Can GNN be Good Adapter for LLMs?

Xuanwen Huang
xwhuang@zju.edu.cn
Zhejiang University
Hangzhou, China

Kaiqiao Han
kaiqiaohan@zju.edu.cn
Zhejiang University
Hangzhou, China

Yang Yang†
yangya@zju.edu.cn
Zhejiang University
Hangzhou, China

Dezheng Bao
baodezheng@zju.edu.cn
Zhejiang University
Hangzhou, China

Quanjin Tao
taoquanjin@zju.edu.cn
Zhejiang University
Hangzhou, China

Ziwei Chai
zwchai@zju.edu.cn
Zhejiang University
Hangzhou, China

Qi Zhu*
qzhuamazon@amazon.com
Amazon Web Services
Santa Clara, USA

ABSTRACT

Recently, large language models (LLMs) have demonstrated superior capabilities in understanding and zero-shot learning on textual data, promising significant advances for many text-related domains. In the graph domain, various real-world scenarios also involve textual data, where tasks and node features can be described by text. These text-attributed graphs (TAGs) have broad applications in social media, recommendation systems, etc. Thus, this paper explores how to utilize LLMs to model TAGs. Previous methods for TAG modeling are based on million-scale LMs. When scaled up to billion-scale LLMs, they face huge challenges in computational costs. Additionally, they also ignore the zero-shot inference capabilities of LLMs. Therefore, we propose *GraphAdapter*, which uses a graph neural network (GNN) as an efficient adapter in collaboration with LLMs to tackle TAGs. In terms of efficiency, the GNN adapter introduces only a few trainable parameters and can be trained with low computation costs. The entire framework is trained using auto-regression on node text (next token prediction). Once trained, *GraphAdapter* can be seamlessly fine-tuned with task-specific prompts for various downstream tasks. Through extensive experiments across multiple real-world TAGs, *GraphAdapter* based on Llama 2 gains an average improvement of approximately 5% in terms of node classification. Furthermore, GraphAdapter can also adapt to other language models, including RoBERTa, GPT-2. The promising results demonstrate that GNNs can serve as effective adapters for LLMs in TAG modeling.

†Yang Yang is the corresponding author.
*This work was done before the author joined Amazon.
The code is available at: https://github.com/zjunet/GraphAdapter

WWW '24, May 13–17, 2024, Singapore, Singapore

ACM ISBN 979-8-4007-0171-9/24/05…$15.00
https://doi.org/10.1145/3589334.3645627

CCS CONCEPTS

• **Computing methodologies** → *Artificial intelligence*; • **Information systems** → *Data mining*.

KEYWORDS

Graph Neural Networks, Large Language Model, Text-Attributed Graph

ACM Reference Format:
Xuanwen Huang, Kaiqiao Han, Yang Yang, Dezheng Bao, Quanjin Tao, Ziwei Chai, and Qi Zhu. 2024. Can GNN be Good Adapter for LLMs? . In *Proceedings of the ACM Web Conference 2024 (WWW '24), May 13–17, 2024, Singapore, Singapore.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3589334.3645627

1 INTRODUCTION

Graphs are ubiquitous in the real world [1]. In the past, graph structures have been extensively explored and utilized in many machine learning applications [27, 39]. In many practical cases, the nodes in graphs have textual features, known as Textual-Attributed Graphs (TAGs) [37]. For example, in social media [18], nodes represent users and node features are user profiles. Nodes in TAGs have both textual and structural data, which both reflect their intrinsic properties. Combining textual and structural data to modeling TAGs is an exciting new exploration for both graph machine learning and language modeling, which can benefit the application of graphs.

In TAGs, a complex correlation exists between the structural and textual data of nodes. Understanding this correlation can benefit the modeling of TAGs [5]. In Figure 1, user "Bob" frequently browses daily news on social media, as evidenced by the descriptions in his user profile. Users similar to Bob, who have many followers and often browse news nodes, are also likely interested in news. In other words, a graph can supplement textual attributes on a node through structural proximity. Graph Neural Networks (GNNs) are the de facto machine learning models for leveraging textual information alongside graph structures in TAGs. However, there's a lack of a unified GNN architecture compatible with different language models, especially the powerful foundation models.

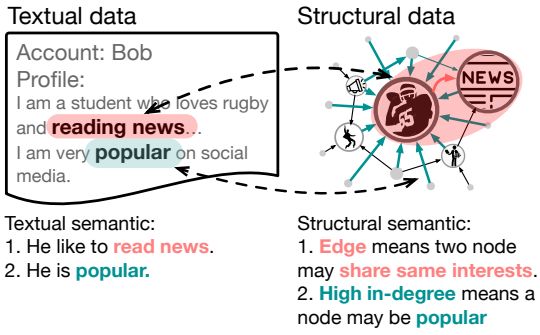

Figure 1: An example of the correlation existing in the structural and textual data of nodes in social networks.

Recently, there has been a surge in studies investigating effective ways to model both textual and structural data in TAGs. Some of these studies emphasize optimizing a cascading architecture that combines GNNs and LMs (**cascading GNN-LMs**) [37, 42]. One major challenge with these models is the extreme amount of additional computational cost brought by the message-passing mechanism. To this end, several studies have successfully reduced the memory and computational overheads of such cascaded models by freezing partial or full parameters of the backbone language models [20, 25]. Large language models exhibit superior multi-task and few-shot learning capabilities across various real-world applications [2]. However, when considering cascading GNN-LMs, existing techniques cannot be scaled up to billion-scale models like Llama 2 [33]. Another pioneering research has ventured to fine-tune language models using unsupervised graph information (**self-supervised GNN-LMs**) [4, 26]. For instance, GIANT [4] fine-tunes language models through a neighbor prediction task, subsequently using the refined language model to extract node representations for downstream tasks. In these methods, PLMs can indirectly incorporate graph information during the tuning process thereby enhancing their capability to process TAGs. However, they separate the training of GNNs and LMs, potentially leading to sub-optimal graph-aware tuning results.

Instead of using graph information as supervision, we believe graph structure can enrich textual features through language modeling. In our previous example, structural proximity can be used to infer the user's preference even if he or she does not mention it in the profile. So, unlike self-supervised methods, we consider pre-training a framework that can combine graph-aware structure and LLMs by leveraging rich textual features. However, traditional frameworks like cascading GNNs and LLMs face efficiency issues in pre-training scenarios. Therefore, inspired by works on parameter-efficient tuning of LLMs [14, 22, 23], we propose the use of GNNs as efficient adapters for LLMs (i.e., *GraphAdapter*). In *GraphAdapter*, the LM is frozen and the final output of the LM is altered by the trainable adapter GNNs. *GraphAdapter* offers several advantages:

- **Lightweight:** A GNN adapter introduces a few trainable parameters and low computational costs.
- **Language-aware graph pre-training**: Using language to supervise the modeling of graph structure, which can help LLMs comprehend both textual and structural information.
- **Convenient tuning:** Once a graph-specific adapter is pre-trained, it can be fine-tuned for multiple downstream tasks.

Now we present the details of *GraphAdapter* with respect to pre-training and fine-tuning of the adapter GNNs. To capture the data distribution of the graph, we employ parameter-efficient tuning of LLMs on node texts. This approach is similar to the continual training of language models [31] except GNN is the tuning parameter, which helps reduce the distribution discrepancy between the pre-training corpus and target data. To further enhance efficiency, we utilize the GNN adapter exclusively at the transformer's final layer. It ensures that all transformer computational processes are executed just once and then can be cached for adapter training. Besides, we perform mean-pooling on the predicted logits from a GNN adapter and LLMs then optimize their final results of the next-word prediction, which can help adapters focus more on the graph-related tokens. Once the adapter is trained, one can use *GraphAdapter* together with the backbone LLMs on various downstream tasks. For instance, we use a classification head atop the embeddings of the last token to fine-tune for node classification.

To verify the effectiveness of *GraphAdapter*, we conduct extensive experiments on multiple real-world TAGs including social and citation networks. *GraphAdapter* achieves an improvement of 4.7% over state-of-the-art cascaded GNN-LM methods and 5.4% over self-supervised GNN-LMs on average, with 30X fewer training parameters and storage. Moreover, once *GraphAdapter* is pre-trained, it can be conveniently fine-tuned for various tasks. Our ablation analysis shows that the pre-training step consistently improves the model performance across different graphs. We summarize our contributions as follows,

- *GraphAdapter* is a novel approach that harnesses the large language models on graph structure data with parameter-efficient tuning.
- We propose a residual learning procedure to pre-train the GNN adapter with the LLMs. The pre-training step significantly improves the fine-tuning performance of *GraphAdapter*.
- We conduct extensive experiments on large-scale TAGs using state-of-the-art open-sourced large language models (GPT-2 1.5B [28] and Llama 2 13B [33]). The results demonstrate that *GraphAdapter* can also reap the benefits of a larger model.

2 RELATED WORK

Modeling text-attributed graphs has attracted much attention in academia, which requires modeling both textual and structural data.

Modeling semantics and graph structure. Understanding the semantics is a key part of modeling TAG. With the advent of Transformers [34], pre-trained language models have made breakthrough progress in modeling semantics [6, 43]. These methods leverage massive unlabeled text through unsupervised methods like auto-regressive [19] and auto-encoding pre-training [12, 24] to pre-train Transformers and then fine-tune to downstream tasks. Since language models have a large number of parameters, fine-tuning efficiency is low and requires numerous training samples. Therefore, some work proposed using adapter modules to reduce the number of fine-tuning parameters. For example, LoRA [14] trains a sparse matrix appended to the original parameters while keeping the language model frozen. Some work proposed using

prompts to directly adapt language models to downstream tasks without fine-tuning. Furthermore, some work proposed prompt tuning [16, 23], which adds a trainable prompt and only trains the added prompt during training, greatly reducing the number of parameters. Another aspect of modeling TAG is modeling the structural information. With the proposal of GNNs [11], modeling graph structure achieved remarkable success. Many works [21, 36] have explored GNN architectures extensively, and these methods have achieved breakthrough progress in graph structure modeling.

Modeling TAGs. However, despite the success of language models and GNNs in their respective areas, how to utilize them to model text-attributed graphs still has many challenges. **(1) Cascading GNN-LM**: Directly cascading these two models is straightforward but has limitations, mainly high computational overhead. Since GNNs are mostly based on message-passing, they need to compute representations for many nodes simultaneously. Using language models to model so many text features requires huge memory and time costs. To address this, some work [25] proposed freezing the language model to reduce the computation needed for cascading. Some work [17, 20] proposed neighbor sampling but that reduces the graph information captured. Therefore, recently some work tried joint training of LMs and GNNs through knowledge distillation [26] or Expectation Maximization algorithms [42]. **(2) Self-supervised GNN-LMs**: some methods [4, 26] directly supervise language model fine-tuning through graph-related tasks, to help language models better understand the textual information in text-attributed graphs. The language model is then combined with GNNs by freezing the language model. This approach demonstrates the inherent connections between graph structure and text in TAGs. However, current research in this direction has limitations in that the LM and graph are separate, and the language model cannot directly perceive graph information. It also does not utilize the inherent connections between language and graphs to help GNNs better learn structural features. **(3) LLMs for Graph**: With the breakthrough progress made by LLMs on textual tasks [33, 41], recently many works have emerged exploring how to directly utilize LLMs to understand text-attributed graphs [3]. For example, by converting the graph to text [10, 40], or by converting it to a graph representation as part of a prompt [32]. Some works also explored using large models to enhance the textual features of text-attributed graphs [7, 13]. However, this paper is more focused on how to leverage the semantic information in text-attributed graphs to help us model text-attributed graphs. Therefore, this type of method not be further elaborated.

3 BACKGROUND

Before introducing the proposed method, it's important to understand some basic concepts and the background of pre-trained language models, graph neural networks, and text-attributed graphs.

3.1 Pretrained Language Model

Textual data. Textual data can be formulated as $\mathbb{D} = \{d_1, d_2 ... d_K\}$. It can be tokenized into a sequence of tokens $\mathbb{S} = \{s_1, s_2, ..., s_L\}$, where s_i represents a specific token-id. In most cases, the first token in the sentence (i.e., s_0) is [**CLS**], indicating the beginning of this sentence.

Framework of PLMs. A PLM consists of a multi-layer transformer encoder that takes a sentence $\mathbb{S}$ as input and outputs the hidden states of each token:

$$\textbf{Transformer}(\{s_0, ..., s_L\}) = \{h_0, ..., h_L\}, \tag{1}$$

where h_k is the dense hidden state of s_k.

Pre-training of PLMs. This paper uses the auto-regression task as an instance of pre-training, which is commonly applied to auto-regressive PLMs [29]. Given a sequence $\mathbb{S} = \{s_0, ..., s_L\}$, the goal is to model the joint probability of the sequence $P(\mathbb{S})$.

$$P(\mathbb{S}) = \prod_{k=1}^{L} p(s_i | s_0, ...s_{k-1}) \tag{2}$$

The transformer block is used to model these conditional probabilities. More specifically, at step k ($0 < k \leq L$), the transformer receives $\{s_0 ... s_{k-1}\}$ and outputs their hidden states $\{h_0, ..., h_k\}$. The h_k are used to predict the probability distribution of the next token.

$$p(s_i | s_0, ...s_{k-1}) = \hat{s}_k = \sigma(\textbf{Head}(h_k)) \tag{3}$$

The model parameters are trained to maximize the likelihood of $p(\mathbb{S})$, which is equivalent to minimizing the negative log-likelihood. Therefore, the loss function is:

$$\mathcal{L}_{LM} = \sum_{k=1}^{L} \textbf{CrossEntropy}(s_k, \hat{s}_k) \tag{4}$$

Sentence representation. Given a sentence $\mathbb{S}$ with length L, its sentence representation W can be obtained by three methods [8, 30]: (1) first token representation, which uses the hidden state of the [**CLS**] token ($h_{i,0}$) as sentence representation. (2) mean-pooling representation, which is obtained by mean-pooling of all hidden states (i.e., $\textbf{Pool}(\{h_0 ... h_L\})$). (3) last token representation, which uses the hidden state of the last token.

PLMs with prompts. Due to the gap between pretraining tasks and downstream tasks, sentence representation may be hard to contain all the sentence information, thereby requiring fine-tuning for specific tasks. To address this issue, some studies utilize prompts to extract task-specific sentence features [16]. For example, suppose a $\mathbb{S}_i$ is a paper titled "Llama 2: Open Foundation and Fine-Tuned Chat Models", and the task is to classify the subject of it belongs. We can add some prompts to the sentence:

$$\{[Title], this, paper, belong, to, which, subject?\} \tag{5}$$

We denote this new sentence with the prompt inserted as $\mathbb{S}_{i|\mathbb{P}}$, where $\mathbb{P}$ represents the newly inserted tokens. We use the hidden state of the last token as the sentence representation, denoted as $W_{i|\mathbb{P}}$. Since the last token is used to predict the next token distribution in the pre-training stage, it can naturally combine the inserted prompt information into the original sentence and extract the prompt-related semantics. Extensive studies [19, 23] show that using prompts can reduce the gap between PLMs and downstream tasks and maximize the utilization of knowledge learned by PLMs during pre-training.

3.2 Graph Neural Network

Graph Neural Networks (GNNs) have achieved remarkable success in modeling graphs [9, 35]. The message-passing framework is a commonly used architecture of GNN.

Graph. Let $G = \{V, A\}$ denote a graph, where V is the node set and A is the adjacency matrix, with $A_{ij} = 1$ meaning there is an edge between node i and node j. Usually, each node i is associated with a node feature x_i^0.

Framework of GNN. The message-passing framework takes a set of node features $\mathcal{X} = \{x_i^0 | i \in V\}$, and an adjacency matrix A as input and iteratively captures neighbors' information via pooling. More specifically, for a given node $i \in V$ in the l-th layer of message-passing, it can be formulated as:

$$x_i^l = f_2(\textbf{Pool}\{f_1(x_j^{l-1}|\theta_1^l)|j \in \mathcal{N}_i\}, x_i|\theta_2^l) \tag{6}$$

where $\textbf{Pool}\{\cdot\}$ is an aggregation function that combines the features of neighboring nodes, such as mean-pooling. And $\mathcal{N}_i$ denotes the set of neighbors of node i. Besides, $f_1(\cdot|\theta_1^l)$ and $f_2(\cdot|\theta_2^l)$ denote two trainable transformations with parameters θ_1^l and θ_2^l respectively. Further, we denote an l_{max} layer message-passing framework as GNN, formally:

$$z_i = \textbf{GNN}(x_i^0, \mathcal{X}^0, A|\Theta_g) \tag{7}$$

where $z_i = x_i^{l_{max}}$, and Θ_g represents all the trainable parameters in the GNN. We use z_i as the structural representation for node i.

3.3 Text-Attributed Graph

Let $\mathcal{G} = \{\mathcal{V}, \mathcal{A}\}$ denote a text-attributed graph, where $\mathcal{V}$ is the node set and $\mathcal{A}$ is the adjacency matrix. Each node $i \in \mathcal{V}$ is associated with a tokenized textual data, represented by $\mathbb{S}_i = \{s_{i,0}, ..., s_{i,L_i}\}$, which represents the textual data of the node.

Problem Definition: Given a text-attributed graph $\mathcal{G}$ and corresponding node labels $\mathcal{Y} = y_i | i \in \mathcal{N}$, this paper addresses the problem of efficiently modeling both the textual data $\mathbb{S}_i | i \in \mathcal{V}$ and the structural data in $\mathcal{G}$ to predict the node labels $\mathcal{Y}$.

4 METHOD

This section introduces the proposed framework, referred to as *GraphAdapter*, which uses GNNs as adapters for LLMs to better model TAGs.

4.1 Overview

Motivation: In the textual data of TAGs, many structure-related semantics are hard to infer from context alone. As illustrated in the example in Figure 1, we can easily infer that this user is "popular" based on his degree in the social network, but it is difficult to infer from their description of habits alone. Combining structural information can enhance language models' ability to model these structure-related semantics in TAGs. Meanwhile, the process of enhancement is learning how to model structure. Therefore, the proposed method *GraphAdapter*, which first uses GNN as adapters for frozen PLMs, to combine structural information with PLMs, and then pre-trains them through the semantic understanding task on TAGs.

Language-structure pre-training: In the field of natural language processing, pre-training is a common strategy used to self-supervised enhance language models' ability for semantic understanding, with techniques such as auto-regressive pre-training (e.g., GPT-2/3 [2, 29], Llama 2 [33], etc.) and auto-encoding pre-training (e.g., BERT[38], RoBERTa[24], etc.). Following our motivation, *GraphAdapter* uses the same pre-training task as these PLMs. To facilitate comprehension, this section only discusses *GraphAdapter* based on auto-regressive pre-training, and further details on how *GraphAdapter* is combined with other pre-training tasks can be found in the appendix. Since the pre-training process uses the context semantic to supervise structure learning, we refer to this pre-training as language-structure pre-training.

Framework: The framework of *GraphAdapter* is shown in Figure 2 (a). We also show how to fine-tune *GraphAdapter* on the downstream tasks in Figure 2 (b), we detail this part in Section 4.3. Given the textual data and graph structural data of a node, during the pre-training process, Step 1. GNN models the node structure information; Step 2. integrates the structural information with the corresponding context hidden-states modeled by PLM; and Step 3. predicts the next token. During this pre-training process, *GraphAdapter* can learn rich information. **Align GNN with the language model.** During the learning process, the node representation obtained by GNN is constantly combined with different representations modeled by the language model for reasoning, and the entire process naturally aligns these two. **Enhance GNN in modeling graph structure.** During the entire pre-training stage, the semantic information in the textual data supervises the GNN to model the graph structural information. **Better understanding the semantics in TAG**. GraphAapter can learn how to combine LLM and GNN to model the semantic information on TAG.

4.2 Pre-training on TAGs

In the training stage, *GraphAdapter* uses the textual data of each node in TAG to train GNN.

Pipeline of pre-training: Given a text-attributed graph $\mathcal{G}$, node i and its textual data $\mathbb{S}_i = \{s_{i,0}, ..., s_{i,L_i}\}$, *GraphAdapter* uses all the tokens in $\mathbb{S}_i$ as supervision. For the k-th token, *GraphAdapter* first extracts its previous tokens $\mathcal{S}_{i,k} = \{s_{i,0}, ..., s_{i,k-1}\}$. Then, GNN models node i's structure information z_i. The structure information is then combined with the previous tokens to predict the probability distribution of the next token, where the ground truth is token $s_{i,k}$.

Structural representation: *GraphAdapter* obtains its structural features z_i through GNN. Here we use a general GNN based on the message-passing framework, which continuously aggregates neighbor features to obtain the new node's structural information. For whole process is formalized as:

$$z_i = \textbf{GNN}(x_i^0, \mathcal{X}, \mathcal{A}|\Theta_g) \tag{8}$$

where x_i^0 and $\mathcal{A}$ represent the initial node feature input and adjacency matrix in GNN, respectively. This paper used the sentence representation of the corresponding node as x_i^0. See more details about GNN in Section 3.2.

Context hidden-states. *GraphAdapter* use the pre-trained transformer in PLM to encode $\mathcal{S}_{i,k}$, it is formalized as:

$$h_{i,k} = \textbf{Transformer}(\{s_{i,0}, s_{i,1}, ..., s_{i,k-1}\}) \tag{9}$$

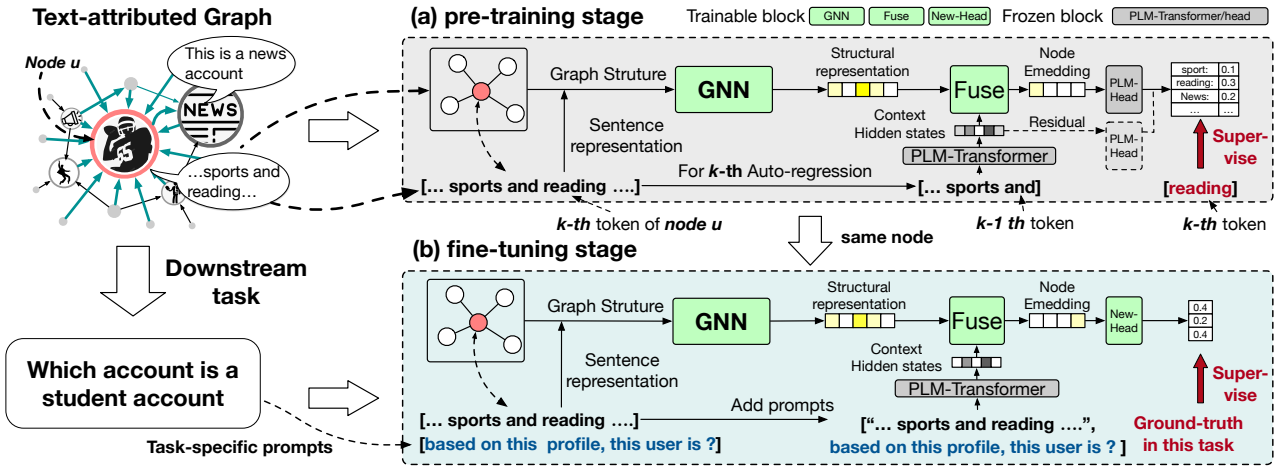

Figure 2: Framework of *GraphAdapter*. In the pre-training stage, Step 1. GNN models the node structure information, Step 2. integrates the structural information with the corresponding text fragment encoded by LM, and Step 3. predicts the masked token.

Where the **Transformer**'s parameters are trained in frozen, and $h_{i,k}$ is the context hidden-states $\mathcal{S}_{i,k}$. Note that in the pretraining stage of PLM, $h_{i,k}$ is directly used to predict the next token, so $h_{i,k}$ contains both the context information and a certain of PLMs' prediction result.

Fusion block: *GraphAdapter* next fuse structural representation into context hidden-states, which is formalized as:

$$r_{i,k} = \textbf{Fusion}(h_{i,k}, z_i|\Theta_{fuse}), \tag{10}$$

The $\textbf{Fusion}(*)$ function is trainable with parameters Θ_{fuse}. In this paper, MLPs are used as the structure of fusion. The process involves concatenating $h_{i,k}$ and z_i, and then feeding the resulting vector into MLPs.

Residual connection: the fused $r_{i,k}$ contains both structure information and context information. However, not every token's prediction requires the graph structure. For example, in the sentence "This paper focuses on graphs," the word "on" is simply a fixed collocation and easily inferred by context. Intuitively, words related to graph structure should be difficult for the language model to predict based on context. Therefore, the results of pre-trained language models are reused. We separately calculated the prediction probabilities of the language model alone and the probabilities that mixed the graph structure and the previous predictions. The two probabilities are then averaged to obtain the final prediction result. Formally:

$$\hat{s}^{LM}_{i,k} = \sigma(\textbf{Head}(h_{i,k})), \hat{s}^{GNN}_{i,k} = \sigma(\textbf{Head}(r_{i,k})) \tag{11}$$

$$\hat{s}^{ALL}_{i,k} = (\hat{s}^{LM}_{i,k} + \hat{s}^{GNN}_{i,k})/2 \tag{12}$$

Where σ denotes the softmax function. Naturally, if a token $s_{i,k}$ can be accurately predicted by the language model and the corresponding score $\hat{s}^{GNN}_{i,k}$ is evenly distributed, the overall result remains correct. Conversely, if the token cannot be predicted by the language model, the *GraphAdapter* needs to predict the correct token precisely to ensure the final result is correct. This difference leads the model to naturally focus on tokens originally predicted poorly by the language model during optimization. It then attempts to use additional structural data to enhance the overall framework's predictive performance.

Optimization: Our goal is to minimize the cross-entropy loss between the predicted probability distribution and the ground-truth distribution. Formally,

$$\mathcal{L}_{i,k} = \textbf{CrossEntropy}(\hat{s}^{ALL}_{i,k}, s_{i,k}) \tag{13}$$

$$\min_{\Theta_g, \Theta_{fuse}} \sum_{i \in V} \sum_{k \in \mathcal{S}_i} \mathcal{L}_{i,k} \tag{14}$$

Note, only $\textbf{GNN}(*|\Theta_G)$ and $\textbf{Fusion}(*|\Theta_{fuse})$ of *GraphAdapter* are trainable in whole pre-training.

GNN as Adapter: In the whole pre-training stage, the GNN combines with the frozen LM's hidden states outputted from the transformer block. The combined hidden states are then input into the PLM's prediction head. Thus, the GNN acts as an adapter, altering the language model's predictions. Since the hidden states outputted by the transformer block can be pre-processed and stored in advance. Therefore, the entire training process only requires training the GNN. Therefore, GraphAdpater can efficiently pre-train based on different scales of PLMs.

4.3 Fine-tuning with Prompts

The pipeline is shown in Figure 2 (b). *GraphAdapter* is pre-trained by token-level semantic understanding tasks. To better utilize the learned knowledge of *GraphAdapter* and the PLMs in downstream tasks, we further proposed prompt-aware fine-tuning. It inserts prompts in textual data to get task-specific sentence embedding of each node. Prompts can transform various downstream tasks on TAGs into next token prediction. E.g., the task "*Which account is a student account*" can be transformed by a next-token prediction task, "[context], *based on this profile, this user is* ". In the pre-training stage,

Table 1: The performance of different methods across three datasets. Each row corresponds to a specific method, and each column presents the performance of the models on a particular dataset. The evaluation metric used is accuracy for the Arxiv and Reddit datasets, and ROC-AUC for Instagram. The LM employed in each method is indicated in parentheses.

		Arxiv	Instagram	Reddit
LM	GNN (Ogb-feature)	0.6980 (0.0013)	-	-
	GNN (RoBERTa)	0.7129 (0.0013)	0.6123 (0.0063)	0.6191 (0.0043)
	GNN (RoBERTa+Prop)	0.7067 (0.0011)	0.6138 (0.0117)	0.6198 (0.0036)
	GIANT (BERT)	0.7262 (0.0011)	0.5986 (0.0022)	0.6379 (0.0045)
	GIANT (BERT+Prop)	0.7252 (0.0012)	0.6029 (0.0123)	0.6348 (0.0039)
	GLEM[1] (DeBERTa)	0.7550 (0.0024)	-	-
	GLEM (DeBERTa)	0.7355 (0.0034)	0.6166 (0.0056)	0.6228 (0.0060)
	GLEM (DeBERTa+Prop)	0.7315 (0.0033)	0.6105 (0.0038)	0.6221 (0.0052)
LLM	MLPs (Llama 2 + Prop)	0.7541 (0.0024)	0.6248 (0.0111)	0.6123 (0.0034)
	LoRA (Llama 2 + Prop)	0.7454 (0.0012)	0.5910 (0.0160)	0.5740 (0.0172)
	GNN (Llama 2)	0.7305 (0.0020)	0.6221 (0.0112)	0.6320 (0.0041)
	GNN (Llama 2+Prop)	0.7336 (0.0018)	0.6312 (0.0051)	0.6324 (0.0033)
	Graph2Text (Llama 2 + Prop)	0.7348 (0.0026)	0.6226 (0.0045)	0.6053 (0.0033)
	TAPE (GPT-3.5)	0.7672 (0.0007)	-	-
Ours	GraphAdapter (w/o Pre)	0.7648 (0.0020)	0.6351 (0.0077)	0.6369 (0.0025)
	GraphAdapter	**0.7707 (0.0015)**	**0.6513 (0.0075)**	**0.6461 (0.0019)**

[1]performance reported in [42]

GraphAdapter has learned how to utilize the structural information captured by GNN to enhance the accuracy of next-token prediction, therefore, under the transformed downstream task can better utilize the learned knowledge from pre-training. Formally, given textual data $\mathbb{S}_i$ of node i, we can combine a sequence of tokens with task-specific prompts behind textual data, namely, $\mathbb{S}_{i|\mathbb{P}} = [\mathbb{S}_i, \mathbb{P}]$, then we can get its sentence hidden states $h_{i|\mathbb{P}}$ through the transformer of PLM. The resulting hidden state is then fused with the node's structural representation as node representation in a specific downstream task.

$$r_{i|\mathbb{P}} = \textbf{Fusion}(h_{i|\mathbb{P}}, z_i) \tag{15}$$

This node representation can be used in various tasks. For example, in the node classification, we can append a new linear transformation to output the result, i.e., $\hat{y}_{i|\mathbb{P}} = f(r_{i|\mathbb{P}}|\theta_{new})$. In fine-tuning stage, the whole parameters $\{\Theta_g, \Theta_{fuse}, \theta_{new}\}$ in *GraphAdapter* are trainable.

5 EXPERIMENT

To comprehensively validate that *GraphAdapter* can mine the intrinsic correlation between the textual and structure data in TAGs, we conduct extensive experiments on three real-world datasets from diverse domains.

Our experimentation centered on the following five questions:

- ***Q1*: How well is *GraphAdapter* in modeling TAGs?**
- ***Q2*: Whether *GraphAdapter* can adapt to other PLMs?**
- ***Q3*: Are all components comprising *GraphAdapter* valid?**
- ***Q4*: What exactly does *GraphAdapter*'s pre-training learn?**
- ***Q5*: How efficient is GraphAdapter?**

5.1 Experiment setup

Dataset and metrics. We select three public and real-world datasets used for evaluation: Ogbn-arxiv [15] (shorted as Arxiv), Instagram [18] and Reddit[2]. The evaluation task involves node classification. The metric used for Arxiv and Reddit is accuracy, while for Instagram, it is ROC-AUC. See more details in Appendix A.

Baselines. We compare the proposed *GraphAdapter* with several TAG modeling methods. They are LM+MLPs, LM+GNN, GIANT [4], GLEM [42], Graph2Text [3], LoRA [14] and TAPE [13]. Since most of these methods can combine with different GNN blocks and PLMs, and the specific GNN framework is not the key point this paper focuses on, this paper uses GraphSAGE [11] as an instance of GNN. And we detail the used PLMs in Table 1. Please refer to Appendix A.2 for more details.

Prompts. Since *GraphAdapter* involves prompts, to make a fair comparison, we also enhance the baselines with prompts (denoted as "+Prop"). For further details, please refer to the Appendix A.3.

Implementation details. In the experiment, Llama 2 defaulted to the 13B version, while other language models defaulted to the large version. For further details, please refer to the Appendix A.4.

5.2 Performance

***Q1*: How well is *GraphAdapter* in modeling TAGs?**

***A1*: *GraphAdapter* can effectively model TAGs and surpass current state-of-the-art baselines on node classification tasks.** We compare *GraphAdapter* with 6 state-of-the-art baselines on 3 different real-world datasets to evaluate its effectiveness. As Table 1 shows, the experiment results suggest:

(1) Frozen LLMs are effective on TAGs. In general, frozen LLMs have an improved performance compared to the previous frozen LM.

[2]https://convokit.cornell.edu/documentation/subreddit.html

Table 2: The performance of the GraphAdapter based on different LM across three datasets. The evaluation metrics used for these datasets align with those outlined in Table 1.

	Arxiv			Instagram			Reddit		
	RoBERTa	GPT2	Llama 2	RoBERTa	GPT-2	Llama 2	RoBERTa	GPT-2	Llama 2
GNN (PLM)	0.7129 (0.0013)	0.7174 (0.0019)	0.7305 (0.0022)	0.6123 (0.0063)	0.6019 (0.0124)	0.6221 (0.0112)	0.6191(0.0043)	0.6282 (0.0036)	0.6320 (0.0041)
GNN (PLM+Prop)	0.7067 (0.0011)	0.6915 (0.0021)	0.7336 (0.0027)	0.6138 (0.0117)	0.6128 (0.0014)	0.6312 (0.0051)	0.6198 (0.0036)	0.6206 (0.0011)	0.6324 (0.0033)
GraphAdapter (w/o Pre)	0.7069 (0.0026)	0.7146 (0.0025)	0.7648 (0.0020)	0.6165 (0.0038)	0.6162 (0.0066)	0.6351 (0.0077)	0.6210 (0.0036)	0.6284 (0.0027)	0.6369 (0.0025)
GraphAdapter	**0.7273 (0.0021)**	**0.7325 (0.0022)**	**0.7707 (0.0015)**	**0.6292 (0.0033)**	**0.6276 (0.0034)**	**0.6508 (0.0033)**	**0.6379 (0.0061)**	**0.6441 (0.0022)**	**0.6461 (0.0019)**

Experiment results show Llama 2 has improved performance on 3 datasets by 1.34% compared to RoBERTa-based methods. LLM can better combine the information in prompts to extract task-relevant sentence representations of nodes. As the results show, prompts can bring a 0.42% improvement on average for LLM, but they could not improve the performance of LM. Frozen LLMs with prompt can surpass many GNN-LM methods that require tuning LM. Results also show that LLMs with prompts can surpass GLEM and GIANT by 0.43% and 0.79% on average, respectively.

(2) Directly fusing GNN and LLM results in unstable improvements. Compared to ordinary GNN, *GraphAdapter* (w/o Pre) only adds one fusion component to fuse the semantic representation from the LM and structural representation from the GNN. Experiment results show that directly fusing language model representations only brings improvements on Arxiv, but not obviously on other datasets. Note that the Arxiv training samples are much larger than the other datasets. This result suggests that training samples may have an impact on GNNs to understand and effectively incorporate the representations inferred by LLMs with prompts.

(3) *GraphAdapter* can effectively combine GNN and LLM, surpassing existing state-of-the-art baselines in terms of performance. The pre-training effect of *GraphAdapter* is significant, bringing an average performance improvement of 1.98% and thus surpassing existing state-of-the-art baselines. Specifically, *GraphAdapter* achieves an improvement of 4.72% over state-of-the-art cascaded GNN-LM methods and 5.40% over self-supervised GNN-LMs on average. At the same time, *GraphAdapter* also surpasses TAPE, another LLM-based method on Arxiv by 0.4% accuracy improvement.

***Q2*: Whether *GraphAdapter* can adapt to other PLMs?**

***A2*: *GraphAdapter* can be effectively pre-trained based on RoB-ERTa, GPT-2, and Llama 2, resulting in performance improvements.** We run GraphAapter based on 3 different LMs. The experiment results are shown in Table 2. *GraphAdapter* improved average performance over directly combining GNNs with frozen PLM by 1.67% on RoBERTa, 1.89% on GPT-2, and 2.77% on Llama 2. Meanwhile, *GraphAdapter* pre-training brings 1.67%, 1.50%, and 1.02% improvements on RoBERTa, GPT-2, and Llama 2 respectively. This result fully demonstrates that ***GraphAdapter* is a general and scalable method.** It is worth noting that the pre-training method of RoBERTa is different from others. *GraphAdapter* uses a pre-training task similar to RoBERTa, so there are some slight differences from the formula in Section 4. The main differences come from the loss function and language model inputs. We describe the details of applying *GraphAdapter* on Roberta in the appendix.

Table 3: The performance of different methods using the same LMs across three datasets. The evaluation metrics employed for these datasets align with those described in Table 1.

	Arxiv	Instagram	Reddit
GNN (BERT)	0.7039 (0.0013)	0.5973 (0.0063)	0.6061 (0.0043)
GIANT (BERT)	**0.7269 (0.0021)**	0.5986 (0.0022)	**0.6379 (0.0045)**
GraphAdapter (BERT)	0.7264 (0.0012)	**0.6156 (0.0032)**	0.6366 (0.0034)
GNN (RoBERTa)	0.7129 (0.0013)	0.6123 (0.0063)	0.6191 (0.0043)
GLEM (RoBERTa)	**0.7308 (0.0029)**	0.6114 (0.0075)	0.6228 (0.0018)
GraphAdapter (RoBERTa)	0.7273 (0.0021)	**0.6276 (0.0034)**	**0.6379 (0.0061)**

Under the same PLM, the performance of *GraphAdapter* is comparable to the SOTA baselines based on fine-tuning the PLM. We evaluate the performance between *GraphAdapter* and SOTA baselines under the same LM. Since the GLEM adopted DeBERTa, however, the pre-training code of DeBERTa is not open-sourced at present. To keep consistent, *GraphAdapter* and GLEM both adopt the same RoBERTa-base. As shown in Table 3, the experiment results suggest that methods based on pre-training like GIANT and *GraphAdapter* perform better on small datasets like Instagram and Reddit. Similarly, Roberta-based *GraphAdapter* outperforms GLEM by 1.57% and BERT-based GIANT outperforms GLEM by 1.15% on small datasets. Compared to baselines based on pre-training, although GIANT fine-tunes the LM, its performance is 0.51% lower than *GraphAdapter* on average. Therefore, overall, even without fine-tuning the LM, the performance of *GraphAdapter* is comparable to current state-of-the-art baselines based on fine-tuning the LM.

5.3 In-depth Analysis.

***Q3*: Are all components comprising *GraphAdapter* valid?**

***A3*: As Table 4 shows, removing any component of GraphAdapter results in performance drops.** Removing pre-training leads to a 0.91% drop, demonstrating that *GraphAdapter*'s improvements indeed come from pre-training. Next, the most significant performance drop is when we simultaneously remove pre-training and graph structure in the fine-tuning stage (keeping only self-loops), which causes a 1.95% drop. This shows having the graph is crucial for *GraphAdapter* to work. Removing the task-related prompt leads to a 0.98% drop, validating our design of aligning pre-training tasks via prompts. Notably, removing the residual learning ("w/o Res Label" that is stated in Equation 12) leads to a 1.02% drop (more than removing pre-training), suggesting that training GNNs directly on all text may introduce excessive noise and hurt performance. Therefore, *GraphAdapter* indeed benefited from residual

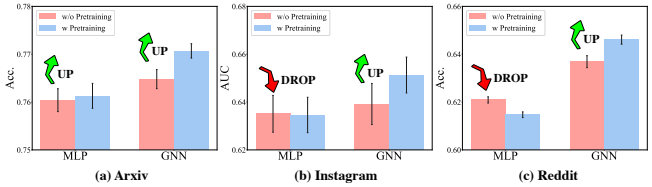

Figure 3: The performance of *GraphAdapter* before and after pre-training, using MLP and GNN as the backbone architectures. The red represents performance without pre-training, while the blue represents performance after pre-training.

learning, which utilizes language model predictions to select words more semantically related to the graph.

***Q4*: What exactly does *GraphAdapter*'s pre-training learn?** We conduct three comparative experiments to demonstrate what *GraphAdapter* pre-training is doing.

(1) GNN can obtain stronger expressive power through pre-training. We first observe the performance change of GNNs before and after pre-training, where we directly use the structural representations from the pre-trained GNN to fine-tune for downstream tasks. As Table 5 shows, the pre-trained GNN performs better on downstream tasks, improving by 0.78% on average. This demonstrates that GNNs are training their ability to model the graph structure during pre-training.

(2) Fusion block is learning how to fuse the knowledge from the language model and GNN during pre-training. We further explore whether the fusion layer learned useful knowledge during training. We randomly initialize the parameters in a specific *GraphAdapter*'s blocks after pre-trained. As Table 6 shows, initializing the parameters of the fusion layer leads to significant performance drops, decreasing by 1.03% on average across 3 datasets. This result shows that the enhanced knowledge from GNN may need to be outputted through the matching fusion layer. To further verify this conjecture, we further reinitialized the parameters of GNN, and some performance decline can also be observed, decreasing by 0.82% on average. This is similar to the impact of reinitializing the fusion layer. The fusion layer alone does not contain much knowledge. Therefore, these results demonstrate that the fusion layer can learn how to fuse the knowledge from GNN and language models.

(3) Graph structure is the basis of pre-training. We further observe the changes in different base models before and after pre-training. In this comparative experiment, we keep all the structures of *GraphAdapter*, only replacing the GNN block with MLPs of equal parameter size. As Figure 3 shows, the MLP-based *GraphAdapter* shows no significant change before and after pre-training (average improvement of 0.19%), and even decreases in performance on Instagram and Reddit (drops of 0.05% and 0.62% respectively). While the GNN improves notably before and after pre-training (average improvement of 0.91%). This result suggests that GNN is a prerequisite for effective pre-training.

These three results demonstrate that *GraphAdapter* is indeed learning graph structures via pre-training. This validates that the pre-training of *GraphAdapter* is reasonable and effective, and further supports the motivation of this paper.

Table 4: The performance of *GraphAdapter* when various components are removed. The evaluation metrics used for these tests align with those described above. The term 'w/o' indicates removing a specific component from the *GraphAdapter*.

	Arxiv	Instagram	Reddit
w/o Pretraining	0.7648 (0.0020)	0.6392 (0.0086)	0.6369 (0.0025)
w/o Graph structure	0.7604 (0.0024)	0.6346 (0.0074)	0.6147 (0.0012)
w/o Res label	0.7605 (0.0013)	0.6408 (0.0130)	0.6363 (0.0036)
w/o task-specific prompt	0.7594 (0.0030)	0.6364 (0.0073)	0.6430 (0.0021)
GraphAdapter	0.7707 (0.0015)	0.6513 (0.0075)	0.6461 (0.0019)

Table 5: The performance changes of the GNN block in *GraphAdapter* before and after pre-training. Here, "w/o Pretraining" signifies no pre-training, while "w Pretraining" indicates the opposite.

	Arxiv	Instagram	Reddit
GNN w/o Pretraining	0.7305 (0.0020)	0.6181 (0.0112)	0.6320 (0.0041)
GNN w Pretraining	**0.7335 (0.0024)**	**0.6294 (0.0038)**	**0.6410 (0.0027)**

Table 6: The performance of *GraphAdapter* after randomly initializing some blocks. Here, "Re-init" represents re-initialization.

	Arxiv	Instagram	Reddit
Re-init All	0.7648 (0.0020)	0.6392 (0.0086)	0.6369 (0.0025)
Re-init GNN	0.7680 (0.0022)	0.6390 (0.0050)	0.6364 (0.0026)
Re-init Fusion	0.7562 (0.0011)	0.6431 (0.0024)	0.6378 (0.0022)
GraphAdapter	0.7707 (0.0015)	0.6513 (0.0075)	0.6461 (0.0019)

Furthermore, we investigate the performance of *GraphAdapter* with different GNN blocks (see Appendix A.3) and conduct more detailed ablation studies (see Appendix B.1). Additionally, we analyze and report the efficiency of *GraphAdapter* (see Appendix B.3) to answer the ***Q5***. Moreover, we conduct a case study on Arxiv to further demonstrate the advantages of the proposed method (see Appendix B.4).

6 CONCLUSION

This paper proposes *GraphAdapter* to harness LLMs for TAGs without fine-tuning. A GNN adapter is trained to reduce LLM next-word errors on node texts. This adapts LLMs for graphs efficiently. Across node classification tasks, *GraphAdapter* improves accuracy by 5% over baselines. We validate with RoBERTa, GPT-2, and Llama 2, efficiently leveraging LLMs for interconnected text-graph data.

ACKNOWLEDGMENTS

This work is supported by Natural Science Foundation of China (No.62322606).

REFERENCES

[1] Claude Berge. 2001. The theory of graphs. In *Courier Corporation*.

[2] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. 2020. Language models are few-shot learners. In *NeurIPS'20*, Vol. 33. 1877–1901.

[3] Zhikai Chen, Haitao Mao, Hang Li, Wei Jin, Hongzhi Wen, Xiaochi Wei, Shuaiqiang Wang, Dawei Yin, Wenqi Fan, Hui Liu, et al. 2023. Exploring the potential of large language models (llms) in learning on graphs. In *arXiv preprint arXiv:2307.03393*.

[4] Eli Chien, Wei-Cheng Chang, Cho-Jui Hsieh, Hsiang-Fu Yu, Jiong Zhang, Olgica Milenkovic, and Inderjit S Dhillon. 2021. Node feature extraction by self-supervised multi-scale neighborhood prediction. In *arXiv preprint arXiv:2111.00064*.

[5] Courtney D Corley, Diane J Cook, Armin R Mikler, and Karan P Singh. 2010. Text and structural data mining of influenza mentions in web and social media. In *International journal of environmental research and public health*, Vol. 7. 596–615.

[6] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. Bert: pre-training of deep bidirectional transformers for language understanding. In *arXiv preprint arXiv:1810.04805*.

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

[8] Tianyu Gao, Xingcheng Yao, and Danqi Chen. 2021. Simcse: simple contrastive learning of sentence embeddings. In *arXiv preprint arXiv:2104.08821*.

[9] Johannes Gasteiger, Aleksandar Bojchevski, and Stephan Günnemann. 2018. Predict then propagate: Graph neural networks meet personalized pagerank. In *arXiv preprint arXiv:1810.05997*.

[10] Jiayan Guo, Lun Du, and Hengyu Liu. 2023. GPT4Graph: can large language models understand graph structured Data? An Empirical Evaluation and Benchmarking. In *arXiv preprint arXiv:2305.15066*.

[11] Will Hamilton, Zhitao Ying, and Jure Leskovec. 2017. Inductive representation learning on large graphs. In *NeurIPS'17*, Vol. 30.

[12] Pengcheng He, Xiaodong Liu, Jianfeng Gao, and Weizhu Chen. 2020. Deberta: decoding-enhanced bert with disentangled attention. In *arXiv preprint arXiv:2006.03654*.

[13] Xiaoxin He, Xavier Bresson, Thomas Laurent, and Bryan Hooi. 2023. Explanations as Features: LLM-Based Features for text-attributed graphs. In *arXiv preprint arXiv:2305.19523*.

[14] Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. 2021. Lora: Low-rank adaptation of large language models. In *arXiv preprint arXiv:2106.09685*.

[15] 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 *NeurIPS'20*, Vol. 33. 22118–22133.

[16] Ting Jiang, Jian Jiao, Shaohan Huang, Zihan Zhang, Deqing Wang, Fuzhen Zhuang, Furu Wei, Haizhen Huang, Denvy Deng, and Qi Zhang. 2022. Prompt-bert: improving bert sentence embeddings with prompts. In *arXiv preprint arXiv:2201.04337*.

[17] Bowen Jin, Wentao Zhang, Yu Zhang, Yu Meng, Xinyang Zhang, Qi Zhu, and Jiawei Han. 2023. Patton: language model pretraining on text-Rich networks. In *arXiv preprint arXiv:2305.12268*.

[18] Seungbae Kim, Jyun-Yu Jiang, Masaki Nakada, Jinyoung Han, and Wei Wang. 2020. Multimodal post attentive profiling for influencer marketing. In *WWW'20*. 2878–2884.

[19] Brian Lester, Rami Al-Rfou, and Noah Constant. 2021. The power of scale for parameter-efficient prompt tuning. In *arXiv preprint arXiv:2104.08691*.

[20] Chaozhuo Li, Bochen Pang, Yuming Liu, Hao Sun, Zheng Liu, Xing Xie, Tianqi Yang, Yanling Cui, Liangjie Zhang, and Qi Zhang. 2021. Adsgnn: behavior-graph augmented relevance modeling in sponsored search. In *SIGIR'21*. 223–232.

[21] Guohao Li, Matthias Müller, Bernard Ghanem, and Vladlen Koltun. 2021. Training graph neural networks with 1000 layers. In *ICML'21*. 6437–6449.

[22] Xiang Lisa Li and Percy Liang. 2021. Prefix-tuning: optimizing continuous prompts for generation. In *arXiv preprint arXiv:2101.00190*.

[23] Xiao Liu, Kaixuan Ji, Yicheng Fu, Weng Lam Tam, Zhengxiao Du, Zhilin Yang, and Jie Tang. 2021. P-tuning v2: prompt tuning can be comparable to fine-tuning universally across scales and tasks. In *arXiv preprint arXiv:2110.07602*.

[24] 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. In *arXiv preprint arXiv:1907.11692*.

[25] Zhenghao Liu, Chenyan Xiong, Maosong Sun, and Zhiyuan Liu. 2019. Fine-grained fact verification with kernel graph attention network. In *arXiv preprint arXiv:1910.09796*.

[26] Costas Mavromatis, Vassilis N Ioannidis, Shen Wang, Da Zheng, Soji Adeshina, Jun Ma, Han Zhao, Christos Faloutsos, and George Karypis. 2023. Train your own GNN teacher: graph-aware distillation on textual graphs. In *arXiv preprint arXiv:2304.10668*.

[27] Mark EJ Newman, Duncan J Watts, and Steven H Strogatz. 2002. Random graph models of social networks. In *Proceedings of the national academy of Sciences*, Vol. 99. 2566–2572.

[28] Alec Radford, Karthik Narasimhan, Tim Salimans, Ilya Sutskever, et al. 2018. Improving language understanding by generative pre-training. In *OpenAI*.

[29] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. 2019. Language models are unsupervised multitask learners. In *OpenAI*.

[30] Nils Reimers and Iryna Gurevych. 2019. Sentence-bert: sentence embeddings using siamese bert-networks. In *arXiv preprint arXiv:1908.10084*.

[31] Yu Sun, Shuohuan Wang, Yukun Li, Shikun Feng, Hao Tian, Hua Wu, and Haifeng Wang. 2020. Ernie 2.0: a continual pre-training framework for language understanding. In *AAAI'20*, Vol. 34. 8968–8975.

[32] Yijun Tian, Huan Song, Zichen Wang, Haozhu Wang, Ziqing Hu, Fang Wang, Nitesh V Chawla, and Panpan Xu. 2023. Graph neural prompting with large language models. In *arXiv preprint arXiv:2309.15427*.

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

[34] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *NeurIPS'17*, Vol. 30.

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

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

[37] Junhan Yang, Zheng Liu, Shitao Xiao, Chaozhuo Li, Defu Lian, Sanjay Agrawal, Amit Singh, Guangzhong Sun, and Xing Xie. 2021. GraphFormers: GNN-nested transformers for representation learning on textual graph. In *NeurIPS'21*, Vol. 34. 28798–28810.

[38] Yiping Yang and Xiaohui Cui. 2021. Bert-enhanced text graph neural network for classification. In *Entropy*, Vol. 23. 1–1.

[39] Rex Ying, Ruining He, Kaifeng Chen, Pong Eksombatchai, William L Hamilton, and Jure Leskovec. 2018. Graph convolutional neural networks for web-scale recommender systems. In *KDD'18*. 974–983.

[40] Shuzhou Yuan and Michael Färber. 2023. Evaluating generative models for graph-to-text generation. In *arXiv preprint arXiv:2307.14712*.

[41] Aohan Zeng, Xiao Liu, Zhengxiao Du, Zihan Wang, Hanyu Lai, Ming Ding, Zhuoyi Yang, Yifan Xu, Wendi Zheng, Xiao Xia, et al. 2022. Glm-130b: an open bilingual pre-trained model. In *arXiv preprint arXiv:2210.02414*.

[42] Jianan Zhao, Meng Qu, Chaozhuo Li, Hao Yan, Qian Liu, Rui Li, Xing Xie, and Jian Tang. 2022. Learning on Large-scale text-attributed graphs via variational inference. In *arXiv preprint arXiv:2210.14709*.

[43] Bolei Zhou, Hang Zhao, Xavier Puig, Tete Xiao, Sanja Fidler, Adela Barriuso, and Antonio Torralba. 2019. Semantic understanding of scenes through the ade20k dataset. In *IJCV'19*, Vol. 127. 302–321.

A EXPERIMENT SETTING

Table 7: Statistics of experiment datasets.

Dataset	# Nodes	# Edges	# Tokens	Split ratio (%)	#Class	Metric
Arxiv	169,343	1,166,243	35,920,710	54/18/28	40	Accuracy
Instagram	11,339	144,010	579,263	10/10/80	2	ROC-AUC
Reddit	33,434	198,448	6,748,436	10/10/80	2	Accuracy

A.1 Dataset Details

We select three public and real-world datasets used for evaluation. Table 7 shows detailed statistics of these datasets. Below are the details of these datasets:

Arxiv. Ogbn-Arxiv (shorted as Arxiv), is a citation network where edges represent citation relationships, nodes represent papers and the text attribute is the abstracts of papers. The task on this graph is to predict paper subjects. This paper uses the public partition, ground truth, and text information provided by OGB[15].

Instagram. Instagram is a social network where edges represent following relationships, nodes represent users, and the prediction task is to classify commercial and normal users in this network.

The original dataset for Instagram is provided by [18]. Since the original dataset did not contain graph information, we obtained users' follow lists, personal introductions, and tags for commercial users through Instagram's public API [3].

Reddit. Reddit is also a social network where each node denotes a user, the node features are the content of users' historically published subreddits, and edges denote whether two users have replied to each other. The prediction task is to classify whether a user is in the top 50% popular (average score of all subreddits). It is constructed on a public dataset [4] that collected replies and scores from Reddit users. The node text feature of this graph is the user's historical post content (limited to the last three posts per user). We divided users into popular and normal categories based on their average score of history posts, with users whose average score is higher than the median considered popular and others considered normal.

A.2 Baselines

We compare the proposed GraphAdapter with several state-of-the-art TAG modeling methods.

- **GNN-based methods**: This method directly combines different frozen PLM with GNNs to model TAGs. Since the specific GNN framework is not the key point this paper focuses on, this paper uses GraphSAGE [11] as an instance of GNN.
- **LM-based methods**: we select GIANT [4], and GLEM [42] as baseline. GIANT uses self-supervised tasks to finetune PLM. Then incorporates the fine-tuned PLM and GNN to model TAG. GLEM jointly trains PLM and GNN. Note that GIANT is based on BERT, and GLEM uses DeBERTa. Considering PLMs have a high influence on performance, we also compare GraphAdapter with them under the same PLM.
- **LLM-based methods**: There are a few LLM-based methods that are suitable in our setting. Therefore, we select TAPE [13] as the LLM-based baseline. This method, due to its need to obtain the interpretation of the text graph through GPT-3.5 and only the interpretation data on Arxiv is published. Therefore, we only report the results of this method on Arxiv. Besides, we also extend MLPs, Graph2Text [3], and LoRA [14] to our experiment setting. Graph2Text directly incorporates the textual data of the 1-hop neighbors of a node to model the graph structure. For example, "*It is a paper node, its abstract is represented as XXX, and the abstracts of its cited papers are represented as follows: 1. YYY, 2. ZZZ.*"

Since many baseline methods involve GNN components, which are mostly optional, and considering that different GNNs have different performances. To make a fair comparison and without loss of generality, all GNNs used in all baselines are fixed to GraphSAGE, which is a classic and general GNN model.

A.3 Prompts

According to the downstream task and graph information, this article has designed simple prompts for each dataset. As shown in Table 8. It should be noted that because PLMs are sensitive to prompts, different prompts may result in significant performance differences. However, how to find suitable prompts is not the focus of this paper, so no search for prompts is conducted.

[3]https://developers.facebook.com/docs/graph-api

[4]https://convokit.cornell.edu/documentation/subreddit.html

Most baselines rely on the sentence representations obtained from the LM. For instance, GNN+LM uses the sentence representation as a node feature. In baselines utilizing BERT or RoBERTa, we append the same prompts used in the original text to obtain prompt-aware sentence representations. When employing Llama 2, we use the same prompt and utilize the last token as the sentence representation.

Table 8: Detailed prompts on three datasets.

Dataset	Node feature	prompts
Arxiv	{ABSTRACT}	*This is a paper's abstract:* {ABSTRACT}. *Question: Based on the abstract above, this paper is published on ___ subject on Arxiv. Answer:*
Instagram	{PROFILE}	*This is a user's profile is:* {PROFILE} *Question: Based on the profile provided, this account is a ___ (answer in one word) account on Instagram. Answer:*
Reddit	{LAST 3 POSTS }	*This is a user on Reddit, his last 3 posts are:* {LAST 3 POSTS}. *Question: Based on the given posts, the style of this user is ___ (answer in one word). Answer:*

A.4 Implementation Details

We independently pre-trained GraphAdapter on three datasets. The GNN used in the pre-training process was a 2-layer GraphSAGE, and the fusion layer used a 2-layer MLP. The pre-training was conducted for 50 rounds, and we used language model techniques such as silt activation function, layer-norm, and warm-up. The hidden side of GNN in GraphAdapter is set to 128, 64, and 128 on Arxiv, Instagram, and Reddit specifically.

When using BERT or RoBERTa with GraphAdapter, there are some modifications to the GraphAdapter pipeline. Since these language models (LMs) utilize a mask-prediction task, we modify the input of $\mathcal{S}_{i,k-1} = \{si, 0...s_{i,k-1}, [mask], s_{i,k+1}...\}$ in Equation 9. Additionally, unlike based auto-regressive models, which use all tokens in pretraining, GraphAdapter based on BERT and RoBERTa only mask 20% of tokens and pre-trained by their corresponding labels.

B EXPERIMENT RESULT

B.1 Ablation Studies

We also conduct experiments that isolate and specifically compare the contributions of the base model and the fusion of the graph-language model, aiming to enhance the robustness of *GraphAdapter*. As shown in Table 10, the ranking of contributions from GNNs and LLMs varies across datasets. However, fusing GNNs and LLMs can achieve better performance in most cases. GraphAdapter not only efficiently combines GNNs and LLMs but also enhances their performance through next-token prediction pre-training.

Additionally, we investigate the effect of different GNNs with the same LM. In this experiment, we fix the LM as Llama 2 13B and compare the performance of GraphAdapter with different GNN blocks, both in pre-training and fine-tuning. We discovered that the original attention mechanism in Graph Attention Networks (GAT) is not effective for the pre-training of GraphAdapter. However, we found that a dot-product-based mechanism [3] yielded positive results. Consequently, we utilized a modified version of

Table 9: Performance of GraphAdapter with various GNN blocks. Here we fix LM as Llama 2 13B.

GNN Block	Arxiv		Instagram		Reddit	
	GAT*	SAGE	GAT*	SAGE	GAT*	SAGE
Only GNN	0.7534 (0.0016)	0.7305 (0.0020)	0.6292 (0.0055)	0.6221 (0.0112)	0.6495 (0.0031)	0.6320 (0.0041)
GraphAdapter(w/o pre)	0.7621 (0.0012)	0.7648 (0.0020)	0.6490 (0.0045)	0.6351 (0.0077)	0.6505 (0.0070)	0.6369 (0.0025)
GraphAdapter	**0.7663 (0.0016)**	**0.7707 (0.0015)**	**0.6545 (0.0034)**	**0.6513 (0.0075)**	**0.6694 (0.0041)**	**0.6461 (0.0019)**

Table 10: Results of additional ablation studies on GraphAdapter. "r Fusion" indicates replacing the Fusion component with sum-pooling, while "o" means using only a specific component.

	Arxiv	Instagram	Reddit
r Fusion	0.7698 (0.0024)	0.6450 (0.0080)	0.6361 (0.0028)
o GNN	0.7335 (0.0024)	0.6294 (0.0038)	0.6410 (0.0027)
o LLM+Prompt	0.7618 (0.0019)	0.6346 (0.0044)	0.6092 (0.0026)
GraphAdapter	0.7707 (0.0015)	0.6513 (0.0075)	0.6461 (0.0019)

Table 11: Running time of different methods on Arxiv using one Nvidia A800 80GB. Since different methods use different PLM, we also report the number of parameters for the PLM (decoded as "# para") and the number of trainable parameters ("# trainable").

	GIANT	GLEM	GraphAdapter
PLM	BERT	DeBERTa-Large	Llama 2-13B
# para of PLM	110M	139M	13B
# trainable in Pre	110M	-	3M
# trainable in Fine	0.7M	139M	2M
Pre-process	-	-	192 min
Pre-training	341 min	-	312 min
Fine-tuning	1 min	612 min	1 min
Total time costs	342 min	612 min	505 min

GAT, denoted as GAT*, and proposed this result to inspire future works. As Table 9 shows, the pre-training of GraphAdapter is also suitable for attention-based GNN.

B.2 Analysis of Prompt

We further investigated GraphAdapter's stability with regard to prompts. We observed the performance of GraphAdapter based on different prompts. As shown in Tables 12, 13, and 14, task-related prompts significantly enhance the capability of large language models (LLMs) to handle text-as-graphs (TAGs). Comparing GraphAdapter (w/o pre) with LLMs+MLPs, it is evident that graph information is beneficial for prompts in most cases. Simultaneously, GraphAdapter achieves additional improvements on top of prompts, demonstrating that the pretraining of GraphAdapter indeed facilitates the integration of graph information and prompts. Therefore, GraphAdapter stands out as a stable method suitable for different prompts

B.3 Efficient

Comparing time complexities of different methods is challenging due to varying compatible base language models. Therefore, we estimate time complexities as follows: Inference time/space complexity for a single node for the language model is T_{infer} and J_{infer}, and training time/space complexity is T_{train} and J_{train}. Complexity for non-linear transformations of PLM representations is T_{MLP} and J_{MLP}. GIANT's complexity is equivalent to the time required for fine-tuning the PLM, i.e., $O(N \times T_{train})$ for training and $O(N \times T_{infer})$ for inference. GLEM has a similar complexity to GIANT. Our approach involves a single inference pass of PLMs, after which all operations are independent of PLMs. GraphAdapter only uses the processed representation to train GNN, resulting in $O(|S_{all}| * T_{GNN})$ complexity. $|S_{all}|$ is the total number of training tokens in the TAGs. Hence, our total complexity is $O(N \times T_{infer} + |S_{all}| \times T_{GNN})$. Our primary advantage is independence from T_{train}, and T_{infer} can be accelerated by many methods. Considering larger language models where $T_{train} \gg T_{GNN}$, our approach holds a significant advantage. In terms of space complexity, our approach doesn't demand loading language model parameters during training, resulting in $O(batchsize \times J_{MLP})$ for Graph compared to $O(batchsize \times J_{GNN})$ for methods involving fine-tuning. Generally, $J_{PLM} \gg J_{GNN}$, allowing GraphAdapter to accommodate larger batch sizes in memory-restricted GPU environments. We also report efficiency comparisons for reproducibility purposes in Table 11.

B.4 Case Studies

We presented three cases in Table 15, considering different scenarios: 1. Node text acting as a distractor, 2. Graph features acting as distractors, 3. Neither the graph nor the text providing a strong signal. We denote these situations as Case A, Case B, and Case C, respectively. As shown in the cases, GraphAdapter without pretraining is effective when the text acts as a distractor (Case A, correct), but it struggles when the graph features are distracting (Case B, incorrect) due to over-reliance on graph information. After pretraining, GraphAdapter can more flexibly combine graph structural features and text features, enabling it to make judgments based on either graph or text information. Moreover, it can even make correct judgments on some very rare samples where neither the graph nor the text exhibits strong features. This indicates that pretrained GraphAdapter effectively utilizes the potential correlations between these two types of information.

Table 12: Performance of different models with different prompts on Arxiv. Here the adopted LM is Llama 2-13B.

	LLM+MLP	GraphAdapter (w/o Pre)	GraphAdapter
"*Question: Based on the abstract above, this paper is published on ___ subject on Arxiv. Answer:*"	0.7541 (0.0024)	0.7648 (0.0020)	0.7707 (0.0015)
"*Question: Please predict the subject it belongs to based on the abstract of this paper, please answer directly. Answer: *"	0.7522 (0.0021)	0.7657 (0.0019)	0.7719 (0.0025)
"*Question: This paper is __. Answer: *"	0.7381 (0.0021)	0.7554 (0.0026)	0.7581 (0.0028)
"*Question: I like __ apple. Answer: *"	0.7314 (0.0010)	0.7478 (0.0017)	0.7559 (0.0017)
None	0.7335 (0.0030)	0.7561 (0.0014)	0.7607 (0.0039)

Table 13: Performance of different models with different prompts on Instagram. Here the adopted LM is Llama 2-13B.

	LLM+MLP	GraphAdapter (w/o Pre)	GraphAdapter
"*Question: Based on the profile provided, this account is a ___ (answer in one word) account on Instagram. Answer:*"	0.6248 (0.0111)	0.6351 (0.0077)	0.6513 (0.0075)
"*Based on the profile provided, please answer the type of this account(answer in one word). Answer:*"	0.6325 (0.0098)	0.6427 (0.0044)	0.6562 (0.0031)
"*Question: This account is a __. Answer: *"	0.6298 (0.0097)	0.6388 (0.0100)	0.6392 (0.0065)
"*Question: This user likes __ apple. Answer: *"	0.6161 (0.0083)	0.6299 (0.0107)	0.6308 (0.0047)
None	0.6203 (0.0071)	0.6306 (0.0052)	0.6378 (0.0055)

Table 14: Performance of different models with different prompts on Reddit. Here the adopted LM is Llama 2-13B.

	LLM+MLP	GraphAdapter (w/o Pre)	GraphAdapter
"*Question: Based on the given posts, the style of this user is ___ (answer in one word). Answer:*"	0.6123 (0.0034)	0.6369 (0.0025)	0.6461 (0.0019)
"*Based on the given posts, please answer the popularity of this user. Answer: *"	0.6019 (0.0021)	0.6324 (0.0033)	0.6380 (0.0031)
"*Question: This user is __. Answer: *"	0.6117 (0.0032)	0.6377 (0.0022)	0.6446 (0.0021)
"*Question: This user likes __ apple. Answer: *"	0.6103 (0.0055)	0.6359 (0.0044)	0.6413 (0.0021)
None	0.6201 (0.0020)	0.6354 (0.0014)	0.6420 (0.0024)

Table 15: Three cases from the Ogbn-Arxiv dataset. LLM + MLP only utilizes abstract to predict paper's subjection, and make a wrong prediction on Case A and Case B. GraphAdapter (w/o Pre) can utilize both graph information and textual data, but also make a wrong prediction on Case B and Case C. After pretraining, GraphAdapter can make an accurate prediction on all cases.

		Case A	Case B	Case C
Feature	Title	Text classification with pixel embedding	Informative Image Captioning with External Sources of Information	A Re-evaluation of Knowledge Graph Completion Methods
	Abstract	Mentioned 3x"convolutional", 5x"kernel" 5X"3D", but only 2x"Text classification".	Focus on "image caption"	Focus on "Knowledge Graph Completion"
	Citation	cited many NLP papers.	cited 5+ papers from AAAI, and 5 papers about "Language"	cited many "Machine Learning" and "Computation and Language" papers.
Predict	LLM+MLP	Computer Vision	Computation and Language	Information Retrieval
	GraphAdapter (w/o Pre)	Computation and Language	Computer Vision	Machine Learning
	GraphAdapter	Computation and Language	Computation and Language	Computation and Language
	Ground truth	Computation and Language	Computation and Language	Computation and Language