DEKOR: DECOMPOSED KNOWLEDGE ROUTING FOR EFFICIENT GNN–LLM FUSION

Anonymous authors
Paper under double-blind review

ABSTRACT

Large Language Models (LLMs) have shown strong potential for text-attributed graph learning by leveraging rich pre-trained knowledge and language understanding capabilities. Combining LLMs with Graph Neural Networks (GNNs) offers a promising paradigm for expanding the modeling capabilities of graph learning. However, LLMs and GNNs follow distinct learning paradigms, leading to a pronounced mismatch between their knowledge spaces. Such discrepancies make effective knowledge fusion nontrivial. Moreover, existing methods often incur substantial additional computation for only limited performance gains. To address these challenges, we propose DeKoR, an efficient framework for GNN and LLM knowledge fusion. To enable efficient knowledge transfer, DeKoR decomposes LLM knowledge into complementary local and global views and uses the LLM as an offline teacher to distill them into lightweight experts. To enable selective knowledge fusion, an adaptive router learns how much each node should rely on each expert. Extensive experiments on multiple benchmarks demonstrate that DeKoR outperforms existing baselines on both node and subgraph classification, achieving superior effectiveness and efficiency.

1 INTRODUCTION

Large Language Models (LLMs) such as DeepSeek-R1 (Guo et al., 2025) and Qwen3 (Yang et al., 2025) have demonstrated strong capabilities in understanding and reasoning through advances in pretraining and post-training. These capabilities motivate exploration of how LLM knowledge can support graph learning (Jin et al., 2024; Li et al., 2023). In text-attributed graphs (TAGs), LLMs draw on rich pre-trained knowledge to interpret node texts, while graph models capture dependencies among nodes (Hu et al., 2020; Yan et al., 2023; Zhou et al., 2025; Li et al., 2026). Their integration opens a promising path toward general graph learning, enabling graph models to draw on broader knowledge and support a wider range of application scenarios (He et al., 2024b; Zhang, 2023).

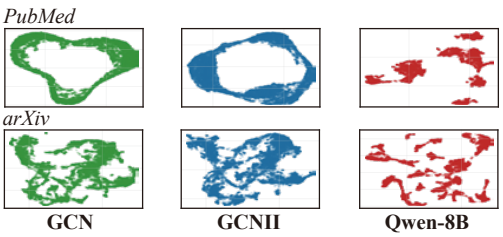


Figure 1: UMAP visualizations of GCN, GCNII, and Qwen-8B on PubMed and arXiv.

Among methods that use LLMs for graph learning, integrating LLMs with Graph Neural Networks (GNNs) offers a promising approach to TAG learning, termed the GNN–LLM paradigm. By combining the powerful textual understanding of LLMs with the relational modeling of GNNs, these methods can better understand node texts in graph context, capturing richer contextual information. Existing GNN–LLM methods mainly differ in how knowledge is transferred and integrated between the two models. Some approaches integrate semantic knowledge from LLMs into GNNs for subsequent graph learning (He et al., 2024a; Liu et al., 2024; Hu et al., 2025); others inject graph structural knowledge into LLMs through graph tokens or parameters (Wang et al.; Chai et al., 2025; Perozzi et al., 2024; Wang et al., 2024; Sun et al., 2026). Recent studies further explore explicit interactions between structural and semantic knowledge, such as (Zeng et al., 2026; Loveland et al., 2026).

Despite these advances, existing GNN–LLM methods still face two challenges in achieving effective and efficient knowledge fusion. **Challenge I: Mismatched knowledge space between LLMs**

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

and GNNs. LLMs and GNNs acquire knowledge under fundamentally different learning paradigms, leading to distinct representation spaces. This mismatch is consistently observed across five datasets: as shown in Tab. 1, the representation gaps between GNNs and Qwen-8B (Yang et al., 2025), measured by $1 - \mathrm{CKA}$ (Kornblith et al., 2019), are consistently larger than those between GCN (Kipf & Welling, 2016) and GCNII (Chen et al., 2020). The UMAP visualizations (McInnes et al., 2018) in Fig. 1 further reveal distinct representation geometries. Such misalignment can cause direct knowledge fusion to introduce redundant or conflicting information, limiting the effective use of the complementary knowledge encoded by the two models.

Table 1: Representation gaps among GCN, GCNII, and Qwen-8B across five datasets.

Dataset	GCN vs. GCNII	GCN vs. Qwen-8B	GCNII vs. Qwen-8B
Cora	0.05	0.26	**0.29**
CiteSeer	0.05	**0.24**	0.21
PubMed	0.14	**0.33**	0.26
WikiCS	0.19	**0.37**	0.30
arXiv	0.20	**0.38**	0.37
Average	0.13	**0.32**	0.29

Challenge II: Limited gains under high computational cost. Many existing GNN–LLM methods require LLM inference for downstream prediction, introducing substantial computational and memory overhead. As shown in Tab. 5, several representative methods incur considerably higher inference costs, while their performance gains remain limited. This imbalance restricts the practicality of GNN–LLM integration, particularly in low-resource settings. Effective integration therefore requires preserving useful LLM knowledge without relying on the LLM during inference.

These challenges suggest that effective GNN–LLM integration should avoid directly coupling the original knowledge spaces of the two models. Our key insight is to separate knowledge transfer from knowledge fusion: LLM knowledge is first transformed into lightweight representations, and fusion is then performed after this transfer rather than directly between the two original models. This design avoids directly fusing knowledge from the mismatched GNN and LLM spaces, while allowing the LLM to be removed during inference.

Based on this insight, we propose DeKoR, a knowledge decomposition and routing framework for GNN–LLM fusion. DeKoR decomposes LLM knowledge into two complementary views: local knowledge that preserves fine-grained node information and global knowledge that characterizes each node through its relations to global prototypes. The decomposed knowledge is distilled into lightweight experts, whose predictions are selectively combined by a node-adaptive router. The LLM is used only as an offline teacher, allowing DeKoR to retain useful LLM knowledge without requiring LLM inference during deployment. In summary, our main contributions are as follows:

- We introduce a new perspective on GNN–LLM knowledge fusion by identifying the mismatch between their knowledge spaces and decoupling knowledge transfer from knowledge fusion. This perspective first transforms LLM knowledge into lightweight forms and then performs fusion on the transferred knowledge.
- Following this idea, we propose DeKoR. It separates LLM knowledge into complementary local and global views and distills them into lightweight experts. For each node, an adaptive router determines how the two experts contribute to prediction, allowing the model to exploit LLM knowledge without requiring LLM inference.
- We conduct extensive experiments on multiple benchmarks, demonstrating that DeKoR consistently improves performance across node and subgraph classification tasks while substantially reducing inference time and GPU memory usage compared with most GNN–LLM baselines.

2 Related Work

LLM for Graph Learning. LLM for Graph Learning aims to leverage the rich pre-trained semantic knowledge of LLMs to assist with graph-related tasks. Methods such as Chen et al. (2024); Zhao et al. (2023); Tang et al. (2024a) investigate how to transform graph data into inputs that can be processed by LLMs. Tang et al. (2024b); Kong et al. (2025) focus on employing instruction tuning to enable LLMs to directly handle diverse graph-related tasks. Furthermore, Zolnai-Lucas et al. (2024); Xia et al. (2024) explore how to leverage LLMs to enhance graph data with semantic knowledge.

GNN–LLM Knowledge Fusion. GNN–LLM knowledge fusion methods aim to jointly leverage the structural knowledge captured by GNNs and the semantic knowledge encoded by LLMs for graph learning. One line of work integrates semantic knowledge from LLMs into GNNs. For example, methods such as He et al. (2024a); Liu et al. (2024) transform LLM semantic knowledge into representations for subsequent GNN learning, while Hu et al. (2025) distills knowledge from an LLM teacher into a GNN through representation alignment. Another line of work injects graph structural knowledge into LLMs. Methods including Wang et al.; Chai et al. (2025); Perozzi et al. (2024); Wang et al. (2024); Sun et al. (2026) encode structural knowledge into graph tokens or parameters, enabling subsequent LLMs to exploit the information for graph-related tasks. Beyond such directional knowledge transfer, recent studies further explore interactions between structural and semantic knowledge. For instance, Zeng et al. (2026) aligns structural and semantic representations in a shared latent space, whereas Loveland et al. (2026) adaptively determines when LLM knowledge should be incorporated for individual nodes. Despite these advances, most existing approaches fuse GNN and LLM knowledge without considering the mismatch between their knowledge spaces. Therefore, their integration may introduce redundant or conflicting information. Moreover, many methods rely on LLM inference, resulting in substantial computational overhead. These limitations motivate a more effective and efficient knowledge integration paradigm.

3 PRELIMINARY

Text-Attributed Graph. A text-attributed graph (TAG) is denoted as $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathbf{X}, \mathcal{T}, \mathcal{Y})$, where $\mathcal{V} = \{v_1, \ldots, v_{|\mathcal{V}|}\}$ and $\mathcal{E} = \{e_1, \ldots, e_{|\mathcal{E}|}\}$ are the node and edge sets, respectively. Each node $v \in \mathcal{V}$ is associated with a text description $t_v \in \mathcal{T}$, which is encoded into a feature vector $x_v \in \mathbf{X}$ by a text encoder. $\mathcal{Y} = \{y_v\}_{v \in \mathcal{V}}$ denotes the set of node labels.

Graph Neural Networks. GNNs learn node representations by propagating and aggregating information from neighbors. Let $\mathbf{h}_v^{(0)} = x_v$ denote the initial feature of node v, for the ℓ-th layer, the representation of node v is updated by

$$\mathbf{h}_v^{(\ell)} = \mathrm{UPD}^{(\ell)}\left(\mathbf{h}_v^{(\ell-1)}, \mathrm{AGG}^{(\ell)}\left(\left\{\mathbf{h}_u^{(\ell-1)} : u \in \mathcal{N}(v)\right\}\right)\right), \tag{1}$$

where $\mathcal{N}(v)$ denotes the neighbors of node v, $\mathrm{AGG}^{(\ell)}(\cdot)$ is an aggregation function (e.g., mean or sum), and $\mathrm{UPD}^{(\ell)}(\cdot)$ combines the node's previous representation with the aggregated information.

Large Language Models. We denote a pre-trained large language model as $\mathrm{LLM}(\cdot)$. Given a text t, the LLM first tokenizes it into a sequence of tokens $\mathbf{x} = (x_1, x_2, \ldots, x_q)$, which are then mapped to continuous embeddings and processed through a stack of Transformer layers. Let $\mathbf{R}^{(L)} = [\mathbf{r}_1, \ldots, \mathbf{r}_q]$ denote the token-level representations produced by the last Transformer layer, where $\mathbf{r}_i \in \mathbb{R}^{d_l}$ corresponds to token x_i. We extract node-level semantic representations from $\mathbf{R}^{(L)}$ using a representation extraction operation $\mathrm{Rep}(\cdot)$:

$$\mathbf{z} = \mathrm{LLM}(t) = \mathrm{Rep}\Big(\mathbf{R}^{(L)}\Big) \in \mathbb{R}^{d_z}, \tag{2}$$

where $\mathrm{Rep}(\cdot)$ can be implemented using mean pooling or the last valid token, and d_z denotes the dimension of the resulting semantic representation. In this work, we construct textual prompts for each node v with text t_v, and use the pre-trained LLM to extract its semantic representation.

4 METHOD

We propose DeKoR, an efficient framework for GNN–LLM knowledge fusion, as illustrated in Fig. 2. DeKoR first constructs structural and semantic representations. It then decomposes the LLM knowledge into local and global semantic views, which capture node-level semantics and prototype-based global relations, respectively. The two forms of knowledge are distilled into lightweight semantic experts, without requiring LLM inference during deployment. Finally, a node-adaptive router assigns node-specific weights to the two experts and combines their predictions.

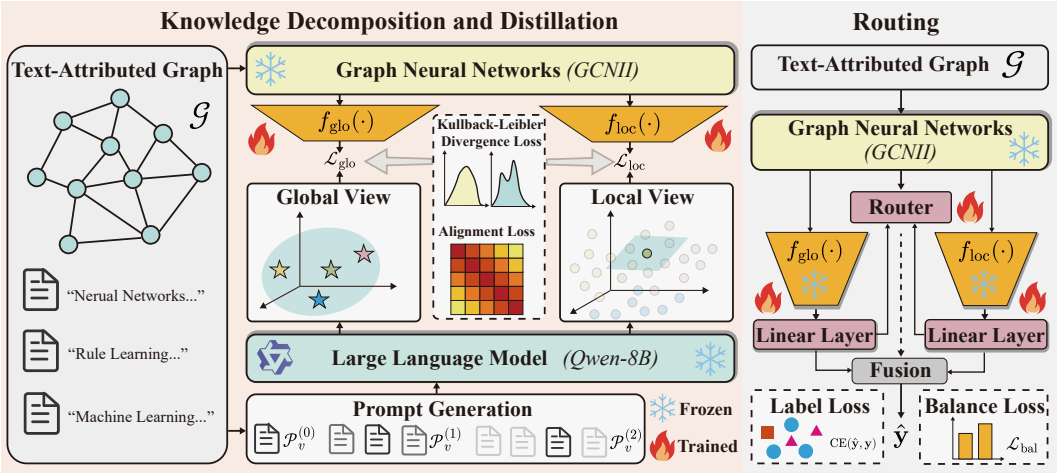


Figure 2: Overall framework of DeKoR. LLM semantic knowledge is decomposed into complementary global and local views and distilled into lightweight experts. During routing, a node-adaptive router dynamically combines the two experts for final prediction.

4.1 Representation Construction

Given a text-attributed graph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathbf{X}, \mathcal{T}, \mathcal{Y})$, we pre-train a GNN encoder $f_{\text{GNN}}(\cdot)$ and fine-tune an $\text{LLM}(\cdot)$ for each dataset. Both models are then frozen and used to construct structural and semantic representations for each node. Detailed training procedures and settings are provided in Appendix D.4 and Appendix D.5.

We adopt the frozen $f_{\text{GNN}}(\cdot)$ to generate structural representations. For each node $v \in \mathcal{V}$, its structural representation is given by $\mathbf{h}_v = f_{\text{GNN}}(\mathbf{X}, \mathcal{E})_v$. To obtain semantic representations, we construct three textual prompts for each node at different context scales: (i) the center node text only, (ii) the center node text with sampled 1-hop neighbors, and (iii) the center node text with sampled 2-hop neighbors. We denote these prompts as $\mathcal{P}_v^{(0)}$, $\mathcal{P}_v^{(1)}$, $\mathcal{P}_v^{(2)}$, respectively. The detailed prompt templates are provided in Appendix C. We then use the frozen $\text{LLM}(\cdot)$ to extract semantic embeddings from these prompts:

$$\mathbf{e}_v^{(m)} = \text{LLM}(\mathcal{P}_v^{(m)}) \in \mathbb{R}^{d_z}, \quad m \in \{0, 1, 2\}. \tag{3}$$

Finally, we concatenate the three embeddings to obtain the semantic representation:

$$\mathbf{z}_v = [\mathbf{e}_v^{(0)}; \mathbf{e}_v^{(1)}; \mathbf{e}_v^{(2)}] \in \mathbb{R}^{3d_z}. \tag{4}$$

4.2 LLM Semantic Knowledge Decomposition

Proposition 1 (Multi-Granularity Semantic Representations). *Let $\mathbf{z} \in \mathbb{R}^d$ denote a node semantic representation, and let $\mathbf{C} \in \mathbb{R}^{K \times d}$ denote a set of K semantic prototypes. We define the prototype-induced representation as*

$$\mathbf{q}(\mathbf{z}) = \text{softmax}\left(\frac{\mathbf{C}\mathbf{z}}{\tau}\right), \tag{5}$$

where $\tau > 0$ is a temperature parameter. The Jacobian (Gao & Pavel, 2017) of this mapping satisfies

$$\text{rank}\left(\mathbf{J}_{\mathbf{q}}(\mathbf{z})\right) \leq \min\left\{K - 1, \text{rank}(\mathbf{C})\right\}. \tag{6}$$

Moreover, for any perturbation $\boldsymbol{\delta} \in \mathbb{R}^d$ satisfying $\mathbf{C}\boldsymbol{\delta} = a\mathbf{1}$ for some scalar a, the prototype-induced representation remains invariant: $\mathbf{q}(\mathbf{z} + \boldsymbol{\delta}) = \mathbf{q}(\mathbf{z})$.

Therefore, $\mathbf{z}$ and $\mathbf{q}(\mathbf{z})$ represent semantics at different granularities: $\mathbf{z}$ preserves original semantics, while $\mathbf{q}(\mathbf{z})$ captures the node's relationships with semantic prototypes. The proof is provided in Appendix A.1.

Connection to Our Method. Proposition 1 characterizes how to decompose semantic knowledge at different granularities. In DeKoR, we preserve the original LLM representations $\mathbf{Z}$ as fine-grained local semantic knowledge, while constructing $\mathbf{Q}$ as prototype-induced global semantic knowledge.

Specifically, we apply K-means (McQueen, 1967) to $\{\mathbf{z}_v\}_{v\in\mathcal{V}}$ and obtain K global semantic prototypes $\mathbf{C} = [\mathbf{c}_1, \ldots, \mathbf{c}_K]^\top$. For each node v, we compute its similarity with these prototypes as:

$$\mathbf{q}_v = \mathrm{softmax}\left(\frac{\mathbf{z}_v\mathbf{C}^\top}{\tau_{\mathrm{teacher}}}\right), \tag{7}$$

where τ_{teacher} controls the sharpness of the distribution. We collect the resulting representations as:

$$\mathbf{Q} = [\mathbf{q}_1, \mathbf{q}_2, \ldots, \mathbf{q}_n]^\top \in \mathbb{R}^{n\times K}. \tag{8}$$

Accordingly, we decompose LLM semantic knowledge into complementary local and global views, capturing semantic information at different granularities.

4.3 Local-Global Semantic Knowledge Distillation

After decomposing the LLM knowledge into local and global semantic views, we transfer the knowledge into lightweight modules through knowledge distillation. We provide a theoretical characterization of the knowledge preserved by the distillation objectives.

Proposition 2 (Information Preservation under Knowledge Distillation)**.** *Let $\mathbf{Z}$ and $\mathbf{H}^{\mathrm{loc}}$ denote the teacher representation and its distilled representation from the local module, respectively. Under the standard InfoNCE assumption (Poole et al., 2019), we have*

$$I\big(\mathbf{H}^{\mathrm{loc}}; \mathbf{Z}\big) \geq \log|\mathcal{B}| - \mathcal{L}_{\mathrm{nce}}, \tag{9}$$

where $\mathcal{B}$ denotes the training batch. Thus, minimizing $\mathcal{L}_{\mathrm{nce}}$ increases the lower bound on the mutual information retained by the lightweight module. For the global semantic view, let $\mathbf{q}_v$ and $\mathbf{p}_v^{\mathrm{glo}}$ denote the teacher and distilled prototype distributions from the global module, respectively. If $\mathcal{L}_{\mathrm{glo},v} = \mathrm{KL}(\mathbf{q}_v \| \mathbf{p}_v^{\mathrm{glo}})$, Pinsker's inequality (Csiszár, 1967) gives

$$\big\|\mathbf{q}_v - \mathbf{p}_v^{\mathrm{glo}}\big\|_1 \leq \sqrt{2\mathcal{L}_{\mathrm{glo},v}}. \tag{10}$$

Therefore, the two objectives support the preservation of local and global semantic knowledge, respectively. The proof is provided in Appendix A.2.

Connection to Our Method. Proposition 2 supports distilling the decomposed LLM knowledge into lightweight modules. The InfoNCE objective preserves local semantic knowledge, while the KL objective preserves the global prototype distribution.

We define lightweight local and global semantic experts as $f_{\mathrm{loc}}(\cdot)$ and $f_{\mathrm{glo}}(\cdot)$. Given the structural representations $\mathbf{H}$ generated by the GNN, we use $f_{\mathrm{loc}}(\cdot)$ and $f_{\mathrm{glo}}(\cdot)$ to encode $\mathbf{H}$ into local and global semantic representations:

$$\mathbf{H}^{\mathrm{loc}} = f_{\mathrm{loc}}(\mathbf{H}), \qquad \mathbf{H}^{\mathrm{glo}} = f_{\mathrm{glo}}(\mathbf{H}). \tag{11}$$

For the local semantic expert, we encourage $\mathbf{h}_v^{\mathrm{loc}} \in \mathbf{H}^{\mathrm{loc}}$ to align with $\mathbf{z}_v \in \mathbf{Z}$ through cosine similarity. For a batch $\mathcal{B}$, the cosine similarity loss $\mathcal{L}_{\cos}$ is defined as:

$$\mathcal{L}_{\cos} = 1 - \frac{1}{|\mathcal{B}|}\sum_{v\in\mathcal{B}} \cos(\mathbf{h}_v^{\mathrm{loc}}, \mathbf{z}_v). \tag{12}$$

Consistent with Proposition 2, we introduce a symmetric InfoNCE loss. The directional loss $\mathcal{L}_{\mathrm{h}\to\mathrm{z}}$ is defined from $\mathbf{h}_v^{\mathrm{loc}}$ to $\mathbf{z}_v$, while $\mathcal{L}_{\mathrm{z}\to\mathrm{h}}$ is obtained by reversing the two representations:

$$\mathcal{L}_{\mathrm{h}\to\mathrm{z}} = -\frac{1}{|\mathcal{B}|}\sum_{v\in\mathcal{B}} \log \frac{\exp\left(\cos(\mathbf{h}_v^{\mathrm{loc}}, \mathbf{z}_v)/\tau_{\mathrm{loc}}\right)}{\sum_{u\in\mathcal{B}}\exp\left(\cos(\mathbf{h}_v^{\mathrm{loc}}, \mathbf{z}_u)/\tau_{\mathrm{loc}}\right)}, \qquad \mathcal{L}_{\mathrm{nce}} = \frac{1}{2}\left(\mathcal{L}_{\mathrm{h}\to\mathrm{z}} + \mathcal{L}_{\mathrm{z}\to\mathrm{h}}\right), \tag{13}$$

where τ_{loc} is the temperature parameter. The local expert is optimized with:

$$\mathcal{L}_{\mathrm{loc}} = \mathcal{L}_{\cos} + \lambda_{\mathrm{nce}}\mathcal{L}_{\mathrm{nce}}. \tag{14}$$

For the global semantic expert, we align $\mathbf{h}_v^{\text{glo}} \in \mathbf{H}^{\text{glo}}$ with $\mathbf{q}_v \in \mathbf{Q}$. We first convert $\mathbf{h}_v^{\text{glo}}$ into a probability distribution using softmax. Motivated by Proposition 2, we then measure its discrepancy from $\mathbf{q}_v$ using the KL divergence (Kullback & Leibler, 1951):

$$\mathcal{L}_{\text{glo}} = \text{KL}\left(\mathbf{q}_v \,\middle\|\, \text{softmax}\left(\frac{\mathbf{h}_v^{\text{glo}}}{\tau_{\text{glo}}}\right)\right), \tag{15}$$

where τ_{glo} is the temperature parameter. Finally, the parameters of $f_{\text{loc}}(\cdot)$ and $f_{\text{glo}}(\cdot)$ are updated by minimizing $\mathcal{L}_{\text{distill}} = \lambda_{\text{loc}}\mathcal{L}_{\text{loc}} + \lambda_{\text{glo}}\mathcal{L}_{\text{glo}}$.

4.4 Node-Adaptive Semantic Knowledge Routing

After distillation, we freeze $f_{\text{loc}}(\cdot)$ and $f_{\text{glo}}(\cdot)$. Given the $\mathbf{h}_v^{\text{loc}}$ and $\mathbf{h}_v^{\text{glo}}$ for each node $v \in \mathcal{V}$, we use prediction heads $g^{\text{loc}}(\cdot)$ and $g^{\text{glo}}(\cdot)$ to obtain their prediction logits:

$$\mathbf{y}_v^{\text{loc}} = g^{\text{loc}}(\mathbf{h}_v^{\text{loc}}), \qquad \mathbf{y}_v^{\text{glo}} = g^{\text{glo}}\big(\mathbf{h}_v^{\text{glo}}\big). \tag{16}$$

To achieve fine-grained knowledge fusion, we train a router $\phi(\cdot)$ to adaptively assign fusion weights to the two experts for each node:

$$\boldsymbol{\alpha}_v = \text{softmax}\big(\phi\big(\big[\mathbf{h}_v; \mathbf{y}_v^{\text{loc}}; \mathbf{y}_v^{\text{glo}}\big]\big)\big) = \big[\alpha_v^{\text{loc}}, \alpha_v^{\text{glo}}\big]^\top, \tag{17}$$

where $\boldsymbol{\alpha}_v$ denotes the fusion weights for node v. Then we can obtain the final prediction $\hat{\mathbf{y}}_v$:

$$\hat{\mathbf{y}}_v = \alpha_v^{\text{loc}}\mathbf{y}_v^{\text{loc}} + \alpha_v^{\text{glo}}\mathbf{y}_v^{\text{glo}}. \tag{18}$$

To encourage balanced utilization of the experts during routing, we define the average routing weight of expert m in a batch as $\bar{\alpha}^m = \frac{1}{|\mathcal{B}|}\sum_{v\in\mathcal{B}} \alpha_v^m$, and the balance loss $\mathcal{L}_{\text{bal}}$ is defined as:

$$\mathcal{L}_{\text{bal}} = \sum_{m\in\{\text{loc},\text{glo}\}} \left(\bar{\alpha}^m - \frac{1}{2}\right)^2. \tag{19}$$

At this stage, we optimize the parameters of the router $\phi(\cdot)$, the prediction $g^{\text{loc}}(\cdot)$ and $g^{\text{glo}}(\cdot)$ by minimizing the routing objective $\mathcal{L}_{\text{route}} = \text{CE}(\hat{\mathbf{y}}_v, y_v) + \lambda_{\text{bal}}\mathcal{L}_{\text{bal}}$.

5 Experiments

In this section, we conduct extensive experiments to evaluate the effectiveness of **DeKoR**, aiming to answer the following research questions: **RQ1:** How effective is DeKoR compared with existing baselines on node and subgraph classification tasks? **RQ2:** Does DeKoR remain effective across different GNN and LLM backbones? **RQ3:** How do the proposed components contribute to the performance of DeKoR? **RQ4:** How does the adaptive routing mechanism behave across different nodes? **RQ5:** How sensitive is DeKoR to key hyper-parameters? **RQ6:** How efficient is DeKoR compared with existing baselines?

5.1 Experimental setup

Datasets. In node and subgraph classification experiments, five widely used text-attributed graph datasets are employed, including Cora (Sen et al., 2008), CiteSeer (Sen et al., 2008), PubMed (Yang et al., 2016), WikiCS (Mernyei & Cangea, 2020), and arXiv (Hu et al., 2020). For details on the utilized datasets, refer to Appendix D.1 and D.2.

Baselines. The baseline models include four typical GNNs: GCN (Kipf & Welling, 2016), GAT (Veličković et al., 2017), GraphSAGE (Hamilton et al., 2017), and GIN (Xu et al., 2018); four graph Transformer models (GTs): DIFFFormer (Wu et al., 2023), NodeFormer (Wu et al., 2022), DUALFormer (Zhuo et al., 2025), and CoBFormer (Xing et al., 2024); and five GNN–LLM methods: UniGraph (He et al., 2025), OFA (Liu et al., 2024), GraphGPT (Tang et al., 2024a), LinguGKD (Hu et al., 2025), and GLANCE (Loveland et al., 2026). Appendix D.3 provides detailed descriptions of the compared baselines.

Table 2: Accuracy(%) of the node classification task across five datasets. Best and runner-up methods are **bolded** and underlined, respectively.

Category	Method	Cora	CiteSeer	PubMed	WikiCS	arXiv
GNNs	GCN *ICLR'17*	87.40 ± 0.96	74.95 ± 1.38	88.70 ± 0.35	82.25 ± 0.36	66.57 ± 0.45
	GAT *ICLR'18*	87.99 ± 0.47	75.83 ± 0.96	87.51 ± 0.77	82.71 ± 0.67	65.64 ± 0.99
	GraphSAGE *NeurIPS'17*	88.53 ± 1.26	76.87 ± 1.34	90.60 ± 0.43	84.01 ± 0.45	68.40 ± 0.63
	GIN *ICLR'19*	86.46 ± 0.87	73.65 ± 1.32	84.85 ± 1.06	61.08 ± 1.51	55.05 ± 2.70
GTs	DIFFormer *ICLR'23*	87.15 ± 1.41	77.12 ± 0.69	91.37 ± 0.27	83.29 ± 0.72	70.43 ± 0.21
	NodeFormer *NeurIPS'22*	86.71 ± 1.82	76.91 ± 1.03	91.12 ± 0.24	80.44 ± 0.29	48.25 ± 0.85
	DUALFormer *ICLR'25*	88.43 ± 0.60	74.95 ± 1.01	87.17 ± 0.88	63.24 ± 1.08	54.57 ± 0.56
	CoBFormer *ICML'24*	84.59 ± 1.91	75.07 ± 0.69	89.98 ± 1.12	79.03 ± 0.75	68.69 ± 0.38
LLM for Graphs	UniGraph *KDD'25*	82.32 ± 2.01	73.48 ± 2.01	76.59 ± 1.19	80.60 ± 0.89	51.19 ± 2.10
	OFA *ICLR'24*	87.49 ± 1.09	74.49 ± 0.31	89.66 ± 0.39	80.52 ± 0.74	67.97 ± 0.65
	GraphGPT *SIGIR'24*	81.60 ± 0.76	65.29 ± 2.43	65.94 ± 0.90	73.67 ± 1.26	54.49 ± 0.58
	LinguGKD *AAAI'25*	87.91 ± 1.23	75.70 ± 0.36	85.05 ± 0.13	82.26 ± 0.22	62.70 ± 2.65
	GLANCE *ICLR'26*	$\underline{89.50 \pm 0.40}$	$\underline{80.72 \pm 2.39}$	$\underline{92.60 \pm 0.10}$	$\mathbf{88.30 \pm 2.51}$	$\underline{71.79 \pm 1.08}$
Ours	DeKoR	$\mathbf{90.79 \pm 0.34}$	$\mathbf{81.47 \pm 1.83}$	$\mathbf{93.13 \pm 0.73}$	$\underline{88.22 \pm 1.85}$	$\mathbf{75.30 \pm 1.15}$

Table 3: Accuracy(%) of the subgraph classification task across five datasets. Best and runner-up methods are **bolded** and underlined, respectively.

Category	Method	Cora	CiteSeer	PubMed	WikiCS	arXiv
GNNs	GCN *ICLR'17*	87.30 ± 0.59	74.74 ± 0.84	84.14 ± 0.75	79.37 ± 0.93	67.67 ± 0.49
	GAT *ICLR'18*	86.95 ± 0.45	74.24 ± 1.47	84.08 ± 0.92	79.06 ± 0.85	67.75 ± 0.56
	GraphSAGE *NeurIPS'17*	87.25 ± 0.60	74.99 ± 1.17	83.77 ± 0.68	79.49 ± 0.55	68.20 ± 0.42
	GIN *ICLR'19*	86.46 ± 1.07	74.49 ± 0.69	84.17 ± 0.38	78.23 ± 0.98	66.20 ± 0.31
GTs	DIFFormer *ICLR'23*	87.30 ± 0.77	75.74 ± 1.70	84.10 ± 0.49	79.36 ± 0.72	67.82 ± 0.61
	NodeFormer *NeurIPS'22*	87.54 ± 0.39	73.99 ± 0.92	83.76 ± 1.06	75.47 ± 1.93	66.33 ± 0.20
	DUALFormer *ICLR'25*	87.44 ± 0.53	75.83 ± 0.97	84.35 ± 1.01	79.54 ± 0.82	68.81 ± 0.44
	CoBFormer *ICML'24*	86.46 ± 0.84	75.45 ± 1.14	83.96 ± 1.24	78.53 ± 1.17	66.23 ± 0.84
LLM for Graphs	UniGraph *KDD'25*	80.40 ± 0.87	71.94 ± 1.27	75.02 ± 0.81	72.56 ± 1.15	46.17 ± 3.17
	OFA *ICLR'24*	86.76 ± 1.26	75.41 ± 0.98	$\mathbf{89.61 \pm 0.36}$	$\underline{79.70 \pm 0.93}$	67.95 ± 0.73
	GraphGPT *SIGIR'24*	52.98 ± 1.96	42.49 ± 1.09	75.97 ± 0.69	44.77 ± 1.27	52.86 ± 2.49
	LinguGKD *AAAI'25*	$\underline{87.64 \pm 0.37}$	77.08 ± 1.45	83.52 ± 0.41	78.94 ± 1.50	63.92 ± 0.52
	GLANCE *ICLR'26*	87.10 ± 1.03	$\underline{77.24 \pm 1.76}$	84.69 ± 1.16	79.30 ± 1.06	$\underline{69.42 \pm 0.46}$
Ours	DeKoR	$\mathbf{88.23 \pm 0.51}$	$\mathbf{78.00 \pm 1.64}$	$\underline{85.11 \pm 0.77}$	$\mathbf{80.23 \pm 0.88}$	$\mathbf{70.10 \pm 0.10}$

5.2 Experimental results

Node Classification (RQ1). Tab. 2 reports the node classification results. DeKoR achieves the highest accuracy on four of the five datasets and achieves competitive performance with the best-performing baseline on WikiCS, demonstrating the effectiveness of our method. DeKoR achieves an average accuracy of 85.78% across the five datasets, exceeding the strongest baseline, GLANCE (84.58%), by 1.20%. The largest improvement occurs on arXiv, where DeKoR achieves 75.30% accuracy, 3.51% above GLANCE. These results show that DeKoR performs consistently across the evaluated datasets, including the large-scale datasets.

Subgraph Classification (RQ1). To further evaluate the effectiveness of our proposed method, we conduct subgraph classification experiments. The subgraph construction is described in Appendix D.2. As shown in Tab. 3, DeKoR outperforms all baseline models on four of the five datasets, achieving the best accuracy on Cora, CiteSeer, WikiCS, and arXiv. In particular, it achieves 78.00% on CiteSeer, outperforming the runner-up by 0.76% points. On PubMed, OFA achieves the highest accuracy of 89.61%, while DeKoR reaches 85.11%. This may be attributed to OFA's unified task formulation, which explicitly incorporates graph-level tasks into its model design. Overall, these results demonstrate the effectiveness of DeKoR on the subgraph classification task.

Table 4: Accuracy(%) of DeKoR with different GNN and LLM backbones on node classification.

(a) GNN Backbones

Backbone	Avg. Accuracy
GCN	82.09 ± 1.25
GraphSAGE	78.02 ± 4.53
GCNII	82.86 ± 1.22
GCN (Ours)	83.99 ± 1.46
GraphSAGE (Ours)	85.97 ± 2.05
GCNII (Ours)	85.78 ± 1.18

(b) LLM Backbones

Backbone	Cora	CiteSeer	WikiCS
Llama-8B	90.00 ± 0.98	78.77 ± 4.53	85.95 ± 1.93
Vicuna-7B	88.58 ± 2.36	79.69 ± 6.03	84.69 ± 2.06
Qwen-8B	86.56 ± 4.23	79.17 ± 3.54	82.21 ± 1.73
Llama-8B (Ours)	90.59 ± 0.68	80.97 ± 2.94	87.99 ± 2.06
Vicuna-7B (Ours)	89.91 ± 0.74	80.89 ± 2.82	87.92 ± 1.91
Qwen-8B (Ours)	90.79 ± 0.34	81.47 ± 1.83	88.22 ± 1.85

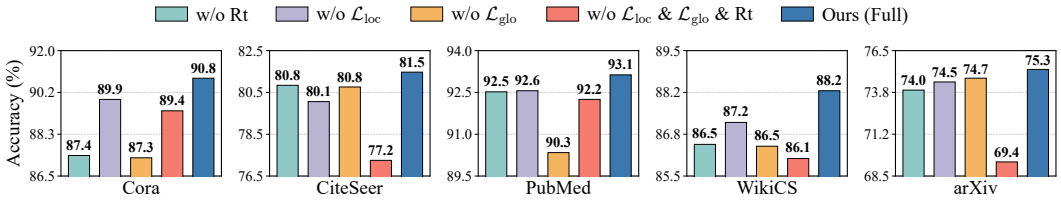


Figure 3: Ablation studies of key components in DeKoR.

Backbone Analysis (RQ2). As shown in Tab. 4, we evaluate DeKoR across different GNN and LLM backbones. In Tab. 4(a), we replace the GNN backbone and report the average accuracy across five datasets. Compared with directly using the GNN backbone for classification, DeKoR increases the average accuracy of GCN, GraphSAGE, and GCNII by 1.90%, 7.95%, and 2.92%, respectively. Moreover, the performance of DeKoR remains stable across different GNN backbones, ranging from 83.99% to 85.97%. In Tab. 4(b), DeKoR improves accuracy over each corresponding LLM-only baseline on all three datasets. Take Qwen-8B as an example, accuracy rises from 86.56% to 90.79% on Cora, from 79.17% to 81.47% on CiteSeer, and from 82.21% to 88.22% on WikiCS. These results suggest that DeKoR effectively coordinates the complementary knowledge from GNNs and LLMs while maintaining robustness to changes in backbones.

Ablation Study (RQ3). To evaluate the contribution of key components of DeKoR, we conduct an ablation study. We design four variants: (1) w/o Rt, which removes the routing mechanism; (2) w/o $\mathcal{L}_{\text{loc}}$, which removes the local distillation loss; (3) w/o $\mathcal{L}_{\text{glo}}$, which removes the global distillation loss; and (4) w/o $\mathcal{L}_{\text{loc}}$ & $\mathcal{L}_{\text{glo}}$ & Rt, which removes both distillation losses and the routing mechanism. As shown in Fig. 3, removing any component leads to a performance drop across all datasets, demonstrating their contributions to DeKoR. Specifically, the average accuracy is reduced by 1.54%, 0.92% and 1.86% by removing the router, the local distillation loss or the global distillation loss. Removing these three components will produce the largest average decline of 2.92%.

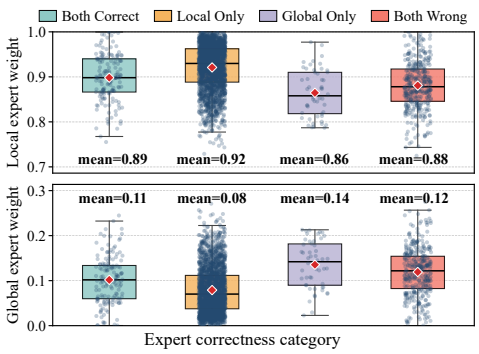


Figure 4: Routing weights of local and global semantic experts grouped by expert classification correctness on WikiCS.

Routing Analysis (RQ4). We further analyze the routing behavior of the proposed method. We partition the nodes into four groups based on the classification correctness. Fig. 4 shows the distributions of the routing weights for these groups on WikiCS, where individual points represent nodes, boxes indicate the interquartile ranges, and red diamonds denote the mean weights. The local semantic expert receives a higher routing weight than the global semantic expert across all groups, suggesting that local semantic knowledge plays a more prominent role on WikiCS. Moreover, the router favors the local semantic expert in the *Local Only* group but gives more weight to the global semantic expert in the *Global Only* group. This suggests that the router assigns higher weights to experts with correct predictions.

Table 5: Inference time (ms) and GPU memory usage (MiB) across five datasets.

Method	Time/Memory	Cora	CiteSeer	PubMed	WikiCS	arXiv	Avg. Accuracy(%)
UniGraph	Time	1363594.3	680603.5	775438.3	1172503.5	2426704.3	72.84
	Memory	14363	14368	14338	14367	14596	
OFA	Time	55285.8	54932.2	69428.5	70333.5	65201.4	80.03
	Memory	13027	13027	13028	13026	13027	
GraphGPT	Time	768910.8	495775.1	471579.6	1548980.4	5966955.7	68.20
	Memory	15351	15359	16213	16253	15821	
LinguGKD	Time	3.5	2.4	4.4	2.3	1.7	78.72
	Memory	163	226	123	145	34	
GLANCE	Time	21433.0	20641.1	21709.5	271763.1	27176.3	<u>84.58</u>
	Memory	22911	22911	22911	22911	22912	
DeKoR	Time	1041.4	854.1	801.0	965.8	810.0	**85.78**
	Memory	60	60	60	60	62	

Hyper-parameter Sensitivity Analysis (RQ5). As shown in Fig. 5, we investigate the sensitivity of DeKoR to the number of global semantic prototypes K. We can observe that our method maintains stable performance across different values of $K \in \{16, 32, 64, 128, 256\}$, with performance variations within 0.7% across all five datasets. This indicates low sensitivity to the number of prototypes, with $K = 32$ or $K = 16$ already providing competitive performance.

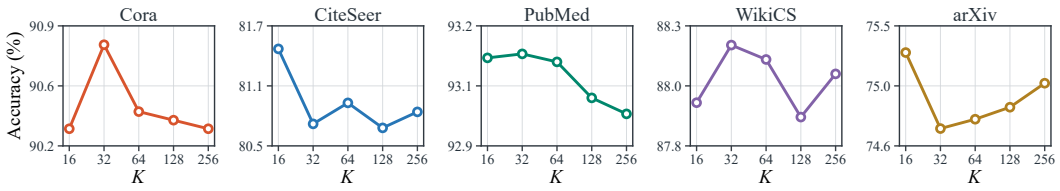


Figure 5: Sensitivity analysis of the number of prototypes K.

Efficiency Analysis (RQ6). To evaluate the time and memory efficiency of DeKoR, we compare its inference time and GPU memory usage with GNN–LLM baselines across five datasets. We randomly select 1,000 nodes from each dataset for evaluation. As shown in Tab. 5, DeKoR requires substantially less GPU memory than most GNN–LLM baselines. Moreover, its inference time is also considerably lower than that of GNN–LLM methods, except LinguGKD. Although LinguGKD achieves lower inference time, it requires more GPU memory than DeKoR on four of the five datasets and achieves 7.06% lower average accuracy. These results demonstrate that DeKoR substantially reduces the inference-time and memory overhead of GNN–LLM methods while maintaining strong predictive performance.

6 Conclusion

In this work, we study how to integrate GNN and LLM knowledge without directly coupling their mismatched knowledge spaces or relying on costly LLM inference. We propose DeKoR, which separates knowledge transfer from knowledge fusion by first reorganizing LLM knowledge before integrating it with graph models. Specifically, DeKoR decomposes LLM knowledge into complementary local and global views, distills them into lightweight experts, and uses a node-adaptive router to determine their contributions to individual predictions. This design allows useful LLM knowledge to be retained while removing the LLM from downstream inference. Extensive experiments on node and subgraph classification benchmarks show consistent improvements in both predictive performance and computational efficiency. These results demonstrate that transferring and reorganizing LLM knowledge before fusion provides a practical way to combine the complementary capabilities of GNNs and LLMs.

AI USE STATEMENT

In this work, we used generative AI tools to improve the clarity, organization, and readability of the manuscript and to support literature retrieval and discovery. All AI-assisted suggestions were manually reviewed and verified by three authors against the original experiments and cited literature. Generative AI was not used to generate experimental results or make scientific conclusions. We take full responsibility for the final content of this work.

REFERENCES

Ziwei Chai, Tianjie Zhang, Liang Wu, Kaiqiang Han, Xiaohai Hu, Xuanwen Huang, and Yang Yang. Graphllm: Boosting graph reasoning ability of large language model. *IEEE Transactions on Big Data*, 2025.

Ming Chen, Zhewei Wei, Zengfeng Huang, Bolin Ding, and Yaliang Li. Simple and deep graph convolutional networks. In *International conference on machine learning*, pp. 1725–1735. PMLR, 2020.

Runjin Chen, Tong Zhao, Ajay Jaiswal, Neil Shah, and Zhangyang Wang. Llaga: Large language and graph assistant. *arXiv preprint arXiv:2402.08170*, 2024.

Imre Csiszár. On information-type measure of difference of probability distributions and indirect observations. *Studia Sci. Math. Hungar.*, 2:299–318, 1967.

Matthias Fey and Jan Eric Lenssen. Fast graph representation learning with pytorch geometric. *arXiv preprint arXiv:1903.02428*, 2019.

Bolin Gao and Lacra Pavel. On the properties of the softmax function with application in game theory and reinforcement learning. *arXiv preprint arXiv:1704.00805*, 2017.

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Peiyi Wang, Qihao Zhu, Runxin Xu, Ruoyu Zhang, Shirong Ma, Xiao Bi, et al. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*, 2025.

Will Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. *Advances in neural information processing systems*, 30, 2017.

Xiaoxin He, Xavier Bresson, Thomas Laurent, Adam Perold, Yann LeCun, and Bryan Hooi. Harnessing explanations: Llm-to-lm interpreter for enhanced text-attributed graph representation learning. In *International conference on learning representations*, volume 2024, pp. 5711–5732, 2024a.

Xiaoxin He, Yijun Tian, Yifei Sun, Nitesh V Chawla, Thomas Laurent, Yann LeCun, Xavier Bresson, and Bryan Hooi. G-retriever: Retrieval-augmented generation for textual graph understanding and question answering. *Advances in Neural Information Processing Systems*, 37:132876–132907, 2024b.

Yufei He, Yuan Sui, Xiaoxin He, and Bryan Hooi. Unigraph: Learning a unified cross-domain foundation model for text-attributed graphs. In *Proceedings of the 31st ACM SIGKDD Conference on Knowledge Discovery and Data Mining V. 1*, pp. 448–459, 2025.

Shengxiang Hu, Guobing Zou, Song Yang, Shiyi Lin, Yanglan Gan, Bofeng Zhang, and Yixin Chen. Large language model meets graph neural network in knowledge distillation. In *Proceedings of the AAAI conference on artificial intelligence*, volume 39, pp. 17295–17304, 2025.

Weihua Hu, Matthias Fey, Marinka Zitnik, Yuxiao Dong, Hongyu Ren, Bowen Liu, Michele Catasta, and Jure Leskovec. Open graph benchmark: Datasets for machine learning on graphs. *Advances in neural information processing systems*, 33:22118–22133, 2020.

Bowen Jin, Gang Liu, Chi Han, Meng Jiang, Heng Ji, and Jiawei Han. Large language models on graphs: A comprehensive survey. *IEEE Transactions on Knowledge and Data Engineering*, 36 (12):8622–8642, 2024.

Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016.

Lecheng Kong, Jiarui Feng, Hao Liu, Chengsong Huang, Jiaxin Huang, Yixin Chen, and Muhan Zhang. Gofa: A generative one-for-all model for joint graph language modeling. In *International Conference on Learning Representations*, volume 2025, pp. 40792–40823, 2025.

Simon Kornblith, Mohammad Norouzi, Honglak Lee, and Geoffrey Hinton. Similarity of neural network representations revisited. In *International conference on machine learning*, pp. 3519–3529. PMlR, 2019.

Solomon Kullback and Richard A Leibler. On information and sufficiency. *The annals of mathematical statistics*, 22(1):79–86, 1951.

Peiyuan Li, Yongqi Huang, Jitao Zhao, Dongxiao He, Di Jin, and Weixiong Zhang. Choe: Cross-domain heterogeneous graph prompt learning via structure-conditioned experts. *arXiv preprint arXiv:2605.15888*, 2026.

Yuhan Li, Zhixun Li, Peisong Wang, Jia Li, Xiangguo Sun, Hong Cheng, and Jeffrey Xu Yu. A survey of graph meets large language model: Progress and future directions. *arXiv preprint arXiv:2311.12399*, 2023.

Hao Liu, Jiarui Feng, Lecheng Kong, Ningyue Liang, Dacheng Tao, Yixin Chen, and Muhan Zhang. One for all: Towards training one graph model for all classification tasks. In *International conference on learning representations*, volume 2024, pp. 20188–20210, 2024.

Donald Loveland, Yao-An Yang, and Danai Koutra. Glance for context: Learning when to leverage llms for node-aware gnn-llm fusion. In *International Conference on Learning Representations*, volume 2026, pp. 90402–90428, 2026.

Leland McInnes, John Healy, and James Melville. Umap: Uniform manifold approximation and projection for dimension reduction. *arXiv preprint arXiv:1802.03426*, 2018.

James B McQueen. Some methods of classification and analysis of multivariate observations. In *Proc. of 5th berkeley symposium on math. stat. and prob.*, pp. 281–297, 1967.

Péter Mernyei and Cătălina Cangea. Wiki-cs: A wikipedia-based benchmark for graph neural networks. *arXiv preprint arXiv:2007.02901*, 2020.

Bryan Perozzi, Bahare Fatemi, Dustin Zelle, Anton Tsitsulin, Mehran Kazemi, Rami Al-Rfou, and Jonathan Halcrow. Let your graph do the talking: Encoding structured data for llms. *arXiv preprint arXiv:2402.05862*, 2024.

Ben Poole, Sherjil Ozair, Aaron Van Den Oord, Alex Alemi, and George Tucker. On variational bounds of mutual information. In *International conference on machine learning*, pp. 5171–5180. PMLR, 2019.

Prithviraj Sen, Galileo Namata, Mustafa Bilgic, Lise Getoor, Brian Gallagher, and Tina Eliassi-Rad. Collective classification in network data. *AI magazine*, 29(3):93–106, 2008.

Junshu Sun, Wanxing Chang, Qingming Huang, and Shuhui Wang. Enhancing llms for graph tasks via graph-aware lora generation. *arXiv preprint arXiv:2606.22429*, 2026.

Jiabin Tang, Yuhao Yang, Wei Wei, Lei Shi, Lixin Su, Suqi Cheng, Dawei Yin, and Chao Huang. Graphgpt: Graph instruction tuning for large language models. In *Proceedings of the 47th international ACM SIGIR conference on research and development in information retrieval*, pp. 491–500, 2024a.

Jiabin Tang, Yuhao Yang, Wei Wei, Lei Shi, Long Xia, Dawei Yin, and Chao Huang. Higpt: Heterogeneous graph language model. In *Proceedings of the 30th ACM SIGKDD conference on knowledge discovery and data mining*, pp. 2842–2853, 2024b.

Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. Graph attention networks. *arXiv preprint arXiv:1710.10903*, 2017.

Duo Wang, Yuan Zuo, Fengzhi Li, and Junjie Wu. Llms as zero-shot graph learners: Alignment of gnn representations with llm token embeddings. *Advances in neural information processing systems*, 37:5950–5973, 2024.

Runze Wang, Mingqi Yang, and Yanming Shen. Graph2token: make llms understand molecule graphs. In *ICML 2024 Workshop on Efficient and Accessible Foundation Models for Biological Discovery*.

Qitian Wu, Wentao Zhao, Zenan Li, David P Wipf, and Junchi Yan. Nodeformer: A scalable graph structure learning transformer for node classification. *Advances in neural information processing systems*, 35:27387–27401, 2022.

Qitian Wu, Chenxiao Yang, Wentao Zhao, Yixuan He, David Wipf, and Junchi Yan. Difformer: Scalable (graph) transformers induced by energy constrained diffusion. *arXiv preprint arXiv:2301.09474*, 2023.

Lianghao Xia, Ben Kao, and Chao Huang. Opengraph: Towards open graph foundation models. In *Findings of the Association for Computational Linguistics: EMNLP 2024*, pp. 2365–2379, 2024.

Yujie Xing, Xiao Wang, Yibo Li, Hai Huang, and Chuan Shi. Less is more: on the over-globalizing problem in graph transformers. *arXiv preprint arXiv:2405.01102*, 2024.

Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? *arXiv preprint arXiv:1810.00826*, 2018.

Hao Yan, Chaozhuo Li, Ruosong Long, Chao Yan, Jianan Zhao, Wenwen Zhuang, Jun Yin, Peiyan Zhang, Weihao Han, Hao Sun, et al. A comprehensive study on text-attributed graphs: Benchmarking and rethinking. *Advances in Neural Information Processing Systems*, 36:17238–17264, 2023.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, et al. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025.

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

Xianlin Zeng, Fan Xia, and Xiangyu Chen. Eralign: Energy-based representation alignment of gnns and llms on text-attributed graphs. *arXiv preprint arXiv:2606.10461*, 2026.

Jiawei Zhang. Graph-toolformer: To empower llms with graph reasoning ability via prompt augmented by chatgpt. *arXiv preprint arXiv:2304.11116*, 2023.

Qifang Zhao, Weidong Ren, Tianyu Li, Hong Liu, Xingsheng He, and Xiaoxiao Xu. Graphgpt: generative pre-trained graph eulerian transformer. *arXiv preprint arXiv:2401.00529*, 2023.

Chuang Zhou, Zhu Wang, Shengyuan Chen, Jiahe Du, Qiyuan Zheng, Zhaozhuo Xu, and Xiao Huang. Taming language models for text-attributed graph learning with decoupled aggregation. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 3463–3474, 2025.

Jiaming Zhuo, Yuwei Liu, Yintong Lu, Ziyi Ma, Kun Fu, Chuan Wang, Yuanfang Guo, Zhen Wang, Xiaochun Cao, and Liang Yang. Dualformer: Dual graph transformer. In *The Thirteenth International Conference on Learning Representations*, 2025.

Aaron Zolnai-Lucas, Jack Boylan, Chris Hokamp, and Parsa Ghaffari. Stage: Simplified text-attributed graph embeddings using pre-trained llms. In *Proceedings of the 1st Workshop on Knowledge Graphs and Large Language Models (KaLLM 2024)*, pp. 92–104, 2024.

A Proofs of Theoretical Results

A.1 Proof of Proposition 1

Proof. Let $\mathbf{z} \in \mathbb{R}^d$ denote a node's semantic representation and let $\mathbf{C} \in \mathbb{R}^{K\times d}$ contain the K semantic prototypes as its rows. For notational convenience, we treat $\mathbf{z}$ as a column vector and define

$$\mathbf{r} = \frac{\mathbf{C}\mathbf{z}}{\tau}, \qquad \mathbf{q} = \mathrm{softmax}(\mathbf{r}), \tag{20}$$

where $\mathbf{q} \in \mathbb{R}^K$ satisfies $q_k \geq 0$ and $\sum_{k=1}^K q_k = 1$.

To examine the local behavior of the prototype-based mapping, we consider the Jacobian of the softmax function with respect to $\mathbf{r}$

$$\frac{\partial \mathbf{q}}{\partial \mathbf{r}} = \mathrm{Diag}(\mathbf{q}) - \mathbf{q}\mathbf{q}^\top. \tag{21}$$

Since $\mathbf{r} = \mathbf{C}\mathbf{z}/\tau$, the chain rule gives

$$\mathbf{J}_\mathbf{q}(\mathbf{z}) = \frac{1}{\tau}\left(\mathrm{Diag}(\mathbf{q}) - \mathbf{q}\mathbf{q}^\top\right)\mathbf{C}. \tag{22}$$

Because $\mathbf{q}^\top\mathbf{1} = 1$, we have

$$\left(\mathrm{Diag}(\mathbf{q}) - \mathbf{q}\mathbf{q}^\top\right)\mathbf{1} = \mathbf{q} - \mathbf{q}\left(\mathbf{q}^\top\mathbf{1}\right) = \mathbf{0}. \tag{23}$$

Therefore, $\mathrm{Diag}(\mathbf{q}) - \mathbf{q}\mathbf{q}^\top$ has rank at most $K-1$. Using $\mathrm{rank}(\mathbf{AB}) \leq \min\{\mathrm{rank}(\mathbf{A}), \mathrm{rank}(\mathbf{B})\}$, Eq. equation 22 yields

$$\mathrm{rank}\left(\mathbf{J}_\mathbf{q}(\mathbf{z})\right) \leq \min\left\{K-1, \mathrm{rank}(\mathbf{C})\right\}. \tag{24}$$

Hence, the prototype-based representation is sensitive to a restricted set of directions defined by the prototype geometry, rather than preserving every variation in the original representation $\mathbf{z}$.

We next establish the invariance property. Consider an arbitrary perturbation $\boldsymbol{\delta} \in \mathbb{R}^d$ satisfying $\mathbf{C}\boldsymbol{\delta} = a\mathbf{1}$ for some scalar a. The prototype logits of the perturbed representation become

$$\frac{\mathbf{C}(\mathbf{z}+\boldsymbol{\delta})}{\tau} = \frac{\mathbf{C}\mathbf{z}}{\tau} + \frac{a}{\tau}\mathbf{1}. \tag{25}$$

The softmax function is invariant to adding the same scalar to all logits, i.e.,

$$\mathrm{softmax}\left(\mathbf{x} + b\mathbf{1}\right) = \mathrm{softmax}(\mathbf{x}) \tag{26}$$

for any vector $\mathbf{x}$ and scalar b. Therefore,

$$\begin{aligned}\mathbf{q}(\mathbf{z}+\boldsymbol{\delta}) &= \mathrm{softmax}\left(\frac{\mathbf{C}(\mathbf{z}+\boldsymbol{\delta})}{\tau}\right)\\ &= \mathrm{softmax}\left(\frac{\mathbf{C}\mathbf{z}}{\tau} + \frac{a}{\tau}\mathbf{1}\right)\\ &= \mathrm{softmax}\left(\frac{\mathbf{C}\mathbf{z}}{\tau}\right) = \mathbf{q}(\mathbf{z}).\end{aligned} \tag{27}$$

This invariance shows that the prototype-induced representation discards variations that affect all prototype similarities equally, while retaining variations that modify their relative relationships. Consequently, the original representation and the prototype-based representation characterize semantic knowledge at different granularities: the former preserves fine-grained node-specific semantics, whereas the latter emphasizes prototype-relative global semantic structure. This completes the proof. □

A.2 Proof of Proposition 2

Proof. We prove the information-preservation properties of the local and global knowledge separately.

Local semantic view. Let the random variables $\mathbf{H}^{\mathrm{loc}}$ and $\mathbf{Z}$ denote the local semantic representation produced by the lightweight modules and the corresponding LLM teacher representation, respectively. Consider a batch $\mathcal{B}$ of $|\mathcal{B}|$ positive pairs $(\mathbf{h}_v^{\mathrm{loc}}, \mathbf{z}_v)$ sampled from their joint distribution $p(\mathbf{H}^{\mathrm{loc}}, \mathbf{Z})$. For each anchor $\mathbf{h}_v^{\mathrm{loc}}$, the representations $\mathbf{z}_u$, $u \neq v$, are treated as negative samples drawn from the marginal distribution of $\mathbf{Z}$.

Using the similarity critic

$$s(\mathbf{h}^{\mathrm{loc}}, \mathbf{z}) = \frac{\cos(\mathbf{h}^{\mathrm{loc}}, \mathbf{z})}{\tau_{\mathrm{loc}}}, \tag{28}$$

the directional InfoNCE objective is

$$\mathcal{L}_{\mathrm{h}\to\mathrm{z}} = -\mathbb{E}\left[\log \frac{\exp s(\mathbf{h}_v^{\mathrm{loc}}, \mathbf{z}_v)}{\sum_{u\in\mathcal{B}} \exp s(\mathbf{h}_v^{\mathrm{loc}}, \mathbf{z}_u)}\right]. \tag{29}$$

Under the standard InfoNCE sampling construction, the InfoNCE lower bound on mutual information gives

$$I\big(\mathbf{H}^{\mathrm{loc}}; \mathbf{Z}\big) \geq \log|\mathcal{B}| - \mathcal{L}_{\mathrm{h}\to\mathrm{z}}. \tag{30}$$

By exchanging the roles of $\mathbf{H}^{\mathrm{loc}}$ and $\mathbf{Z}$, the reverse-direction objective similarly satisfies

$$I\big(\mathbf{Z}; \mathbf{H}^{\mathrm{loc}}\big) \geq \log|\mathcal{B}| - \mathcal{L}_{\mathrm{z}\to\mathrm{h}}. \tag{31}$$

Since mutual information is symmetric,

$$I\big(\mathbf{H}^{\mathrm{loc}}; \mathbf{Z}\big) = I\big(\mathbf{Z}; \mathbf{H}^{\mathrm{loc}}\big). \tag{32}$$

Therefore, Eqs. equation 30 and equation 31 imply

$$\begin{aligned} I\big(\mathbf{H}^{\mathrm{loc}}; \mathbf{Z}\big) \geq \max\{&\log|\mathcal{B}| - \mathcal{L}_{\mathrm{h}\to\mathrm{z}}, \\ &\log|\mathcal{B}| - \mathcal{L}_{\mathrm{z}\to\mathrm{h}}\}. \end{aligned} \tag{33}$$

Since the maximum of two values is no smaller than their average, we further obtain

$$\begin{aligned} I\big(\mathbf{H}^{\mathrm{loc}}; \mathbf{Z}\big) &\geq \log|\mathcal{B}| - \frac{1}{2}\left(\mathcal{L}_{\mathrm{h}\to\mathrm{z}} + \mathcal{L}_{\mathrm{z}\to\mathrm{h}}\right) \\ &= \log|\mathcal{B}| - \mathcal{L}_{\mathrm{nce}}. \end{aligned} \tag{34}$$

Hence, minimizing the symmetric InfoNCE objective increases its corresponding lower bound on the mutual information shared between the local semantic expert and the LLM representation.

Global semantic view. For node v, let $\mathbf{q}_v$ denote the prototype-association distribution from the LLM teacher representation, and define the distribution predicted by the global semantic module as

$$\mathbf{p}_v^{\mathrm{glo}} = \mathrm{softmax}\left(\frac{\mathbf{h}_v^{\mathrm{glo}}}{\tau_{\mathrm{glo}}}\right). \tag{35}$$

Both $\mathbf{q}_v$ and $\mathbf{p}_v^{\mathrm{glo}}$ are probability distributions over the K semantic prototypes. The node-wise global distillation loss is

$$\mathcal{L}_{\mathrm{glo},v} = \mathrm{KL}\big(\mathbf{q}_v \,\big\|\, \mathbf{p}_v^{\mathrm{glo}}\big). \tag{36}$$

By Pinsker's inequality, the total variation distance between two probability distributions satisfies

$$D_{\mathrm{TV}}\left(\mathbf{q}_v, \mathbf{p}_v^{\mathrm{glo}}\right) \leq \sqrt{\frac{1}{2}\mathrm{KL}\Big(\mathbf{q}_v \,\Big\|\, \mathbf{p}_v^{\mathrm{glo}}\Big)}. \tag{37}$$

Using

$$D_{\mathrm{TV}}\left(\mathbf{q}_v, \mathbf{p}_v^{\mathrm{glo}}\right) = \frac{1}{2}\left\|\mathbf{q}_v - \mathbf{p}_v^{\mathrm{glo}}\right\|_1, \tag{38}$$

we obtain

$$\left\|\mathbf{q}_v - \mathbf{p}_v^{\mathrm{glo}}\right\|_1 \le \sqrt{2\mathrm{KL}\Big(\mathbf{q}_v \,\Big\|\, \mathbf{p}_v^{\mathrm{glo}}\Big)} = \sqrt{2\mathcal{L}_{\mathrm{glo},v}}. \tag{39}$$

Therefore, decreasing the KL distillation loss directly tightens an upper bound on the discrepancy between the prototype-association distribution from global semantic knowledge and the distribution reconstructed by the lightweight global expert. In particular, as $\mathcal{L}_{\mathrm{glo},v} \to 0$, we have

$$\left\|\mathbf{q}_v - \mathbf{p}_v^{\mathrm{glo}}\right\|_1 \to 0. \tag{40}$$

Together, Eq. equation 34 and Eq. equation 39 show that the two distillation objectives provide complementary preservation guarantees for the decomposed knowledge: the local objective preserves shared semantic information with the original LLM representation, whereas the global objective preserves the prototype-based semantic distribution. This completes the proof. □

B Algorithm Description

The overall procedure of DeKoR is summarized in Algorithm 1.

Algorithm 1: DeKoR

Input : Text-attributed graph $\mathcal{G}$; pre-trained f_{GNN} and LLM; number of prototypes K
Output : Predictions $\hat{\mathbf{y}}$
Obtain structural representations $\mathbf{H}$ using f_{GNN}
Construct prompts and obtain LLM representations $\mathbf{Z}$ as in equation 3–equation 4
Cluster $\mathbf{Z}$ into K semantic prototypes and construct global knowledge $\mathbf{Q}$ as in equation 7–equation 8
Initialize semantic experts f_{loc} and f_{glo}
while *distillation not converged* **do**
  Compute $\mathbf{H}^{\mathrm{loc}}$ and $\mathbf{H}^{\mathrm{glo}}$ as in equation 11
  Compute local and global distillation losses as in equation 12–equation 15
  Update f_{loc} and f_{glo} with $\mathcal{L}_{\mathrm{distill}}$
end
Freeze f_{loc} and f_{glo}
while *routing not converged* **do**
  Obtain expert predictions as in equation 16
  Compute node-adaptive routing weights and fused predictions as in equation 17–equation 18
  Compute $\mathcal{L}_{\mathrm{route}}$ with the balance loss in equation 19
  Update ϕ, g^{loc}, and g^{glo}
end
return $\hat{\mathbf{y}}$

C Large Language Model Prompting

We adopt the prompt design of Loveland et al. (2026), which constructs prompts with different neighborhood ranges for each node:

- **ego prompt**: contains only the textual description of the target node.
- **one-hop prompt**: contains the textual descriptions of the target node and its sampled one-hop neighbors.
- **two-hop prompt**: contains the textual descriptions of the target node and its sampled two-hop neighbors, excluding the one-hop neighbors.

Each prompt begins with a node classification instruction and candidate categories. The target-node and neighborhood texts are then organized into labeled fields. The category names provide information about the prediction space, while the field labels distinguish the target node from its structural context. The three prompt templates are shown below:

Ego Prompt example:

```
Instruct: Predict the node's category from the provided
ego-node context. Possible categories: <candidate categories>.
Query:
EGO:
<target-node text>
Category?
</END>
```

One-hop Prompt example:

```
Instruct: Predict the node's category from the provided
1-hop context. Possible categories: <candidate categories>.
Query:
EGO:
<target-node text>
HOP1:
- <one-hop neighbor 1 text>
- <one-hop neighbor 2 text>
...
Category?
</END>
```

Two-hop Prompt example:

```
Instruct: Predict the node's category from the provided
2-hop context. Possible categories: <candidate categories>.
Query:
EGO:
<target-node text>
HOP2:
- <two-hop neighbor 1 text>
- <two-hop neighbor 2 text>
...
Category?
</END>
```

D Experimental Details

D.1 Introduction of Datasets

To characterize the limitations of current methods, as well as evaluate DeKoR, we use five node classification datasets: Cora (Sen et al., 2008), CiteSeer (Sen et al., 2008), PubMed (Yang et al., 2016), WikiCS (Mernyei & Cangea, 2020), and arXiv (Hu et al., 2020). Table 6 presents the statistical information for the five graph datasets.

- **Cora.** Cora is a citation network of scientific publications in the field of machine learning. Each node is a paper, and an edge indicates a citation relationship between two papers. Each paper is described by a bag-of-words feature vector. The nodes are categorized into 7 different AI fields.
- **CiteSeer.** CiteSeer is a citation network containing scientific publications from multiple research areas. Nodes correspond to publications, while edges represent citation relationships between papers. Each node is associated with a bag-of-words feature vector derived from its text content. The node classification task involves 6 research categories.
- **PubMed.** PubMed is a larger citation network of biomedical publications, where papers have an edge if they have a citation relationship. Each article is represented by a TF–IDF word vector, with labels corresponding to 3 different disease types.

- **WikiCS.** WikiCS is a text-attributed graph constructed from Wikipedia computer science articles. Each node corresponds to a computer science article, and edges capture relationships between related articles. Each node is represented by a 300-dimensional feature vector, and the nodes are categorized into 10 different computer science topics.
- **arXiv.** arXiv is a large-scale citation network of computer science papers collected from arXiv. Nodes represent scientific papers, and edges are citation relationships between papers. Each paper is associated with a 128-dimensional feature vector, and the node classification task covers 40 research categories.

Table 6: Summary statistics of the datasets used in our experiments.

Dataset	#Nodes	#Edges	Original Feature Dim.	#Classes
Cora	2,708	5,429	1,433	7
CiteSeer	3,186	4,277	3,703	6
PubMed	19,717	44,335	500	3
WikiCS	11,701	215,863	300	10
arXiv	169,343	1,166,243	128	40

D.2 Construction of Subgraph Classification Datasets

To evaluate our method on graph-level tasks, we transform each node classification dataset into a subgraph classification dataset. Given a text-attributed graph, we construct a 2-hop subgraph centered at each node and use the center node's label as the subgraph label. For each center node v_i, we construct an adjacency list from the graph structure, symmetrize the adjacency list and remove self-loops. We then randomly sample at most five 1-hop neighbors of v_i. For each selected 1-hop neighbor, we further sample at most five candidate neighbors to form the 2-hop expansion set. During this process, the center node and selected 1-hop neighbors are excluded, and duplicate 2-hop nodes reached through different intermediate neighbors are removed. The 2-hop expansion set therefore contains at most $5 \times 5 = 25$ nodes, resulting in at most $1 + 5 + 25 = 31$ nodes in each sampled subgraph. Formally, the node set of the subgraph centered at v_i is defined as:

$$\mathcal{V}_i^{\text{sub}} = \{v_i\} \cup \mathcal{N}_i^{(1)} \cup \mathcal{N}_i^{(2)}, \tag{41}$$

where $\mathcal{N}_i^{(1)}$ and $\mathcal{N}_i^{(2)}$ denote the sampled one-hop and two-hop neighbors. The subgraph edge set contains the original edges whose endpoints are both in $\mathcal{V}_i^{\text{sub}}$. The nodes in each subgraph are relabeled with local indices, with the center node assigned index zero. The corresponding node features and labels are extracted from the original graph, and each node is additionally assigned a hop type of 0, 1, or 2 according to its distance from the center. We also retain the original node IDs and raw texts to align the subgraph nodes with the LLM teacher. Using a fixed random seed, we construct and store the resulting subgraphs as PyTorch Geometric `Data` objects.

D.3 Introduction of Baselines

Graph Neural Networks (GNNs). The following outlines the GNN baselines used for comparison.

- GCN (Kipf & Welling, 2016): GCN applies graph convolution to aggregate information from neighboring nodes, combining node features with local graph structure to generate node representations.
- GAT (Veličković et al., 2017): GAT introduces an attention mechanism into graph convolution. It assigns different weights to neighboring nodes when aggregating their information.
- GraphSAGE (Hamilton et al., 2017): GraphSAGE generates node representations by sampling neighboring nodes and aggregating their features with a learnable aggregation function. It enables scalable graph representation learning.

- GIN (Xu et al., 2018): GIN combines a node's own features with aggregated neighbor information through a learnable multilayer perceptron to learn expressive node representations.

Graph Transformers (GTs). We next introduce the GT baselines used for comparison.

- DIFFormer (Wu et al., 2023): This GT architecture employs a diffusion-based attention mechanism to efficiently capture global dependencies between nodes.
- NodeFormer (Wu et al., 2022): This GT architecture employs a kernelized Gumbel-softmax operator to efficiently learn graph structures in a differentiable manner.
- DUALFormer (Zhuo et al., 2025): This GT architecture employs a dual-dimensional design to capture local inter-node dependencies and global inter-feature correlations efficiently.
- CoBFormer (Xing et al., 2024): This GT architecture employs self-attention mechanisms both within and across clusters to mitigate the over-globalization problem.

GNN–LLM Fusion Methods. We next introduce the GNN–LLM fusion baselines used for comparison.

- UniGraph (He et al., 2025): This framework builds a cross-domain graph foundation model by combining language models with GNNs, using masked graph modeling for pre-training and graph instruction tuning to support few-shot and zero-shot transfer.
- OFA (Liu et al., 2024): This framework unifies cross-domain graph data as text-attributed graphs, encodes node, edge, and task descriptions with language models, and uses nodes-of-interest and graph prompting to enable a single GNN to handle classification tasks.
- GraphGPT (Tang et al., 2024a): This framework integrates graph structural knowledge into LLMs through text-graph grounding, a lightweight graph-text alignment projector, and dual-stage graph instruction tuning.
- LinguGKD (Hu et al., 2025): This framework uses instruction tuning to train an LLM teacher and adopts layer-adaptive contrastive knowledge distillation to transfer semantic and relational knowledge from the LLM to a lightweight GNN.
- GLANCE (Loveland et al., 2026): This framework employs a lightweight router to selectively invoke the LLM for nodes where the GNN is likely to underperform, and then fuses the LLM embeddings with GNN representations for classification through a refiner.

For the four GNN baselines, i.e., GCN, GAT, GraphSAGE, and GIN, we use the public library PyTorch Geometric (PyG) (Fey & Lenssen, 2019) for their implementation. For other baselines, we leverage their source codes. The sources are outlined as:

- GNNs: `https://github.com/pyg-team/pytorch_geometric/tree/master/torch_geometric/nn/conv`
- DIFFormer: `https://github.com/qitianwu/DIFFormer`
- NodeFormer: `https://github.com/qitianwu/NodeFormer`
- DUALFormer: `https://github.com/JiamingZhuo/DUALFormer`
- CoBFormer: `https://github.com/null-xyj/CoBFormer`
- UniGraph: `https://github.com/yf-he/UniGraph`
- OFA: `https://github.com/LechengKong/OneForAll`
- GraphGPT: `https://github.com/HKUDS/GraphGPT`
- LinguGKD: `https://github.com/SXiangHu/LinguGKD`
- GLANCE: `https://github.com/dloveland/GLANCE-ICLR26`

For all node classification and subgraph classification experiments, we preserve the original design of each baseline as much as possible. Specifically, we retain the original model architecture, graph pre-processing strategy, and training objective. For GNN–LLM methods, we also preserve the original prompt templates.

For subgraph classification experiments that require adapting a node-level method to graph-level prediction, we append a global mean-pooling readout after the final representation layer to obtain

the graph-level representation. This readout operation is applied only at the final stage and does not modify the baseline backbone. For baselines that already provide a graph-level readout, we retain their original prediction mechanism.

D.4 Pretrain the GNN Encoder

We pre-train a graph neural network encoder for each dataset to capture structural knowledge, using GCNII (Chen et al., 2020) as the backbone. The GNN uses a hidden dimension of 64 and a dropout probability of 0.2. For data splitting, we use an independent random seed of 123 to partition the nodes into training, validation, and test sets with a ratio of 50%/25%/25%, respectively. The GNN trainer uses at most 50,000 training nodes and 50,000 validation nodes. We optimize the GNN parameters using AdamW with a learning rate of 1×10^{-3} and a weight decay of 1×10^{-4}. The maximum number of training epochs is set to 200, with an early-stopping patience of 50 epochs.

D.5 Fine-tuning the LLM

We separately fine-tune a pre-trained LLM for each dataset to produce semantic representations. We adopt Qwen-8B with LoRA-based parameter-efficient fine-tuning, where the LoRA rank is set to 8. The LLM is optimized using the AdamW optimizer with a learning rate of 1×10^{-4}. The maximum number of training epochs is set to 10, with an early-stopping patience of 2 epochs. The LLM trainer follows the same deterministic data-splitting scheme as the GNN trainer, partitioning the nodes into training, validation, and test sets with a ratio of 50%/25%/25%, respectively, using a default random seed of 123. To reduce the computational cost of fine-tuning, we limit the training and validation sets to at most 3,000 and 1,000 nodes, respectively.

The LLM is trained using three prompts described in Appendix C: the ego prompt, one-hop prompt, and two-hop prompt. We use left padding and left truncation during tokenization. The maximum sequence length is 1,024 tokens for the ego prompt and 4,096 tokens for one-hop and two-hop prompts. For each prompt, the hidden state corresponding to the last non-padding token is used as the pooled representation. We concatenate the representations from the three prompts and use an MLP to map the embeddings to the label space for each dataset. After fine-tuning, we remove the MLP head and use the fine-tuned LLM to generate semantic representations for our method.

D.6 Node and Subgraph Classification Experiments

For the final node and subgraph classification experiments, we use a 50%/25%/25% split for the training, validation, and test sets, respectively. To reduce the computational cost on large datasets, we limit the numbers of training, validation, and test nodes to 3,000, 1,000, and 10,000, respectively. For the node and subgraph classification baselines, we preserve the original hyperparameter settings and only tune the learning rate over $0.01, 0.001, 0.0001, 0.00001$. For our method, in addition to tuning the learning rates of the expert distillation and routing, we search $\lambda_{\text{loc}}, \lambda_{\text{glo}}$ and $\tau_{\text{loc}}, \tau_{\text{glo}}$ over $0.1, 1, 10, 100$, and the number of prototypes K over $8, 16, 32, 64$. All experiments are conducted with three random seeds, 123, 456, and 789, and we report the mean accuracy and standard deviation over three runs.