# Efficient Tuning and Inference for Large Language Models on Textual Graphs

**Yun Zhu**[1,], **Yaoke Wang**[1,], **Haizhou Shi**[2] and **Siliang Tang**[1,†]
[1]Zhejiang University
[2]Rutgers University
{zhuyun_dcd, wangyaoke}@zju.edu.cn, haizhou.shi@rutgers.edu, siliang@zju.edu.cn

## Abstract

Rich textual and topological information of textual graphs need to be modeled in real-world applications such as webpages, e-commerce, and academic articles. Practitioners have been long following the path of adopting a shallow text encoder and a subsequent graph neural network (GNN) to solve this problem. In light of recent advancements in large language models (LLMs), it is apparent that integrating LLMs for enhanced textual encoding can substantially improve the performance of textual graphs. Nevertheless, the efficiency of these methods poses a significant challenge. In this paper, we propose ENGINE, a *parameter- and memory-efficient fine-tuning method* for textual graphs with an LLM encoder. The key insight is to combine the LLMs and GNNs through a *tunable side structure*, which significantly reduces the training complexity without impairing the joint model's capacity. Extensive experiments on textual graphs demonstrate our method's effectiveness by achieving the best model performance, meanwhile having the lowest training cost compared to previous methods. Moreover, we introduce two variants with caching and dynamic early exit to further enhance training and inference speed. Specifically, caching accelerates ENGINE's training by 12x, and dynamic early exit achieves up to 5x faster inference with a negligible performance drop (at maximum 1.17% relevant drop across 7 datasets). Our codes are available at: https://github.com/ZhuYun97/ENGINE

## 1 Introduction

Textual graphs are pervasive in the real world, like academic networks [Wang *et al.*, 2020], webpages [Mernyei and Cangea, 2020] and e-commerce datasets [Chiang *et al.*, 2019]. In the early stages, shallow embedding methods [Mikolov *et al.*, 2013; Zhang *et al.*, 2010] and graph neural networks [Kipf and Welling, 2017; Hamilton *et al.*, 2017; Veličković *et al.*, 2018] are combined to analyze and process this data like Figure 2a. However, static shallow embedding methods struggle to capture context-aware information and complex semantic relationships, limiting their ability to exploit the richness of text attributes, particularly in graph tasks.

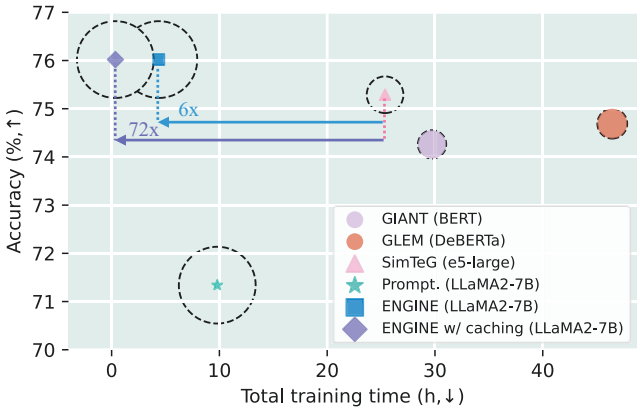


Figure 1: Comparison of performance and training efficiency between ENGINE and baselines on the large-scale textual graph dataset OGBN-ArXiv, where the x-axis denotes total training time and the y-axis denotes accuracy. Here, the radius of dashed circles is proportional to the 4th Root of the parameters in the incorporated language models, and the radius of the internal marker is proportional to the number of tunable parameters. Our method with caching achieves a remarkable 72x faster training compared the previous SoTA method, SimTeG, while simultaneously achieving superior performance. Please refer to Table 3 for more detailed results.

Large language models (LLMs) such as ChatGPT [OpenAI, 2023], LLaMA2 [Touvron *et al.*, 2023], and Falcon [Almazrouei *et al.*, 2023] have recently demonstrated significant potential in understanding language, effectively capturing the semantic richness of text attributes. This success has prompted researchers to utilize LLMs on textual graphs for enhanced performance. Some works [Chien *et al.*, 2021; He *et al.*, 2023b; Duan *et al.*, 2023] explore the application of Language Models (LMs) on textual graphs to improve node features. These features are subsequently aggregated by GNNs to generate the final representation like Figure 2b. However, these approaches, as pointed out by many preceeding researchers [Zhao *et al.*, 2022; Xue *et al.*, 2023;

†Corresponding Author

Pan *et al.*, 2024b], may be sub-optimal: the nodes' textual and structural features are encoded separately by the LLMs and GNNs at two different stages. An apparent solution to this issue is the joint training of LMs and GNNs, while this option introduces the efficiency challenges, as memory and time complexity during training and inference may become prohibitively expensive for both institutions and customers.

This work aims to provide an efficient and effective solution to enable the joint modeling of the textual and topological information on textual graphs. We propose an efficient tuning algorithm for large language models on textual graphs, named ENGINE, to address the challenges mentioned above. As illustrated in Figure 2d, during training, we freeze the parameters of LLMs and add a tiny tunable ladder (*G-Ladder*) alongside each layer of LLMs. Within each G-Ladder, we adopt message passing to integrate structural information, thereby enhancing the quality of node representations. The key idea behind the G-Ladder is that only an extremely small portion of the parameters are updated, resulting in a significant reduction in memory consumption. Furthermore, since the parameter update of ENGINE does not depend on the gradient computation of the LLMs, which allows us to precompute node embeddings, storing them in the cache for subsequent reuse, thereby significantly reducing the time complexity during training, as shown in Figure 1. In the inference phase, where LLM-induced latency is typically high, we introduce an early exit operator post each ladder to expedite the inference process. This variant, termed ENGINE (Early), facilitates dynamic early exit of node embedding encoding, accelerating inference and mitigating the risk of overthinking [Zhou *et al.*, 2020]. Through extensive experiments, our method demonstrates SoTA performance compared to baselines on textual graphs across various domains. Compared with existing LLM-based approaches, our proposed framework significantly improves training efficiency and reduces inference latency. This suggests a promising direction for combining LLMs with GNNs in textual graph analysis.

Our contributions can be concluded as:

- We propose a memory- and time-efficient tuning method for LLMs on textual graphs named ENGINE. The LLMs and GNNs are combined through a *tunable side structure* which significantly reduces the training complexity without impairing the joint model's capacity.
- Two additional variants of ENGINE are proposed to further improve the training and inference: (i) ENGINE (caching) precomputes node embeddings for training samples, places them into cache, and reuses them during training; and (ii) dynamic early exit dynamically determines whether to exit the feed forward of the LLM layers and returns the prediction.
- Extensive experiments demonstrate the effectiveness of our method with the lowest training cost compared to previous methods. Moreover, ENGINE with caching speeds up training time by 12x, and ENGINE (Early) achieves up to 5x faster inference with a slight maximum performance drop of 1.17% across 7 datasets.

# 2 Related Works

## 2.1 LM-based Methods for Textual Graphs

Representation learning for textual graphs has gained significant attention in the field of graph machine learning. Rather than relying solely on shallow or hand-crafted features (*e.g.*, bag-of-words, skip-gram) as seen in previous works [Kipf and Welling, 2017; Veličković *et al.*, 2018; Hamilton *et al.*, 2017; Zhu *et al.*, 2023a; Zhu *et al.*, 2023b; Zhu *et al.*, 2023c] , LM-based methods harness the potential power of Language Models (LMs) to handle textual graphs. These methods combine LMs and GNNs in a cascading (Figure 2b) or iterative structure (Figure 2c). Specifically, in cascading structure methods, the initial step involves fine-tuning pre-trained language models on downstream tasks. Subsequently, these fine-tuned models are employed to generate text embeddings, serving as the initial node features for GNNs. For example, GIANT [Chien *et al.*, 2021] enhances the meaningfulness of node embeddings by fine-tuning LMs using neighbor information. SimTeG [Duan *et al.*, 2023] adopts a two-stage training paradigm: initially fine-tuning LMs through graph-related tasks (*e.g.*, node classification, link prediction), and subsequently training GNNs based on the embeddings generated by LMs. TAPE [He *et al.*, 2023b] utilizes LLMs to augment text descriptions and fine-tunes PLMs based on these augmented texts. The subsequent step involves using the fine-tuned PLMs to generate text embeddings, serving as the initial node attributes for GNNs. These methods only model partial information, limiting their ability to learn comprehensive features.

Instead of adopting the cascading structure, GraphFormers [Yang *et al.*, 2021], GLEM [Zhao *et al.*, 2022], and LEADING [Xue *et al.*, 2023] combine LMs and GNNs in an iterative or co-training structure, training LLMs and GNNs in a co-training way. However, these co-training paradigms face significant scalability issues. Encoding all neighbors by LMs introduces high-cost fine-tuning and inference overhead due to the substantial number of parameters in language models.

To address these non-trivial challenges, we propose to combine LMs and GNNs in a *side structure* as depicted in Figure 2d. We introduce a novel method designed to implement a *parameter- and memory-efficient tuning method* for LLMs on textual graphs named ENGINE.

## 2.2 Parameter-Efficient Fine-Tuning (PEFT)

Recently, large language models (LLMs) have achieved remarkable success in NLP domain [OpenAI, 2023; Touvron *et al.*, 2023]. PEFT methods aim to adapt large pre-trained models to various downstream applications without fine-tuning all parameters due to prohibitive costs. For instance, adapter-based methods [Houlsby *et al.*, 2019] inject lightweight neural networks (adapters) into Transformer layers and exclusively fine-tune these adapters for model adaptation. LoRA [Hu *et al.*, 2021] assumes a low intrinsic rank in weight changes during model tuning. It proposes optimizing the decomposition of original weight matrices in self-attention modules, multiplying optimized matrices during deployment to obtain the delta of self-attention weights, thus significantly reducing the parameters requir-

ing fine-tuning. Similarly, IA3 [Liu *et al.*, 2022] modifies attention weights for both key and value, as well as the feedforward activation, through element-wise multiplication. Prompt-based tuning [Lester *et al.*, 2021; Li and Liang, 2021; Pan *et al.*, 2023] prepends trainable soft prompts to the input text while keeping the entire pre-trained models unchanged. While most methods mainly focus on achieving competitive performance through tuning few parameters (parameter-efficient), they often fall short in terms of memory efficiency. Ladder Side-Tuning (LST) [Sung *et al.*, 2022] addresses this issue by training a ladder-side network that separates trainable parameters from the backbone model. Because it does not require backpropagation through the backbone network (LMs), LST significantly reduces memory requirements.

Although PEFT has achieved great success in the Euclidean domain including natural language and computer vision [Jia *et al.*, 2022; He *et al.*, 2023a; Pan *et al.*, 2024a], how to *effectively* and *efficiently* apply such approaches to non-Euclidean domains like graphs is still under exploration. The main challenge lies in the lack of experience of how to fully utilize structural information to enhance node-level representations. In this work, we propose an efficient tuning method for LLMs on textual graphs, named ENGINE, which explicitly utilizes structural information and enhances node-level representations through our designed lightweight *G-Ladder*.

# 3 Method

In this section, we will introduce the notations used in this paper. Subsequently, we will present the proposed efficient tuning method for LLMs tailored to textual graphs in Section 3.2. Finally, efficient inference will be discussed in Section 3.3.

## 3.1 Notations

Given a textual graph $G = \{\mathcal{V}, \{t_n\}_{n\in\mathcal{V}}, A, Y\}$, where $\mathcal{V}$ is the node set consisting of $N$ instances, $t_n \in \mathcal{T}^{Q_n}$ represents a sequential text for its node $n \in \mathcal{V}$, $\mathcal{T}$ is the tokens dictionary, and $Q_n$ is the sequence length, $A \in \mathbb{R}^{N\times N}$ denotes the adjacency matrix, and $Y$ denotes labels for each node. To enhance scalability, a sampling function $\Gamma(\cdot)$ is applied to a large graph to obtain a set of small subgraphs $\{G_n\}_{n\in\mathcal{V}}$, where $G_n$ represents the subgraph for the target node $n \in \mathcal{V}$. In this study, the focus is on the node classification task of textual graphs. Specifically, given a set of training nodes $\mathcal{V}_{tr}$, a classification model is trained on these nodes and evaluated on the remaining test nodes $\mathcal{V}_{te}$. Formally, given a set of training nodes and their induced subgraphs $\mathcal{G}^{tr} = \{G_n\}_{n\in\mathcal{V}_{tr}}$, the optimal predictor $f_{\theta^*}$ is formulated as

$$f_{\theta^*} \in \arg\max_{\theta} \mathbb{E}_{G_n\in\mathcal{G}^{tr}} P_\theta(\hat{y}_n = y_n \mid G_n), \tag{1}$$

where $y_n$ denotes the true label of target node $n \in \mathcal{V}_{tr}$ and $\hat{y}_n$ is the predicted label. It is noteworthy that our method can be applied to other tasks as well, including graph classification and link prediction. For instance, in graph classification, each instance aligns with a graph, analogous to the subgraph of the target node in the node classification task. The exploration of these applications is left for future work.

## 3.2 Efficient Tuning for LLM on Textual Graphs

The efficient tuning of large language models for graph data is under exploration [Duan *et al.*, 2023; Xue *et al.*, 2023]. However, existing methods are parameter-efficient but not memory-efficient, primarily focusing on fine-tuning small-scale language models like BERT [Kenton and Toutanova, 2019] and DeBERTa [He *et al.*, 2020] for textual graphs. Moreover, these methods encounter challenges in effectively utilizing structural information explicitly during fine-tuning. In this work, we present a parameter-efficient and memory-efficient tuning method for LLMs while also leveraging structural knowledge explicitly, as depicted in Figure 2 (Right).

Given an input graph $G$, a sampling function $\Gamma(\cdot)$, such as random walk with restart [Qiu *et al.*, 2020; Zhu *et al.*, 2022], will be applied to obtain subgraphs $\mathcal{G}^{tr} = \{G_i\}_{i\in\mathcal{V}_{tr}}$ for target nodes. Each subgraph consists of several nodes, and each node contains a textual description $t_i$. The textual descriptions of nodes within a subgraph will be packed into a batch $\mathcal{B} = \{t_i\}_{i=1}^{B}$ and fed into LLMs. In each layer of LLM, token-level representations of each node will be calculated:

$$H^l = \mathrm{LLM\_Layer}^l(H^{l-1}), \tag{2}$$

where $\mathrm{LLM\_Layer}^l(\cdot)$ denotes the $l$-th layer of LLM, and $H^l \in \mathbb{R}^{B\times Q\times D}$ means the token-level representations in $i$-th layer. In the first layer, the input $H^0$ is equivalent to $\mathcal{B}$.

**Lightweight *G-Ladders***

In this work, we focus on node-level tasks, intricately related to the quality of node-level representations. Our goal is to enhance these representations during model tuning. To achieve this, a readout function $\mathcal{R}$, such as mean pooling [Mesquita *et al.*, 2020], is applied to token-level representations:

$$z_i^l = \mathcal{R}(h_{i,1}^l, h_{i,2}^l, ..., h_{i,Q}^l), \tag{3}$$

where $z_i^l \in \mathbb{R}^{1\times D}$ denotes the representation of node $i \in \mathcal{V}_{tr}$. Then these node-level representations are fed into GNN Ladders (*G-Ladders*) to improve the quality of node embeddings through structural information, ensuring accurate comprehension of the node's semantics from a global perspective. The improved node-level representations $\hat{Z}^l$ are derived as

$$\hat{Z}^l = \lambda^l \cdot \mathrm{GNN}^l\left(\mathcal{P}^l\left(Z^l\right), A\right) + \left(1-\lambda^l\right)\cdot \hat{Z}^{l-1}, \tag{4}$$

where $\mathcal{P}^l(\cdot) : \mathbb{R}^{B\times D} \to \mathbb{R}^{B\times K}$ is a projector that maps the node-level representations into low dimensions ($K \ll D$), thereby reducing the subsequent computation in G-Ladders. GNN is one type of graph neural networks that includes message passing. In this paper, for simplicity, we benchmark three different architectures, including GCN [Kipf and Welling, 2017], SAGE [Hamilton *et al.*, 2017], and GAT [Veličković *et al.*, 2018] and pick the best one (SAGE) for modeling G-Ladders. Nevertheless, the choice of the GNN architectures does not bring heavy influence to the final performance (refer to Table 11 in Appendix). $\lambda^l$ is a learnable coefficient for combining information in the current layer $l$ and the previous layer $l-1$. It is modeled by $\lambda^l = \mathrm{sigmoid}(\frac{\omega^l}{T})$, where $\omega^l$ is a learnable zero-initialized scalar, and $T$ is a temperature set as $0.1$.

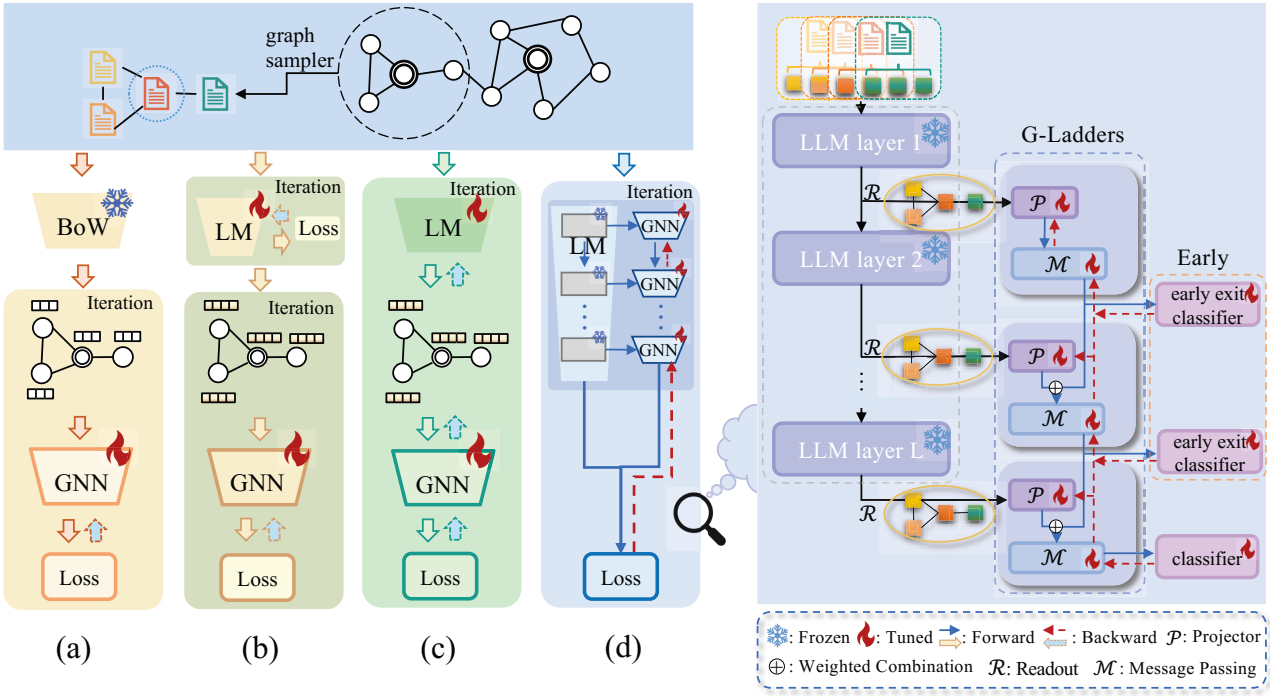


Figure 2: **Left:** strategies for processing textual graphs on graph tasks: (a) Static shadow embedding method and GNN. (b) Cascading Structure: LMs and GNNs are trained independently. (c) Iterative or Co-training Structure: LMs and GNNs are trained jointly. (d) Side Structure: Frozen LMs combined with tunable GNNs in a side structure. **Right:** detailed pipeline of ENGINE, where frozen LLM layers are combined with G-Ladders through a side structure. The dynamic early exit classifier is added after G-Ladders for ENGINE (Early).

Finally, the representation of the target node $\hat{z}_i^L$ from the final layer's output $\hat{Z}^L$ is fed into a linear classifier $\mathcal{C}$ for classification. The training loss $\mathcal{L}$ is computed using the cross-entropy loss $\mathrm{CE}(\cdot,\cdot)$ between predictions and true labels:

$$\mathcal{L} = \mathbb{E}_{i\in\mathcal{V}_{tr}}\,\mathrm{CE}(\hat{y}_i, y_i), \text{ where } \hat{y}_i = \mathcal{C}(\hat{z}_i^L). \tag{5}$$

**What Makes ENGINE An Efficient-Tuning Method?** As depicted in Figure 2 (Right), the efficiency of ENGINE stems from three aspects. First, similar to existing PEFT methods, the parameters of LLMs are frozen and only lightweight G-Ladders are updated during training, ensuring parameter-efficient. Second, our method innovatively integrates *G-Ladders* with LLMs via a *side structure*, eliminating the need for backpropagation through the LLMs. This contrasts with prior methods that insert additional tunable parameters within LLMs, requiring costly backpropagation memory, such as cascading or iterative structures [Chien *et al.*, 2021; Zhao *et al.*, 2022]. Last, our side-structure design allows us to precompute and cache node embeddings for reuse, further boosting computational efficiency. In short, our method is both parameter-efficient and memory-efficient, as demonstrated by empirical evidence in Section 4.5.

### 3.3 Efficient Inference for LLM on Textual Graphs

The high latency linked to LLM inference can hinder model deployment in real-world settings. Various strategies are proposed to speed up model inference, including model pruning [Gordon *et al.*, 2020], model distillation [Sanh *et al.*, 2019], and dynamic early exit [Xin *et al.*, 2020; Hu *et al.*, 2023].

In this work, for simplicity, we adopt a similar mechanism as dynamic early exit, which can be seamlessly integrated with our method. Specifically, we add a lightweight single-layer MLP as an early exit classifier $\mathcal{C}^l$ after each *G-Ladder*. During the model tuning, these classifiers are directly connected to the downstream task's training objective, *e.g.*, the cross-entropy loss between the true label $y$:

$$C^{l*} \in \arg\min_{\mathcal{C}^l} \mathbb{E}_{i\in\mathcal{V}_{tr}}\,\mathrm{CE}(\mathcal{C}^l(\hat{z}_i^l), y_i). \tag{6}$$

Later during inference, early exit will be triggered based on the classifiers' uncertainty measure, such as information entropy [Xin *et al.*, 2020; Hu *et al.*, 2023] and confidence [Xie *et al.*, 2021]. However, these criteria heavily depend on datasets, and careful selection is necessary for different datasets. For example, on simpler datasets, a lower value of information entropy may be chosen as the threshold, considering the trade-off between efficiency and performance. In this work, we utilize 'patience' criteria [Zhou *et al.*, 2020] to guide dynamic early exit, allowing us to avoid the need for meticulous design. Patience-based criteria imply that if consecutive $p$ early exit classifiers predict the same results, where $p$ represents the number of layers set at 2 across different datasets in our experiments, the model stops inference early and makes a prediction.

Since these early exit classifiers are also included in the side structure, the gradients continue to flow exclusively within the side structure as shown in Figure 2 (Right), making our method tuning-efficient and inference-efficient.

# 4 Experiments

In this section, we first introduce the datasets used in Section 4.1. Then, we will illustrate the baselines and experimental setup in Section 4.2 and 4.3 respectively, and conduct experiments on these datasets to demonstrate the effectiveness of our proposed method in Section 4.4. Then, we will study the training efficiency of ENGINE compared to other methods in Section 4.5. Lastly, the analysis of hyper-parameters and an ablation study will be provided. Furthermore, additional experiments (*e.g.*, link prediction) are in Appendix.

## 4.1 Datasets

In this work, we adopt seven commonly used textual graphs to evaluate our proposed ENGINE: Cora [Sen *et al.*, 2008], CiteSeer [Giles *et al.*, 1998], WikiCS [Mernyei and Cangea, 2020], OGBN-ArXiv [Hu *et al.*, 2020], ArXiv-2023 [He *et al.*, 2023b], OGBN-Products [Hu *et al.*, 2020] and Ele-Photo [Yan *et al.*, 2023]. We utilize collected raw text data of these datasets by previous works [Chen *et al.*, 2023; He *et al.*, 2023b; Yan *et al.*, 2023]. Details of these datasets can be found in Appendix A.

| Dataset | #Nodes | #Edges | #Classes |
|---|---|---|---|
| Cora | 2,708 | 5,429 | 7 |
| CiteSeer | 3,186 | 4,277 | 6 |
| WikiCS | 11,701 | 215,863 | 10 |
| OGBN-ArXiv | 169,343 | 1,166,243 | 40 |
| ArXiv-2023 | 46,198 | 78,543 | 40 |
| OGBN-Products (subset) | 54,025 | 74,420 | 47 |
| Ele-Photo | 48,362 | 500,928 | 12 |

Table 1: Statistics of textual graphs used in this work.

## 4.2 Baselines

To assess the effectiveness of our proposed method, 17 baselines in 5 main categories of approaches are employed. The 5 categories are: (i) traditional GNN models, (ii) Graph Transformers, (iii) LM-Based methods, (iv) recent works designed for textual graphs, and (v) PEFT methods. Briefly, traditional GNN models include **GCN** [Kipf and Welling, 2017], **SAGE** [Hamilton *et al.*, 2017], and **GAT** [Veličković *et al.*, 2018]. Graph Transformers include **GraphFormers** [Yang *et al.*, 2021] and **NodeFormer** [Wu *et al.*, 2022]. Full fine-tuned LM-based methods include **BERT** [Kenton and Toutanova, 2019], **SentenceBERT** [Reimers and Gurevych, 2019], and **DeBERTa** [He *et al.*, 2020]. Recent works for textual graphs include Node Feature Extraction by Self-Supervised Multi-scale Neighborhood Prediction (**GIANT**) [Chien *et al.*, 2021], Learning on Large-scale Text-attributed Graphs via Variational Inference (**GLEM**) [Zhao *et al.*, 2022], LLM-to-LM Interpreter for Enhanced Text-Attributed Graph Representation Learning (**TAPE**) [He *et al.*, 2023b], and A Frustratingly Simple Approach Improves Textual Graph Learning (**SimTeG**) [Duan *et al.*, 2023]. PEFT methods include Low-rank Adaptation of Large Language Models (**LoRA**) [Hu *et al.*, 2021], IA3 [Liu *et al.*, 2022], The Power of Scale for Parameter-Efficient Prompt Tuning (**Prompt Tuning**) [Lester *et al.*, 2021], and Ladder Side-Tuning (**LST**) [Sung *et al.*, 2022]. Details of these methods are in Appendix B.

## 4.3 Experimental Setup

For traditional GNN methods, we utilize grid search to obtain optimal results. For LM-based methods, *i.e.*, BERT, SentenceBERT, DeBERTa, we fine-tune all parameters of these models on training nodes. Methods tailored for textual graphs are implemented using their official codes and reproduced under our settings. The GNNs utilized in these methods are selected from GCN, SAGE, and GAT, choosing the most effective one. Notably, GraphFormers is originally designed for link prediction task, GraphFormers$^\star$ refers to our adaptation of their official codes[*] to the node classification tasks. The hyperparameters of the baselines can be found in Appendix C. Regarding our method, ENGINE can be applied to any LLMs. In the main content, we applied ENGINE to the widely used open LLM named LLaMA2-7B [Touvron *et al.*, 2023] to show the effectiveness. **Experimental results for other language models, like e5-large [Wang *et al.*, 2022], can be found in Appendix D.** We report the mean accuracy with a standard deviation across five different random seeds.

## 4.4 Performance Analysis

From Table 2, we can draw the following conclusions:

Firstly, static shallow embedding methods combined with GNNs (*i.e.*, GCN, SAGE, GAT) perform inferiorly compared to recent methods that combine LMs with GNNs. This indicates that static shallow embedding methods may struggle to capture context-aware information and complex semantic relationships, limiting their ability to fully exploit the richness of text attributes, thereby achieving suboptimal results. For instance, on OGBN-ArXiv and OGBN-Products datasets, LM+GNN methods (*i.e.*, SimTeG, GLEM, GIANT) significantly outperform GNNs with shallow embedding methods, exhibiting an absolute performance gap of around 5%.

Secondly, pure LM methods (*i.e.*, BERT, SentenceBERT, DeBERTa) perform inferiorly to LM+GNN methods on textual graphs. This demonstrates that, compared to pure LM methods which neglect intrinsic graph structure, combining LMs with GNNs can generate more semantic and structure-aware node embeddings. For instance, on the Ele-Photo dataset, GIANT achieves 81% accuracy, outperforming its base LM model BERT by approximately 13%.

Lastly, our method is superior to current LM+GNN methods. Specifically, ENGINE further outperforms the previous SoTA method SimTeG, achieving an absolute improvement of over 2% on the Cora dataset and an absolute 3% improvement on the WikiCS dataset. Besides, our method also shows significant improvements over all the other PEFT methods (*i.e.*, LoRA, IA3, Prompt Tuning, LST), showcasing the superiority of ENGINE in fine-tuning LLMs on textual graphs. Additionally, ENGINE (Early), which combines dynamic early exiting, achieves comparable performance with ENGINE while improving the efficiency of model inference, as further discussed in Section 4.5.

[*] https://github.com/microsoft/GraphFormers

| Methods | Cora | CiteSeer | WikiCS | OGBN-ArXiv | ArXiv-2023 | OGBN-Products | Ele-Photo |
|---|---|---|---|---|---|---|---|
| MLP | 74.32±2.75 | 71.13±1.37 | 68.41±0.65 | 55.54±0.11 | 65.39±0.39 | 56.66±0.10 | 61.21±0.11 |
| GCN | 86.90±1.51 | 72.98±1.32 | 76.33±0.81 | 71.51±0.33 | 67.60±0.28 | 69.86±0.14 | 79.00±0.22 |
| SAGE | 85.73±0.65 | 73.61±1.90 | 79.56±0.22 | 71.92±0.32 | 69.06±0.24 | 69.75±0.10 | 80.35±0.26 |
| GAT | 85.73±0.65 | 74.23±1.78 | 78.21±0.66 | 71.64±0.27 | 67.84±0.23 | 69.57±0.18 | 82.08±0.11 |
| GraphFormers$^\star$ | 80.44±1.89 | 71.28±1.17 | 72.07±0.31 | 67.25±0.22 | 62.87±0.46 | 68.15±0.76 | 75.44±0.56 |
| NodeFormer | 88.48±0.33 | 75.74±0.54 | 75.47±0.46 | 69.60±0.08 | 67.44±0.42 | 67.26±0.71 | 77.30±0.06 |
| BERT | 80.15±1.67 | 73.17±1.75 | 78.33±0.43 | 72.78±0.03 | 77.46±0.27 | 76.01±0.14 | 68.88±0.05 |
| SentenceBERT | 78.82±1.39 | 72.79±1.71 | 77.92±0.07 | 71.42±0.09 | 77.53±0.45 | 75.07±0.13 | 68.74±0.07 |
| DeBERTa | 77.79±2.26 | 73.13±1.94 | 75.11±1.97 | 72.90±0.05 | 77.25±0.20 | 75.61±0.28 | 70.82±0.08 |
| GIANT$_{\text{(BERT)}}$ | 85.52±0.74 | 72.38±0.83 | 75.81±0.26 | 74.26±0.17 | 72.18±0.24 | 74.06±0.42 | 81.27±0.41 |
| GLEM$_{\text{(DeBERTa)}}$ | 85.60±0.09 | 75.89±0.53 | 78.92±0.19 | 74.69±0.25 | 78.58±0.09 | 73.77±0.12 | 76.10±0.23 |
| TAPE$_{\text{(DeBERTa)}}$ | 88.52±1.12 | – | – | 74.65±0.10 | 79.23±0.52 | 79.76±0.11 | – |
| SimTeG$_{\text{(e5-large)}}$ | 88.04±1.36 | 77.22±1.43 | 79.07±0.65 | 75.29±0.23 | <u>79.51±0.48</u> | 74.51±1.49 | 83.07±0.20 |
| LoRA (🦙) | 79.95±0.44 | 73.61±1.89 | 78.91±1.26 | 74.94±0.03 | 78.85±0.21 | 75.50±0.05 | 73.25±0.03 |
| IA3 (🦙) | 76.43±1.29 | 71.07±1.24 | 70.08±1.26 | 71.87±0.03 | 78.14±0.30 | 75.82±0.10 | 69.27±0.37 |
| Prompt Tuning (🦙) | 73.73±2.05 | 69.62±2.14 | 67.14±1.50 | 71.34±0.58 | 74.78±0.70 | 74.50±0.99 | 62.84±0.27 |
| LST (🦙) | 77.60±0.76 | 75.05±1.36 | 77.59±0.70 | 73.68±0.90 | 77.82±0.37 | 76.10±0.79 | 68.93±0.21 |
| ENGINE (🦙) | **91.48±0.32** | **78.46±0.49** | **81.56±0.97** | **76.02±0.29** | **79.76±0.14** | **80.05±0.45** | **83.75±0.08** |
| ENGINE (Early) (🦙) | <u>90.41±0.52</u> | <u>78.34±0.75</u> | <u>81.23±0.78</u> | <u>75.66±0.54</u> | 79.29±0.18 | <u>79.78±0.62</u> | <u>83.13±0.44</u> |

Table 2: Experimental results of node classification. 🦙 denotes LLaMA2-7B model. ENGINE (Early) means that use dynamic early exit to accelerate model inference. $\star$ denotes our adaption of their official codes to our tasks. We use **boldface** and <u>underlining</u> to denote the best and the second-best performance, respectively.

These statistics show the effectiveness of our method for generating context-aware and complex semantic nodes embeddings on textual graphs.

## 4.5 Efficiency Analysis

### Training Efficiency

In this section, we assess the training efficiency of several baselines and ENGINE. The additional trainable parameters in our method are integrated with a frozen LLM through a side structure. This allows us to precompute node embeddings using frozen LLMs and store them in the cache for reuse, consequently enhancing training efficiency. We also present the training efficiency of our method with caching.

| Methods | Update Param. | Memory (GB) | Total Time |
|---|---|---|---|
| GIANT$_{\text{(BERT)}}$ | 114,535,896 | 7.5 | 29h 44m |
| GLEM$_{\text{(DeBERTa)}}$ | 138,731,759 | 6.6 | 46h 27m |
| SimTeG$_{\text{(e5-large)}}$ | 2,013,328 | 3.7 | 25h 22m |
| LoRA (🦙) | 4,358,144 | 44.8 | 10h 18m |
| IA3 (🦙) | 425,984 | 28.5 | 9h 55m |
| Prompt. (🦙) | 245,760 | 27.8 | 9h 48m |
| ENGINE (🦙) | 3,909,032 | 14.9 | 4h 23m |
| w/ caching | 3,909,032 | 3.3 | 21m |

Table 3: The efficiency analysis of training different methods on OGBN-ArXiv dataset. The batch size is set as 1 for tuning LMs, and the total training time is reported on the CPU of a 48-core Intel(R) Xeon(R) @ 2.50GHz and GPUs of 6 NVIDIA GeForce RTX 3090.

In Table 3, GIANT and GLEM employ full fine-tuning of language models with the largest trainable parameters. In contrast, SimTeG utilizes LoRA to fine-tune e5-large and subsequently trains GNNs. These methods consume low memory because they employ small language models. However, scaling them up to large language models proves challenging due to their high training costs. Furthermore, it is evident that previous PEFT methods demand substantial time and memory resources for tuning LLaMA2-7B. In contrast, ENGINE achieves the lowest time and memory costs, and the inclusion of caching in ENGINE significantly reduces computation expenses, achieving 12x faster training compared to without caching (21m vs 4h 23m).

### Inference Efficiency

For inference, ENGINE goes through all layers in LLMs, which may be impractical in some realistic scenarios. Therefore, we incorporate dynamic early exit seamlessly into our method, named ENGINE (Early). ENGINE (Early) dynamically exit based on the complexity of samples to save inference time. Table 4 records the inference time for running all test samples using our method (with or without dynamic exit). Figure 3 depicts the early exit ratios across different layers.

| Methods | Cora | CiteSeer | WikiCS |
|---|---|---|---|
| GIANT$_{\text{(BERT)}}$ | 41s | 47s | 2m 58s |
| GLEM$_{\text{(DeBERTa)}}$ | 1m 8s | 1m 19s | 5m 27s |
| SimTeG$_{\text{(e5-large)}}$ | 1m 21s | 1m 35s | 5m 53s |
| ENGINE (🦙) | 1m 30s | 1m 54s | 15m 55s |
| ENGINE (Early) (🦙) | 15s | 20s | 2m 38s |

Table 4: Inference time of ENGINE and baselines across datasets.

In Table 4, the speed advantage of ENGINE (Early) over ENGINE is clearly evident, primarily due to reduced in-

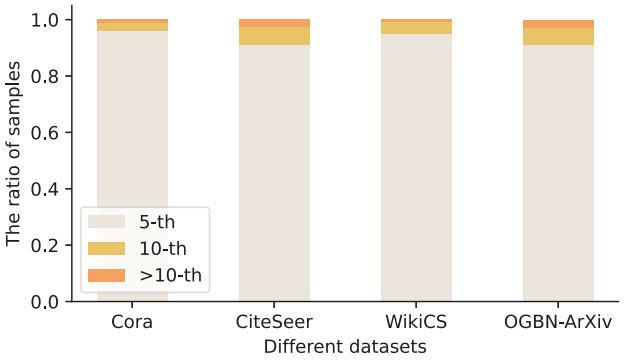


Figure 3: The statistics of samples early exit at each layer.

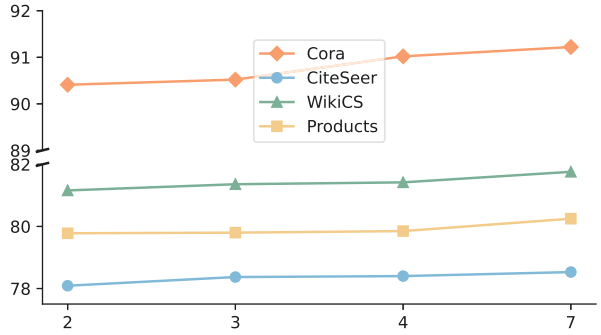


Figure 4: Sensitivity analysis of patience $p$ in dynamic early exit.

tensive computation within LLM layers. Notably, ENGINE (Early) attains 5x faster inference speed, mainly attributed to the fact that over 90% of samples exit early in the 5th layer as depicted in Figure 3. Additionally, ENGINE (Early) is faster than other textual graph baselines while achieving better performance.

### 4.6 Sensitivity Analysis

**The number of G-Ladders.** G-Ladder can be incorporated alongside each LLM layer. In our investigation, we examine varying numbers of G-Ladders. For simplicity, we analyze three strategies: (1) Adding a G-Ladder every 10 layers (*i.e.*, $0, 10, 20, 32$ where the 0-th layer represents the word embedding layer, and the 32-nd layer is the last layer used in all strategies). (2) Adding a G-Ladder every 5 layers (*i.e.*, $0, 5, ..., 25, 32$). (3) Adding a G-Ladder every 2 layers (*i.e.*, $0, 2, ..., 30, 32$).

Table 5 presents the results. The first strategy consistently performs worse than others, likely due to having the fewest learnable parameters. Although the last strategy involves the most learnable parameters, it falls short of the second strategy on small datasets (*e.g.*, Cora), suggesting that a moderate number of parameters is optimal. Throughout our experiments, we adopt the second strategy due to its balanced consideration of effectiveness and efficiency.

| Inserted Layers | Cora | CiteSeer | WikiCS |
|---|---|---|---|
| $0, 10, 20, 32$ | 90.22±1.05 | 78.31±0.68 | 81.20±0.86 |
| $0, 5, ..., 25, 32$ | 91.48±0.32 | 78.46±0.49 | 81.56±0.97 |
| $0, 2, ..., 30, 32$ | 90.77±0.56 | 78.71±0.77 | 81.90±0.76 |

Table 5: Sensitivity analysis of the number of G-Ladders. We report the mean accuracy with a standard deviation across 5 different random seeds.

**The patience in dynamic early exit.** To accelerate inference, we adopt patience-based early dynamic exiting in our work. In this section, experiments are conducted with varying patience values: 2, 3, 4, and 7. The patience of 2 means terminating in advance and returning results if two consecutive layers produce identical results. Notably, the patience of 7 signifies the absence of dynamic early exiting.

Results from Figure 4 illustrate the trade-off between performance and efficiency. In our experiments, we set the patience to 2 for most datasets, achieving comparable performance while significantly reducing inference time, denoted as ENGINE (Early).

### 4.7 Ablation Study

In this section, we investigate the impact of components in our method. Specifically, 'constant $\lambda$ (0.5)' indicates setting $\lambda$ as a constant value (*i.e.*, 0.5) in Equation 4 rather than a learnable coefficient. 'w/o struct. info.' denotes the removal of message passing in G-Ladders.

Table 6 highlights that structural information is crucial for enhancing the quality of node representations, showing around a 14% absolute improvement on Cora and WikiCS datasets. Additionally, a learnable coefficient $\lambda$ outperforms a constant value because the coefficient can adapt to the characteristics of datasets.

| | Cora | CiteSeer | WikiCS |
|---|---|---|---|
| ENGINE | 91.48±0.32 | 78.46±0.49 | 81.56±0.97 |
| constant $\lambda$ (0.5) | 90.07±0.91 | 78.06±0.36 | 80.66±0.66 |
| w/o struct. info. | 76.94±1.98 | 69.97±3.46 | 68.44±2.75 |

Table 6: Experimental results of ablation study. We report the mean accuracy with a standard deviation across 5 different random seeds.

## 5 Conclusion

In this paper, we present ENGINE, an efficient and effective framework for incorporating large language models in textual graphs. Our proposed approach introduces a lightweight, tunable GNN-based side structure (G-Ladder) alongside each layer of the LLM, to explicitly model the structural information of the textual graphs. The key insight is that the parameter update of ENGINE does not depend on the gradient computation of the LLMs, resulting in exceptionally efficient training compared to concurrent counterparts. Building upon this, we introduce two variants with caching and dynamic early exit to further enhance training and inference speed. Empirical studies demonstrate that our ENGINE outperforms the state-of-the-art approaches across multiple realistic textual graph datasets, in terms of performance, training efficiency, and inference efficiency.

## Acknowledgments

This work was supported by the Key Research and Development Projects in Zhejiang Province (No. 2024C01106), NSFC (No. 62272411), the Tencent WeChat Rhino-Bird Special Research Program (Tencent WXG-FR-2023-10), the National Key Research and Development Project of China (2018AAA0101900), and Research funding from FinVolution Group.

## Contribution Statement

The paper's methodology and experimental design were collaboratively conceived by all contributing authors. Both **Yun Zhu** and **Yaoke Wang** made significant and equal contributions to this work, thereby meriting the designation of co-first authors. **Siliang Tang** holds the role of corresponding author for this publication.

## References

[Almazrouei *et al.*, 2023] Ebtesam Almazrouei, Hamza Alobeidli, Abdulaziz Alshamsi, Alessandro Cappelli, Ruxandra Cojocaru, Merouane Debbah, Etienne Goffinet, Daniel Heslow, Julien Launay, Quentin Malartic, et al. Falcon-40b: an open large language model with state-of-the-art performance. *Findings of the Association for Computational Linguistics: ACL*, 2023.

[Chen *et al.*, 2023] Zhikai Chen, Haitao Mao, Hongzhi Wen, Haoyu Han, Wei Jin, Haiyang Zhang, Hui Liu, and Jiliang Tang. Label-free node classification on graphs with large language models (llms). *arXiv preprint arXiv:2310.04668*, 2023.

[Chiang *et al.*, 2019] Wei-Lin Chiang, Xuanqing Liu, Si Si, Yang Li, Samy Bengio, and Cho-Jui Hsieh. Cluster-gcn: An efficient algorithm for training deep and large graph convolutional networks. In *Proc. of KDD*, 2019.

[Chien *et al.*, 2021] Eli Chien, Wei-Cheng Chang, Cho-Jui Hsieh, Hsiang-Fu Yu, Jiong Zhang, Olgica Milenkovic, and Inderjit S Dhillon. Node feature extraction by self-supervised multi-scale neighborhood prediction. In *Proc. of ICLR*, 2021.

[Duan *et al.*, 2023] Keyu Duan, Qian Liu, Tat-Seng Chua, Shuicheng Yan, Wei Tsang Ooi, Qizhe Xie, and Junxian He. Simteg: A frustratingly simple approach improves textual graph learning. *arXiv preprint arXiv:2308.02565*, 2023.

[Giles *et al.*, 1998] C Lee Giles, Kurt D Bollacker, and Steve Lawrence. Citeseer: An automatic citation indexing system. In *Proceedings of the third ACM conference on Digital libraries*, 1998.

[Gordon *et al.*, 2020] Mitchell A. Gordon, Kevin Duh, and Nicholas Andrews. Compressing BERT: studying the effects of weight pruning on transfer learning. *CoRR*, 2020.

[Hamilton *et al.*, 2017] Will Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. *Proc. of NeurIPS*, 2017.

[He *et al.*, 2020] Pengcheng He, Xiaodong Liu, Jianfeng Gao, and Weizhu Chen. Deberta: Decoding-enhanced bert with disentangled attention. In *Proc. of ICLR*, 2020.

[He *et al.*, 2023a] Haoyu He, Jianfei Cai, Jing Zhang, Dacheng Tao, and Bohan Zhuang. Sensitivity-aware visual parameter-efficient fine-tuning. In *Proc. of ICCV*, 2023.

[He *et al.*, 2023b] Xiaoxin He, Xavier Bresson, Thomas Laurent, and Bryan Hooi. Explanations as features: Llm-based features for text-attributed graphs. *arXiv preprint arXiv:2305.19523*, 2023.

[Houlsby *et al.*, 2019] Neil Houlsby, Andrei Giurgiu, Stanislaw Jastrzebski, Bruna Morrone, Quentin De Laroussilhe, Andrea Gesmundo, Mona Attariyan, and Sylvain Gelly. Parameter-efficient transfer learning for nlp. In *Proc. of ICML*, 2019.

[Hu *et al.*, 2020] Weihua Hu, Matthias Fey, Marinka Zitnik, Yuxiao Dong, Hongyu Ren, Bowen Liu, Michele Catasta, and Jure Leskovec. Open graph benchmark: Datasets for machine learning on graphs. *Proc. of NeurIPS*, 2020.

[Hu *et al.*, 2021] Edward J Hu, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, Weizhu Chen, et al. Lora: Low-rank adaptation of large language models. In *Proc. of ICLR*, 2021.

[Hu *et al.*, 2023] Boren Hu, Yun Zhu, Jiacheng Li, and Siliang Tang. Smartbert: A promotion of dynamic early exiting mechanism for accelerating bert inference. In *Proc. of IJCAI*, 2023.

[Jia *et al.*, 2022] Menglin Jia, Luming Tang, Bor-Chun Chen, Claire Cardie, Serge Belongie, Bharath Hariharan, and Ser-Nam Lim. Visual prompt tuning. In *Proc. of ECCV*, 2022.

[Kenton and Toutanova, 2019] Jacob Devlin Ming-Wei Chang Kenton and Lee Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. In *Proc. of AACL*, 2019.

[Kipf and Welling, 2017] Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. In *Proc. of ICLR*, 2017.

[Lester *et al.*, 2021] Brian Lester, Rami Al-Rfou, and Noah Constant. The power of scale for parameter-efficient prompt tuning. In *Proc. of EMNLP*, 2021.

[Li and Liang, 2021] Xiang Lisa Li and Percy Liang. Prefix-tuning: Optimizing continuous prompts for generation. In *Proc. of ACL*, 2021.

[Liu *et al.*, 2022] Haokun Liu, Derek Tam, Mohammed Muqeeth, Jay Mohta, Tenghao Huang, Mohit Bansal, and Colin A Raffel. Few-shot parameter-efficient fine-tuning is better and cheaper than in-context learning. *Proc. of NeurIPS*, 2022.

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

[Mesquita *et al.*, 2020] Diego Mesquita, Amauri Souza, and Samuel Kaski. Rethinking pooling in graph neural networks. *Proc. of NeurIPS*, 2020.

[Mikolov *et al.*, 2013] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. *Proc. of NeurIPS*, 2013.

[OpenAI, 2023] OpenAI. Gpt-4 technical report, 2023.

[Pan *et al.*, 2023] Kaihang Pan, Juncheng Li, Hongye Song, Jun Lin, Xiaozhong Liu, and Siliang Tang. Self-supervised meta-prompt learning with meta-gradient regularization for few-shot generalization. *arXiv preprint arXiv:2303.12314*, 2023.

[Pan *et al.*, 2024a] Kaihang Pan, Juncheng Li, Wenjie Wang, Hao Fei, Hongye Song, Wei Ji, Jun Lin, Xiaozhong Liu, Tat-Seng Chua, and Siliang Tang. I3: Intent-introspective retrieval conditioned on instructions, 2024.

[Pan *et al.*, 2024b] Shirui Pan, Yizhen Zheng, and Yixin Liu. Integrating graphs with large language models: Methods and prospects. *IEEE Intelligent Systems*, 39(1):64–68, 2024.

[Qiu *et al.*, 2020] Jiezhong Qiu, Qibin Chen, Yuxiao Dong, Jing Zhang, Hongxia Yang, Ming Ding, Kuansan Wang, and Jie Tang. Gcc: Graph contrastive coding for graph neural network pre-training. In *Proc. of KDD*, 2020.

[Reimers and Gurevych, 2019] Nils Reimers and Iryna Gurevych. Sentence-bert: Sentence embeddings using siamese bert-networks. In *Proc. of EMNLP*, 2019.

[Sanh *et al.*, 2019] Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter. *arXiv preprint arXiv:1910.01108*, 2019.

[Sen *et al.*, 2008] Prithviraj Sen, Galileo Namata, Mustafa Bilgic, Lise Getoor, Brian Galligher, and Tina Eliassi-Rad. Collective classification in network data. *AI magazine*, 2008.

[Sung *et al.*, 2022] Yi-Lin Sung, Jaemin Cho, and Mohit Bansal. Lst: Ladder side-tuning for parameter and memory efficient transfer learning. *Proc. of NeurIPS*, 2022.

[Touvron *et al.*, 2023] Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, et al. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*, 2023.

[Veličković *et al.*, 2018] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph attention networks. In *Proc. of ICLR*, 2018.

[Wang *et al.*, 2020] Kuansan Wang, Zhihong Shen, Chiyuan Huang, Chieh-Han Wu, Yuxiao Dong, and Anshul Kanakia. Microsoft academic graph: When experts are not enough. *Quantitative Science Studies*, 2020.

[Wang *et al.*, 2022] Liang Wang, Nan Yang, Xiaolong Huang, Binxing Jiao, Linjun Yang, Daxin Jiang, Rangan Majumder, and Furu Wei. Text embeddings by weakly-supervised contrastive pre-training. *arXiv preprint arXiv:2212.03533*, 2022.

[Wu *et al.*, 2022] Qitian Wu, Wentao Zhao, Zenan Li, David P Wipf, and Junchi Yan. Nodeformer: A scalable graph structure learning transformer for node classification. *Proc. of NeurIPS*, 2022.

[Xie *et al.*, 2021] Keli Xie, Siyuan Lu, Meiqi Wang, and Zhongfeng Wang. Elbert: Fast albert with confidence-window based early exit. In *Proc. of ICASSP*, 2021.

[Xin *et al.*, 2020] Ji Xin, Raphael Tang, Jaejun Lee, Yaoliang Yu, and Jimmy Lin. DeeBERT: Dynamic early exiting for accelerating BERT inference. In *Proc. of ACL*, 2020.

[Xue *et al.*, 2023] Rui Xue, Xipeng Shen, Ruozhou Yu, and Xiaorui Liu. Efficient large language models fine-tuning on graphs. *arXiv preprint arXiv:2312.04737*, 2023.

[Yan *et al.*, 2023] Hao Yan, Chaozhuo Li, Ruosong Long, Chao Yan, Jianan Zhao, Wenwen Zhuang, Jun Yin, Peiyan Zhang, Weihao Han, Hao Sun, et al. A comprehensive study on text-attributed graphs: Benchmarking and rethinking. In *Proc. of NeurIPS*, 2023.

[Yang *et al.*, 2021] Junhan Yang, Zheng Liu, Shitao Xiao, Chaozhuo Li, Defu Lian, Sanjay Agrawal, Amit Singh, Guangzhong Sun, and Xing Xie. Graphformers: Gnn-nested transformers for representation learning on textual graph. *Proc. of NeurIPS*, 2021.

[Zhang *et al.*, 2010] Yin Zhang, Rong Jin, and Zhi-Hua Zhou. Understanding bag-of-words model: a statistical framework. *International journal of machine learning and cybernetics*, 2010.

[Zhao *et al.*, 2022] Jianan Zhao, Meng Qu, Chaozhuo Li, Hao Yan, Qian Liu, Rui Li, Xing Xie, and Jian Tang. Learning on large-scale text-attributed graphs via variational inference. In *Proc. of ICLR*, 2022.

[Zhou *et al.*, 2020] Wangchunshu Zhou, Canwen Xu, Tao Ge, Julian McAuley, Ke Xu, and Furu Wei. Bert loses patience: Fast and robust inference with early exit. *Proc. of NeurIPS*, 2020.

[Zhu *et al.*, 2022] Yun Zhu, Jianhao Guo, Fei Wu, and Siliang Tang. Rosa: A robust self-aligned framework for node-node graph contrastive learning. In *Proc. of IJCAI*, 2022.

[Zhu *et al.*, 2023a] Yun Zhu, Jianhao Guo, and Siliang Tang. Sgl-pt: A strong graph learner with graph prompt tuning. *arXiv preprint arXiv:2302.12449*, 2023.

[Zhu *et al.*, 2023b] Yun Zhu, Haizhou Shi, Zhenshuo Zhang, and Siliang Tang. Mario: Model agnostic recipe for improving ood generalization of graph contrastive learning. *arXiv preprint arXiv:2307.13055*, 2023.

[Zhu *et al.*, 2023c] Yun Zhu, Yaoke Wang, Haizhou Shi, Zhenshuo Zhang, and Siliang Tang. Graphcontrol: Adding conditional control to universal graph pre-trained models for graph domain transfer learning, 2023.