathrm{LLM\_Layer}^l(\mathbf{H}^{l-1}), \tag{2.6}$$

where all parameters involved are frozen. Since node-level tasks require node representations rather than token representations, a readout function R (typically mean pooling) aggregates the token embeddings for each node:

$$\boldsymbol{z}_i^l = R(\mathbf{h}_{i,1}^l, \mathbf{h}_{i,2}^l, \ldots, \mathbf{h}_{i,Q}^l), \tag{2.7}$$

At this point, the core component of ENGINE is used. The model passes the node-level representations $\boldsymbol{z}^l$ into the *G-Ladder*, which runs in parallel with the LLM layer to further improve the node embedding quality. Each *G-Ladder* first applies a low-dimensional projection ($K \ll D$) to reduce the computational cost, and then performs message passing over the graph:

$$\tilde{\mathbf{Z}}^l = \lambda^l \cdot \mathrm{GNN}^l(\mathbf{P}^l(\mathbf{Z}^l), A) + (1-\lambda^l)\cdot \tilde{\mathbf{Z}}^{l-1}, \tag{2.8}$$

where $\mathbf{P}^l$ projects the D-dimensional representations into a K-dimensional space, producing a compact form that preserves the essential information for subsequent graph operations. The GNN^l module performs message passing over the adjacency matrix A. Zhu et al. [17] evaluate several GNN variants at this stage, including GCN [7], GraphSAGE [8], and GAT [9], and find that GraphSAGE yields slightly better performance. However, the choice of GNN has a limited overall impact. The coefficient λ^l is not fixed but a learnable weight that balances new structural information with the output from the previous layer:

$$\lambda^l = \sigma(\omega^l / T), \tag{2.9}$$

where ω^l is initialized to zero and $T = 0.1$ is a temperature parameter. Initializing $\omega^l = 0$ makes the model depend more on the information from the earlier layers at the start of training, and over time, it learns to incorporate more graph structures. This setup has two main advantages. First, it facilitates gradient propagation, which reduces the vanishing gradient problem in deep learning networks. Second, it allows each layer to adaptively determine the amount of new structural information to use, while maintaining the representations computed in the previous layers. [17]

In the final stage, ENGINE feeds the node representations $\tilde{\mathbf{Z}}^L$ produced by the final *G-Ladder* layer into a simple linear classifier for node classification:

$$\mathcal{L} = \mathbb{E}_{i\in V_{\mathrm{tr}}}\left[\mathrm{CE}(\hat{y}_i, y_i)\right], \quad \text{where } \hat{y}_i = C(\tilde{\boldsymbol{z}}_i^L). \tag{2.10}$$

ENGINE models text and graph data together by introducing a lightweight graph-aware structure *G-Ladder* that operates alongside a frozen LLM. Instead of altering the LLM or adding adapters, ENGINE uses *G-Ladders* to add structural information by message passing on node-level representations at each layer. All LLM parameters stay fixed throughout training, while only the *G-Ladders* are optimized. This enables the efficient integration of textual semantics and graph topology with low computational and memory costs.

2.2.2. LLaGA

LLaGA formulates node classification as a prompt-based reasoning problem. The overall architecture is demonstrated in Figure 2.2. Within LLaGA, the graph structure and node attributes are translated into a structured token sequence and embedded into a natural language prompt, which is then processed by a frozen large language model to generate predictions.

Rather than learning task-specific classifiers, LLaGA relies on the pretrained reasoning capability of the language model to interpret graph-structured inputs expressed in linguistic form. Therefore, the model does not explicitly optimize the classification accuracy at the representation level. Instead, it learns a mapping that enables the LLM to reason over graph neighborhoods via text generation.

The LLaGA framework consists of three main components: (i) structure-aware graph translation via node-level templates, (ii) a versatile projector that aligns the embedding spaces, and (iii) a frozen LLM for reasoning and generation [16]. Notably, only the projector parameters require training. The templates do not have parameters, and the LLM remains frozen throughout the training. LLaGA focuses on the graph translation and projection components, as they define how graph structure is exposed to the language model during prompt-based reasoning.

Structure-Aware Graph Translation. The node serves as the fundamental unit of graph analysis, and LLaGA explicitly builds on this observation. Accordingly, LLaGA introduces two node-level templates that encode the structural information surrounding each central node while preserving the node features. In our experiments, we adopt the Neighborhood Detail Template (ND Template) because it explicitly encodes the local neighborhood structure around each node while maintaining its original feature information, which aligns well with our node-level classification setting.

Given node v, the ND Template constructs a fixed-shape sampled computational tree centered on v. For every hop of the neighbors, it uses a set sample size, with n_i as the sample size for the i-th hop. This process is recursive. Specifically, we take v as the root node and randomly sample n_1 neighbors from its 1-hop neighbor set $\mathcal{N}_v^1$ to form a new set $\widetilde{\mathcal{N}}_v^1$. If $|\mathcal{N}_v^1| < n_1$, we append placeholder nodes `[pad]` until the set reaches size n_1. The nodes in $\widetilde{\mathcal{N}}_v^1$ serve as the children of v. This procedure is repeated for each node at the next hop, sampling n_2 neighbors recursively. Whenever a set lacks enough neighbors, we fill the rest with placeholders. Once a placeholder node appears, all its children are placeholders as well.

Next, we apply a level-order traversal on the computational tree to generate a fixed-length node sequence that captures the central node and its entire neighborhood. Each position in the sequence uniquely corresponds to a relative structural location within the original graph, thereby preserving the structure in the specific order.

For TAGs, the original experiment [16] uses SBERT [44] to encode text features. Placeholder nodes are represented by zero vectors of the same size.

To further enhance the structural expressiveness, LLaGA incorporates Laplacian positional embeddings derived from the adjacency matrix of the computational tree A_{tree}. The Laplacian embedding is obtained as the eigenvector of the normalized Laplacian:

$$\mathbf{L} = \mathbf{I} - \mathbf{D}^{-\frac{1}{2}} A_{\text{tree}} \mathbf{D}^{-\frac{1}{2}} = \mathbf{U}^{\top} \Lambda \mathbf{U}, \tag{2.11}$$

where $\mathbf{D}$ denotes the degree matrix. Because the tree structure is fixed for given sample sizes, we compute the Laplacian embedding and use it for all graphs. The final node embedding $\mathbf{h}_{v_i}$ at sequence position i is defined as

$$\mathbf{h}_{v_i} = \begin{cases} \mathbf{0} \,\|\, \mathbf{U}_i, & v_i = \texttt{[pad]}, \\ \omega(x_{v_i}) \,\|\, \mathbf{U}_i, & \text{otherwise}, \end{cases} \tag{2.12}$$

where $\|$ denotes the concatenation. This results in a node embedding sequence that jointly encodes node attributes and structural context.

Versatile Projector. To enable effective interaction between graph representations and LLMs, LLaGA aligns the node embedding space with the LLM token embedding space using a learnable projector. This approach avoids feeding raw graph features directly into the language model and instead employs a lightweight multilayer perceptron:

$$\mathbf{e}_i = f_\omega(\mathbf{h}_i), \tag{2.13}$$

where f_ω denotes the projector with parameters ω that maps each node embedding into the token embedding space.

By concatenating the projected node embeddings with the standard token embeddings in the prompt, LLaGA effectively treats the graph structure as a specialized sequence of tokens. This design allows frozen LLMs to reason on graph-structured inputs without modifying their internal parameters. During training, the model optimizes only the projector parameters while keeping the templates fixed and the LLM entirely frozen.

2.2.3. Use case of ENGINE and LLaGA

To make the practical usage of ENGINE and LLaGA explicit, we provide concrete input examples for them. Starting from a target node v_i, these examples clarify how node representations are constructed and how different models utilize language and graph information for node classification. They highlight not only the interface differences, but also what each method treats as the primary source of information for prediction.

ENGINE: text encoding + graph-side inference (no task prompt). ENGINE treats the LLM purely as a frozen semantic encoder. For a target node v_i, a sampler first extracts a local k-hop or random-walk subgraph $\mathcal{G}_{v_i}$ while preserving the original node indices. Importantly, the language model is not invoked during subgraph construction or training-time inference.

Instead, each node v_j in the graph is associated with a raw text description (e.g., `Title:\nAbstract:`), which is encoded offline in a separate caching stage. In this step, the frozen LLM processes each node text independently as a plain-text sequence and produces token-level hidden states, which are pooled into a fixed-dimensional node embedding. No task instructions, prompts, or class labels are involved in this encoding phase.

During training and inference, the model operates exclusively on these cached node embeddings. The subgraph loader batches PyG subgraphs centered at v_i and retrieves the corresponding node representations by indexing the cache using the preserved original node IDs. As a result, node texts are neither retokenized nor concatenated with neighboring texts at runtime. Instead, the GNN operates directly on precomputed semantic embeddings.

A simplified example of the offline LLM encoding step is shown below:

Independent node texts encoded once by the frozen LLM
`[Node` v_i`]` 'Title: Neural network models for ...
Abstract: ...'
`[Node` v_j`]` 'Title: Graph-based semi-supervised learning ...
Abstract: ...'
`[Node` v_k`]` 'Title: A survey on citation networks ...
Abstract: ...'

The resulting node embeddings are then refined by *G-Ladders* through message passing on $\mathcal{G}_{v_i}$, and a discriminative classifier produces logits over classes. In summary, ENGINE relies on the LLM for semantic representation learning, while structural reasoning and classification are performed entirely outside the language model.

LLaGA: prompt-based generation with explicit `<graph>` injection. In contrast, LLaGA formulates node classification for a target node v_i as an instruction-following generation problem. Starting from v_i, LLaGA constructs a fixed-length node-centered neighborhood representation, which is projected into a sequence of continuous graph tokens. These graph tokens are then injected into a natural-language prompt via a reserved placeholder `<graph>`.

Unlike ENGINE, the LLM input explicitly includes not only the graph-derived token sequence but also a task instruction and a closed-set label list. In our codebase, prompts are wrapped using the default LLaMA-2 chat template (`conv_llaga_llama_2`). At runtime, the prompt string is first generated by `conv = conv_templates[args.conv_mode]` followed by `conv.get_prompt()`, where the system prefix and the `<graph>` placeholder are preserved. Subsequently, `tokenizer_graph_token` replaces `<graph>` with a fixed-length sequence of projected graph tokens corresponding to the neighborhood of v_i.

A real node classification prompt used in our codebase is shown below:

```
Given a node-centered graph: <graph>, ...
the node feature of center node is {text}.
We need to classify the center node into {C} classes: ...
Please tell me which class the center node belongs to? Directly tell me
the class name.
```

Here, the label list depends on the dataset. For example, Cora uses 7 classes, while WikiCS uses 10 classes.

These examples illustrate a fundamental difference in how node representations are generated and used. ENGINE encodes each node v_i into a semantic embedding before graph learning and never exposes the LLM to the task definition or label space. In contrast, LLaGA embeds the neighborhood of v_i directly into the LLM input and assigns both reasoning and prediction to the generative language model.

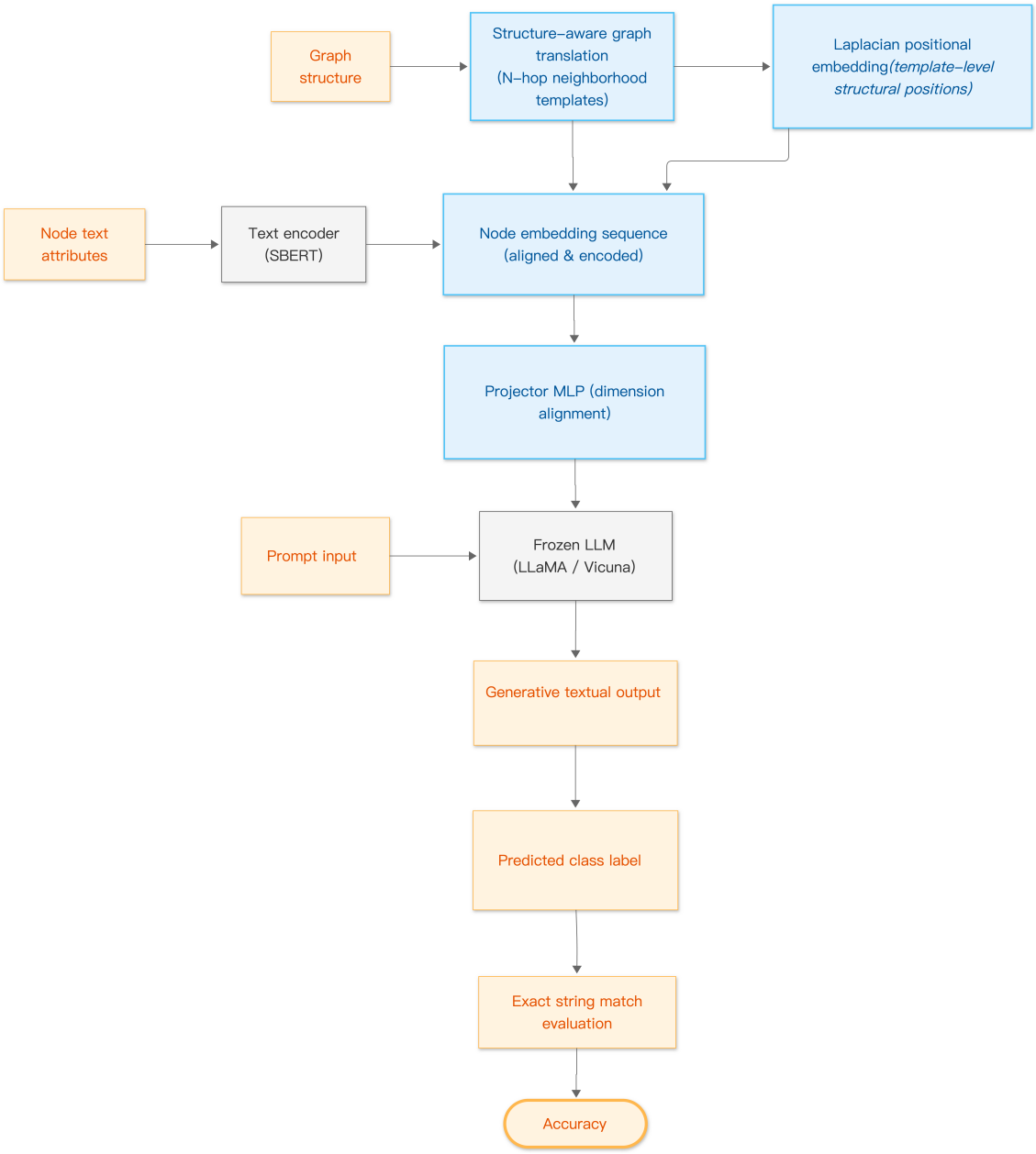


Figure 2.2: Illustration of the LLaGA architecture. Graph structures are translated into structure-aware node sequences and combined with the textual node attributes. A projector maps graph-aware embeddings into the token space of a frozen LLM that performs prompt-based generative reasoning to produce class labels.

3

Methodology

In this chapter, we first introduce our proposed lightweight hybrid model for transductive node classification, detailing its token construction, prompt-guided cross-attention, and the fusion and prediction mechanisms. We then describe the experimental implementation and adaptation details required to run both classical GNN baselines and LLM-based graph frameworks in a unified and reproducible setup. In particular, we summarize key engineering adaptations for ENGINE and LLaGA to ensure stable execution in our HPC environment, while preserving their original model designs and evaluation protocols.

3.1. A Lightweight Language-guided Hybrid Model

We propose a hybrid node classification framework that combines (i) graph-aware node representations, (ii) lightweight topological features, and (iii) natural-language label semantics encoded as label tokens. Given a graph $G = (\mathcal{V}, \mathcal{E})$ with $N = |\mathcal{V}|$ nodes and C classes, we build a *node token* $\mathbf{v}_i$ by fusing a structure-aware text representation and a positional representation. We then perform *prompt-guided cross-attention* where each node attends to the set of label tokens to obtain a label-conditioned context vector $\mathbf{u}_i$. Finally, a node-wise gating module produces a fused representation $\mathbf{f}_i$, and three lightweight classifier heads (structure branch $\mathbf{v}_i$, label-context branch $\mathbf{u}_i$, and fusion branch $\mathbf{f}_i$) are combined with learnable mixture weights to produce the final prediction. The overall architecture of our proposed hybrid model is shown in Figure 3.1.

3.1.1. Token Construction

Token design. Our framework uses three types of tokens: (i) *label tokens* encode class semantics from natural language prompts; (ii) *graph-aware text tokens* encode node textual features while injecting neighborhood information via a GCN; (iii) *position tokens* encode hand-crafted global/topological features via a small MLP. We then concatenate projected graph-aware text and position tokens to form the final node token.

Label tokens from natural-language prompts. For each class $c \in \{1, \dots, C\}$, we define a prompt prompt_c (e.g., "This node belongs to category: ·") and encode it using a frozen Sentence-BERT encoder [44]:

$$\mathbf{l}_c = \mathrm{SBERT}(\text{prompt}_c) \in \mathbb{R}^{d_{\text{label}}}. \tag{3.1}$$

Stacking all label embeddings yields $\mathbf{L} = [\mathbf{l}_1^\top; \dots; \mathbf{l}_C^\top] \in \mathbb{R}^{C \times d_{\text{label}}}$. Besides, we allow learnable label offsets to adapt label tokens to the dataset:

$$\tilde{\mathbf{L}} = \mathbf{L} + \Delta, \quad \Delta \in \mathbb{R}^{C \times d_{\text{label}}}, \tag{3.2}$$

where Δ is initialized with small Gaussian noise and is trained jointly with the rest of the model. In all cases, we ℓ_2-normalize label tokens before projection to encourage cosine-space alignment.

Graph-aware text token. Each node i has a precomputed text embedding $\mathbf{t}_i \in \mathbb{R}^{d_{\text{text}}}$ (Sentence-BERT [44]). To inject local structural context, we apply a two-layer GCN over the graph:

$$\mathbf{h}_i^{(1)} = \mathrm{ReLU}(\mathrm{GCNConv}_1(\mathbf{t}_i, \mathbf{E})), \tag{3.3}$$

$$\mathbf{u}_i^{\text{text}} = \mathrm{GCNConv}_2(\mathbf{h}_i^{(1)}, \mathbf{E}) \in \mathbb{R}^{d_{\text{gcn}}}. \tag{3.4}$$

where $\mathbf{E}$ denotes the edge index. Specifically, the first GCN layer injects one-hop structural context by aggregating the text embeddings of directly connected neighbors into the representation of node i according to the graph structure. As a result, the updated node representation captures not only the semantic content of the node itself but also information from its immediate neighborhood. The second GCN layer further propagates these representations, enabling the node to incorporate two-hop local neighborhood semantics, such that the resulting textual representation reflects the aggregated semantic context of the surrounding local subgraph.

Position token. We compute a vector of topological features $\mathbf{p}_i$, such as degree statistics and PageRank, and encode it using a lightweight two-layer MLP:

$$\mathbf{u}_i^{\text{pos}} = \mathrm{MLP}_{\text{pos}}(\mathbf{p}_i) \in \mathbb{R}^{d_{\text{pos}}}. \tag{3.5}$$

Node token via projection and concatenation. We project the graph-aware text token and the position token into a shared latent space of dimension d:

$$\mathbf{v}_i^{\text{text}} = \phi_t(\mathbf{u}_i^{\text{text}}) \in \mathbb{R}^{d/2}, \tag{3.6}$$

$$\mathbf{v}_i^{\text{pos}} = \phi_p(\mathbf{u}_i^{\text{pos}}) \in \mathbb{R}^{d/2}. \tag{3.7}$$

Here, the projections serve two purposes. First, they control the relative representational capacity of textual and positional information by allocating an equal-dimensional subspace to each modality. Second, mapping both components to the same dimension ensures compatibility with subsequent attention and gating operations, which are sensitive to feature magnitude and dimensional imbalance.

The final node token is then formed by concatenation:

$$\mathbf{v}_i = \mathbf{v}_i^{\text{text}} \,\|\, \mathbf{v}_i^{\text{pos}} \in \mathbb{R}^d. \tag{3.8}$$

3.1.2. Prompt-Guided Cross-Attention

We treat label tokens as a compact semantic memory of classes and let each node actively retrieve relevant label context. Here, prompt-guided refers to the use of natural-language-derived label representations as explicit semantic prompts that guide the node representation learning process. Instead of treating class labels as abstract indices, each label is encoded from a textual prompt and participates directly in the attention mechanism to provide semantic guidance during node-label alignment.

Compared to directly classifying $\mathbf{v}_i$ with an MLP, prompt-guided attention provides an explicit mechanism for aligning node representations with label semantics and offers an interpretable affinity distribution between nodes and labels.

Projection of label tokens to attention space. We map label tokens to the model space with a trainable projection:

$$\mathbf{q}_c = \psi(\tilde{\mathbf{l}}_c) \in \mathbb{R}^d, \tag{3.9}$$

$$\mathbf{Q} = \psi(\tilde{\mathbf{L}}) \in \mathbb{R}^{C \times d}. \tag{3.10}$$

where $\psi(\cdot)$ is a linear layer and $\tilde{\mathbf{l}}_c$ denotes the possibly adapted label token.

Node-as-query cross-attention. Our implementation uses *node tokens as queries* and *label tokens as keys/values*. Specifically, we employ a standard multi-head attention mechanism [45], which computes attention-weighted combinations of label token representations conditioned on each node token. Within each attention head, MHA computes similarity scores between node queries and label keys via scaled dot-product attention, subsequently applying softmax normalization and weighted summation to extract label-aware semantic content. By employing multiple attention heads with distinct projection matrices, MHA enables node-label matching across different representational subspaces, allowing the model to capture complementary alignment patterns. Thereby, this mechanism can prevent the model from relying solely on a single similarity metric. This mechanism is shown in Figure 3.2.

For each node i, we compute a label-conditioned representation using multi-head attention:

$$\mathbf{u}_i^{\text{mha}} = \text{MHA}(\mathbf{v}_i, \mathbf{Q}, \mathbf{Q}) \in \mathbb{R}^d, \tag{3.11}$$

where $\mathbf{v}_i$ serves as the query and $\mathbf{Q}$ denotes the projected label tokens used as both keys and values.

Intuitively, $\mathbf{u}_i^{\text{mha}}$ aggregates information from the label token set, allowing each node to attend to the most semantically relevant class descriptions under the attention mechanism.

Cosine-softmax label aggregation. To further encourage stable semantic alignment between node representations and label semantics, we additionally introduce an explicit cosine-based label aggregation mechanism. From a conceptual perspective, this component can be interpreted as a prototype matching module, where each label token serves as a semantic prototype representing a class, and each node is matched against this prototype set.

Concretely, the cosine similarity scores define a relative matching distribution between a node and all label prototypes. The resulting coefficients α_{ic} therefore quantify how strongly node i aligns with each class prototype in semantic space, rather than producing an absolute similarity score. Based on this distribution, we construct a label-aware representation by aggregating label prototypes weighted by their relative relevance to the node.

Using cosine similarity ensures that the matching process is governed by directional alignment, thereby preventing certain label tokens from being inherently favored due to larger embedding norms. This normalization is particularly important when label tokens are derived from natural language prompts and adapted via learnable offsets, as their magnitudes may vary during training.

Formally, we compute the cosine-based similarity and the corresponding softmax distribution as:

$$s_{ic} = \frac{\langle \mathbf{v}_i, \mathbf{q}_c \rangle}{\tau}, \tag{3.12}$$

$$\alpha_{ic} = \frac{\exp(s_{ic})}{\sum_{c'=1}^{C} \exp(s_{ic'})}, \tag{3.13}$$

where $\tau > 0$ is a temperature hyper-parameter controlling the sharpness of the matching distribution. The final cosine-based label context vector is then obtained as a weighted sum over label prototypes:

$$\mathbf{u}_i^{\text{cos}} = \sum_{c=1}^{C} \alpha_{ic}\, \mathbf{q}_c \in \mathbb{R}^d. \tag{3.14}$$

Finally, we combine both context vectors and apply layer normalization:

$$\mathbf{u}_i = \text{LN}\left(\mathbf{u}_i^{\text{mha}} + \mathbf{u}_i^{\text{cos}}\right) \in \mathbb{R}^d. \tag{3.15}$$

3.1.3. Node-wise Dynamic Gating and Prediction

Feature-wise gating. We adaptively fuse the original node token $\mathbf{v}_i$ and the label-aware context $\mathbf{u}_i$ using a *feature-wise* gate:

$$\mathbf{g}_i = \sigma\big(\mathrm{MLP}_{\mathsf{gate}}([\mathbf{v}_i \| \mathbf{u}_i])\big) \in (0,1)^d, \tag{3.16}$$

$$\mathbf{f}_i = \mathbf{g}_i \odot \mathbf{u}_i + (1 - \mathbf{g}_i) \odot \mathbf{v}_i, \tag{3.17}$$

where σ is the sigmoid function and $\odot$ denotes element-wise multiplication. This gate allows each dimension to decide how much to rely on label context versus the original node representation.

Three-branch classification and learnable mixing. We attach three lightweight classifier heads to the structure branch ($\mathbf{v}_i$), label-context branch ($\mathbf{u}_i$), and fusion branch ($\mathbf{f}_i$):

$$\mathbf{o}_i^v = \mathrm{Head}_v(\mathbf{v}_i),\ \mathbf{o}_i^u = \mathrm{Head}_u(\mathbf{u}_i),\ \mathbf{o}_i^f = \mathrm{Head}_f(\mathbf{f}_i), \tag{3.18}$$

where each head is a small MLP producing logits in $\mathbb{R}^C$. We then combine these logits with learnable mixture coefficients:

$$\pi = \mathrm{softmax}(\gamma) \in \mathbb{R}^3, \tag{3.19}$$

$$\mathbf{o}_i = \pi_v \mathbf{o}_i^v + \pi_u \mathbf{o}_i^u + \pi_f \mathbf{o}_i^f, \tag{3.20}$$

where $\gamma \in \mathbb{R}^3$ is a trainable parameter vector.

Single-label prediction and training objective. We perform single-label node classification by taking $\hat{y}_i = \arg\max_c \mathbf{o}_{ic}$, and train the model using cross-entropy loss on the training nodes:

$$\mathcal{L}_{\mathsf{main}} = \frac{1}{|\mathcal{V}_{\mathsf{train}}|} \sum_{i \in \mathcal{V}_{\mathsf{train}}} \mathrm{CE}(\mathbf{o}_i, y_i). \tag{3.21}$$

In our experiments, we optimize the fused prediction loss as the primary training objective.

Our hybrid method is characterized by: (i) a graph-aware text encoder (GCN over SBERT node embeddings) combined with a lightweight positional encoder, (ii) prompt-derived label tokens that enable node-to-label semantic retrieval via cross-attention and cosine-softmax aggregation, and (iii) feature-wise gating and a three-branch classification scheme for robust single-label prediction.

3.2. Implementation and Adaptation Details

Before presenting the individual model implementations, we first want to outline how different classes of methods are handled in our experimental setup. Specifically, we distinguish between classical GNN models and LLM-based graph frameworks, as these two categories differ substantially in their training pipelines and practical deployment requirements.

For classical GNNs, we focus on standard message-passing implementations that operate directly on node features and graph structure. For LLM-based methods, we describe how pretrained language models are integrated into graph learning, including necessary adaptations for efficient training and inference in our computational environment. By structuring this section accordingly, we aim to make clear which implementation choices are shared across models and which are specific to particular integration paradigms, thereby ensuring transparency and reproducibility in the subsequent experiments.

3.2.1. Learning Setting

All experiments in this work are conducted under a *transductive node classification* setting. For each dataset, the full graph structure, including all nodes and edges, is available during training and inference.

Nodes are partitioned into training, validation, and test sets via fixed node-level masks, where labels are only provided for training nodes.

To study robustness under limited supervision, we further vary the number of labeled training nodes by subsampling from the original training split. Importantly, validation and test nodes remain present in the graph throughout training, but their labels are never exposed. As a result, performance differences reflect sensitivity to label scarcity rather than inductive generalization to unseen nodes.

This setup follows the standard evaluation protocol used in prior work on transductive node classification benchmarks such as Cora and WikiCS.

3.2.2. GNN Models

Shared Experimental Protocol We run all GNN-based models in a unified training setup that eliminates confounding factors while preserving each method's architectural characteristics. All baselines call the same `load_data()` function and receive a homogeneous PyG `Data` object containing `(x, edge_index, y, train_mask, val_mask, test_mask)`. The predefined masks determine which nodes are used for training, validation, and testing, ensuring identical supervision across models.

To study robustness under limited supervision, we construct reduced training sets by subsampling only the original training split. For example, a train ratio of 10% indicates that approximately 10% of all nodes are labeled for training, obtained by uniformly sampling from the original training pool. Importantly, the validation and test splits are kept fixed across all settings, and only the training mask is modified. This design ensures that performance differences across train ratios reflect sensitivity to label scarcity, rather than changes in validation or test data.

All models are trained using the cross-entropy objective with an Adam optimizer. We apply early stopping based on validation loss, with patience values following original recommendations for each GNN architecture. The checkpoint achieving the best validation performance is selected for final testing.

During inference, evaluation is always performed on the full graph. The complete data object is transferred to the device, a single forward pass is executed, and classification accuracy is computed over nodes specified by the test mask.

GCN Model The `GCN` [7] model follows the standard two-layer architecture introduced by Kipf and Welling. The forward computation consists of input feature dropout, graph convolution with symmetric normalization, ReLU nonlinearity, dropout on hidden representations, and a second graph convolution that outputs unnormalized class logits.

Both convolutional layers use `GCNConv` with `cached=True` and `normalize=True`. By caching the normalized adjacency matrix $\tilde{D}^{-1/2}\tilde{A}\tilde{D}^{-1/2}$, the model avoids recomputing it at each forward pass. This setup aligns with the original approach that assumes a fixed graph structure during training.

Furthermore, we follow the regularization strategy described in the original paper. We apply dropout and weight decay to mitigate overfitting and stabilize optimization. As in the original design, regularization is primarily applied to the transformation of raw input features in the first layer, while the second layer focuses on class discrimination.

Then we perform training using full-batch optimization. At each epoch, the model processes the entire graph in a single forward pass, computes the loss over the nodes specified by the `train_mask` and updates the parameters accordingly.

GAT Model The `GAT` [9] implementation follows the standard two-layer transductive architecture described by Veličković et al. The first layer applies multi-head graph attention with eight heads and concatenation, followed by an ELU activation. The second layer uses a single attention head without concatenation to produce class logits.

Both input features and intermediate representations employ a dropout mechanism with a probability of 0.6. In addition, the `GATConv` layers internally apply dropout to attention coefficients, randomly

masking normalized attention weights during training. This mechanism regularizes neighborhood influence and prevents over-reliance on individual edges.

Like GCN, training is conducted in a full-batch manner over the complete graph to preserve the training approach.

GraphSAGE Model GraphSAGE [8] differs fundamentally from GCN and GAT in its design. Rather than assuming access to a fixed graph during training, GraphSAGE is designed for inductive learning and scalability through neighborhood sampling.

The GraphSAGE model implements the mean-aggregator variant using PyG's `SAGEConv`. The architecture consists of two layers with ReLU nonlinearity and dropout applied to hidden representations. `SAGEConv` provides an optional internal normalization, but the original GraphSAGE algorithm calls for explicit ℓ_2 normalization after each layer.

Our training follows the inherent sample-and-aggregate approach of GraphSAGE. We perform the mini-batch optimization by using neighborhood sampling with two hops and fixed fan-outs $S_1 = 25$ and $S_2 = 10$. In each batch, we sample seed nodes from the `train_mask`, build neighborhoods using these sizes, and compute loss only on the seed nodes.

3.2.3. ENGINE

To ensure a fair and reproducible comparison, we reimplement the ENGINE pipeline in the TU Delft high-performance computing (HPC) environment while strictly preserving its original model architecture and training logic. Specifically, ENGINE is conducted on GPU compute nodes equipped with NVIDIA A40 GPUs (46 GB memory), running on an x86_64 architecture with up to 64 CPU cores and 500 GB system memory per node. The software environment consists of Python 3.10 with PyTorch 2.0.1 (CUDA 11.7), executed within a Conda-managed environment on the SLURM workload manager.

Since the official ENGINE codebase is designed for a specific software stack and hardware configuration, directly deploying it in this HPC setting required a series of engineering adaptations. These adaptations include adjustments to model loading routines, offline embedding caching, and configuration management to ensure compatibility with the cluster environment and resource constraints. Crucially, all modifications are restricted to infrastructure-level components and do not alter ENGINE's architectural design, training objective, or graph–language interaction mechanism. As a result, the reimplemented ENGINE faithfully reflects the original method while enabling stable and reproducible execution under our experimental setting.

Extension of the LLM Encoder and Caching Pipeline As we do not run the ENGINE in its original environment but instead adapt it to the HPC cluster using PyTorch version 2.0.1, we accordingly update the transformer/accelerate library and do not enable DeepSpeed acceleration support.

Concretely, we modify `cache.py` and `llm.py` at the infrastructure level to ensure stable execution on the TU Delft HPC cluster, without altering ENGINE's model architecture or learning objective. Specifically, we:

- explicitly set tokenizer padding tokens and propagate the corresponding `pad_token_id` to the language model configuration, ensuring consistent sequence pooling behavior across different LLM backbones;
- replace the original automatic model sharding and quantized loading (`device_map="auto"` with 8-bit weights) with explicit single-GPU placement (via `.to(device)`) in our HPC setup, and run the LLM forward pass in FP16 to ensure deterministic memory usage and stable execution.
- preserve the original hidden-state extraction logic by caching mean-pooled representations from all transformer layers, exactly as in the original ENGINE implementation, but adapt the caching pipeline to operate reliably under the HPC execution environment.

All modifications are restricted to model loading, precision control, and embedding caching. The graph-aware *G-Ladder* modules, training objective, and graph-language interaction mechanism remain unchanged.

Unified Configuration-Driven Training Pipeline Given that ENGINE uses a YAML-based configuration mechanism, we follow the original LLaMA setup and rely exclusively on configuration files to keep experiments consistent and reproducible. All model-specific parameters are specified in the YAML configuration files. These are automatically merged into runtime arguments, creating a configuration-driven training and evaluation pipeline.

We did not introduce any architecture-specific modification beyond what is already in the original ENGINE implementation. Specifically, the input dimensionality, transformer layer selection, and model loading routines strictly follow the default LLaMA configuration used in ENGINE [17]. This approach keeps the intended transformer-layer abstraction and avoids model-specific biases or extra memory usage.

Importantly, we confine all implementation choices to engineering-level components, including configuration management, data preprocessing, caching procedures, and library compatibility. These components affect only the embedding extraction and caching stages and do not alter the core architecture of ENGINE. The downstream graph inference module and the prompt-based aggregation remain identical to those of the original ENGINE model.

3.2.4. LLaGA

Like ENGINE, LLaGA is also executed on the same GPU nodes under Slurm. To ensure compatibility with our experimental environment, we modify the original LLaGA project while preserving its core architectural and optimization design.

Specifically, our adaptations are limited to engineering-level components and execution settings, and include:

- enabling stable training and inference in our HPC environment under a single-GPU setting;
- preserving the original LLaGA model architecture and optimization objective;
- restricting modifications to infrastructure and execution details such as data I/O, trainer wrappers, logging, and cache handling.

In addition, to ensure robustness and reproducibility under constrained resources, we disable specialized decoding branches relying on key-value caching and enforce a unified graph encoding path during inference.

This change does not alter the model structure or training objective, but constrains the generative execution strategy by removing reliance on cached past key-value states and advanced decoding optimizations. All predictions are generated through a unified forward pass that recomputes the graph conditional representation at each step, ensuring consistent inference under limited computational resources. All reported LLaGA results should therefore be interpreted under this controlled decoding setting.

Library and Environment Compatibility Adjustments The original LLaGA implementation assumes the availability of advanced system-level components, including DeepSpeed ZeRO-3, Triton-based kernels, flash-attention, and customized CUDA extensions. However, these dependencies are not supported in our execution environment due to restricted system libraries and limited kernel compilation support.

We therefore remove all DeepSpeed-related logic and simplify the corresponding utility functions (e.g., `maybe_zero_3`) to operate on standard tensors. We also adjust the checkpoint saving mechanism to enable direct storage of multimodal projection weights without invoking ZeRO-3.

Furthermore, since the cluster does not support bf16, tf32, or flash-attention kernels, we explicitly disable these options during training. To maintain compatibility with the current HuggingFace `transformers` API, we update several internal interfaces, including sampler initialization, optimizer calls, and model wrapping behavior, ensuring that training proceeds correctly under a single-GPU setup. All logging to external services (e.g., WandB) is disabled, as outbound network connections are restricted in the HPC environment.

Serialization Format Handling To ensure compatibility when loading preprocessed datasets that contain complex Python objects, including `torch_geometric.data.Data` instances, Python lists, and textual label metadata, we explicitly enable full deserialization when reading trusted local files. Specifically, we load dataset files using

```
torch.load(path, weights_only=False).
```

This setting ensures that graph structures and associated metadata are correctly restored during data loading, without affecting any model parameters or training behavior.

Generation Cache Management and Multimodal Input Handling During inference, we encounter instability when the multimodal embedding injection pipeline interacts with the HuggingFace OPT generation loop. Specifically, the `past_key_values` cache occasionally becomes partially undefined, producing errors such as:

```
AttributeError: 'NoneType' object has no attribute 'shape'.
```

The original LLaGA code contains a special branch inside `prepare_inputs_labels_for_multimodal` that attempts to reuse cached key-value states. This branch is fragile under recent Transformers versions.

To guarantee stable behavior, we implement the following modifications:

- disable key-value caching entirely by setting `model.generation_config.use_cache = False`;
- pass `use_cache=False` explicitly to `model.generate`;
- disable the special cached-attention branch, ensuring that all decoding steps follow a unified `encode_graphs` path.

Although this increases inference time, it guarantees determinism and correctness and does not affect semantic outputs.

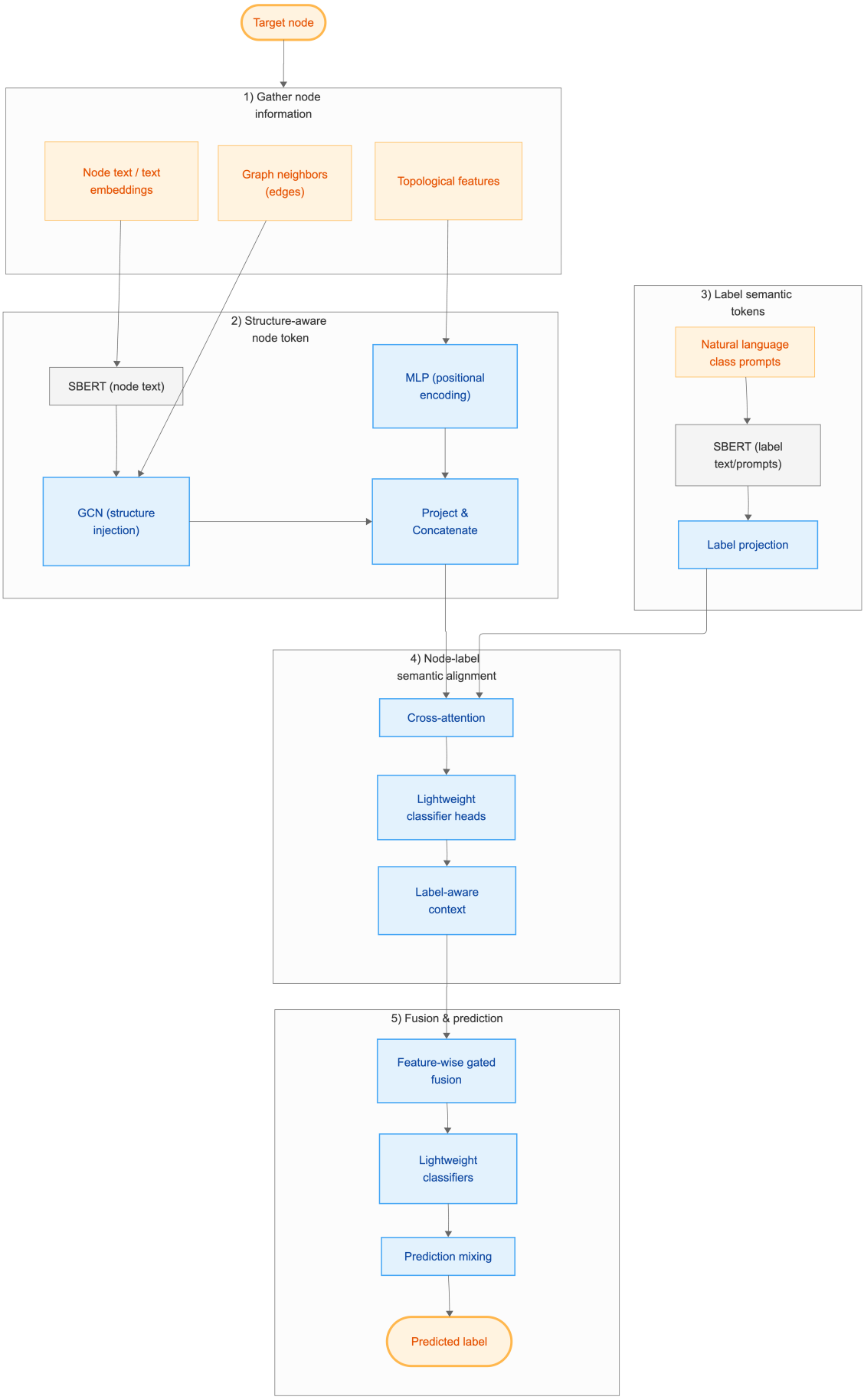


Figure 3.1: Illustration of our hybrid model architecture. Node textual features are encoded by a frozen SBERT encoder and injected with neighborhood structure via a GCN, while positional features are encoded by a lightweight MLP. Label semantics are represented as prompt-based label tokens encoded by SBERT. A prompt-guided cross-attention module aligns node tokens with label tokens, followed by gated fusion and lightweight classifier heads for final node prediction.

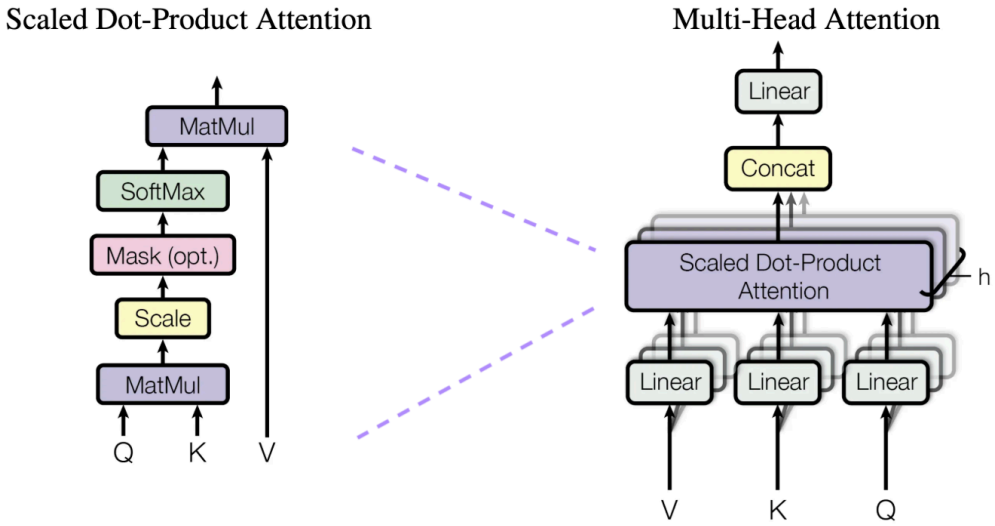


Figure 3.2: (Left) Single attention head (scaled dot-product attention). (Right) Multi-head attention with multiple attention heads in parallel. Source: Attention Is All You Need [45].

4

Experimental Results

To systematically study how large language models affect node classification on textual graphs, we conduct a comparative study covering three categories of approaches: classical GNNs (GCN, GAT, and GraphSAGE), LLM-based graph reasoning frameworks (ENGINE and LLaGA), and a lightweight hybrid model proposed in this work.

All models are evaluated on identical graph structures and node-level supervision, ensuring that performance differences cannot be attributed to data access or supervision differences. Instead, the models differ fundamentally in (i) how language representations are integrated into graph learning, (ii) whether prediction is formulated as a discriminative classification or a prompt-based generative process, and (iii) how robust each paradigm is under reduced training supervision.

Based on the above-mentioned considerations, our experimental analysis centres on the following research questions:

- **RQ1:** What are the relative advantages of parameter-efficient LLM-GNN integration (ENGINE) vs. prompt-based graph reasoning (LLaGA)?
- **RQ2:** Does integrating LLM representations yield consistent improvements over strong GNN baselines when the graph structure is identical?
- **RQ3:** Does the proposed hybrid model provide complementary benefits beyond existing GNN and LLM-based approaches?
- **RQ4:** How robust are different model families under reduced training supervision?

4.1. Experimental Setup

4.1.1. Datasets

We evaluate all models on two widely used textual graph benchmarks for node classification: Cora [46] and WikiCS [21]. These datasets vary in scale, semantic richness, and the number of labels. This allows us to assess model behavior under different structural and textual conditions.

To better understand these differences and to contextualize the experimental results, we further analyze the structural properties and textual characteristics of both datasets. Table 4.3 summarizes key statistics, including graph size and density, average node degree, and the richness of node-associated text. This analysis highlights the contrasting regimes represented by Cora and WikiCS, ranging from small, sparsely connected graphs with short abstracts to larger, denser graphs with substantially richer textual content.

Table 4.1: Node classification accuracy (%) under the standard training setting (60%). Results are reported as mean ± standard deviation over five random seeds.

Group	Model	Cora	WikiCS
GNN	GCN	86.34 ± 1.28	81.35 ± 0.30
	GAT	86.63 ± 1.45	81.85 ± 0.59
	GraphSAGE	86.52 ± 1.01	85.40 ± 0.28
LLM-based	ENGINE	$\mathbf{89.85 \pm 0.63}$	$\mathbf{86.99 \pm 0.79}$
	LLaGA	85.09 ± 1.37	82.60 ± 1.60
Hybrid (ours)	Hybrid	86.69 ± 1.21	83.11 ± 0.85

Cora. Cora is a citation network in which nodes correspond to scientific publications and edges denote citation links between papers. The dataset contains $2{,}708$ nodes and $5{,}429$ edges, resulting in a sparse graph with a mean degree of 4.00. Each publication in the dataset is described by a word vector with a 0/1 value, indicating the absence or presence of the corresponding word from the dictionary [47]. Each node contains an average of 186.53 tokens and is associated with a brief textual description consisting of a title and abstract. The classification task involves 7 research topics. Due to its small scale, sparse connectivity, and limited textual richness, Cora serves as a standard benchmark for transductive node classification and has been widely used to evaluate the effectiveness of message-passing GNNs under relatively simple semantic conditions.

WikiCS. WikiCS is a larger and more text-rich graph constructed from Wikipedia articles. The dataset comprises $11{,}701$ nodes and $216{,}123$ edges, yielding a substantially higher mean degree of 36.94. The raw text associated with each node includes the name and content of a Wikipedia article. The label of each node corresponds to the article's category. Compared to Cora, WikiCS exhibits significantly richer node-level textual information, with an average of 642.04 tokens per node. The classification task covers 10 categories. These characteristics make WikiCS a more challenging benchmark that combines dense graph structure with long-form textual semantics, providing a complementary evaluation setting for assessing how models scale with increased structural complexity and semantic richness.

Splits and Pre-processing. For both datasets, we split the nodes into 60% training, 20% validation, and 20% test sets, and reused the same splits across all models and random seeds.

Classical GNN models operate on sparse textual features as formulated in the original models [7, 9, 8]. In contrast, ENGINE and LLaGA additionally employ language model encoders to obtain more detailed representations. Our hybrid model lies between these two paradigms. It retains a lightweight GNN backbone for structural message passing, while incorporating pretrained language representations in the form of fixed text embeddings and prompt-derived label semantics, without relying on generative inference or end-to-end LLM fine-tuning. All models use identical graph structures and supervision signals.

This unified setup ensures that performance differences stem from the modeling and integration strategies rather than data preprocessing.

Table 4.2: Node classification accuracy (%) under reduced training data. Train ratio denotes the fraction of the full dataset used for training (10%/20%/30%), while validation and test splits remain unchanged. Results are reported as mean ± standard deviation over five random seeds.

Group	Model & Setting	Cora	WikiCS
GNN	GraphSAGE (10%)	81.10 ± 2.71	82.39 ± 0.42
	GraphSAGE (20%)	84.35 ± 0.89	83.77 ± 0.50
	GraphSAGE (30%)	$\mathbf{85.60 \pm 1.23}$	$\mathbf{84.66 \pm 0.59}$
LLM-based	ENGINE (10%)	83.03 ± 0.13	83.44 ± 0.65
	ENGINE (20%)	84.87 ± 0.13	84.44 ± 1.21
	ENGINE (30%)	$\mathbf{86.90 \pm 0.13}$	$\mathbf{85.74 \pm 0.41}$
	LLaGA (10%)	57.09 ± 3.70	70.97 ± 2.23
	LLaGA (20%)	74.50 ± 2.48	76.85 ± 1.74
	LLaGA (30%)	$\mathbf{80.74 \pm 1.75}$	$\mathbf{78.33 \pm 1.56}$
Hybrid (ours)	Hybrid (10%)	82.92 ± 1.69	78.98 ± 0.37
	Hybrid (20%)	$\mathbf{85.18 \pm 1.74}$	80.31 ± 0.18
	Hybrid (30%)	84.44 ± 2.17	$\mathbf{81.90 \pm 0.25}$

4.1.2. Evaluation Protocol

To ensure fair comparison across classical GNNs, LLM-based frameworks, and hybrid architectures, we adopt a unified multi-seed evaluation protocol.

Each model is evaluated over five independent runs with different random seeds. Unless specified otherwise in the original implementation, we use the seed list `[0,1,2,3,4]`. In all cases, the total number of runs is kept identical across models.

For each run, the full pipeline is executed independently, including data loading, model initialization, training, and inference. This procedure applies uniformly to full-batch and mini-batch GNNs, as well as LLM-based methods involving embedding extraction, prompt construction, and prediction.

To evaluate data efficiency, we report results under training ratios of {10%, 20%, 30%}. Each ratio refers to the proportion of the entire dataset used for training. For a given seed, the train/validation/test split is first constructed, after which the training subset is sampled from that run's training pool to match the target ratio. Validation and test sets remain fixed for all ratios under the same seed.

We report the results as the mean accuracy and standard deviation across the five runs:

$$\mu \pm \sigma.$$

This consistent multi-seed evaluation protocol helps to reduce stochastic effects of initialization, sampling procedures, and generation variability. This ensures that results reflect each model's expected performance rather than favorable random outcomes.

4.2. Results Analysis

We analyze the experimental results from two perspectives: (i) *horizontal comparisons*, which compare different models on the same dataset, and (ii) *vertical comparisons*, which examine the behavior of the

Table 4.3: Dataset statistics for Cora and WikiCS.

Property	Cora	WikiCS
nodes	2,708	11,701
edges	5,429	216,123
classes	7	10
Mean degree	4.00	36.94
Avg. tokens per node	186.53	642.04

same model across datasets and training regimes.

4.2.1. Horizontal Comparison: Different Models on the Same Dataset

Cora. On the Cora dataset, all three classical GNN models achieve similar performance, with accuracies between 86% and 87% under the standard 60% training setting (Table 4.1). This indicates that, given Cora's citation-based structure and strong class homophily, standard message-passing GNNs are already sufficient to propagate label information effectively across local neighborhoods under the 60% training setting. Among them, GraphSAGE performs slightly better, suggesting that sampling-based neighborhood aggregation is well suited for Cora's connectivity structure.

ENGINE achieves the highest accuracy at 89.85%, outperforming all classical GNN baselines. This gain indicates that injecting pre-trained language representations into graph reasoning provides additional semantic information beyond purely structural propagation.

LLaGA attains 85.09% on Cora, slightly below discriminative baselines. Under the closed-set classification setting, the model predicts a label from a predefined class set, and evaluation is performed by exact label matching rather than free-form text generation. Within this setting, generative reasoning offers a limited advantage over discriminative training and appears more sensitive to optimization and decoding constraints.

The proposed hybrid model reaches 86.69%, comparable to strong GNN baselines and substantially above the generative LLM-based method. This suggests that lightweight semantic-structural fusion can enhance classical GNN representations without relying on full generative decoding.

WikiCS. On WikiCS, GCN and GAT achieve 81.35% and 81.85% respectively, while GraphSAGE improves to 85.40%. This indicates that richer neighborhood sampling benefits larger and semantically diverse graphs.

ENGINE again achieves the best performance at 86.99%, demonstrating consistent gains across datasets with different sizes and label distributions. The hybrid model attains 83.11%, remaining competitive with GNN baselines while underperforming compared to ENGINE. This suggests that semantic label priors contribute useful guidance, but full LLM-enhanced architectures still provide stronger global semantic transfer on WikiCS.

LLaGA obtains 82.60%, remaining below discriminative methods. Overall, under strict closed-set evaluation, discriminative semantic-structural integration appears more effective than generative prompting for node classification.

4.2.2. Vertical Comparison: Model Behavior Across Training Regimes

To better illustrate model robustness under limited supervision, we visualize the results using line plots.

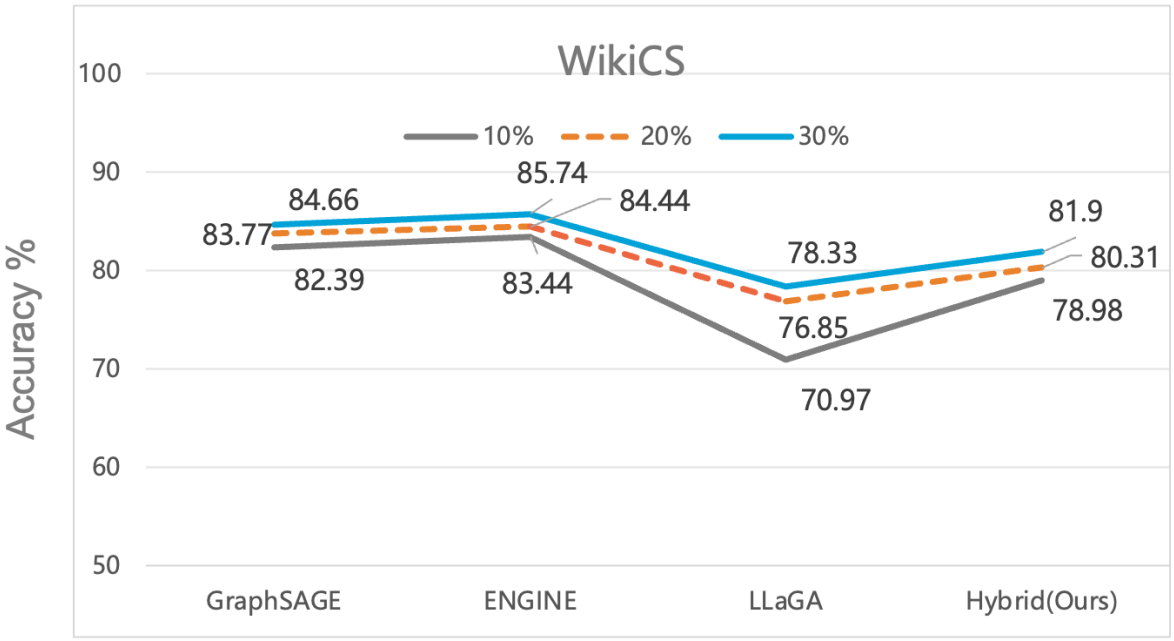


Figure 4.1: **Robustness Under Limited Supervision on WikiCS.** Classification accuracy under varying proportions of training data (10%, 20%, and 30%) for different models within WikiCS.

This representation emphasizes how smoothly or abruptly each model's performance changes with increasing supervision, thereby revealing differences in stability and sensitivity. In particular, steep drops or irregular trends indicate a higher dependence on labeled data or sensitivity to training conditions, whereas smoother curves suggest stronger robustness under reduced supervision.

We analyze these trends on both WikiCS and Cora in the following. We further analyze model robustness under reduced supervision using training ratios of 10%, 20%, and 30% of the full dataset. The results are presented in Table 4.2.

Classical GNNs exhibit gradual and predictable degradation as labeled data decreases. For example, GraphSAGE on Cora drops from 85.60% (30%) to 81.10% (10%), with a similar trend on WikiCS. This decline reflects the reliance of message passing on available labeled seed nodes.

ENGINE remains comparatively robust in low-label regimes, preserving high accuracy even at 10% training data. This stability indicates that pre-trained language representations provide transferable semantic priors that reduce dependence on large, labeled training sets.

LLaGA shows substantial performance drops under limited supervision, particularly on Cora, where accuracy falls sharply at 10% training data. This suggests that generative prompt-based reasoning requires sufficient labeled signals to reliably align graph structure with text-based prediction.

The hybrid model follows a degradation trend comparable to classical GNNs, without abrupt performance collapse. On Cora, accuracy decreases from 84.44% (30%) to 82.92% (10%), and on WikiCS from 81.90% to 78.98%.

This places the hybrid approach between purely structural GNNs and LLM-based frameworks in terms of robustness, which benefits from semantic label priors while maintaining stable discriminative training.

4.2.3. Overall Discussion

Across all experiments, several patterns emerge. Classical GNNs provide strong and efficient baselines with stable behavior under varying label availability. LLM-based graph frameworks such as ENGINE deliver the highest accuracy and superior robustness under limited supervision, confirming the benefit of injecting rich semantic priors into graph reasoning. Generative prompting approaches such as LLaGA

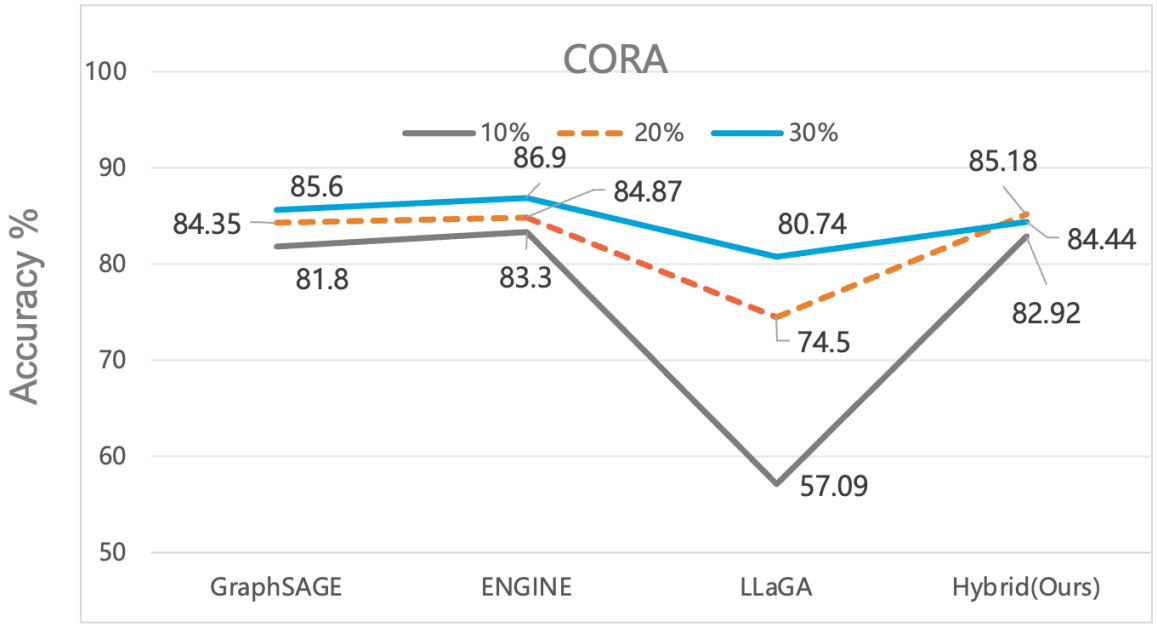


Figure 4.2: **Robustness Under Limited Supervision on Cora.** Classification accuracy under varying proportions of labeled training data (10%, 20%, and 30%) for different models within Cora.

are less effective for closed-set node classification under strict evaluation, particularly in low-resource regimes.

The proposed hybrid model occupies a middle ground. By incorporating LLM-derived semantic label embeddings through lightweight cross-attention while retaining standard GNN message passing and discriminative training, it achieves competitive accuracy and avoids catastrophic failures under label scarcity. These results suggest that lightweight semantic-structural fusion offers a practical and computationally efficient alternative for integrating language priors into graph learning.

4.3. Answers to the Research Questions

In this section, we systematically answer the research questions introduced earlier, grounded in the controlled experimental setting described in Section 4.1. The discussion integrates results from both horizontal and vertical comparisons, with particular emphasis on model behavior under identical data splits, unified evaluation protocols, and constrained computational resources. The goal is not to claim universal superiority of any single approach, but to clarify how different integration strategies behave under realistic and reproducible conditions.

4.3.1. Parameter-Efficient Integration vs. Prompt-Based Reasoning(RQ1)

This section addresses the research question:

What are the relative advantages of parameter-efficient LLM-GNN integration (ENGINE) versus prompt-based graph reasoning (LLaGA)?

This question compares ENGINE's parameter-efficient integration strategy with LLaGA's prompt-based generative reasoning paradigm.

ENGINE follows a discriminative learning approach, in which the language model is kept entirely frozen and only lightweight graph-side components are optimized. Specifically, training updates are restricted to the *G-Ladder* modules, while inference combines cached LLM embeddings with graph message passing and a discriminative classifier. As a result, the optimization objective remains directly aligned with closed-set node classification throughout training.

The engineering adaptations applied in our experiments, such as removing DeepSpeed dependencies,

resolving library conflicts, and modifying the embedding caching mechanism, do not alter ENGINE's core design or learning paradigm. In all cases, the LLM serves purely as a fixed semantic encoder, while structural reasoning and task-specific learning are handled by parameter-efficient graph components. This makes ENGINE inherently robust to inference-time constraints and well-suited for reproducible evaluation under our limited computational resources.

Beyond computational efficiency, the parameter-efficient design of ENGINE also contributes to its strong empirical performance in our setting. By freezing the large language model, ENGINE avoids overfitting high-capacity semantic encoders when labeled supervision is limited. Besides, ENGINE achieves a more stable optimization process and reduces sensitivity to training noise by limiting gradient updates to lightweight graph modules. Furthermore, ENGINE directly optimizes classification accuracy through a discriminative objective function. This method enables improvements in representation quality to be immediately reflected in node-level predictions. These characteristics together explain why ENGINE consistently outperforms classical GNN baselines and remains robust across training regimes in our experiments.

LLaGA uses a fundamentally different approach. It treats graph classification as a generation task, turning the graph into a sequence that is processed with a question prompt by a large language model, which subsequently generates the answer. The model does not directly optimize for classification accuracy. Instead, it learns to generate coherent text that may include class labels. This generative setup is a core part of LLaGA's design, not just an implementation detail.

The adaptations we made affect LLaGA more, since they limit its generation flexibility. Turning off `bf16`/`tf32` precision, flash-attention, and especially key-value caching during generation slows down decoding and may make it less stable. Moreover, our strict string-matching evaluation, which requires exactly one clear class label in the output, does not fit well with how LLMs work. These limits do not make LLaGA inherently worse. Instead, they force it to work in a way that blocks its strengths, like interpretability and flexibility, while making its weaknesses, such as slower inference and prompt sensitivity, more obvious.

Thus, LLaGA's lower performance compared to ENGINE shows that prompt-based generative reasoning struggles under strict evaluation with limited decoding. This observation does not imply that the underlying approach is fundamentally flawed. Instead, this observation indicates a poor match with the evaluation setting we used.

4.3.2. Do LLM Representations Help?(RQ2)

This section addresses the research question:

Does integrating LLM representations yield consistent improvements over strong GNN baselines when the graph structure is identical?

The second research question is most influenced by the adaptations we applied.

Direct comparison is difficult, since the classical GNN models use sparse text features such as Bag-of-Words or TF-IDF, while ENGINE and LLaGA use full LLM forward passes to create contextualized representations from multiple transformer layers.

As a result, ENGINE's advantage over the GNN models comes from two factors: (i) its improved architecture for combining graph and language (like *G-Ladder* for joint reasoning), and (ii) its use of rich, contextualized LLM embeddings that encode substantially more semantic information than sparse text features.

The situation is less clear for LLaGA. LLaGA uses LLM representations, since the language model encodes the whole graph before generating an answer. However, it does not outperform strong GNN baselines in our experiment. This result does not mean LLM representations are ineffective. Instead, it suggests that using language knowledge in a generative manner does not always lead to better accuracy under strict closed-set evaluation. In particular, LLaGA may generate semantically correct but non-standard answers that exhibit uncertainty or use synonymous label descriptions, but our exact string-matching criterion rejects these outputs. Hence, the results of the experiment make it difficult to find a clear answer to RQ2. LLM representations yield consistent improvements when used with

discriminative training that directly optimizes classification accuracy, as exemplified by ENGINE.

However, these improvements are not consistently observed when LLM representations are integrated via generative reasoning and evaluated under strict label-matching rules, as seen in the current LLaGA setup. It remains unclear whether this is a genuine limitation of generative methods or simply a consequence of our evaluation protocol.

4.3.3. Complementary Effects of Our Lightweight model.(RQ3)

This section addresses the research question:

Does the proposed hybrid model provide complementary benefits beyond existing GNN and LLM-based approaches?

Table 4.1 and Table 4.2 demonstrate that our proposed hybrid model consistently achieves competitive performance across both datasets and training regimes, while exhibiting robustness that are distinct from those of classical GNNs and LLM-based graph reasoning frameworks. Under the standard 60% training setting, the hybrid model slightly outperforms strong message-passing baselines on Cora (86.69% vs. 86.34%–86.63%) and remains competitive on WikiCS, although it does not surpass ENGINE. More importantly, when supervision is reduced, the hybrid model shows a notably smooth performance degradation, without the abrupt accuracy drops observed in some LLM-based generative approaches. For example, on Cora with only 10% labeled training nodes, the hybrid model achieves 82.92%, compared to 81.10% for GraphSAGE and 57.09% for LLaGA. Similar robustness trends are consistently observed on WikiCS, indicating that the hybrid model maintains stable predictive behavior even under limited-label regimes.

This behavior suggests that the hybrid model provides complementary benefits by integrating semantic priors from pretrained language models into a lightweight graph learning architecture. Compared to classical GNNs, which rely primarily on sparse textual features and local message passing, the hybrid model explicitly incorporates label semantics through prompt-derived label tokens and node-label cross-attention. This mechanism supplies additional semantic guidance during training, which becomes particularly valuable when labeled data is scarce. As a result, the hybrid model can better align node representations with class semantics than purely structure-driven GNNs, without requiring deep or computationally expensive language model integration.

In contrast to LLM-based frameworks, especially generative approaches such as LLaGA, the hybrid model avoids formulating node classification as a text generation problem. Instead, it adopts a discriminative prediction objective that directly optimizes classification accuracy. This design choice circumvents the instability and prompt sensitivity commonly associated with generative decoding under closed-set classification protocols. At the same time, unlike ENGINE, which relies on frozen LLM backbones and layer-wise graph-side modules, the hybrid model replaces full-scale LLM encoders with compact Sentence-BERT embeddings and shallow GCN layers. This results in a substantially lighter architecture with reduced computational and memory requirements, while still retaining meaningful semantic information.

Overall, the hybrid model does not aim to universally outperform either classical GNNs or LLM-based graph reasoning frameworks. Instead, it occupies a complementary middle ground between these paradigms. It inherits the stable optimization dynamics and structural efficiency of message-passing GNNs, while selectively incorporating semantic priors from pretrained language models through explicit node–label alignment. This balance enables the hybrid model to remain robust under reduced supervision, avoid the instability of generative LLM reasoning, and achieve competitive accuracy with significantly lower computational overhead.

4.3.4. Model Robustness Under Limited Supervision.(RQ4)

This section addresses the research question:

How robust are different model families under reduced training supervision?

Table 4.2 reports model performance when the training set is reduced to 10-30% of the full dataset, while validation and test splits remain fixed. Across all methods, decreasing supervision consistently

leads to performance degradation, but with markedly different sensitivity levels. Classical GNNs show a gradual and stable decline. For instance, GraphSAGE on Cora decreases from 85.60% (30%) to 81.10% (10%), and a similar trend is observed on WikiCS. ENGINE remains comparatively robust under label scarcity, preserving high accuracy even at 10% training data on both datasets. In contrast, LLaGA exhibits substantial drops in low-data regimes, particularly on Cora, where performance falls sharply at 10% training data.

The hybrid model follows a degradation trend comparable to classical GNNs. As training data diminishes, their performance declines gradually without an abrupt collapse. On Cora, hybrid accuracy ranges from 84.44% (30%) to 82.92% (10%), while on WikiCS it varies from 81.90% (30%) to 78.98% (10%). This places the hybrid model between strong GNN baselines and LLM-based frameworks in terms of robustness.

These results indicate that robustness under limited supervision depends strongly on how semantic and structural information are integrated. Message-passing GNNs exhibit predictable degradation due to their reliance on local label propagation. ENGINE benefits from pre-trained language representations that provide transferable semantic priors, allowing it to remain stable with fewer labeled nodes. By contrast, generative LLM-based reasoning in LLaGA appears more sensitive to reduced supervision, likely because prompt-based decoding requires sufficient labeled signals to align graph structure with text generation. The hybrid model displays behavior closer to classical GNNs, with no sudden performance collapse in low-data regimes, while still leveraging semantic label information through prompt-based embeddings. Overall, these results indicate that lightweight hybrid semantic-structural fusion achieves robustness comparable to classical GNNs, while retaining part of the semantic generalization advantage of LLM-based models, without relying on large generative backbones.

4.4. Implications

The engineering adaptations and execution constraints discussed above directly affect how the experimental results should be interpreted and limit the reliability of the conclusions that can be drawn from them. Rather than treating the reported accuracies as absolute performance indicators, they should be understood in the context of a controlled and resource-constrained evaluation setting.

Within this setting, the advantages of ENGINE appear robust and consistent. Although several engineering modifications are required to ensure compatibility with the HPC environment, these changes are confined to infrastructure-level components and do not alter the model's core architectural design or learning objective. Across both datasets, ENGINE demonstrates strong and stable performance, suggesting that parameter-efficient integration of LLM representations with graph neural networks is a practical and effective strategy for node classification tasks. Therefore, the empirical evidence supports the claim that discriminative LLM-GNN hybrids, which decouple semantic encoding from task-specific inference, can reliably leverage language representations without incurring excessive computational or optimization overhead.

In contrast, the results of LLaGA require more cautious interpretation. While its performance is consistently lower under the present experimental setup, this outcome should not be taken as definitive evidence that generative, prompt-based graph reasoning is inherently inferior to discriminative approaches. LLaGA relies heavily on prompt formulation, decoding strategies, and flexible generative inference, all of which are sensitive to computational constraints and evaluation protocols. It is therefore likely that relaxed resource constraints or evaluation schemes that go beyond exact label matching could lead to improved performance.

A more precise interpretation is that generative LLM-based approaches face substantial challenges under strict accuracy-driven evaluation settings, particularly when inference is constrained, and predictions must exactly match a fixed label set. In such scenarios, the flexibility of language generation does not directly translate into higher classification accuracy and may instead introduce instability or inefficiency.

Finally, classical GNN models serve as an important reference point due to their architectural

simplicity and well-understood inductive biases. Their performance establishes a strong baseline that reflects what can be achieved using graph structure and sparse textual features alone, without the added complexity of large language models. The consistent improvements of ENGINE over these GNN baselines indicate that deeper language integration can be beneficial when aligned with discriminative training objectives. Conversely, the inability of LLaGA to consistently surpass classical GNNs highlights that the effectiveness of language integration depends not only on the richness of semantic representations but also on how these representations are incorporated into the learning and inference process.

5

Conclusion

5.1. Overview

Our work presents a systematic comparison of classical graph neural networks and LLM-based frameworks for node classification on TAGs. We evaluate three classical GNN architectures, including GCN, GAT, and GraphSAGE, alongside two representative LLM-based methods, including ENGINE and LLaGA, and our proposed lightweight hybrid model under identical experimental settings. All models are trained and tested on the same data splits using a consistent multi-seed evaluation protocol.

The experimental results reveal substantial differences across model families. Classical GNNs achieve strong accuracy on both Cora and WikiCS, with accuracies ranging from 81% to 86% under standard training conditions. ENGINE consistently outperforms all baselines, with accuracies reaching 89.85% on Cora and 86.99% on WikiCS. This advantage stems from its parameter-efficient design, which injects pretrained semantic representations through lightweight trainable modules while keeping the language model frozen. By contrast, LLaGA underperforms relative to discriminative methods, particularly on Cora (85.09%). This outcome reflects the constraints imposed by the controlled and reproducible evaluation protocol adopted throughout this study, which favors deterministic, closed-set prediction over flexible generative inference. Under this protocol, discriminative models that directly optimize classification accuracy benefit from stable training and evaluation conditions, whereas prompt-based generative approaches are more sensitive to inference constraints. Our hybrid model achieves competitive accuracy (86.69% on Cora, 83.11% on WikiCS) while using a substantially lighter architecture.

When training data is reduced to 10-30% of the full dataset, accuracy differences become more pronounced. Classical GNNs degrade gradually and predictably. ENGINE remains robust, maintaining high accuracy even at 10% supervision. LLaGA exhibits sharp performance drops, falling to 57.09% on Cora at 10% training data. The hybrid model follows a controlled degradation pattern similar to classical GNNs, avoiding abrupt collapse while retaining partial semantic benefits from label-based supervision.

5.2. Limitations

Although the present study provides a systematic and controlled comparative analysis, several limitations should be acknowledged when interpreting the results. First, the empirical comparison includes only two representative LLM-based frameworks, ENGINE and LLaGA. While these models capture distinct integration paradigms, the findings may not directly generalize to other emerging hybrid or prompt-based architectures that employ different forms of graph–language interaction, parameterization, or training objectives.

Besides, to ensure fair comparison and strict reproducibility across all methods, we disable several optimizations commonly used in large-scale generative systems, including distributed inference and

advanced decoding strategies. Although these constraints are necessary to maintain a controlled and deterministic evaluation protocol, they limit LLaGA's ability to fully exploit its generative flexibility and efficiency. As a result, the performance of generative approaches should be interpreted as reflecting robustness under practical deployment constraints, rather than their maximal achievable performance under unconstrained generative settings. In addition, the scope of the present evaluation is limited to single-label, closed-set node classification, with accuracy as the primary performance metric. While this setting is appropriate for controlled comparison and aligns with common practice in benchmark evaluations, it inherently favors discriminative classification paradigms. As a result, the current setup does not capture scenarios in which generative or prompt-based approaches may offer advantages, such as open-set classification, long-tailed label distributions, multi-label prediction, or tasks that require natural-language explanations. Consequently, the conclusions regarding the relative effectiveness of discriminative and generative integration strategies should be interpreted within the confines of closed-set accuracy-driven evaluation.

Moreover, although the proposed hybrid model is designed to be lightweight, it comprises multiple interacting components, including prompt-guided cross-attention, cosine-based label aggregation, feature-wise gating, and a multi-branch prediction mechanism with learnable mixture weights. The present study does not include a comprehensive ablation analysis of these components. As a result, it is not possible to precisely disentangle the individual contributions of each module, assess potential redundancy, or determine which components are most critical under different supervision regimes. A more systematic ablation study would be necessary to clarify the extent to which the observed performance gains stem from specific architectural choices, as opposed to the overall model complexity or the use of label semantic priors.

Furthermore, the choice of the semantic text encoder is fixed throughout the experiments. Both the proposed hybrid model and the LLaGA framework rely on frozen Sentence-BERT representations to encode textual information. While this design choice ensures computational efficiency and isolates the effect of lightweight semantic priors, it also constrains the representational capacity of the language component. Accordingly, the reported results primarily reflect the effectiveness of incorporating static, pretrained sentence-level embeddings and do not address how the relative performance of different integration paradigms might change if stronger language encoders were employed, or if text encoders were partially or fully fine-tuned.

Finally, although the study emphasizes practical feasibility and reproducibility under constrained computational resources, it does not provide a systematic quantification of training and inference costs. Metrics such as GPU-hours, peak memory consumption, throughput, and inference latency are not explicitly reported. This limits the ability to fully assess accuracy-cost trade-offs across models, particularly when comparing parameter-efficient discriminative approaches with generative frameworks that incur higher computational overhead.

5.3. Future Directions

This work opens several concrete directions for future research. First, while our comparative experiments focus on a small set of representative frameworks, a natural next step is to more systematically explore the design space of lightweight hybrid models that avoid full generative reasoning yet leverage semantic priors from language models. Concretely, future work could implement and benchmark alternative hybrid architectures that vary where and how semantic information is injected, such as replacing cross-attention with label-conditioned adapters or low-rank conditioning layers attached to intermediate GNN representations. By controlling architectural complexity while varying the semantic injection mechanism, such studies could help identify which design choices offer the best trade-off between performance, stability, and computational cost, thereby reducing ambiguity in the rapidly growing space between classical GNNs and fully generative graph-LLM models.

Second, our results indicate that robustness under limited supervision varies substantially across model families. Future work could systematically investigate how different mechanisms for

incorporating semantic information, such as using pretrained node text embeddings or prompt-based label conditioning, interact with graph learning when labeled data are lacking. In particular, an open question is which forms of semantic supervision are most effective at improving data efficiency and stabilizing training, without introducing the optimization instability observed in fully generative approaches.

Third, beyond classification accuracy, future research should explicitly evaluate LLM-based graph models along additional practical dimensions relevant to real-world deployment. In particular, systematic benchmarking of inference latency, peak memory usage, and run-to-run reproducibility across different hardware and software configurations would provide a more realistic assessment of model suitability in production settings. By coupling performance metrics with system-level measurements, future studies could better characterize when the additional complexity of LLM integration is justified, and when simpler graph learning approaches remain preferable despite slightly lower accuracy.

Finally, our proposed hybrid model suggests a promising direction for combining discriminative graph learning with generative language models in a more controlled manner. While the current framework relies on lightweight cross-attention and label-conditioned representations for efficient closed-set classification, future work may explore the use of generative language models as high-level reasoning or explanation modules operating on top of discriminative predictions. Such a design could enhance interpretability and flexibility without sacrificing the efficiency and stability of discriminative graph inference. Exploring when and how generative language models can be integrated into hybrid graph architectures remains an open and practically relevant research direction.

Bibliography

[1] Weihua Hu et al. “Open graph benchmark: Datasets for machine learning on graphs”. In: *Advances in neural information processing systems* 33 (2020), pp. 22118–22133.

[2] Masaki Eto. “Extended co-citation search: Graph-based document retrieval on a co-citation network containing citation context information”. In: *Information Processing & Management* 56.6 (2019), p. 102046.

[3] Aidan Hogan et al. “Knowledge graphs”. In: *ACM Computing Surveys (Csur)* 54.4 (2021), pp. 1–37.

[4] Hongming Zhang et al. “ASER: A large-scale eventuality knowledge graph”. In: *Proceedings of the web conference 2020*. 2020, pp. 201–211.

[5] Xiao Li et al. “A survey of graph neural network based recommendation in social networks”. In: *Neurocomputing* 549 (2023), p. 126441.

[6] Vineeta Anand and Ashish Kumar Maurya. “A survey on recommender systems using graph neural network”. In: *ACM Transactions on Information Systems* 43.1 (2025), pp. 1–49.

[7] Thomas N Kipf and Max Welling. “SEMI-SUPERVISED CLASSIFICATION WITH GRAPH CONVOLUTIONAL NETWORKS”. In: ().

[8] Will Hamilton, Zhitao Ying, and Jure Leskovec. “Inductive representation learning on large graphs”. In: *Advances in neural information processing systems* 30 (2017).

[9] Petar Veličković et al. “Graph attention networks”. In: *arXiv preprint arXiv:1710.10903* (2017).

[10] Hugo Touvron et al. “Llama: Open and efficient foundation language models”. In: *arXiv preprint arXiv:2302.13971* (2023).

[11] Josh Achiam et al. “Gpt-4 technical report”. In: *arXiv preprint arXiv:2303.08774* (2023).

[12] Ebtesam Almazrouei et al. *Falcon-40B: an open large language model with state-of-the-art performance*. 2023.

[13] Yu Wang et al. “Knowledge graph prompting for multi-document question answering”. In: *Proceedings of the AAAI conference on artificial intelligence*. Vol. 38. 17. 2024, pp. 19206–19214.

[14] Wei Wei et al. “Llmrec: Large language models with graph augmentation for recommendation”. In: *Proceedings of the 17th ACM international conference on web search and data mining*. 2024, pp. 806–815.

[15] Aitor Lewkowycz et al. “Solving quantitative reasoning problems with language models”. In: *Advances in neural information processing systems* 35 (2022), pp. 3843–3857.

[16] Runjin Chen et al. “Llaga: Large language and graph assistant”. In: *arXiv preprint arXiv:2402.08170* (2024).

[17] Yun Zhu et al. “Efficient tuning and inference for large language models on textual graphs”. In: *arXiv preprint arXiv:2401.15569* (2024).

[18] Xi Zhu et al. “Llm as gnn: Graph vocabulary learning for text-attributed graph foundation models”. In: *arXiv preprint arXiv:2503.03313* (2025).

[19] Hao Liu et al. “One for all: Towards training one graph model for all classification tasks”. In: *arXiv preprint arXiv:2310.00149* (2023).

[20] Prithviraj Sen et al. “Collective classification in network data”. In: *AI magazine* 29.3 (2008), pp. 93–93.

[21] Péter Mernyei and Cătălina Cangea. “Wiki-cs: A wikipedia-based benchmark for graph neural networks”. In: *arXiv preprint arXiv:2007.02901* (2020).

[22] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. “Deepwalk: Online learning of social representations”. In: *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*. 2014, pp. 701–710.

[23] Aditya Grover and Jure Leskovec. “node2vec: Scalable feature learning for networks”. In: *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*. 2016, pp. 855–864.

[24] Kezhao Huang et al. “Understanding and bridging the gaps in current GNN performance optimizations”. In: *Proceedings of the 26th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*. 2021, pp. 119–132.

[25] Yin Zhang, Rong Jin, and Zhi-Hua Zhou. “Understanding bag-of-words model: a statistical framework”. In: *International journal of machine learning and cybernetics* 1.1 (2010), pp. 43–52.

[26] Tomas Mikolov et al. “Distributed representations of words and phrases and their compositionality”. In: *Advances in neural information processing systems* 26 (2013).

[27] Xixi Wu et al. “When Do LLMs Help With Node Classification? A Comprehensive Analysis”. In: *arXiv preprint arXiv:2502.00829* (2025).

[28] Gerard Salton and Christopher Buckley. “Term-weighting approaches in automatic text retrieval”. In: *Information processing & management* 24.5 (1988), pp. 513–523.

[29] Bowen Jin et al. “Patton: Language model pretraining on text-rich networks”. In: *arXiv preprint arXiv:2305.12268* (2023).

[30] Zhikai Chen et al. “Label-free node classification on graphs with large language models (llms)”. In: *arXiv preprint arXiv:2310.04668* (2023).

[31] Xiaoxin He et al. “Harnessing explanations: Llm-to-lm interpreter for enhanced text-attributed graph representation learning”. In: *arXiv preprint arXiv:2305.19523* (2023).

[32] William Brannon et al. “Congrat: Self-supervised contrastive pretraining for joint graph and text embeddings”. In: *Proceedings of TextGraphs-17: Graph-based Methods for Natural Language Processing*. 2024, pp. 19–39.

[33] Junhan Yang et al. “Graphformers: Gnn-nested transformers for representation learning on textual graph”. In: *Advances in Neural Information Processing Systems* 34 (2021), pp. 28798–28810.

[34] Costas Mavromatis et al. “Train your own gnn teacher: Graph-aware distillation on textual graphs”. In: *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*. Springer. 2023, pp. 157–173.

[35] Tao Zou et al. “Pretraining language models with text-attributed heterogeneous graphs”. In: *arXiv preprint arXiv:2310.12580* (2023).

[36] Jianan Zhao et al. “Learning on large-scale text-attributed graphs via variational inference”. In: *arXiv preprint arXiv:2210.14709* (2022).

[37] Jiabin Tang et al. “Graphgpt: Graph instruction tuning for large language models”. In: *Proceedings of the 47th International ACM SIGIR Conference on Research and Development in Information Retrieval*. 2024, pp. 491–500.

[38] Ziwei Chai et al. “Graphllm: Boosting graph reasoning ability of large language model”. In: *IEEE Transactions on Big Data* (2025).

[39] Jin Huang et al. “Can llms effectively leverage graph structural information through prompts, and why?” In: *arXiv preprint arXiv:2309.16595* (2023).

[40] Jiayan Guo et al. “Gpt4graph: Can large language models understand graph structured data? an empirical evaluation and benchmarking”. In: *arXiv preprint arXiv:2305.15066* (2023).

[41] Rui Xue et al. “Efficient large language models fine-tuning on graphs”. In: *arXiv preprint arXiv:2312.04737* (2023).

[42] Shirui Pan, Yizhen Zheng, and Yixin Liu. “Integrating graphs with large language models: Methods and prospects”. In: *IEEE Intelligent Systems* 39.1 (2024), pp. 64–68.

[43] Edward J Hu et al. “Lora: Low-rank adaptation of large language models.” In: *ICLR* 1.2 (2022), p. 3.

[44] Nils Reimers and Iryna Gurevych. “Sentence-bert: Sentence embeddings using siamese bert-networks”. In: *arXiv preprint arXiv:1908.10084* (2019).

[45] Ashish Vaswani et al. “Attention is all you need”. In: *Advances in neural information processing systems* 30 (2017).

[46] Zhilin Yang, William Cohen, and Ruslan Salakhudinov. “Revisiting semi-supervised learning with graph embeddings”. In: *International conference on machine learning*. PMLR. 2016, pp. 40–48.

[47] Andrew McCallum. *Cora*. 2024. doi: 10.21227/jsg4-wp31. url: https://dx.doi.org/10.21227/jsg4-wp31.