

LLM-Powered Holistic Enhancement for Text-Attributed Graphs

Xiaotong Zhang Haoliang Ming Ze Wang Han Liu*

Dalian University of Technology, Dalian, China

zxt.dut@hotmail.com hlming.dut@hotmail.com

wangze@dlut.edu.cn liu.han.dut@gmail.com

Abstract

Text-attributed graphs (TAGs) have recently garnered significant attention due to their rich text information for each node. Graph Neural Networks (GNNs) have been widely used to leverage the text attributes and graph topology. However, traditional GNNs use shallow text embeddings and struggle with sparse and noisy structure. The advent of Large Language Models (LLMs) has introduced powerful capabilities in reasoning and text generation, promoting recent works to integrate LLMs with GNNs. However, most existing studies focus on a specific aspect of graph data, such as node features or graph structure, without fully exploring the entire graph. In this paper, we propose EFS (Enhancing Features and Structures), a novel holistic enhancement method that enhances both node features and graph structure with LLMs, which thoroughly explores the text attributes and efficiently addresses both missing and noisy edges compared to existing methods. We first design elaborate prompt templates and adopt LLMs to generate summarized keywords, ranked predictions and corresponding explanations for enriching the node features. Then we construct a lightweight edge refiner to improve the graph structure, which utilizes the previously generated ranked predictions without additional LLM API calls. Extensive experiments on various benchmark datasets show the superior performance and efficiency of EFS compared with the state-of-the-art baselines.

1 Introduction

Graphs are a form of structured data commonly found in various real-life applications, such as citation networks, product networks, social networks, and molecular graphs. A prevalent type of graph data is Text-Attributed Graphs (TAGs) (Yang et al., 2021), where nodes represent entities with textual information and edges signify relationships

between these entities. For instance, in a citation network, each paper is a node with attributes like the title and abstract, while edges indicate citation relationships between papers. In a social network, each user is a node with attributes like user profiles, and edges indicate user relationships. TAGs integrate text attributes with graph topology, which provide rich information for real-world data and have been widely applied in recommendation systems (Zhu et al., 2021), social networks (Myers et al., 2014), fact verification (Liu et al., 2020), molecular property prediction (Li et al., 2021b), and etc.

The unique characteristics of TAGs make them unsuitable for traditional machine learning methods. To effectively leverage both node features and graph structure in TAGs, GNNs (Kipf and Welling, 2017) that propagate feature information through edges are developed and have become a cornerstone in graph learning, achieving notable success by exploiting message passing and aggregation strategies to learn node representations.

The quality of node features and graph structure in TAGs highly affects the performance of GNNs. Early studies encode the node text attributes with non-contextual shallow embedding techniques such as Bag-of-Words and TF-IDF, which are inadequate in capturing context-aware information and complex semantic relationships (Miaschi and Dell’Orletta, 2020; Qiu et al., 2020). Moreover, real-world graphs exhibit sparsity and noisy links that undermine the reliability of graph structure. For example, real-world graphs may contain missing or noisy links (Dai et al., 2022). These limitations hinder the performance of GNNs in learning from graphs.

Most recently, with the advent of LLMs like GPT-3 (Brown et al., 2020) and LLaMA (Touvron et al., 2023a), there have been attempts to apply LLMs to graph learning tasks. They achieve more promising results compared to previous GNN-

*Corresponding author.

based methods without LLMs. For example, TAPE (He et al., 2024) uses LLMs to generate predictive and explanatory information about nodes, which augments text information and improves node features. GraphEdit (Guo et al., 2024) leverages LLMs to identify node dependencies and handle noisy connections between uncorrelated nodes, which requires traversing the whole graph to fine-tune the LLM, thus is too time-consuming. However, current methods using LLMs often focus on specific aspects of graph data, such as node features or graph structure, without fully exploring the graph data as a whole. This limitation restricts the ability of the model to comprehensively understand and characterize complex graphs. The only work considering both node features and graph structure is GAUGLLM (Fang et al., 2024), it enhances them through self-supervised contrastive learning by combining prompt experts with a collaborative edge editor. However, this approach requires invoking a large language model to evaluate every candidate edge, leading to high inference costs. Its feature perturbation strategy also depends solely on generating diverse texts from multiple prompts, without extracting explicit internal key information or incorporating external explanatory knowledge.

In this paper, we propose a new method called EFS that uses LLMs to synthesize and enhance both node features and graph structure in TAGs, fully exploiting the rich information within graphs to improve the performance of GNNs in practical applications. EFS has two main objectives: first, to thoroughly explore and utilize the semantic information in the text attributes of nodes, and second, to refine the graph structure by adding missing links and weakening spurious edges without incurring significant additional costs. To address the former objective, we first design prompts for the LLM to extract keywords and generate concise explanations for each keyword based on the raw text attributes of each node. Then by leveraging the same raw text attributes of each node, the LLM produces a ranked list of predictions along with textual explanations for each prediction. Finally, we enhance the node features by transforming the raw text, extracted keywords, ranked predictions, and their corresponding explanations into enriched and informative representations using the LM. To avoid significant additional costs when addressing the missing and noisy links, we introduce a lightweight edge refiner for graph structure learning. This refiner is trained using the raw text attributes and graph topology to

identify candidate nodes that have potential connections or disconnections with each node. We then use the previously generated ranked predictions to assess the quality of these candidate nodes, with only high-quality candidates retained for connection or disconnection. This enables us to reweight the adjacency matrix and improve the graph structure. At the end, we train multiple GNN models with various augmented node features and the enhanced graph structure. The outputs of these models are fused for downstream node classification tasks by assigning adaptive weights, ensuring that the most relevant information is prioritized for each node. The resulting graph is a task-beneficial topology for downstream classification rather than an exact reconstruction of the physical graph.

In summary, the contributions of this paper are as follows:

- (1) The vast majority of existing works focus on enhancing either node features or graph structure using LLMs. We propose to improve both jointly, achieving better performance without incurring significant additional costs.
- (2) We propose a novel framework that integrates enhanced node features and graph structure with the previously extracted keywords, generated ranked predictions, and raw text embeddings. We devise elaborate prompt templates for the guidance of LLMs and design a lightweight edge refiner for graph structure learning.
- (3) EFS is modular and plug-and-play, allowing the modules to be freely combined with any GNN model for downstream node classification tasks.
- (4) Extensive experiments show that EFS achieves the state-of-the-art performance while maintaining high efficiency.

2 Related Work

2.1 GNNs in TAGs

GNNs have been utilized to enhance various tasks by capturing the node attributes and the relationships between nodes. Graph Convolutional Network (GCN) (Kipf and Welling, 2017) has been widely adopted due to its ability of using graph convolution to aggregate and propagate information across nodes, thus enhancing node representations and improving the performance of downstream tasks. Graph Attention Network (GAT) (Veličković et al., 2018) applies self-attention to enable the model to focus on the most relevant neighbors when aggregating information, which has

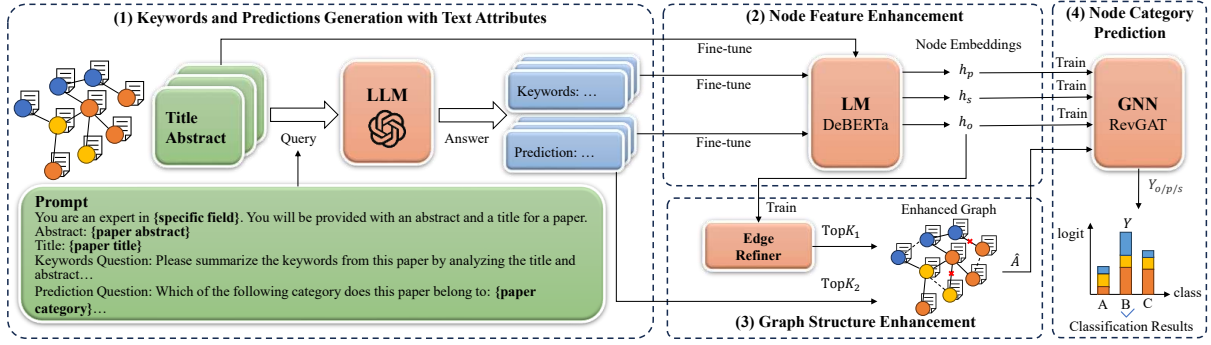


Figure 1: The overview of the EFS method. This method involves four main steps: (1) keywords and predictions generation with text attributes: this step uses LLMs to generate keywords and categories along with predictive explanations for each node; (2) node feature enhancement: this step learns three kinds of node embeddings based on original text attributes, keywords and their explanations, ranked predictions and their explanations; (3) graph structure enhancement: this step uses the original text attributes to construct an edge refiner, which is then combined with the generated category predictions for identifying the potentially connected nodes and the nodes that should not be directly connected; (4) node category prediction: this step feeds each kind of node embedding and the enhanced graph structure into a GNN model, then adaptively weights the outputs of these GNN models to obtain the combined node classification results.

shown improvements over traditional GCN. Current GNNs for processing textual information in TAGs usually transform text attributes into shallow features, e.g., Bag-of-Words and TF-IDF, which may struggle with capturing long-range dependencies in textual information, leading to suboptimal performance in tasks like node classification and link prediction.

2.2 LLMs in TAGs

LLMs like GPT-3 (Brown et al., 2020) and LLaMA (Touvron et al., 2023a) have revolutionized Natural Language Processing (NLP) by demonstrating exceptional language understanding and advanced reasoning, which significantly enhance the performance of various NLP tasks. However, for graph-structured data, LLMs have not been fully explored. Recently there has been an influx of methods that utilize LLMs to solve graph problems. They can be categorized into two types according to the role played by LLMs in graph-related tasks, namely LLMs as predictors and LLMs as enhancers. LLMs as predictors refer to inputting graph data into LLMs to directly accomplish graph tasks by transforming different data forms within graphs. For example, GraphGPT (Tang et al., 2024) and InstructGLM (Ye et al., 2024) designs a series of prompts to fine-tune the LLM using natural language descriptions of the target node and its multi-hop neighbors. LLaGA (Chen et al., 2024) uses node-level templates to convert graph data into organized sequences. However, there are some dif-

ficulties in transforming the whole graph structure due to the huge number of relationships. Hence existing methods only transform and retain the local neighborhood information, resulting in information loss of graph structure. The key of LLMs as enhancers is to enhance the quality of node embeddings with the help of powerful LLMs or to capture and generate higher level information. For example, OFA (Liu et al., 2024) utilizes LLMs to uniformly encode graph data from different domains into a common embedding space. ENGINE (Zhu et al., 2024) adds a lightweight and tunable G-Ladder module to each layer of the LLM to explicitly capture the structural information of TAGs. CasMLN (Wang et al., 2024) uses LLMs to capture the implicit nature of temporal heterogeneous graphs and node types, which can enhance type-level and graph-level representations. Some popular methods such as TAPE (He et al., 2024) and other methods are summarized in the survey (Ren et al., 2024). Current methods primarily focus on enhancing either node features or graph structure. However, techniques that improve node features often fail to fully leverage the rich information of text attributes. On the other hand, methods that aim at enhancing graph structure tend to be costly. This raises the question of how to effectively integrate and improve both aspects.

In contrast, EFS reuses the same node-level LLM outputs for both feature and structure enhancement, so graph refinement requires neither per-edge LLM queries nor a second generation

stage. Unlike graph-reconstruction methods, EFS learns a task-beneficial topology for downstream node classification. This unified and lightweight design distinguishes EFS from feature-only, structure-only, and per-edge LLM approaches.

3 Preliminaries

3.1 Text-Attributed Graphs

Define the text-attributed graphs (TAGs) as $\mathcal{G} = \{\mathcal{V}, \mathbf{A}, \mathcal{S}\}$. $\mathcal{V}$ represents a set of nodes. Each node $v \in \mathcal{V}$ corresponds to an entity or an item within the graph. $\mathbf{A} \in \{0, 1\}^{N \times N}$ is the adjacency matrix, which defines the graph structure by indicating the connections between nodes. $\mathcal{S}$ denotes the set of text attributes associated with the nodes. For each node $v \in \mathcal{V}$, there is an associated piece of textual sentence $s_v \in \mathcal{S}$.

3.2 Node Classification

In this paper, we address the problem of classifying nodes in a TAG. Specifically, the objective is to learn a function $f : \mathcal{V} \rightarrow Y$ that uses the TAG $\mathcal{G}$ and a set of labeled nodes $\mathcal{N} \subset \mathcal{V}$ to predict categories for unlabeled nodes $\mathcal{I} \subset \mathcal{V} \setminus \mathcal{N}$.

3.3 Graph Neural Networks

GNNs learn node or graph representations by propagating and aggregating node features via edges. This iterative process enables GNNs to capture both local structures and feature information effectively. Formally, GNNs learn the representation of the node v_i at the k -th layer by:

$$\mathbf{h}_i^{(k)} = \text{Update}(\mathbf{h}_i^{(k-1)}, \text{Aggregate}(\mathbf{h}_j^{(k-1)} | v_j \in \mathcal{N}_i)), \quad (1)$$

where $\mathcal{N}_i$ is the neighboring nodes of node v_i . Aggregate function processes the features of these neighbors from the previous layer, employing common methods such as mean, sum, or max pooling. Update function combines the previous representation $\mathbf{h}_i^{(k-1)}$ of v_i with the aggregated neighborhood information to form the new node representation $\mathbf{h}_i^{(k)}$ of v_i . This combination can be achieved through various methods, such as a neural network layer, a gating mechanism, or a simple concatenation followed by a non-linear transformation.

4 The Proposed Method

In this section, we propose a new pipeline for node classification on TAGs. As shown in Figure 1, our key idea is to leverage LLM to enhance both node

features and topology for downstream GNNs without incurring significant additional costs.

4.1 Keywords and Predictions Generation with Text Attributes

As demonstrated by TAPE (He et al., 2024), LLMs can effectively predict node categories and provide explanations due to their deep semantic understanding and reasoning capabilities. However, we find that the generated category predictions and corresponding explanations cannot exhibit key information of each node, due to the redundancy present in the text. For instance, in the Wiki-CS dataset, many nodes contain numerous domain-specific terms and redundant descriptions due to the comprehensive and specialized nature of Wikipedia content. To address this, we propose a keyword summarization method that removes redundant information while retaining the semantic core by extracting and interpreting key terms from the node text.

In this section, for each node, we aim to use LLMs to first extract keywords and predict its categories based on the original text information, while also generating corresponding explanations using its text attributes. Then these additional information can be combined with the original text information in TAGs, enhancing both node features and graph structure.

We first devise an elaborate prompt template for each dataset, then directly access the LLM via its API, avoiding the need for fine-tuning or direct interaction with its embeddings or logits. The prompt template varies with datasets due to the different data types. For example, if we have a dataset about research paper, its prompt format is outlined in Appendix (Table 5), which directs the LLM to summarize the paper’s keywords and predicted categories, and to generate explanations for each keyword and prediction based on the title and abstract.

Afterwards, for each node $v_i \in \mathcal{V}$, the keywords summary prompt Prompt_s and category prediction prompt Prompt_p are generated according to the prompt template. The LLM is then tasked with summarizing the keywords based on the node’s textual attributes, providing a concise explanation for each keyword. Following this, the LLM predicts the node’s categories based on its text attributes, ranks the predictions by likelihood, and provides an explanation for each prediction:

$$\begin{aligned} s_s &= \text{LLM}(\text{Prompt}_s(s_o)), \\ s_p &= \text{LLM}(\text{Prompt}_p(s_o)), \end{aligned} \quad (2)$$

where s_o denotes the original text attributes of the node, s_s denotes the summarized keywords along with their explanations, and s_p denotes the ranked category predictions and their corresponding explanations, respectively.

An example of the keywords, ranked prediction list and explanations is shown in Appendix (Table 6), the later parts of the keywords and explanations are omitted due to space limitation. Then these keywords, ranked prediction lists and explanations generated by LLM will be utilized for node feature enhancement and graph structure enhancement.

4.2 Node Feature Enhancement

In addition to using the original text attributes, we enhance the node features with the keywords, ranked predictions, and their corresponding explanations generated by LLMs. Specifically, for each node, the original text attributes, keyword information, and ranked category prediction details are separately input into the DeBERTa LM:

$$\mathbf{h}_o = \text{LM}(s_o), \mathbf{h}_s = \text{LM}(s_s), \mathbf{h}_p = \text{LM}(s_p), \quad (3)$$

where $\mathbf{h}_o$, $\mathbf{h}_s$ and $\mathbf{h}_p$ denote the embeddings of the original text attributes s_o , keywords information s_s and the category predictions information s_p , respectively. These embeddings are further input into a Multi-Layer Perceptron (MLP) to obtain three vectors $\mathbf{y}_o$, $\mathbf{y}_s$ and $\mathbf{y}_p$ representing the class predictions for each node:

$$\mathbf{y}_o = \text{MLP}(\mathbf{h}_o), \mathbf{y}_s = \text{MLP}(\mathbf{h}_s), \mathbf{y}_p = \text{MLP}(\mathbf{h}_p). \quad (4)$$

Then the LM and MLP are fine-tuned with cross-entropy loss between the class prediction vectors and ground truth of the nodes.

Finally, $\mathbf{h}_o$, $\mathbf{h}_s$ and $\mathbf{h}_p$ will be taken as enhanced features of the node v_i to feed into downstream GNNs. And $\mathbf{h}_o$ will be further used for enhancing the graph structure.

4.3 Graph Structure Enhancement

Graph structure is the most fundamental part of GNNs, which influences how information is propagated, aggregated, and represented, ultimately impacting the performance of node classification. But real-world graphs often encounter challenges such as noisy links and sparsity, which can compromise the reliability of graph structure. Traditional methods refine the graph structure with node features. However, they generally use the shallow feature

embeddings or the deep embeddings from LMs. Most recently, GraphEdit (Guo et al., 2024) uses LLMs to learn node features so that an edge refiner is constructed, then fine-tunes an LLM to predict and improve the graph structure, achieving superior performance for node classification tasks. But it requires traversing the whole graph to fine-tune the LLM, which will incur a huge cost, especially for large graphs.

In this section, we present a graph structure enhancement technique that refines connections without any additional LLM API calls. We begin by using the previously learned node embeddings $\mathbf{h}_o$ to train an edge refiner. Next, we merge the predicted results with the ranked prediction s_p from Eq. (2) to construct an enhanced adjacency matrix. The objective is not to reconstruct the observed physical graph exactly, but to construct a task-beneficial topology for downstream node classification.

Specifically, we first concatenate the representations of the two nodes and feed them into an MLP to predict a task-related connection score:

$$\hat{y}_e(\mathbf{h}_{o_i}, \mathbf{h}_{o_j}) = \text{MLP}(\mathbf{h}_{o_i} || \mathbf{h}_{o_j}), \quad (5)$$

where $\mathbf{h}_{o_i} = \text{LM}(s_{o_i})$ and $\mathbf{h}_{o_j} = \text{LM}(s_{o_j})$ are the representations of node v_i and node v_j , respectively. $||$ denotes the concatenation operator.

After that, we construct a set of positive node pairs and a set of negative node pairs using labeled training nodes. For any two training nodes, if their labels are the same then they are taken as a positive node pair, and if different then they are considered as a negative node pair:

$$y_e(\mathbf{h}_{o_i}, \mathbf{h}_{o_j}) = \begin{cases} 1 & \text{if } y_i = y_j, \\ 0 & \text{if } y_i \neq y_j, \end{cases} \quad (6)$$

where y_i and y_j denote the labels of nodes $v_i, v_j \in \mathcal{V}_{\text{train}}$. Importantly, Eq. (6) uses training labels only. Validation labels are used only for hyperparameter selection, and test labels are used only for final evaluation; neither is used to construct pair targets or decide individual edges. We utilize the cross-entropy loss to train the MLP in the edge refiner:

$$\mathcal{L}_{\text{CE}}(y_e, \hat{y}_e) = -[y_e \log(\hat{y}_e) + (1 - y_e) \log(1 - \hat{y}_e)], \quad (7)$$

where $\hat{y}_e \in [0, 1]$ and $y_e \in \{0, 1\}$ are the predicted score and pair target defined by Eq. (5) and Eq. (6), respectively.

Before applying the trained edge refiner, we construct bounded candidate sets rather than scoring all $|\mathcal{V}|^2$ node pairs. For deletion, S_i^{del} contains only the existing neighbors of v_i . For addition, block-wise cosine-similarity search over $\mathbf{h}_o$ retains the top- M non-neighbors ($M = 50$) as S_i^{add} . Then we use the trained edge refiner to obtain the top K_1 potentially connected nodes J_i^{add} in S_i^{add} and the bottom K_1 potentially connected nodes J_i^{del} in S_i^{del} :

$$\begin{aligned} J_i^{add} &= \text{Top}K_1(\text{Refiner}(v_i)), J_i^{add} \subseteq S_i^{add}, \\ J_i^{del} &= \text{Btm}K_1(\text{Refiner}(v_i)), J_i^{del} \subseteq S_i^{del}, \end{aligned} \quad (8)$$

where $\text{Refiner}(v_i)$ ranks only nodes in $S_i^{add} \cup S_i^{del}$ by $\hat{y}_e(\mathbf{h}_{o_i}, \mathbf{h}_{o_j})$. Thus, MLP scoring costs $O(|\mathcal{V}|M + |\mathcal{E}|)$ rather than $O(|\mathcal{V}|^2)$. Blockwise retrieval stores only the current similarity block and its top- M results instead of materializing the full all-pair matrix.

We further use the ranked prediction list s_p in Eq. (2) to determine the final connected and disconnected nodes for each node. Mathematically, for each node v_i , if we have a node $v_j \in J_i^{add}$, and there is an intersection between their top K_2 ranked categories, then v_i and v_j are connected:

$$A_{i,j}^{add} = \begin{cases} 1 & \text{if } \text{Top}K_2(s_p(v_i)) \\ & \cap \text{Top}K_2(s_p(v_j)) \neq \emptyset, \\ 0 & \text{if } \text{Top}K_2(s_p(v_i)) \\ & \cap \text{Top}K_2(s_p(v_j)) = \emptyset, \end{cases} \quad (9)$$

where $s_p(v_i)$ and $s_p(v_j)$ are the ranked category predictions of v_i and v_j , respectively. $\text{Top}K_2$ denotes the top K_2 categories in the set, and K_2 is a hyperparameter to be set.

Similarly, if we have a node $v_j \in J_i^{del}$ and there is no intersection between their top K_2 ranked categories, then v_i and v_j are disconnected:

$$A_{i,j}^{del} = \begin{cases} 1 & \text{if } \text{Top}K_2(s_p(v_i)) \\ & \cap \text{Top}K_2(s_p(v_j)) = \emptyset, \\ 0 & \text{if } \text{Top}K_2(s_p(v_i)) \\ & \cap \text{Top}K_2(s_p(v_j)) \neq \emptyset. \end{cases} \quad (10)$$

Finally, the node connectivity matrix A^{add} and disconnectivity matrix A^{del} are merged with the original adjacency matrix A to reweight the edge weights, and the enhanced graph structure $\hat{A}$ is obtained by

$$\hat{A} = A + A^{add} - A^{del}. \quad (11)$$

The benchmark graphs are treated as undirected, so additions and deletions are applied symmetrically.

4.4 Node Category Prediction

We input the enhanced node features into the GNN along with the enhanced graph structure $\hat{A}$ to get the final classification results. We construct a separate GNN for each kind of feature among $\mathbf{h}_o$, $\mathbf{h}_p$ and $\mathbf{h}_s$ to generate three predicted logits: Y_o , Y_p , and Y_s .

$$\begin{aligned} Y_o &= \text{GNN}_o(\mathbf{h}_o, \hat{A}), \\ Y_p &= \text{GNN}_p(\mathbf{h}_p, \hat{A}), \\ Y_s &= \text{GNN}_s(\mathbf{h}_s, \hat{A}). \end{aligned} \quad (12)$$

Finally, we combine these logits with adaptive weights to derive the combined node classification results. Specifically, we concatenate Y_o , Y_p , and Y_s , and input it into a one-layer MLP to predict the categories for all the nodes:

$$Y = \text{MLP}(Y_o || Y_p || Y_s), \quad (13)$$

where Y is the prediction of nodes, which will be used to train the GNNs and MLP with the cross-entropy loss.

5 Experiments

5.1 Experimental Settings

5.1.1 Datasets

We use six real-world datasets including Cora (McCallum et al., 2000), Pubmed (Sen et al., 2008), Citeseer (Giles et al., 1998), ogbn-arxiv (Hu et al., 2020), ogbn-products (Hu et al., 2020), and Wiki-CS (Mernyei and Cangea, 2020) to evaluate EFS, which are widely used for node classification tasks. In Cora, Pubmed, Citeseer, and ogbn-arxiv, each node represents a publication, and the edges represent citations between the publications. For the ogbn-products dataset, nodes represent Amazon products, with edges indicating that two products were purchased together. In the Wiki-CS dataset, nodes correspond to Wikipedia entries related to computer science, and edges represent hyperlinks connecting these entries. The statistics of datasets are shown in Table 1.

5.1.2 Baselines

We compare EFS with 11 baselines from 3 kinds of methods. (1) GNN: GCN (Kipf and Welling, 2017) and RevGAT (Li et al., 2021a); (2) LLM: GPT-3.5; (3) GNN+LLM: GIANT (Chien et al., 2022), TAPE (He et al., 2024), GraphEdit (Guo et al., 2024), GAUGLLM (Fang et al., 2024), OFA (Liu et al.,

Dataset	#Nodes	#Edges	#Classes
Cora	2,708	5,429	7
Pubmed	19,717	44,338	3
Citeseer	3,186	4,277	6
ogbn-arxiv	169,343	1,166,243	40
ogbn-products	54,025	74,420	47
Wiki-CS	48,362	500,928	12

Table 1: Statistics of datasets.

Datasets	K_1	K_2
Cora	4	3
Pubmed	4	1
Citeseer	4	3
ogbn-arxiv	4	3
ogbn-products	4	3
Wiki-CS	4	3

Table 2: Hyperparameter settings.

2024), InstructGLM (Ye et al., 2024), ENGINE (Zhu et al., 2024), and LLaGA (Chen et al., 2024). More details are given in the Appendix A.1.

5.1.3 Implementation Details

EFS uses gpt-3.5-turbo-0125 (OpenAI, 2024) as the LLM for ranked predictions and explanation generation with text attributes, and takes DeBERTa (He et al., 2021) as the LM to encode the texts associated with each node. To make a fair comparison with GNN+LLM baselines, we take RevGAT (Li et al., 2021a) as the GNN model for our EFS method and GNN+LLM baselines. In node feature enhancement, we add a one-layer MLP upon DeBERTa to fine-tune the LM, using the AdamW optimizer with a learning rate of 0.002. In graph structure enhancement, we apply a four-layer MLP as the edge refiner and train it with a learning rate of 0.01. For the division of the dataset, we follow the previous GNN+LLM studies (He et al., 2024; Guo et al., 2024), where the training, validation, and test sets are randomly divided according to the ratio of 6:2:2. For the hyperparameters, we use the validation set to select K_1 and K_2 in graph structure enhancement, as shown in Table 2, with their investigation given in Appendix A.2. For the DeBERTa and RevGAT used in EFS, we follow their original papers to set the hyperparameters.

The API settings are temperature = 1.0, top- p = 1.0, and a maximum output length of 4,096 tokens. Each node is queried once for keywords and once for ranked predictions, yielding $2|\mathcal{V}|$ one-time node-level calls and no per-edge or per-training-round calls; all outputs are cached before training. The original-text, keyword, and prediction views use separately fine-tuned DeBERTa encoders and separate one-layer MLP heads. Positive and negative edge-refiner pairs are sampled only from the same-class and different-class pools within $\mathcal{V}_{\text{train}}$. Validation labels are used only for hyperparameter selection, and test labels are used only for final evaluation. Appendix A.4 provides the remaining prompt, parsing, preprocessing, and graph-update

details.

For each baseline method, we used the official source code and ran it within the same experimental setup to ensure reproducibility and fairness. This included consistent data splits, feature preprocessing, hardware settings, and hyperparameter search ranges. All experiments were conducted on an NVIDIA A800 GPU with 80GB of VRAM.

5.2 Result Analysis

We evaluate the performance of EFS and the baselines with Classification Accuracy. The results in Table 3 are averaged over five runs by randomly selecting seeds. We show the mean results and the standard deviations in Table 3, and draw the following conclusions.

In comparison to existing GNN+LLM methods, EFS delivers superior results across all six datasets. These findings demonstrate the significance of enhancing the node features and graph structure simultaneously.

(1) Compared to the graph representation learning methods such as GIANT and TAPE, EFS captures more discriminative textual information by leveraging LLMs to extract keywords, ranked predictions, and corresponding explanations. Node features in EFS are enriched holistically by integrating internal summary information with external prediction information, while GIANT only uses internal information within TAGs and TAPE explores limited external prediction information.

(2) Compared to the graph structure learning method like GraphEdit, EFS constructs a more lightweight edge refiner that leverages original features and existing edges of TAGs as well as the ranked category predictions generated by LLMs, without finetuning the LLM model with the whole graph. This operation makes EFS identify spurious edges and potential connections at a low cost, thereby improving graph quality.

(3) Compared to the integrated graph representation and structure learning method like GAUGLLM

	Method	Cora	Pubmed	Citeseer	ogbn-arxiv	ogbn-products	Wiki-CS
GNN	GCN	86.46 $\pm$ 0.56	80.42 $\pm$ 0.42	72.56 $\pm$ 0.46	71.21 $\pm$ 0.45	70.12 $\pm$ 0.56	75.96 $\pm$ 0.24
	RevGAT	85.58 $\pm$ 0.86	88.26 $\pm$ 0.43	75.76 $\pm$ 0.62	70.82 $\pm$ 0.36	69.85 $\pm$ 0.56	78.64 $\pm$ 0.48
LLM	GPT	67.69 $\pm$ 0.00	93.42 $\pm$ 0.00	59.29 $\pm$ 0.00	73.50 $\pm$ 0.00	73.56 $\pm$ 0.00	72.67 $\pm$ 0.00
	GIANT	85.75 $\pm$ 0.88	91.21 $\pm$ 0.29	73.45 $\pm$ 0.72	74.16 $\pm$ 0.12	75.14 $\pm$ 0.58	76.36 $\pm$ 0.35
	TAPE	89.42 $\pm$ 1.45	94.88 $\pm$ 0.16	78.58 $\pm$ 0.63	77.37 $\pm$ 0.12	80.86 $\pm$ 0.44	82.67 $\pm$ 0.29
	GraphEdit	88.21 $\pm$ 1.27	93.65 $\pm$ 0.33	80.84 $\pm$ 1.26	75.24 $\pm$ 0.25	78.18 $\pm$ 0.28	79.86 $\pm$ 0.78
	GAugLLM	78.84 $\pm$ 0.75	83.50 $\pm$ 0.84	74.12 $\pm$ 1.58	73.71 $\pm$ 0.08	71.87 $\pm$ 0.64	73.42 $\pm$ 0.41
	OFA	77.24 $\pm$ 0.98	79.62 $\pm$ 1.06	70.58 $\pm$ 0.81	75.24 $\pm$ 0.79	78.98 $\pm$ 0.61	78.27 $\pm$ 0.26
	InstructGLM	87.08 $\pm$ 0.32	93.84 $\pm$ 0.25	77.04 $\pm$ 0.12	76.42 $\pm$ 0.09	79.18 $\pm$ 0.41	80.79 $\pm$ 0.72
	ENGINE	89.96 $\pm$ 0.36	92.56 $\pm$ 0.17	78.34 $\pm$ 0.78	76.07 $\pm$ 0.34	77.75 $\pm$ 0.59	81.33 $\pm$ 0.87
	LLaGA	89.45 $\pm$ 0.25	94.95 $\pm$ 0.12	75.46 $\pm$ 0.36	75.78 $\pm$ 0.14	79.85 $\pm$ 0.23	81.85 $\pm$ 0.28
	EFS (ours)	91.64 $\pm$ 0.23	96.36 $\pm$ 0.08	87.49 $\pm$ 0.81	78.86 $\pm$ 0.19	82.27 $\pm$ 0.10	84.08 $\pm$ 0.09

Table 3: Accuracy comparison between EFS and other baselines. The best results are **in bold**.

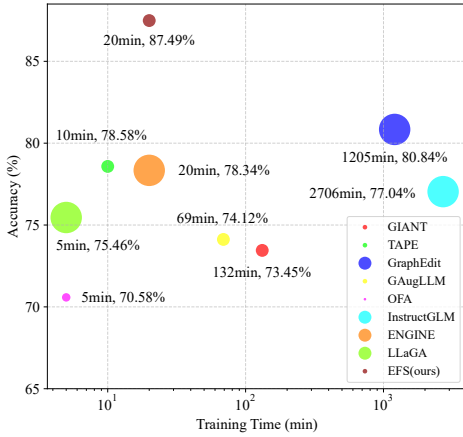


Figure 2: Training time and accuracy on Citeseer, with the size of the marker indicating the number of parameters.

and others, EFS achieves the highest accuracy. This demonstrates that the combination of deep textual encoding from LMs, discriminative information extracted from LLMs, and graph structure optimization collectively enables more nuanced and effective graph representations, resulting in superior node classification performance.

Appendix A.5 reports compact robustness and additional-comparison results. EFS remains effective with LLaMA2-7B-Chat (Touvron et al., 2023b) and with BERT/RoBERTa text encoders (Devlin et al., 2019; Liu et al., 2019), and it outperforms the simple rewiring and structure-learning baselines on Cora and Citeseer.

5.3 Time Efficiency Analysis

To assess the balance between training efficiency and classification performance, we compare the training time and accuracy of EFS with other GNN+LLM baselines on the Citeseer dataset. Fig-

ure 2 highlights the relationship between training time and accuracy, with marker sizes representing the parameter count for each method. The figure reports post-LLM model-side training time because node-level LLM generations are produced once and cached. The complete model-side EFS pipeline takes 20.20 minutes on Citeseer and 411 minutes on ogbn-arxiv. Edge refinement makes no additional LLM calls and scores bounded candidate sets rather than all node pairs. Appendix A.6 reports candidate counts, runtime decomposition, parameters, FLOPs, LLM calls, and token usage. Because billing records are unavailable, we report tokens rather than an estimated monetary cost.

This analysis underscores the strength of EFS in achieving state-of-the-art performance with dramatically reduced training time. This efficiency is attributed to the streamlined design of our edge refiner and the modular integration of LLMs for enhancing both node features and graph structure. The reported model-side time does not include complete LLM API end-to-end latency.

5.4 Ablation Study

We investigate the effectiveness of each component of EFS on the Cora and Citeseer datasets. The results are averaged over five runs. Specifically, we conduct separate experiments using the original node features h_o with original graph structure (denoted as $h_o + A$), the summarized keywords features h_s with the original graph structure (denoted as $h_s + A$), the ranked prediction features h_p with original graph structure (denoted as $h_p + A$), all the features $h = h_o/h_s/h_p$ with the original graph structure (denoted as $h + A$), all the features with the original graph structure and added edges (denoted as $h + A + A^{add}$), all the features with the original graph structure and deleted edges (denoted as

Method	Cora	Citeseer
$h_o + A$	0.8708 ± 0.0105	0.7765 ± 0.0184
$h_s + A$	0.8576 ± 0.0118	0.7761 ± 0.0133
$h_p + A$	0.8808 ± 0.0157	0.7664 ± 0.0109
$h + A$	0.9007 ± 0.0120	0.8031 ± 0.0101
$h + A + A^{add}$	0.9126 ± 0.0094	0.8561 ± 0.0069
$h + A - A^{del}$	0.9118 ± 0.0086	0.8228 ± 0.0072
$h + A + A^{add} - A^{del}$	0.9164 ± 0.0023	0.8749 ± 0.0081

Table 4: Accuracy results of ablation study.

$h + A - A^{del}$), and all the features with the enhanced graph structure (denoted as $h + A + A^{add} - A^{del}$).

The results in Table 4 show that using only the original node features ($h_o + A$) provides a moderate baseline performance. Only using summarized keywords ($h_s + A$) or ranked prediction embeddings ($h_p + A$) is not enough for enhancing accuracy, but combining all the features ($h + A$) such as original attributes, keywords, and ranked predictions yields a substantial improvement. This demonstrates that integrating diverse feature types fosters a more comprehensive understanding of node characteristics, highlighting the value of semantic information extracted by LLMs in enriching node representations.

Enhancing the graph structure further boosts the accuracy. Adding edges ($h + A + A_{add}$) to address missing connections improves performance by reinforcing the graph’s topology. Similarly, removing spurious edges ($h + A - A_{del}$) increases accuracy by eliminating noisy or irrelevant links. The best results, however, are achieved through a dual approach that combines added and deleted edges to optimize the adjacency matrix ($h + A + A_{add} - A_{del}$). This refinement effectively balances the graph structure, capturing latent node relationships while reducing noisy links.

Under 20% random edge corruption, graph refinement improves Cora from 84.37 ± 2.51 to 87.21 ± 2.06 and Citeseer from 75.94 ± 2.33 to 79.29 ± 2.78 . In addition, full EFS retains 75.74% accuracy on the LLM-wrong subset of Citeseer and 78.77% on Cora, indicating that the final GNN does not mechanically copy the LLM top-1 prediction.

This study underscores the value of a combined approach that enhances both node features and graph structure. Though individual improvements contribute to better accuracy, integrating enriched node features with a refined graph structure creates a synergistic effect, enabling a more comprehensive representation and superior performance in

node classification tasks.

6 Conclusion

In this paper, we propose a novel method called EFS to fully exploit the semantic information in the text attributes of nodes and uncover the implicit dependencies within the graph structure. Extensive experiments have demonstrated superior results of EFS, confirming that LLMs could achieve simultaneous improvement of node features and graph structure without incurring significant additional costs. In the future, we will adapt our method to more downstream tasks in the graph domain.

Limitations

In this paper, we use LLMs to output keywords and ranked predictions along with their explanations for each node, which rely on the prediction ability of the LLMs, like the other GNN+LLM methods. So we select a relatively simple LLM, gpt-3.5-turbo-0125, as the predictor for all methods in our experiments. This ensures that our approach is evaluated under the same condition as the baselines, providing a controlled comparison that isolates the effect of LLM differences. Errors or biases in LLM-generated outputs may propagate through feature enhancement, graph refinement, and downstream message passing, especially in sensitive or user-facing applications. Moreover, the current experiments mainly focus on commonly used homophilous TAGs; generalization to heterophilous or relation-driven graphs requires further study.

Acknowledgments

This work was supported by National Natural Science Foundation of China (No. 62206038), the Strategic Priority Research Program of the Chinese Academy of Sciences (No. XDA0490301), Liaoning Binhai Laboratory Project (No. LBLF-2023-01), Xiaomi Young Talents Program, and the Interdisciplinary Institute of Smart Molecular Engineering, Dalian University of Technology. We also would like to thank Shanghai Mosuan Guichao Technology Company for providing inclusive computing power and technical support.

References

Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind

- Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, and 12 others. 2020. Language models are few-shot learners. In *Advances in Neural Information Processing Systems* 33.
- Runjin Chen, Tong Zhao, Ajay Kumar Jaiswal, Neil Shah, and Zhangyang Wang. 2024. LLaGA: Large language and graph assistant. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pages 7809–7823. PMLR.
- Eli Chien, Wei-Cheng Chang, Cho-Jui Hsieh, Hsiang-Fu Yu, Jiong Zhang, Olgica Milenkovic, and Inderjit S. Dhillon. 2022. Node feature extraction by self-supervised multi-scale neighborhood prediction. In *The Tenth International Conference on Learning Representations*.
- Enyan Dai, Wei Jin, Hui Liu, and Suhang Wang. 2022. Towards robust graph neural networks for noisy graphs with sparse labels. In *Proceedings of the Fifteenth ACM International Conference on Web Search and Data Mining*, pages 181–191.
- Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 4171–4186.
- Yi Fang, Dongzhe Fan, Daochen Zha, and Qiaoyu Tan. 2024. GAUGLLM: Improving graph contrastive learning for text-attributed graphs with large language models. In *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pages 747–758.
- 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.
- Zirui Guo, Lianghao Xia, Yanhua Yu, Yuling Wang, Zixuan Yang, Wei Wei, Liang Pang, Tat-Seng Chua, and Chao Huang. 2024. GraphEdit: Large language models for graph structure learning. *arXiv*, 2402.15183.
- Pengcheng He, Xiaodong Liu, Jianfeng Gao, and Weizhu Chen. 2021. DeBERTa: Decoding-enhanced BERT with disentangled attention. In *The Ninth International Conference on Learning Representations*.
- Xiaoxin He, Xavier Bresson, Thomas Laurent, Adam Perold, Yann LeCun, and Bryan Hooi. 2024. Harnessing explanations: LLM-to-LM interpreter for enhanced text-attributed graph representation learning. In *The Twelfth International Conference on Learning Representations*.
- Weihua Hu, Matthias Fey, Marinka Zitnik, Yuxiao Dong, Hongyu Ren, Bowen Liu, Michele Catasta, and Jure Leskovec. 2020. Open graph benchmark: Datasets for machine learning on graphs. In *Advances in Neural Information Processing Systems* 33.
- Thomas N. Kipf and Max Welling. 2017. Semi-supervised classification with graph convolutional networks. In *The Fifth International Conference on Learning Representations*.
- Guohao Li, Matthias Müller, Bernard Ghanem, and Vladlen Koltun. 2021a. Training graph neural networks with 1000 layers. In *Proceedings of the 38th International Conference on Machine Learning*, pages 6437–6449.
- Shuangli Li, Jingbo Zhou, Tong Xu, Liang Huang, Fan Wang, Haoyi Xiong, Weili Huang, Dejing Dou, and Hui Xiong. 2021b. Structure-aware interactive graph neural networks for the prediction of protein-ligand binding affinity. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*, pages 975–985.
- Hao Liu, Jiarui Feng, Lecheng Kong, Ningyue Liang, Dacheng Tao, Yixin Chen, and Muhan Zhang. 2024. One for all: Towards training one graph model for all classification tasks. In *The Twelfth International Conference on Learning Representations*.
- Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019. RoBERTa: A robustly optimized BERT pretraining approach. *arXiv preprint arXiv:1907.11692*.
- Zhenghao Liu, Chenyan Xiong, Maosong Sun, and Zhiyuan Liu. 2020. Fine-grained fact verification with kernel graph attention network. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 7342–7351.
- Andrew Kachites McCallum, Kamal Nigam, Jason Rennie, and Kristie Seymore. 2000. Automating the construction of internet portals with machine learning. *Information Retrieval*, 3(2):127–163.
- Péter Mernyei and Cătălina Cangea. 2020. Wiki-CS: A wikipedia-based benchmark for graph neural networks. *arXiv*, 2007.02901.
- Alessio Miaschi and Felice Dell’Orletta. 2020. Contextual and non-contextual word embeddings: an in-depth linguistic investigation. In *Proceedings of the 5th Workshop on Representation Learning for NLP*, pages 110–119.
- Seth A. Myers, Aneesh Sharma, Pankaj Gupta, and Jimmy Lin. 2014. Information network or social network? the structure of the twitter follow graph. In *Companion Proceedings of the 23rd International Conference on World Wide Web*, pages 493–498.

- OpenAI. 2024. GPT-3.5 Turbo model documentation. OpenAI API documentation. Documents the gpt-3.5-turbo-0125 snapshot; accessed 2026-08-22.
- Xipeng Qiu, Tianxiang Sun, Yige Xu, Yunfan Shao, Ning Dai, and Xuanjing Huang. 2020. Pre-trained models for natural language processing: A survey. *arXiv*, 2003.08271.
- Xubin Ren, Jiabin Tang, Dawei Yin, Nitesh V. Chawla, and Chao Huang. 2024. A survey of large language models for graphs. In *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pages 6616–6626.
- Prithviraj Sen, Galileo Namata, Mustafa Bilgic, Lise Getoor, Brian Gallagher, and Tina Eliassi-Rad. 2008. Collective classification in network data. *AI Magazine*, 29(3):93–106.
- 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.
- Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, Aurélien Rodriguez, Armand Joulin, Edouard Grave, and Guillaume Lample. 2023a. LLaMA: Open and efficient foundation language models. *arXiv*, 2302.13971.
- Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajwal Bhargava, Shruti Bhosale, Dan Bikel, Lukas Blecher, Cristian Canton Ferrer, Moya Chen, Guillem Cucurull, David Esiobu, Jude Fernandes, Jeremy Fu, Wenyin Fu, and 49 others. 2023b. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*.
- Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. 2018. Graph attention networks. In *The Sixth International Conference on Learning Representations*.
- Fengyi Wang, Guanghui Zhu, Chunfeng Yuan, and Yihua Huang. 2024. LLM-enhanced cascaded multi-level learning on temporal heterogeneous graphs. In *Proceedings of the 47th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 512–521.
- Junhan Yang, Zheng Liu, Shitao Xiao, Chaozhao Li, Defu Lian, Sanjay Agrawal, Amit Singh, Guangzhong Sun, and Xing Xie. 2021. GraphFormers: GNN-nested transformers for representation learning on textual graph. In *Advances in Neural Information Processing Systems 34*.
- Ruosong Ye, Caiqi Zhang, Runhui Wang, Shuyuan Xu, and Yongfeng Zhang. 2024. Language is all a graph needs. In *Findings of the Association for Computational Linguistics: EACL 2024*, pages 1955–1973. Association for Computational Linguistics.
- Jason Zhu, Yanling Cui, Yuming Liu, Hao Sun, Xue Li, Markus Pelger, Tianqi Yang, Liangjie Zhang, Ruofei Zhang, and Huasha Zhao. 2021. TextGNN: Improving text encoder via graph neural network in sponsored search. In *Proceedings of the Web Conference*, pages 2848–2857.
- Yun Zhu, Yaoke Wang, Haizhou Shi, and Siliang Tang. 2024. Efficient tuning and inference for large language models on textual graphs. In *Proceedings of the Thirty-Third International Joint Conference on Artificial Intelligence*, pages 5734–5742.

A Appendix

A.1 Baselines

We compare EFS with 11 baselines, which can be divided into 3 kinds of methods.

(1) **GNN**: GCN (Kipf and Welling, 2017) and GAT (Veličković et al., 2018) are popular GNNs for learning graph-structured data. GCN extends convolutional operations to graphs, aggregating information from neighboring nodes. GAT improves this with an attention mechanism to weigh neighbors differently, enhancing accuracy and interpretability. Here we adopt an advanced GAT variant called RevGAT (Li et al., 2021a), and follow their experimental settings to use shallow embeddings.

(2) **LLM**: GPT (OpenAI, 2024) is an advanced LLM that could understand and generate human-like text. Here we use gpt-3.5-turbo-0125 as our baseline with the same prompts as EFS.

(3) **GNN+LLM**: this kind of methods can be further divided into 4 types.

Graph Representation Learning: Recent methods like GIANT (Chien et al., 2022) and TAPE (He et al., 2024) focus on enhancing node features by combining LLM with GNN for node classification tasks. GIANT leverages neighborhood prediction to pre-train and fine-tune language models, improving node feature representation. TAPE enriches node features by using LLMs to generate class predictions and provide explanatory context.

Graph Structure Learning: Recent methods like GraphEdit (Guo et al., 2024) focus on enhancing graph structure by integrating LLM with GNN for node classification tasks. GraphEdit enhances graph structure by utilizing LLM to capture complex node relationships and optimize the graph. It fine-tunes the LLM by predicting whether each pair

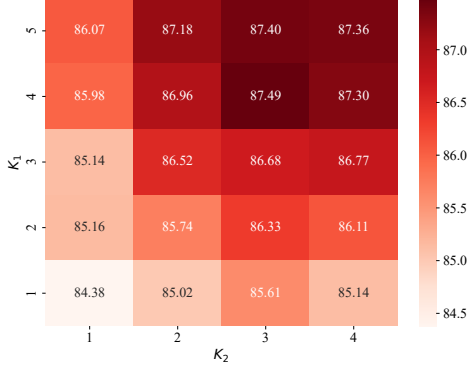


Figure 3: Effect of different K_1 and K_2 on node classification accuracy of Citeseer.

of nodes belong to the same category, then takes it as the edge refiner to optimize the graph structure.

Integrated Graph Representation and Structure Learning: GAUGLLM (Fang et al., 2024) combines LLM and GNN to enable self-supervised learning through graph contrastive learning. This method simultaneously enhances node features and graph structure, but it does not use any annotated data during graph representation and structure learning, which is different from our supervised setting. Node feature enhancement leverages multiple prompt experts to perturb the original text attributes, and integrates various features through weighted fusion. Structural enhancement uses an edge modifier prompt to query LLM for determining the connectivity of any two nodes based on their semantic similarity.

Others: We also compare additional advanced methods that utilize LLM in graph learning tasks, including OFA (Liu et al., 2024), InstructGLM (Ye et al., 2024), ENGINE (Zhu et al., 2024), and LLaGA (Chen et al., 2024). OFA standardizes graph data from diverse domains to support cross-domain learning. InstructGLM designs prompts that enable LLMs to handle graph issues directly. ENGINE integrates LLM with GNN for efficient model training. LLaGA transforms graph data into sequences to facilitate processing graphs by LLM.

A.2 Influence of Edge Choice

The significance of graph structure in node classification tasks is evident from our previous experiments. To further investigate this, we aim to explore how the hyperparameters K_1 and K_2 in the graph structure enhancement affect node classification accuracy. Specifically, we analyze the performance of the Citeseer dataset by searching

the grid $K_1 \in \{1, 2, 3, 4, 5\}$ and $K_2 \in \{1, 2, 3, 4\}$. The results are shown in Figure 3.

As K_1 and K_2 increase, classification accuracy initially improves, peaking at $K_1 = 4$ and $K_2 = 3$, before declining with further increases. This trend highlights the need of balancing edge additions and deletions. When K_1 is small, the limited edge adjustments may fail to sufficiently enhance the graph structure, leaving meaningful node relationships unaddressed. In contrast, larger K_1 values risk introducing excessive adjustments, potentially adding noise or removing valuable connections, which weakens the model’s ability to propagate meaningful information. The choice of K_2 also significantly impacts edge relevance based on ranked categories. At $K_2 = 3$, the overlap of top-ranked categories provides a reliable criterion for assessing node similarity and determining edges to be adjusted. However, as K_2 exceeds this optimal value, incorporating lower-ranked categories increases the risk of adding noisy or less relevant edges, ultimately reducing accuracy.

So we set $K_1 = 4$ and $K_2 = 3$ to obtain favourable results for Cora, Citeseer, ogbn-arxiv, ogbn-products, and Wiki-CS. As there are only three classes in Pubmed, the number of ranked categories is usually much smaller than three, so we set $K_2 = 1$, and remain $K_1 = 4$ for Pubmed.

A.3 Case Study

In order to explore the reasons for the specific impact of K_1 and K_2 in graph structure enhancement, we take the Citeseer dataset as an example and select a first-order subgraph of a node A, where nodes denote papers, and edges indicate citation relationships, as shown in Figure 4. The categories of the nodes are as follows: A, B, J, E, and F belong to “Machine Learning”, C, D, and H belong to “Information Retrieval”, G belongs to “Human-Computer Interaction”, and I belongs to “Database”. Solid lines between nodes indicate edges that originally existed and are retained, dashed lines represent newly added edges that did not exist initially, and a cross on an edge denotes an edge that previously existed but has been removed.

As shown in Figure 4 (a), we adjust the edges of node A under the condition of $K_1 = 4$, $K_2 = 3$. It can be seen that some edges of nodes that are originally highly correlated with node A are retained, such as A-J, etc. Nodes that have very low correlation with node A and are falsely connected are successfully identified and the edges between

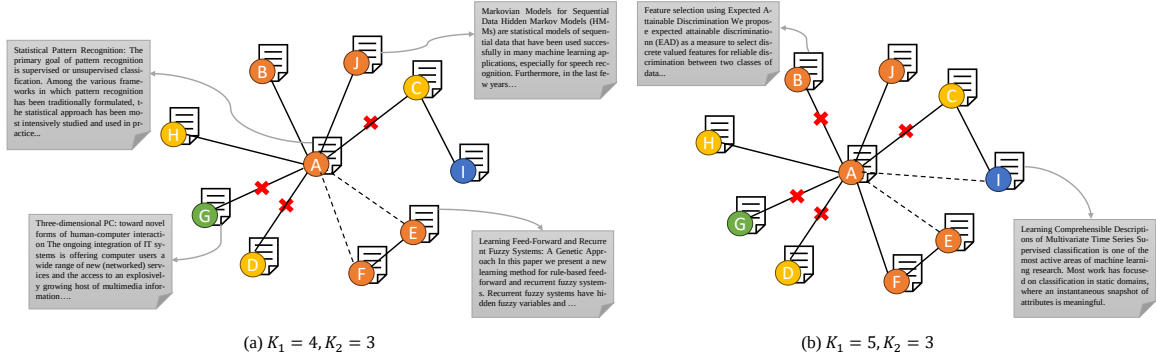


Figure 4: The impact of K_1 and K_2 on enhancing graph structure of Citeseer.

them are disconnected, e.g., A-G, etc. Highly relevant nodes that are not connected with node A are also successfully identified and connections are established, e.g., A-E, etc.,

As shown in Figure 4 (b), we then adjust the edges for node A under the setting $K_1 = 5, K_2 = 3$. It can be seen that when K_1 is increased, there exists a node B that has a higher correlation with node A. The connection between them is incorrectly disconnected, so that effective information propagation between nodes is lost. It can also be found that node I, which has a relatively low correlation with node A is incorrectly identified, which not only introduces noisy edges but also invalidates the originally disconnected A-C edge due to the existence of edge between node C and node I.

This example shows that a reasonable edge selection strategy can optimize the graph structure and improve the model performance. Moderately adding highly relevant edges and deleting irrelevant edges can help to enhance the quality of information propagation and bring significant improvement to the node classification task, while excessive edge addition and deletion may lead to information loss or introduce noise.

A.4 Examples of Prompts and LLM Outputs

The keyword and prediction questions in Table 5 are issued as two separate calls with the same dataset-specific context. For scientific-paper datasets, the input contains the title and abstract. For ogbn-products, the instruction is changed to product-category classification, the input contains the product title and description, and all 47 valid Amazon categories are listed. Responses are parsed using the requested Keywords:, Ranked Predictions:, and Explanation: fields; ranked labels are matched to the fixed dataset category list,

while explanations are retained as generated text. The raw and generated strings are tokenized independently by their respective encoder tokenizers.

For EdgeRefiner training, positive and negative pairs are drawn only from same-class and different-class pools within $\mathcal{V}_{\text{train}}$. At inference, addition candidates are the top- M ($M = 50$) cosine-similar non-neighbors found by blockwise retrieval, whereas deletion candidates are existing neighbors. All graph updates are symmetrized and the refined benchmark graphs are treated as undirected.

You are an expert in **{specific field}**. You will be provided with an abstract and a title for a paper.

Abstract: **{paper abstract}**

Title: **{paper title}**

Keywords Question: Please summarize the keywords from this paper by analyzing the title and abstract. For each identified keyword, provide a brief description in the following format: "Keyword1: Description, Keyword2: Description, ...".

Prediction Question: Which of the following category does this paper belong to: **{paper category}**. If multiple categories apply, provide a comma-separated list ordered from most to least relevant as "Ranked Predictions:". For each chosen category, offer a detailed explanation with quotes from the text to justify its relevance under "Explanation:".

Table 5: Format of the prompt.

A.5 Additional Results

Table 7 evaluates robustness to LLM and text-encoder choices and presents additional comparisons. The low-label row uses only 10% training labels.

Keywords:

1. sequential consistency: A memory model ensuring that all memory operations appear to occur in some total order, consistent with the order seen by each individual processor.
 2. shared-memory multiprocessors: A system where multiple processors share a common memory space, allowing them to communicate and coordinate through memory operations.
- ...

Ranked Predictions:

DB, AI

Explanation:

1. Database (DB): This paper can be most likely categorized under database research because it deals with the memory model of shared-memory multiprocessors ...
2. Artificial Intelligence (AI): While the paper primarily focuses on memory systems and model checking, it also touches upon concepts related to causality and data independence ...

Table 6: Example of keywords, ranked predictions and explanations for a paper in the Citeseer dataset.

Setting	Variant	Cora	Citeseer
LLM	GPT-3.5	91.64 $\pm$ 0.23	87.49 $\pm$ 0.81
	LLaMA2-7B-Chat	89.97 $\pm$ 0.84	84.18 $\pm$ 1.26
Encoder	BERT-base	89.72 $\pm$ 1.06	84.93 $\pm$ 1.37
	RoBERTa-base	90.86 $\pm$ 0.74	86.58 $\pm$ 1.02
	DeBERTa-base	91.64 $\pm$ 0.23	87.49 $\pm$ 0.81
Other	Class-prob. rewiring	85.44 $\pm$ 3.05	82.81 $\pm$ 3.29
	Text- k NN graph	90.56 $\pm$ 0.92	83.74 $\pm$ 0.83
	Non-LLM structure	90.41 $\pm$ 0.88	84.31 $\pm$ 0.79
	EFS (10% labels)	85.84 $\pm$ 1.73	82.78 $\pm$ 0.90
	EFS	91.64 $\pm$ 0.23	87.49 $\pm$ 0.81

Table 7: Accuracy (%) for backbone robustness and additional comparisons.

A.6 Efficiency and Scalability Details

Candidate-Pair Efficiency. Table 8 reports the number of candidate node pairs actually scored by the EdgeRefiner and the corresponding runtime. *Pairs* includes the addition and deletion candidates retained after candidate generation, while *Reduction* is the ratio of all unordered node pairs to *Pairs*. *Search* and *MLP* denote the time for blockwise top- M addition-candidate retrieval and candidate-pair scoring, respectively; neither includes EdgeRefiner training or graph reconstruction.

Dataset	Pairs	Reduction	Search	MLP
Citeseer	167,854	30 $\times$	0.8 s	2.4 s
ogbn-arxiv	10,782,748	1,329 $\times$	35.2 s	46.1 s
ogbn-products	2,845,888	512 $\times$	9.3 s	13.7 s

Table 8: Candidate-pair construction and EdgeRefiner scoring efficiency. Reduction is relative to exhaustive scoring over all unordered node pairs.

Model-Side Runtime and Complexity. On Citeseer, the precise post-LLM model-side runtime is 20.20 minutes (rounded to 20 minutes in Figure 2), consisting of 14.52 minutes for text-encoder fine-tuning and embedding export, 1.28 minutes for EdgeRefiner training, 0.8 seconds for candidate search and 2.4 seconds for MLP scoring (the corresponding entries in Table 8), 1.9 seconds for graph reconstruction, and 4.31 minutes for the RevGAT branches and fusion. The same post-LLM pipeline takes 411 minutes on ogbn-arxiv. EFS has 568.35M trainable parameters, including 559.9M in three independently fine-tuned DeBERTa encoders, 1.31M in the EdgeRefiner, and approximately 7.14M in the three RevGAT branches. Since text embeddings and the refined graph are cached, online GNN inference uses only the RevGAT branches and fusion module. On Citeseer, one full-batch GNN inference requires 46.8 GFLOPs and takes 127.1 ± 4.3 ms on an A800, averaged over 15 runs.

LLM Calls and Token Usage. EFS makes $2|\mathcal{V}|$ one-time node-level API calls and zero LLM calls per edge or training round. Under the same API token-counting protocol, EFS uses 138,881,159 prompt tokens and 78,544,448 completion tokens (217,425,607 total; 834.07 tokens per node), whereas TAPE uses 80,555,887 prompt and 42,170,349 completion tokens (122,726,236 total; 470.79 per node). Thus, EFS uses $1.77\times$ TAPE’s tokens because it generates both keyword- and prediction-based semantics. Monetary API cost is not reported because the corresponding billing records are unavailable.