HopRank: Self-Supervised LLM Preference-Tuning on Graphs for Few-Shot Node Classification

Anonymous ACL submission

Abstract

Large language models (LLMs) show great promise for text-attributed graphs (TAGs), yet current methods require abundant labeled data while underexploring the structural information in graph topology. We address this by leveraging the homophily principle to design a self-supervised approach. Our key insight is that training LLMs to predict connections implicitly teaches them class structure, since edges predominantly connect similar nodes. Based on this insight, we present HopRank, which trains LLMs through self-supervised learning on graph structure. We construct preference data via hierarchical hop-based sampling, then employ adaptive preference learning that prioritizes informative training signals without requiring any class labels. During inference, we classify nodes by predicting which class's labeled anchors they would most likely connect to. Extensive experiments show HopRank matches GNN performance and substantially outperforms existing graph LLM methods without any labeled training data. Code is available at https://anonymous.4open.science/r/HopRank/.

1 Introduction

Large language models (LLMs) show great promise for text-attributed graphs (TAGs) due to their powerful text understanding capabilities (Jin et al., 2024; Chen et al., 2024b). Recent methods leverage various techniques—including prompting, instruction tuning, and representation alignment—to adapt LLMs for graph tasks (Ye et al., 2023; He et al., 2023; Tang et al., 2024; Sun et al., 2023; Chen et al., 2024a; Xu and Ding, 2025). However, these approaches require abundant labeled data to learn the mapping from node text to class labels, while the rich structural information already encoded in graph topology remains underexplored. These limitations hinder their applicability in scenarios where annotations are scarce.

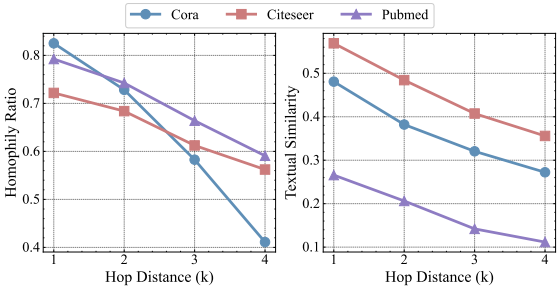

Figure 1: **Graph Topology Encodes Class Structure.** Class homophily and textual similarity are highest for directly connected nodes and decay predictably with hop distance.

To address these limitations, we leverage the homophily principle (McPherson et al., 2001)—that similar nodes tend to connect—to design a self-supervised fine-tuning approach for LLMs that requires no labeled data. Figure 1 demonstrates that both class homophily and textual similarity decay predictably with hop distance, creating a natural hierarchy: directly connected nodes (1-hop) are highly similar, while distant nodes (3-hop, 4-hop) are increasingly dissimilar. This pattern reveals that graph topology inherently encodes connection preferences without any labels. The key question is: *how can we train LLMs to capture these structural preferences?*

Our key insight is to train LLMs on link prediction: by learning to predict which nodes are most likely to connect, the model implicitly captures the similarity patterns encoded in graph structure. Since edges predominantly connect similar nodes (same class), predicting connections teaches the model about class structure without requiring any class labels during training. This learned understanding of connection patterns directly enables few-shot classification: at inference, we determine a node's class by predicting which class's labeled anchors it would most likely connect to.

Based on this insight, we introduce HopRank, which trains LLMs through self-supervised learn-

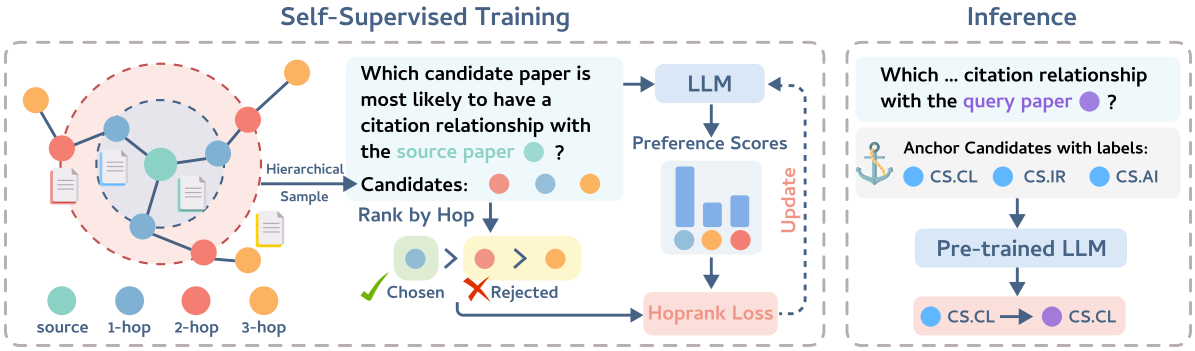

Figure 2: **Overview of the HopRank framework. Left:** Self-supervised training learns to predict connections from structural preferences using hierarchical hop-based sampling, without any labels. **Right:** Few-shot inference classifies nodes by predicting their connection preferences to labeled anchors.

ing on graph structure (Figure 2). We construct preference data through hierarchical hop-based sampling, where each pair compares the actual 1-hop neighbor (chosen) against nodes sampled from increasing hop distances (rejected). We then employ adaptive preference learning to weight these pairs by (1) hop distance, emphasizing challenging nearby distinctions, and (2) ranking difficulty, focusing on pairs where the model currently makes errors. This creates an adaptive curriculum that prioritizes informative training signals. During inference, it achieves few-shot classification using only a few labeled anchors per class. Extensive experiments demonstrate that HopRank matches GNN performance and substantially outperforms existing graph LLM methods without any labeled training data.

2 The HopRank Framework

2.1 Problem Formulation

Given a text-attributed graph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathbf{X})$ where $\mathcal{V}$ denotes nodes, $\mathcal{E}$ denotes edges, and $\mathbf{X} = \{x_1, ..., x_N\}$ represents textual attributes, we reformulate node classification as learning connection preferences. Our framework operates in two stages: **(1) Self-supervised training** learns a link prediction function $f : (\mathcal{V} \times \mathcal{V}) \to [0, 1]$ from graph topology and node texts, without requiring class labels. **(2) Few-shot inference** classifies test nodes by predicting their connection preferences to K labeled anchor nodes per class, transforming classification into a preference ranking problem.

2.2 Hierarchical Hop-based Sampling

To leverage the structural preferences encoded in graph topology, we construct training instances through hierarchical hop-based sampling. For each edge $(u, v) \in \mathcal{E}$, we create a preference learning task with the true neighbor v as the chosen response and negative candidates sampled from increasing hop distances as rejected responses. Specifically, we sample negatives from $\{2\text{-hop}, 3\text{-hop}, ...\}$ neighborhoods of node u, where 2-hop negatives are challenging (semantically similar but structurally unconnected) and distant negatives are progressively easier to distinguish. This creates a natural curriculum that focuses learning on fine-grained distinctions, as validated by the predictable decay patterns in Figure 1.

We format each instance as a natural language prompt. For example, with two negative samples:

Link Prediction Training Prompt

```
Given a source paper and multiple candidate
papers, identify which candidate paper is
most likely to have a citation relationship
with the source paper.
Source Paper: '{title_u}'
Abstract: {abstract_u}
Candidate Papers:
A. '{title_v}'
Abstract: {abstract_v} [1-hop: actual neighbor]
B. '{title_n2}'
Abstract: {abstract_n2} [2-hop: hard negative]
C. '{title_n3}'
Abstract: {abstract_n3} [3hop: easy negative]
Which candidate paper is most likely to be
cited by or cite the source paper?
```

2.3 Adaptive Preference Learning

After hierarchical sampling, we obtain a listwise preference dataset where each instance contains one chosen response (1-hop neighbor) and multiple rejected responses at varying distances. We train the LLM to predict these connections by learning which pairs are most informative.

Not all preference pairs are equally informative for learning. Distinguishing a 1-hop neighbor from

a 2-hop node requires understanding subtle distinctions—both are semantically similar but structurally different—while separating 1-hop from 5-hop is trivial, as they differ vastly in both semantics and structure. Moreover, during training, the model's predictions vary in quality: some pairs are already correctly ranked while others require more attention. To address this, we introduce adaptive preference learning through two complementary mechanisms that emphasize challenging and informative pairs.

We build upon Direct Preference Optimization (DPO) (Rafailov et al., 2023), which parameterizes preferences through implicit rewards:

$$\psi(y|x) = \beta \log \frac{\pi_\theta(y|x)}{\pi_{\text{ref}}(y|x)} \tag{1}$$

where $\psi(y|x)$ measures the log-likelihood ratio of response y under the current policy π_θ versus the reference policy π_{ref}, scaled by β. The base DPO loss for a chosen-rejected pair (y_c, y_r) is:

$$\mathcal{L}_{\text{DPO}}(y_c, y_r) = -\log \sigma(\psi(y_c|x) - \psi(y_r|x)) \tag{2}$$

We extend DPO with two complementary weights that implement adaptive preference learning:

Distance weight w_{dist} emphasizes challenging nearby distinctions:

$$w_{\text{dist}}(y_c, y_r) = \frac{1}{|d(y_c) - d(y_r)|} \tag{3}$$

where $d(\cdot)$ denotes hop distance from the source node (with $d(y_c) = 1$ for the chosen neighbor). The weight is maximized for 2-hop negatives ($w = 1$), where distinguishing structurally unconnected but semantically similar nodes is most challenging, and decreases for distant negatives (e.g., $w = 0.25$ for 5-hop), where the distinction is more obvious.

Ranking weight w_{rank} focuses on the model's current errors:

$$w_{\text{rank}}(y_c, y_r) = \left| \frac{1}{\text{rank}(y_c)} - \frac{1}{\text{rank}(y_r)} \right| \tag{4}$$

where $\text{rank}(y)$ is the position when all candidates in the current instance are sorted by model scores $\psi(y|x)$ in descending order. The reciprocal weighting emphasizes top-position errors: if the model incorrectly places a negative at rank 1 and the true neighbor at rank 2, this pair receives higher weight than lower-rank pairs. This focuses learning on ranking errors most critical for downstream classification.

Final objective combines weighted preference learning with SFT regularization:

$$\mathcal{L}_{\text{HopRank}} = \sum_{(y_c, y_r) \in \mathcal{P}} w_{\text{dist}}(y_c, y_r) \cdot w_{\text{rank}}(y_c, y_r) \cdot \mathcal{L}_{\text{DPO}}(y_c, y_r) + \gamma \cdot \mathcal{L}_{\text{SFT}}(y_c) \tag{5}$$

where $\mathcal{P}$ contains all chosen-rejected pairs from our listwise preferences. The SFT term $\mathcal{L}_{\text{SFT}}(y_c) = -\log \pi_\theta(y_c|x)$ serves a dual purpose: it maintains the model's text generation quality while providing supervised signal on actual graph connections, ensuring the model learns both to distinguish preferences and to accurately predict true neighbors. The hyperparameter γ balances the two objectives.

2.4 Few-Shot Inference via Anchor Sampling

During inference, we leverage the learned preference model for few-shot classification. Given K labeled anchor nodes per class $\mathcal{A}_c = \{a_1^c, ..., a_K^c\}$, directly comparing a test node against all anchors in one step is impractical due to context length constraints when K is large. Instead, we employ a sampling-based voting strategy.

In each round, we randomly sample one anchor from each class and classify the test node by predicting which sampled anchor it would most likely connect to. After repeating this R times with different anchor combinations, the final class is determined via majority voting, where the class receiving the most votes across all rounds is selected as the prediction. The number of rounds R can be flexibly set: when $R = K$, each anchor is used once (standard few-shot setting); when $R > K$, **ensemble sampling** with replacement allows each anchor to be evaluated multiple times in different contexts. This ensemble approach reduces variance and improves classification reliability while efficiently handling scenarios with many labeled examples per class.

3 Experiments

Experimental Setup. We evaluate HOPRANK on three TAG benchmarks: Citeseer, Cora, and Pubmed (Sen et al., 2008; Yang et al., 2016). All results are averaged over 5 runs with different random seeds. We compare against tradi-

Table 1: **Main Results on Node Classification Benchmarks.** We report Accuracy (%) for few-shot learning settings. The overall best performance is in **bold**, and the second best is underlined.

Method	Citeseer			Cora			Pubmed		
	5-shot	10-shot	20-shot	5-shot	10-shot	20-shot	5-shot	10-shot	20-shot
Non-LLM Baselines									
GCN	$63.21_{\pm 0.87}$	$68.38_{\pm 1.49}$	$69.75_{\pm 0.42}$	$\mathbf{71.72}_{\pm 0.22}$	$\underline{78.22}_{\pm 0.89}$	$\underline{81.30}_{\pm 0.59}$	$70.58_{\pm 0.49}$	$75.33_{\pm 0.94}$	$79.64_{\pm 0.98}$
GraphSAGE	$62.35_{\pm 0.83}$	$67.02_{\pm 0.56}$	$69.95_{\pm 0.66}$	$70.66_{\pm 1.09}$	$77.98_{\pm 0.32}$	$\mathbf{81.37}_{\pm 0.43}$	$68.59_{\pm 1.01}$	$75.12_{\pm 0.55}$	$77.64_{\pm 0.92}$
SGC	$62.08_{\pm 0.77}$	$67.44_{\pm 0.60}$	$\underline{70.92}_{\pm 0.77}$	$71.48_{\pm 0.35}$	$\mathbf{78.44}_{\pm 0.37}$	$79.84_{\pm 0.68}$	$70.74_{\pm 0.94}$	$74.98_{\pm 1.91}$	$78.45_{\pm 0.56}$
GAT	$63.48_{\pm 1.54}$	$68.10_{\pm 1.77}$	$70.02_{\pm 0.44}$	$\underline{71.70}_{\pm 0.57}$	$77.58_{\pm 0.23}$	$80.94_{\pm 0.56}$	$69.56_{\pm 1.55}$	$72.84_{\pm 1.36}$	$78.37_{\pm 0.38}$
LLM-based Baselines (LLaMA-3-8b)									
Zero-Shot	$58.17_{\pm 0.64}$	$58.17_{\pm 0.64}$	$58.17_{\pm 0.64}$	$65.54_{\pm 0.26}$	$65.54_{\pm 0.26}$	$65.54_{\pm 0.26}$	$74.51_{\pm 0.39}$	$74.51_{\pm 0.39}$	$74.51_{\pm 0.39}$
w/ Neighbors	$54.93_{\pm 1.28}$	$54.93_{\pm 1.28}$	$54.93_{\pm 1.28}$	$68.72_{\pm 1.56}$	$68.72_{\pm 1.56}$	$68.72_{\pm 1.56}$	$74.98_{\pm 3.16}$	$74.98_{\pm 3.16}$	$74.98_{\pm 3.16}$
w/ Few Shot	$62.40_{\pm 1.47}$	$62.92_{\pm 1.32}$	$63.28_{\pm 1.50}$	$64.50_{\pm 2.40}$	$65.57_{\pm 2.13}$	$67.86_{\pm 2.22}$	$77.12_{\pm 1.34}$	$79.75_{\pm 1.12}$	$81.97_{\pm 1.25}$
InstructTuning	$\underline{68.77}_{\pm 2.66}$	$\underline{69.10}_{\pm 1.63}$	$68.44_{\pm 0.93}$	$69.33_{\pm 1.50}$	$69.53_{\pm 1.21}$	$71.89_{\pm 1.76}$	$\underline{83.71}_{\pm 0.64}$	$\underline{85.03}_{\pm 0.16}$	$\underline{88.16}_{\pm 2.17}$
LLaGA	$43.71_{\pm 4.36}$	$51.22_{\pm 1.43}$	$66.17_{\pm 0.43}$	$62.88_{\pm 2.19}$	$69.25_{\pm 0.97}$	$76.51_{\pm 0.22}$	$58.63_{\pm 1.05}$	$67.29_{\pm 2.26}$	$68.77_{\pm 0.35}$
GraphGPT	$51.83_{\pm 2.24}$	$55.40_{\pm 3.16}$	$62.84_{\pm 0.49}$	$60.17_{\pm 1.44}$	$61.58_{\pm 0.77}$	$63.25_{\pm 0.34}$	$57.39_{\pm 3.67}$	$71.33_{\pm 2.81}$	$78.36_{\pm 1.24}$
HopRank (Ours)	$\mathbf{69.03}_{\pm 1.11}$	$\mathbf{71.31}_{\pm 1.06}$	$\mathbf{72.91}_{\pm 1.02}$	$70.37_{\pm 1.42}$	$73.73_{\pm 1.33}$	$77.56_{\pm 1.08}$	$\mathbf{88.00}_{\pm 1.97}$	$\mathbf{88.62}_{\pm 1.24}$	$\mathbf{89.98}_{\pm 0.60}$

tional GNNs (GCN, GraphSAGE, SGC, GAT) and LLM-based methods using LLaMA-3-8B, including various prompting strategies (zero-shot with-/without neighbors, few-shot) and fine-tuned approaches (InstructTuning, LLaGA (Chen et al., 2024a), GraphGPT (Tang et al., 2024)). Detailed experimental settings are in Appendix B and C.

Main Results. Table 1 shows that HopRank achieves strong performance across all datasets. On Pubmed and Citeseer, HopRank matches or exceeds structure-aware GNN methods, while on Cora it achieves competitive results. Crucially, HopRank requires zero labeled data during training—labels are only used at inference as anchors for comparison, not for model updates. Despite this, HopRank consistently outperforms zero-shot and few-shot prompting baselines. This advantage is particularly evident compared to fine-tuned methods like InstructTuning, LLaGA, and GraphGPT, which require substantial labeled data during training to integrate graph structure, whereas HopRank learns entirely through adaptive preference learning on graph structure, without requiring labeled training data.

Ablation Study. Table 2 validates each component's contribution. Without self-supervised training, performance drops dramatically, confirming that base LLMs lack inherent structural awareness and cannot predict connections without learning from graph topology. Hierarchical sampling provides consistent 2-3% gains across datasets by focusing learning on challenging nearby distinctions rather than trivial distant comparisons, validating our curriculum design. The voting mechanism with ensemble sampling ($R > K$) provides 2-3% gains over standard voting ($R = K$) by allowing each anchor to be evaluated multiple times in different contexts, reducing variance through ensemble effects. Among loss components, SFT has the largest individual impact, while the distance and rank weights each contribute 1-3%, with their combination creating an adaptive curriculum that emphasizes challenging pairs and critical errors. Notably, all components show positive contributions, confirming the complementary nature of our design.

Table 2: **Ablation Study of HopRank Components.**

Model Configuration	Citeseer	Cora
HopRank (Full Model)	$\mathbf{72.91}_{\pm 1.02}$	$\mathbf{77.56}_{\pm 1.08}$
— Ablation of Core Components —		
w/o Hierarchical Sampling	$70.19_{\pm 1.54}$	$74.42_{\pm 1.50}$
w/o Self-supervised Training	$38.71_{\pm 2.07}$	$51.51_{\pm 3.80}$
w/o Ensemble Sampling ($R = K$)	$69.31_{\pm 1.94}$	$71.36_{\pm 2.55}$
— Ablation of Loss Components —		
w/o Distance Weight (w_{dist})	$72.19_{\pm 1.34}$	$74.79_{\pm 1.49}$
w/o Rank Weight (w_{rank})	$71.63_{\pm 1.45}$	$75.16_{\pm 1.27}$
w/o SFT ($\mathcal{L}_{\text{SFT}}$)	$68.87_{\pm 1.30}$	$73.25_{\pm 1.33}$

4 Conclusion

This paper introduced HopRank, which trains LLMs through self-supervised learning on graph structure. By leveraging the natural preference hierarchy encoded in graph topology, HopRank learns entirely from structural signals without requiring labeled training data. Our adaptive preference learning prioritizes informative training signals, enabling HopRank to match GNN performance while substantially outperforming existing graph LLM methods. This work demonstrates that self-supervised learning on graph structure provides a label-efficient alternative to supervised fine-tuning, opening new directions for applying LLMs to graph-structured data.

5 Limitations

As an initial exploration of self-supervised preference learning for graph LLMs, our work focuses on establishing the core framework using LLaMA-3-8B on citation network benchmarks. While this scope demonstrates the viability of our approach, broader validation would strengthen these findings. Future work could explore diverse model architectures (e.g., Qwen, Mistral, or larger LLaMA variants) and graph types (e.g., social networks, knowledge graphs, or heterophilic graphs). Similarly, our simple random anchor sampling already achieves competitive results, demonstrating the robustness of the link prediction mechanism, but more sophisticated selection strategies based on node centrality, structural diversity, or learned embeddings remain unexplored and could potentially yield further improvements.

References

Runjin Chen, Tong Zhao, Ajay Jaiswal, Neil Shah, and Zhangyang Wang. 2024a. Llaga: Large language and graph assistant. In *ICML*.

Zhikai Chen, Haitao Mao, Hang Li, Wei Jin, Hongzhi Wen, Xiaochi Wei, Shuaiqiang Wang, Dawei Yin, Wenqi Fan, Hui Liu, et al. 2024b. Exploring the potential of large language models (llms) in learning on graphs. In *SIGKDD*.

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

C Lee Giles, Kurt D Bollacker, and Steve Lawrence. 1998. Citeseer: An automatic citation indexing system. In *Proceedings of the third ACM conference on Digital libraries*, pages 89–98.

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

Xiaoxin He, Xavier Bresson, Thomas Laurent, Adam Perold, Yann LeCun, and Bryan Hooi. 2023. Harnessing explanations: Llm-to-lm interpreter for enhanced text-attributed graph representation learning. In *ICLR*.

Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, Weizhu Chen, et al. 2022. Lora: Low-rank adaptation of large language models. *ICLR*, 1(2):3.

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

Diederik P Kingma. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

TN Kipf. 2016. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*.

Miller McPherson, Lynn Smith-Lovin, and James M Cook. 2001. Birds of a feather: Homophily in social networks. *Annual review of sociology*, 27(1):415–444.

Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, et al. 2022. Training language models to follow instructions with human feedback. *Advances in neural information processing systems*, 35:27730–27744.

Rafael Rafailov, Archit Sharma, Eric Mitchell, Christopher D Manning, Stefano Ermon, and Chelsea Finn. 2023. Direct preference optimization: Your language model is secretly a reward model. *Advances in neural information processing systems*, 36:53728–53741.

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

Jiashuo Sun, Chengjin Xu, Lumingyuan Tang, Saizhuo Wang, Chen Lin, Yeyun Gong, Lionel M Ni, Heung-Yeung Shum, and Jian Guo. 2023. Think-on-graph: Deep and responsible reasoning of large language model on knowledge graph. *arXiv preprint arXiv:2307.07697*.

Jiabin Tang, Yuhao Yang, Wei Wei, Lei Shi, Lixin Su, Suqi Cheng, Dawei Yin, and Chao Huang. 2024. Graphgpt: Graph instruction tuning for large language models. In *Proceedings of the 47th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 491–500.

Petar Velickovic, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, Yoshua Bengio, et al. 2017. Graph attention networks. *stat*, 1050(20):10–48550.

Felix Wu, Amauri Souza, Tianyi Zhang, Christopher Fifty, Tao Yu, and Kilian Weinberger. 2019. Simplifying graph convolutional networks. In *International conference on machine learning*, pages 6861–6871. Pmlr.

Xixi Wu, Yifei Shen, Fangzhou Ge, Caihua Shan, Yizhu Jiao, Xiangguo Sun, and Hong Cheng. 2025. When do llms help with node classification? a comprehensive analysis. *arXiv preprint arXiv:2502.00829*.

Ruiyao Xu and Kaize Ding. 2025. Gnn-as-judge: Unleashing the power of llms for graph semi-supervised learning with gnn feedback. In *Machine Learning on Graphs in the Era of Generative Artificial Intelligence*.

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

Ruosong Ye, Caiqi Zhang, Runhui Wang, Shuyuan Xu, and Yongfeng Zhang. 2023. Language is all a graph needs. *arXiv preprint arXiv:2308.07134*.

A Datasets

We conduct our experiments on three widely-used citation network benchmarks: Cora, Citeseer, and Pubmed. These datasets are standard in the evaluation of graph-based node classification methods. Each node represents a scientific publication, and edges represent citation links between them. The task is to classify each publication into its respective academic field based on its textual content (title and abstract) and its position within the citation network.

Following the setup of recent comprehensive benchmarks (Wu et al., 2025; Xu and Ding, 2025), we utilize versions of these datasets where the raw text for each node is readily available. This ensures that all methods, including our LLM-based framework and the baselines, have access to the same rich semantic information. The key statistics for these datasets are summarized in Table 3.

Table 3: Summary statistics of the evaluation datasets. Homophily Ratio is the fraction of edges connecting nodes of the same class.

Dataset	Nodes	Edges	Classes	Homophily Ratio (%)
Cora	2,708	5,278	7	82.5
Citeseer	3,327	4,676	6	72.9
Pubmed	19,717	44,324	3	79.2

Note that all three datasets exhibit high homophily ratios (>70%), validating the core assumption of our approach that graph topology encodes class structure through connection patterns.

A brief description of each dataset is provided below:

- **Cora** (Sen et al., 2008): The Cora dataset consists of machine learning papers categorized into seven classes, including *Neural Networks*, *Reinforcement Learning*, and *Probabilistic Methods*. Cora is a relatively small graph but exhibits high homophily (82.5%), making it a standard benchmark for methods that leverage neighborhood similarity.
- **Citeseer** (Giles et al., 1998): The Citeseer dataset contains scientific publications classified into six categories covering areas such as *Artificial Intelligence*, *Machine Learning*, and *Human-Computer Interaction*. Compared to Cora, Citeseer has a slightly lower homophily ratio (72.9%), presenting a more challenging scenario where cross-class connections are more frequent.
- **Pubmed** (Yang et al., 2016): The Pubmed dataset comprises biomedical articles from the PubMed database concerning diabetes, classified into three categories: *Experimentally induced diabetes*, *Type 1 diabetes*, and *Type 2 diabetes*. It is a larger and denser graph with high homophily (79.2%). Its distinct biomedical domain makes it valuable for testing generalization beyond computer science literature.

Few-shot evaluation protocol. For all experiments, we follow the standard few-shot evaluation setup where $K \in \{5, 10, 20\}$ labeled nodes per class are available. Crucially, for our HOPRANK, these labels are only used during inference as anchor nodes for comparison, not during training. For baseline methods that require supervised training (e.g., InstructTuning, LLaGA, GraphGPT), the same K labeled nodes per class are used for fine-tuning. We follow the data splits provided by (Wu et al., 2025; Xu and Ding, 2025) to ensure fair comparison.

B Baseline Methods

We compare HOPRANK against two categories of baselines: (1) traditional Graph Neural Networks (GNNs) that operate on node features and graph structure, and (2) recent LLM-based approaches that leverage text understanding for node classification. Below we describe each baseline and its implementation details.

B.1 Non-LLM Baselines

This category includes representative GNN architectures that are widely considered strong baselines for node classification:

- **GCN (Graph Convolutional Network)** (Kipf, 2016): A foundational GNN model that learns node representations by aggregating information from immediate neighbors through spectral graph convolution.
- **GraphSAGE (Graph Sample and Aggregate)** (Hamilton et al., 2017): An inductive framework that generates node embeddings by sampling neighbors and applying aggregation functions (e.g., mean, max-pooling).
- **SGC (Simple Graph Convolution)** (Wu et al., 2019): A simplified variant of GCN that removes non-linear activations between layers, reducing complexity while maintaining competitive per-

formance on homophilous graphs.

- **GAT (Graph Attention Network)** (Velickovic et al., 2017): A GNN that incorporates attention mechanisms, allowing nodes to assign different weights to neighbors during aggregation.

Implementation Details. Our GNN baselines are implemented using PyTorch Geometric (Fey and Lenssen, 2019).For the few-shot evaluation, each GNN is trained in a semi-supervised (transductive) setting. This means the model is trained using only the labels of the K nodes per class from the training set, but it has access to the full graph structure and the features of all nodes during training. We adopt a consistent set of hyperparameters for all GNN models to ensure comparability. We use the Adam optimizer (Kingma, 2014) with a learning rate of `1e-2` and a weight decay of `5e-4`. Models are trained for a maximum of 500 epochs with an early stopping mechanism that has a patience of 100 epochs, monitored on a validation set. Following common practice, the hidden dimension for all GNN models is set to 128, and we generally use a 2-layer architecture with a dropout rate of 0.5.

B.2 LLM-based Baselines

This category includes methods that leverage LLMs for node classification, all using `LLaMA-3-8B-Instruct` as the backbone for fair comparison. These baselines cover different paradigms of integrating LLMs with graph data:

- **Zero-Shot**: The LLM classifies a node based solely on its textual attributes (title and abstract) without any examples. The prompt provides the node's text and asks the model to choose from the list of possible class labels.

- **w/ Neighbors**: An enhanced zero-shot baseline that incorporates structural information by including the titles of 1-hop neighbors in the prompt, providing local graph context.

- **w/ Few Shot (In-Context Learning)**: This baseline performs in-context learning by providing demonstration examples in the prompt. However, due to context length constraints, we cannot include all K examples per class (which would be $K \times C$ total examples) in a single prompt. Instead, we employ a voting strategy similar to our inference approach: in each round, we randomly sample one demonstration example per class (without replacement from the pool of K examples), present them to the LLM along with the query node, and collect the prediction. We repeat this process K times with different example combinations, and determine the final classification via majority voting. This approach enables the model to learn from multiple diverse example combinations while respecting context length limitations. The demonstrations are formatted as: `"Example: [text]` $\rightarrow$ `Answer: [label]"` before the query.

- **InstructTuning** (Ouyang et al., 2022): Standard supervised fine-tuning of `LLaMA-3-8B-Instruct` on the K labeled training nodes per class. We employ Parameter-Efficient Fine-Tuning (PEFT) using Low-Rank Adaptation (LoRA) (Hu et al., 2022) with rank 8 and alpha 16. Training data is formatted as instruction-response pairs.

- **LLaGA** (Chen et al., 2024a): A state-of-the-art method that uses a GNN to create structure-aware node representations, which are then projected and fed into the LLM along with node text. Only the projection layer is trained, making it parameter-efficient. We follow their official implementation and training protocol.

- **GraphGPT** (Tang et al., 2024): A graph-specific instruction-tuning method that employs multi-stage training to align textual semantics with graph structure. We follow the official implementation for fine-tuning on the downstream classification task.

Implementation Details. All LLM baselines use `LLaMA-3-8B-Instruct` (8 billion parameters) from Hugging Face Transformers. For zero-shot and few-shot prompting methods, we use greedy decoding with temperature 0.0 for deterministic outputs. For fine-tuning methods (InstructTuning, LLaGA, GraphGPT), we use AdamW optimizer with learning rate 1e-4, batch size 4, and gradient accumulation over 4 steps. LoRA parameters: rank=8, alpha=16, target modules include query and value projection layers.

Prompt Design. For zero-shot and few-shot baselines, we employ a consistent instruction prompt template shown below. The template uses placeholders that are replaced with dataset-specific information: `{raw_text}` contains the node's title and abstract, `{num_classes}` specifies the number of

classes, and {labels} lists the class names.

General Node Classification Prompt

```
Instruction: Given the following text from a scientific paper, classify it into one of the {num_classes} possible categories. The paper's text is:
{raw_text}
The possible categories are: {labels}.
Question: Which category does this paper belong to? Please respond with only the category name.
Answer:
```

For the **w/ Few Shot** baseline, we prepend K examples per class before the query in the format: "Example [i]: [text] → Answer: [label]". For the **w/ Neighbors** baseline, we append neighbor information after the main text: "This paper cites/is cited by: [neighbor titles]".

C HopRank Training Configuration

We provide detailed hyperparameters and training configurations for HopRank to ensure reproducibility.

Model Architecture. For fair comparison, we use LLaMA-3-8B-Instruct as the base model with LoRA (Hu et al., 2022) for parameter-efficient fine-tuning. LoRA is applied to all linear layers (lora_target="all") with rank 8 and alpha 16. This results in approximately 0.5% trainable parameters relative to the full model, significantly reducing memory requirements while maintaining performance.

HopRank Loss Configuration. Our loss function combines three components as described in Eq. (5):

- **DPO temperature** (β): 0.1, controlling the strength of preference optimization
- **SFT weight** (γ): 5.0, balancing preference learning with supervised fine-tuning

Training Hyperparameters. The complete training configuration is:

- **Optimizer**: AdamW with learning rate 2e-4
- **Learning rate schedule**: Cosine annealing with 10% warmup
- **Batch size**: 2 per device with 4 gradient accumulation steps (effective batch size = 8)
- **Training epochs**: 10 epochs (sufficient for convergence with preference learning)
- **Max sequence length**: 4096 tokens
- **Gradient clipping**: 1.0 (default)

Hierarchical Sampling Details. For each training edge, we sample one negative from each of {2-hop, 3-hop} neighborhoods, creating instances with 1 chosen response and 2 rejected responses. Nodes at each hop distance are sampled uniformly at random. If a particular hop neighborhood is empty, we skip it and only use available hop distances.

Evaluation and Model Selection. We evaluate every 20 training steps and employ early stopping with patience of 8 evaluations (160 training steps) based on evaluation loss. The best model checkpoint is saved and used for inference. Training typically converges within 2-3 epochs across all datasets.

Inference Configuration. During few-shot inference:

- **Anchor selection**: Random sampling with replacement when $R > K$
- **Voting rounds**: $R = 100$ for all experiments
- **Decoding**: Greedy decoding with temperature 0.8

The ensemble sampling strategy with $R = 100$ provides a good balance between accuracy and computational cost, allowing each anchor to be evaluated approximately more times on average across different contexts.

Computational Resources. All experiments are conducted on 2 NVIDIA H100 GPUs (80GB). Training time varies by dataset: Cora and Citeseer complete in approximately 1-2 hours, while Pubmed (larger graph) takes 3-4 hours.

D Training Data Format and Case Study

To illustrate the hierarchical hop-based sampling approach, we provide a complete example showing the exact prompt format used during training. The prompt structure follows the same format as shown in Section 2.2, with abstracts truncated here for brevity.

Complete Training Prompt Example. Below is a representative training instance from the Cora dataset:

Preference Structure. This instance creates three pairwise comparisons for the HopRank Loss:

Training Prompt Example from Cora Dataset

Given a source paper and multiple candidate papers, identify which candidate paper is most likely to have a citation relationship with the source paper.
Source Paper: 'Statistical Ideas for Selecting Network Architectures'
Abstract: Choosing the architecture of a neural network is one of the most important problems in making neural networks practically useful...
Candidate Papers:
A. 'Reasoning about Time and Probability'
Abstract: An essential component of an intelligent agent is the ability to notice, encode, store, and utilize information about...
[3-hop: easy negative]
B. 'Bayesian Graphical Models for Discrete Data'
Abstract: York's research was supported by a NSF graduate fellowship. The authors are grateful to Julian Besag, David Bradshaw...
[1-hop: actual neighbor]
C. 'Model Selection for Generalized Linear Models via GLIB, with Application to Epidemiology'
Abstract: This is the first draft of a chapter for Bayesian Biostatistics, edited by Donald A. Berry and Darlene K. Strangl...
[2-hop: hard negative]
Which candidate paper is most likely to be cited by or cite the source paper? Provide your answer by reproducing the title and abstract of the selected paper.

Chosen Response (Ground Truth):
Bayesian Graphical Models for Discrete Data
Rejected Responses:
- Model Selection for Generalized Linear Models via GLIB, with Application to Epidemiology `[2-hop]`
- Reasoning about Time and Probability `[3-hop]`

- **Chosen vs. 2-hop negative** (Paper B vs. Paper C): Both papers discuss statistical model selection, making this a challenging distinction. Distance weight: $w_{\text{dist}} = 1/(2-1) = 1.0$ (highest).
- **Chosen vs. 3-hop negative** (Paper B vs. Paper A): Semantic topics differ significantly (Bayesian graphical models vs. agent planning). Distance weight: $w_{\text{dist}} = 1/(3-1) = 0.5$ (lower).

The ranking weight w_{rank} adapts during training based on the model's current ranking of these candidates, emphasizing pairs where ranking errors occur at top positions.

E Additional Experimental Results

E.1 Impact of Voting Rounds

Table 4 shows how classification accuracy varies with the number of voting rounds R in the 20-shot setting on Citeseer. We observe that performance improves rapidly as R increases from $K = 20$ to $R = 100$, with diminishing returns after $R = 100$. This validates our ensemble sampling strategy: using $R > K$ allows each anchor to be evaluated multiple times in different contexts, reducing prediction variance. For computational efficiency, we use $R = 100$ in all main experiments, which provides a good balance between accuracy and inference cost.

Table 4: Impact of voting rounds R on Citeseer (20-shot). Performance improves with more rounds but plateaus after $R = 100$.

R	20	50	100	150	200
Accuracy (%)	69.31	71.18	72.91	73.08	73.15
Gain over $R = 20$	–	+1.87	+3.60	+3.77	+3.84

E.2 Sensitivity to Hyperparameters

We analyze the sensitivity of HOPRANK to key hyperparameters β (DPO temperature) and γ (SFT weight) on Citeseer (20-shot).

The role of γ. Since our task uses short responses (paper titles), the SFT loss $\mathcal{L}_{\text{SFT}}$ decreases rapidly during early training, as predicting exact titles is relatively easy once the model sees a few examples. However, the SFT term serves a crucial role throughout training: it maintains generation quality and provides supervised signal on true connections. To ensure the SFT term remains effective in later training stages when the DPO loss dominates, we use a relatively large $\gamma = 5.0$, which keeps the SFT gradient contribution balanced with the preference learning objective.

Table 5 shows that HOPRANK is relatively robust

to β within a reasonable range, but performance degrades significantly with very low γ (1.0), confirming that adequate SFT regularization is essential. Very high γ (10.0) slightly hurts performance as the model focuses too much on exact title generation rather than learning structural preferences. We select $\beta = 0.1$ and $\gamma = 5.0$ based on validation performance.

Table 5: Sensitivity to hyperparameters β and γ on Citeseer (20-shot).

Varying β (DPO temperature) with fixed $\gamma = 5.0$				
β	0.05	0.1	0.2	0.3
Accuracy (%)	72.34	**72.91**	72.56	72.18
Varying γ (SFT weight) with fixed $\beta = 0.1$				
γ	0.5	1.0	5.0	10.0
Accuracy (%)	69.45	70.12	**72.91**	72.45

E.3 Impact of Hop Distances

We evaluate the contribution of different hop distances to training. Table 6 shows that including both 2-hop and 3-hop negatives yields the best performance.

Table 6: Impact of hop distances used in training on Citeseer (20-shot). Including 2-hop and 3-hop negatives achieves the best balance between challenge and coverage.

Configuration	**Accuracy (%)**
Only 2-hop negatives	71.45
2-hop + 3-hop (default)	**72.91**
2-hop + 3-hop + 4-hop	72.83

As shown in Figure 1, nodes at larger hop distances (4-hop, 5-hop) are vastly different from the source node in both semantics and structure, making them trivial to distinguish. While these easy negatives help in early training, they provide limited informative signal for learning fine-grained class structure. Adding 4-hop negatives slightly degrades performance (72.83 vs. 72.91) because the model's training capacity is diluted across too many samples, with the additional trivial distinctions providing little value. This validates our hierarchical curriculum design: the most informative learning signal comes from challenging nearby distinctions (2-hop) and moderately difficult cases (3-hop), while distant negatives add more noise than signal. The distance weight $w_{\text{dist}} = 1/|d(y_c) - d(y_r)|$ naturally emphasizes this principle, giving 4-hop negatives only 1/3 the weight of 2-hop negatives.