Data-centric Prompt Tuning for Dynamic Graphs

Yufei Peng*
Beijing University of Posts and Telecommunications
Beijing, China
astral_requiem@bupt.edu.cn

Cheng Yang*
Beijing University of Posts and Telecommunications
Beijing, China
yangcheng@bupt.edu.cn

Zhengjie Fan†
Tsinghua University
Beijing, China
zjfanster@gmail.com

Chuan Shi†
Beijing University of Posts and Telecommunications
Beijing, China
shichuan@bupt.edu.cn

Abstract

Dynamic graphs have attracted increasing attention due to their ability to model complex and evolving relationships in real-world scenarios. Traditional approaches typically pre-train models using dynamic link prediction and directly apply the resulting node temporal embeddings to specific downstream tasks. However, the significant differences among downstream tasks often lead to performance degradation, especially under few-shot settings. Prompt tuning has emerged as an effective solution to this problem. Existing prompting methods are often strongly coupled with specific model architectures or pretraining tasks, which makes it difficult to adapt to recent or future model designs. Moreover, their exclusive focus on modifying node or temporal features while neglecting spatial structural information leads to limited expressiveness and degraded performance. To address these limitations, we propose DDGPrompt, a data-centric prompting framework designed to effectively refine pre-trained node embeddings at the input data level, enabling better adaptability to diverse downstream tasks. We first define a unified node expression feature matrix that aggregates all relevant temporal and structural information of each node, ensuring compatibility with a wide range of dynamic graph models. Then, we introduce three prompt matrices (temporal bias, edge weight, and feature mask) to adjust the feature matrix completely, achieving task-specific adaptation of node embeddings. We evaluate DDGPrompt under a strict few-shot setting on four public dynamic graph datasets. Experimental results demonstrate that our method significantly outperforms traditional methods and prompting approaches in scenarios with limited labels and cold-start conditions.

CCS Concepts

• **Computing methodologies** → **Machine learning**.

Keywords

Dynamic Graphs, Graph Neural Networks, Prompt Learning

ACM Reference Format:
Yufei Peng, Cheng Yang, Zhengjie Fan, and Chuan Shi. 2025. Data-centric Prompt Tuning for Dynamic Graphs. In *Proceedings of the 34th ACM International Conference on Information and Knowledge Management (CIKM '25), November 10–14, 2025, Seoul, Republic of Korea.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3746252.3761161

1 Introduction

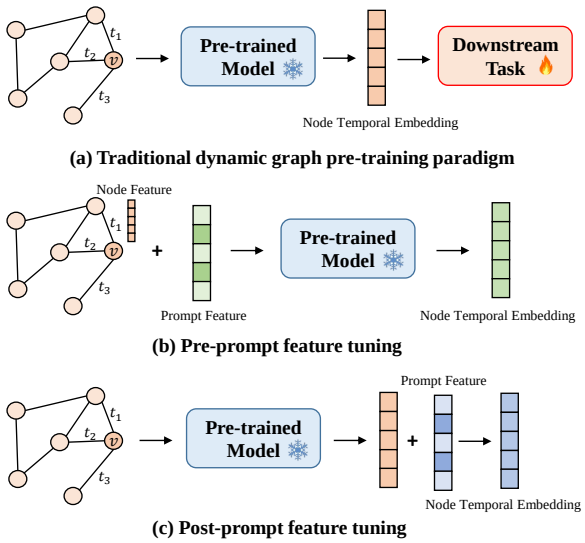

Figure 1: Comparison of (a) traditional dynamic graph methods and (b) (c) existing dynamic graph prompt works

Dynamic graphs have attracted increasing attention [7] for their ability to model real-world systems and capture, predict, and explain time-evolving behaviors more effectively. Compared with static graphs, dynamic graphs can more effectively represent complex relationships and interactions between entities in domains such as social networks [42] and recommendation systems [24], which are critical for understanding and predicting entity behavior.

Trodictional dynamic graph methods [3, 14, 25, 29, 30, 33, 37] learn node temporal embeddings through dynamic link prediction

*Co-first author with equal contribution.
†Corresponding author

CIKM '25, Seoul, Republic of Korea.

ACM ISBN 979-8-4007-2040-6/2025/11
https://doi.org/10.1145/3746252.3761161

losses using node interaction data during pretraining, and then directly apply these embeddings to downstream tasks. These methods have shown excellent performance, particularly on dynamic link prediction. Despite this progress, current dynamic graph methods still have some shortcomings. First, when adapting to various downstream tasks, traditional methods typically take one of two approaches: (1) freezing the parameters of the pre-trained model and training a lightweight decoder to map node temporal embeddings to the task; or (2) fully retraining the model for each new task. The former may introduce noise due to discrepancies between tasks, while the latter incurs considerable computational and memory costs. Second, although current methods perform well in data-rich scenarios, they often struggle in real-world applications where labeled data is scarce or where cold-start issues arise. In such few-shot or sparsely labeled settings, models that rely heavily on large-scale labeled data typically fail to generalize.

In order to solve the above problems in dynamic graphs, some recent approaches have explored prompt tuning [2, 39], which has proven effective in the field of static graphs [4, 13, 18, 27]. As shown in Fig. 1, these methods introduce lightweight prompt modules that adjust graph input features or output embeddings with minimal additional parameters. Specifically, TIGPrompt [2] incorporates recent interaction features as post-prompt added to pre-trained node temporal embeddings. DyGPrompt [39] applies a unified learnable embedding to all node and time features, further fine-tuned by pre-prompt based on the interaction of node and time features.

However, recent prompting methods for dynamic graphs still suffer from two critical limitations. First, existing approaches often require intervention into model-specific structures or adopt entirely different pretraining and inference paradigms from prior works. For instance, DyGPrompt aligns downstream tasks with link-based pretraining objectives through its prompting mechanism, deviating from the common paradigm where node embeddings are directly fed into task-specific classifiers. This design makes it difficult to integrate DyGPrompt into existing models and may result in significant performance loss. Additionally, both TIGPrompt and DyGPrompt focus solely on modifying node or time features, completely ignoring the spatial neighborhood structure that is intrinsic to dynamic graphs. This under-expressive prompting strategy hampers their ability to comprehensively capture task-relevant patterns, ultimately constraining downstream performance.

To address these two challenges, we propose DDGPrompt, a novel data-centric prompting framework for dynamic graphs. DDGPrompt adjusts the input data structure in a unified and model-agnostic manner, enabling better alignment between pre-trained models and downstream tasks. Specifically, to tackle the first challenge, we define a node expression feature matrix inspired by existing dynamic graph models. This matrix is compatible with a wide range of traditional architectures and designed to remain extensible to future frameworks. It encodes a node's recent first-order neighborhood information, including node features, edge features, and time features, and serves as input to various backbone models to get node temporal embeddings. To address the second challenge, we introduce three complementary prompt matrices (temporal bias, edge weight, and feature mask) to comprehensively adjust the node expression feature matrix. The temporal bias prompt dynamically adjusts the temporal features of each neighbor; the edge weight prompt assigns a learnable importance score to each neighbor, capturing spatial structural relevance; and the feature mask prompt leverages a task-aware enhancement network to selectively modulate feature dimensions. These prompts are integrated into the node expression matrix through different weights, allowing the resulting temporal embeddings to be more expressive and better adapted to diverse downstream tasks.

Finally, we conduct extensive experiments on four public dynamic graph datasets. Thanks to our proposed node expression feature matrix and prompting strategy, DDGPrompt achieves strong performance across various challenging few-shot scenarios. It is compatible with existing dynamic graph models and consistently outperforms both backbone baselines and recent prompt-based methods.

Our contributions are summarized as follows:

- We define a node expression feature matrix for efficient prompt tuning in dynamic graphs. This matrix encodes the historical interaction information of each node and maintains compatibility with all existing methods. It serves as the model input to generate node temporal embeddings that are transferable across different downstream tasks.
- We propose a novel data-centric prompting framework, DDGPrompt, which integrates temporal bias, edge weight, and feature mask prompts. By jointly capturing node, temporal, and spatial information, it adaptively refines the node expression matrix for task-specific tuning.
- Extensive few-shot experiments on four dynamic graph datasets demonstrate that DDGPrompt consistently outperforms various backbones and prompt methods, highlighting its robustness and effectiveness.

2 Related work

2.1 Dynamic Graph Learning

Dynamic graphs can be categorized into discrete-time dynamic graph (DTDG) and continuous-time dynamic graph (CTDG) based on the granularity of their temporal information [1, 12, 26]. In recent years, CTDGs have attracted increasing attention due to their closer alignment with real-world scenarios [3, 14, 25, 30, 33, 37]. Most existing CTDG methods share a common architecture: they design a backbone encoder to extract spatio-temporal information from the evolving graph and obtain node representations, followed by a decoder that leverages these embeddings to perform downstream tasks. For instance, TGN [25] introduces a memory module to store and update node embeddings over time. GraphMixer [3] employs a lightweight MLP-based architecture that significantly reduces memory and computation cost while maintaining competitive performance. TCL [30], built upon contrastive learning, uses a transformer-based backbone to capture temporal dynamics. Our work also focuses on the CTDG setting.

While these methods have achieved perfect performance on dynamic link prediction, they often fall short when applied to other downstream tasks. This limitation is particularly pronounced in dynamic graphs due to inherent temporal gaps and node preference variability over time, leading to even greater performance degradation across tasks compared to static graphs. Furthermore, most

existing approaches overlook few-shot and cold-start scenarios. In practical settings where labeled data is scarce or user-item interactions are sparse, such models tend to generalize poorly and lack robustness.

2.2 Graph pretraining

Pretraining has emerged as a prevalent paradigm in graph learning [8–10, 19, 23]. By training models on large-scale datasets and then fine-tuning them for specific downstream tasks, pretraining significantly boosts performance. This paradigm leverages the power of transfer learning [20, 31, 43], allowing models to generalize better by reusing the knowledge acquired during pretraining.

Recent dynamic graph learning methods have begun to adopt pretraining strategies to enable transferability across multiple tasks [37]. Typically, these methods perform supervised pretraining using dynamic link prediction task. When adapting to new downstream tasks, they freeze the parameters of the pre-trained model and retrain only a task-specific classifier based on the node temporal embeddings produced during pretraining.

However, this approach has inherent limitations. Since the temporal embeddings are optimized for a specific task during supervised pretraining, they may encode task-biased information, resulting in suboptimal generalization to other downstream tasks. To address this issue, it is critical to incorporate task-agnostic self-supervised signals during pretraining. It can enhance the versatility of node embeddings, improve performance across diverse downstream tasks, and offer greater robustness in low-resource or few-shot scenarios where labeled data are scarce.

2.3 Prompt Learning on Graphs

In the past two years, prompt learning has been increasingly applied to the graph learning domain and has achieved remarkable success [4, 5, 13, 18, 27, 28, 38, 40]. These studies explore ways to align diverse downstream tasks with pre-trained graph models and have shown strong performance, particularly in few-shot settings. Some works even aim to unify various tasks and datasets under a single graph foundation model [6, 16, 17, 32, 40]. Existing prompt learning approaches for static graphs can be broadly categorized into two types [34]: (1) *Pre-prompt* usually modifies the model input. This idea is analogous to the prompt in natural language processing. For instance, GPF [4] proposes universal feature prompts for all nodes, while All-in-one [13] explicitly alters the graph structure to encode task-specific signals. (2) *Post-prompt* usually uses the prompt vector to modify the node embedding obtained by the model to adapt to different downstream tasks. For example, GraphPrompt [18] introduces a prompt tensor that multiplies the node embeddings, enabling the generation of task-specific node representations via a unified prompt template. Beyond general graph tasks, prompt learning has also been applied to more specialized settings [5, 35].

Recently, a few works have extended prompt learning to dynamic graphs [2, 39]. TIGPrompt [2] injects recent interaction history as prompt signals to enhance node embeddings. DyGPrompt [39], inspired by GraphPrompt, designs global feature prompts for both node and time features while incorporating their interactions. However, despite these efforts, such methods often struggle to generalize across different datasets due to the limited expressiveness of the learned prompt representations.

3 Preliminary

Definition 1. Dynamic Graph Dynamic graph can be represented as $G = (V, E)$ [11]. V and E are denoted as the node set and the edge set, respectively. Each edge consists of a triple $e = (u, v, t) \in E$, where $u \in V$ is the source node, $v \in V$ is the destination node, and t is the timestamp, indicating that u and v interact at timestamp t.

The features corresponding to nodes u, v and edge e are denoted as $f_u, f_v \in \mathbb{R}^{d_N}$ and $f_e \in \mathbb{R}^{d_E}$, respectively. d_N and d_E are feature dimensions.

Definition 2. Problem Formalization.

Dynamic link prediction: For a pair of nodes u and v in a dynamic graph G and a given time t, the purpose of dynamic link prediction is to predict whether there will be a link between the two nodes at time t based on all interactions $\{(u', v', t') | t' < t\}$ before the timestamp.

Dynamic node classification: For each edge $e = (u, v, t)$ in the dynamic graph G, source node u corresponds to a label $l \in L$ at timestamp t, where the labels of all source nodes constitute the label set L. The goal of dynamic node classification is to predict the label of the source node u at timestamp t.

In this paper, we focus on two fundamental tasks on dynamic graphs under the few-shot setting, as real-world dynamic graphs often suffer from limited supervision signals [15, 22, 36, 41].

4 Methodology

4.1 Overall Framework

The overall framework consists of two parts: pretraining and downstream task fine-tuning. In the pretraining stage, we first train the model using a large amount of unlabeled data through self-supervised contrastive learning [21] based on link prediction. Afterwards, we first define a node expression feature matrix for each node. This matrix is then adjusted using three prompt matrices. The result is used as the input of the pre-trained model with frozen parameters, and the node temporal embedding modified by prompt is obtained for different downstream tasks. The overall framework of DDGPrompt is shown in Fig. 2.

4.2 Node Expression Feature Matrix

In this section, we first define a node expression feature matrix for each node based on the common design patterns of existing dynamic graph models, aiming to optimize node temporal embeddings through prompt tuning. Previous works [3, 30, 37] have shown that it is sufficient to capture the general characteristics of a node through its first-order interactions in its recent history. Specifically, for an interaction (u, v, t), we take node u as an example and extract the most recent sequence of interactions $S_u^t = \{(u, v', t') | t' < t\} \in G$ of node u from the interaction history. The original features of all neighbor nodes are $\mathbf{F}_{u,neigh}^t \in \mathbb{R}^{|S_u^t| \times d_N}$. Consequently, the edge features and the time features are $\mathbf{F}_{u,edge}^t \in \mathbb{R}^{|S_u^t| \times d_E}$ and $\mathbf{F}_{u,time}^t \in \mathbb{R}^{|S_u^t| \times d_T}$. For time features, previous works [3, 33] use time encoding function $f_{time}(\Delta t; \omega)$ to encode the time interval

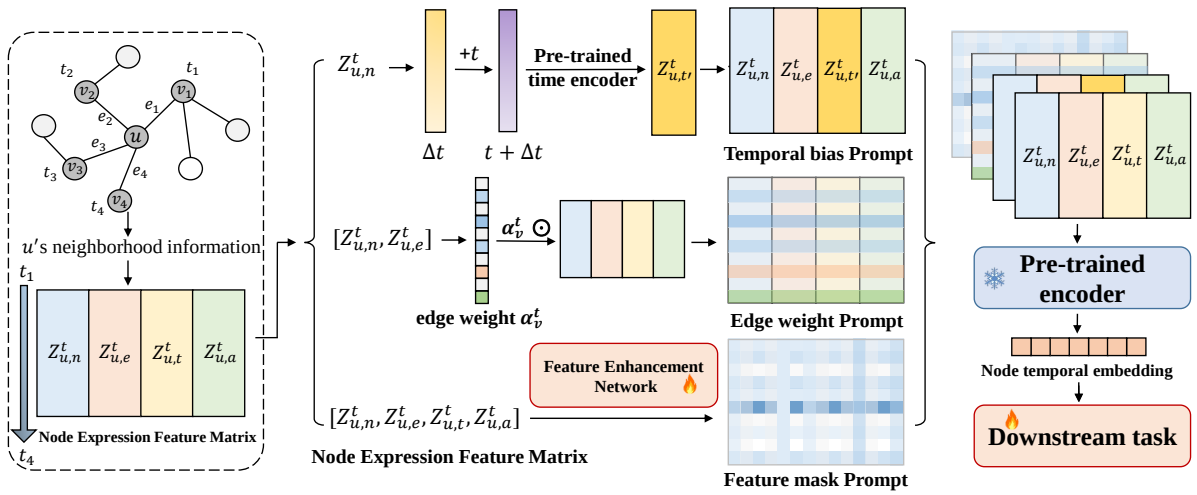

Figure 2: The overall framework of our proposed DDGPrompt

$\Delta t = t - t'$ into T-dimensional time features $\mathbf{F}^t_{u,time}$, where ω is the parameter of the time encoding function.

Next, we use a linear layer to project each original feature into the same feature space so that they have the same dimension d and obtain the embedding of each feature.

$$\mathbf{Z}^t_{u,neigh} = \text{Projection}(\mathbf{F}^t_{u,neigh}) \in \mathbb{R}^{|S^t_u| \times d} \tag{1}$$

$$\mathbf{Z}^t_{u,edge} = \text{Projection}(\mathbf{F}^t_{u,edge}) \in \mathbb{R}^{|S^t_u| \times d} \tag{2}$$

$$\mathbf{Z}^t_{u,time} = \text{Projection}(\mathbf{F}^t_{u,time}) \in \mathbb{R}^{|S^t_u| \times d} \tag{3}$$

We define the node expression feature matrix of node u at timestamp t as follows:

$$\mathbf{Z}^t_u = \mathbf{Z}^t_{u,neigh} || \mathbf{Z}^t_{u,edge} || \mathbf{Z}^t_{u,time} \in \mathbb{R}^{|S^t_u| \times 3d} \tag{4}$$

where $||$ represents the concatenate operation.

In addition, for some specific methods, they may propose some additional features to improve the performance of specific tasks. For example, DyGFormer [37] proposes a co-occurrence feature $\mathbf{Z}^t_{u,occ} \in \mathbb{R}^{|S^t_u| \times d}$ between a pair of nodes to extract the implicit information. We will collectively denote this additional features as $\mathbf{Z}^t_{u,add}$. If $\mathbf{Z}^t_{u,add}$ exists, we add it to the end as $\mathbf{Z}^t_u || \mathbf{Z}^t_{u,add}$. This makes it compatible with existing methods and directly extensible to future work. Obviously, the matrix $\mathbf{Z}^t_u$ is a combination of features for each historical neighbor of the node u at timestamp t.

Afterwards, the feature matrix $\mathbf{Z}^t_u$ will be used as input to different backbone models to extract the information from node u and output the node temporal embedding h^t_u.

4.3 Self-supervised pretraining

We leverage the node expression feature matrix introduced in Section 4.2 to pretrain the backbone model using a large amount of unlabeled dynamic graph data, resulting in the initial node temporal embeddingh^t_u. To optimize the pre-trained model, we adopt a self-supervised contrastive learning loss based on link prediction, as the dynamic interactions between nodes inherently capture rich temporal and structural information from the graph.

Specifically, for an interaction $(u, v, t) \in G_{pre-train}$ of node u in the dynamic graph G, where $G_{pre-train}$ is the pretraining dataset split on G. We randomly sample another node $v-$, which forms a triple $(u, v-, t)$ with u as a negative sample, indicating that there is no edge between u and v up to the timestamp t. We use contrastive learning to make the embeddings of two nodes with a link relationship similar and the embeddings of two nodes without a link relationship far away from each other. Therefore, for each interaction, we use the following loss function to optimize the pre-trained model:

$$\mathcal{L}_{\text{pre}}(\Phi) = -\sum_{(u,v,t) \in G_{pre-train}} \ln \frac{e^{\frac{1}{\tau}\text{sim}(h^t_u, h^t_v)}}{e^{\frac{1}{\tau}\text{sim}(h^t_u, h^t_v)} + e^{\frac{1}{\tau}\text{sim}(h^t_u, h^t_{v-})}} \tag{5}$$

where Φ is the trainable parameter of the pre-trained backbone. τ is the temperature parameter. h^t_u, h^t_v, h^t_{v-} is the temporal embedding of the node at timestamp t for downstream tasks, which is calculated by the pre-trained model with Φ. $\text{sim}(\cdot)$ is used to measure the similarity between node embeddings, and we use the cosine similarity.

4.4 Data-centric Dynamic Graph Prompt

To bridge the gap between pre-trained embeddings and downstream tasks, we introduce three types of prompts that refine the node expression feature matrix from node, temporal, and spatial perspectives. These prompts adjust the resulting temporal embeddings, enabling the pre-trained model to adapt to downstream tasks more efficiently and effectively.

4.4.1 Temporal bias prompt matrix. The relationship between the node attribute and its time feature is a key point that distinguishes dynamic graphs from traditional graphs. However, prior methods often model the features of neighboring nodes and their corresponding timestamps independently, overlooking the crucial interactions between them [3, 33, 37]. In fact, enhancing the time features based

on the neighbor node feature can more accurately capture the behavioral preferences of the current node. For instance, adjusting a neighbor's interaction timestamp to be closer to the current time may indicate a stronger preference or relevance to that neighbor. To capture this intuition, we propose a temporal bias prompt that explicitly modifies the time feature to reflect such dynamics.

We first extract the neighbor node feature $\mathbf{Z}^t_{u,neigh}$ of the node expression feature matrix and generate a one-dimensional temporal bias δt through a linear layer as the time gap that needs to be modified for different neighbor nodes.

$$\delta t = \text{Linear}(\mathbf{Z}^t_{u,time}; \eta) \tag{6}$$

where η is the learnable parameter of the linear layer. Subsequently, we modified the the original time interval Δt using temporal bias δt so that the interaction time of neighbor are adjusted to either a more recent or more distant point in time. We use ReLU as the activation function to prevent the time interval from being less than 0.

$$\overline{\Delta t} = \text{ReLu}(\Delta t + \delta t) \tag{7}$$

The modified time interval $\overline{\Delta t}$ will be used as input to obtain new time features $\overline{\mathbf{Z}}^t_{u,time}$ using the time encoding function and projection layer of the pre-trained model as before.

$$\overline{\mathbf{Z}}^t_{u,time} = \text{Projection}(\text{TimeEncoder}(\overline{\Delta t})) \tag{8}$$

We use $\overline{\mathbf{Z}}^t_{u,time}$ to replace the original time feature and obtain the temporal bias prompt matrix $\mathbf{P}_{temp}$of the same size as the original node expression feature matrix.

$$\mathbf{P}_{temp} = \mathbf{Z}^t_{u,neigh} || \mathbf{Z}^t_{u,edge} || \overline{\mathbf{Z}}^t_{u,time} \in \mathbb{R}^{|S^t_u| \times 3d} \tag{9}$$

4.4.2 Edge weight prompt matrix. Previous traditional and prompt methods overlook the varying importance of different neighbors when aggregating neighborhood information. They typically assign equal weights to all neighbors [2, 3, 30, 37, 39]. However, this uniform treatment becomes a critical limitation when adapting to different downstream tasks, often resulting in significant performance degradation. For example, in dynamic link prediction, it is beneficial to assign higher weights to frequently interacting node pairs, whereas dynamic node classification may prioritize the most recent interactions. Therefore, capturing task-specific neighbor importance is essential for improving generalization across diverse tasks.

Therefore, we generate the edge weights according to the different neighbor node features and edge features to adaptively adjust the information expression on different edges. Specifically, for node u's neighbor node v at timestamp t, we concatenate the neighbor node features $z^t_{uv,neigh}$ and edge features $z^t_{uv,edge}$ and use them as the input of a linear layer to get the edge weight w^t_{uv}.

$$w^t_{uv} = \text{Linear}(z^t_{uv,neigh} || z^t_{uv,edge}; \zeta) \tag{10}$$

where ζ is the learnable parameter of the linear layer. The weight of all edges is denoted by w^t_u. We then multiply the weight by the corresponding edge.

$$\mathbf{P}_{edge} = w^t_u \odot \mathbf{Z}^t_u \tag{11}$$

where $(\odot)$ is the element-wise multiplication. It modifies all feature information on an edge through a broadcast mechanism. Finally, we get the edge weight prompt matrix $\mathbf{P}_{edge}$.

4.4.3 Feature mask prompt matrix. Finally, we introduce a feature mask prompt matrix to enhance the representation of each neighbor's features. The goal is to adaptively refine the original node feature matrix through a Feature Enhancement Network (FEN). To keep the architecture lightweight, we employ a two-layer MLP with a bottleneck structure. The enhancement process is defined as follows:

$$\mathbf{P}_{feat} = \text{MLP}(\mathbf{Z}^t_u; \Omega) = \mathbf{W_2} \cdot \text{Relu}(\mathbf{W_1} \cdot \mathbf{Z}^t_u + b_1) + b_2 \tag{12}$$

Here, $\mathbf{W}_1, \mathbf{W}_2$ and b_1, b_2 denoted as Ω are the learnable parameters of the MLP. The output $\mathbf{P}_{feat}$ also maintains the same dimensionality as the original feature matrix, ensuring compatibility for subsequent operations. This design is also commonly adopted in traditional static graph learning. It does not assume any specific structure or distribution of node features, making it applicable to a wide range of graph-based tasks.

Finally, we add the three prompt matrices to the original node expression feature matrix with different weights.

$$\overline{\mathbf{Z}}^t_u = \mathbf{Z}^t_u + \alpha \odot \mathbf{P}_{temp} + \beta \odot \mathbf{P}_{edge} + \gamma \odot \mathbf{P}_{feat} \tag{13}$$

α, β, γ are the weight hyperparameters corresponding to the three prompts, which are used to further adjust the expression of the three prompts on the original feature matrix. $\overline{\mathbf{Z}}^t_u$ is the prompt-adjusted node expression feature matrix. It is treated as input and passes through pre-trained backbone Pre-train($\mathbf{\Phi}$) to generate new node temporal embedding $\overline{h}^t_u$.

$$\overline{h}^t_u = \text{Pre-train}(\overline{\mathbf{Z}}^t_u; \Phi) \tag{14}$$

4.5 Downstream Task Tuning

In line with previous methods [30, 33, 37], we finally train MLP as a classifier to apply prompt-adjusted node temporal embeddings to different downstream tasks. Both dynamic link prediction and dynamic node classification are formulated as binary classification problems, and we adopt the binary cross-entropy (BCE) loss function for optimization. Additionally, a regularization term is introduced for the FEN to mitigate overfitting. For dynamic link prediction, we apply the following loss on the downstream training set $G_{fine-tune} \in G$, optimizing only the prompt parameters and the classifier, while keeping the backbone frozen. Here, l denotes the ground-truth label in $G_{fine-tune}$.

$$\mathcal{L}_{\text{link}}(\eta, \zeta, \Omega, \Theta) = \text{Cross-Entropy}(\text{MLP}(\overline{h}^t_u || \overline{h}^t_v; \Theta), l) + \lambda ||\Omega||^2 \tag{15}$$

η, ζ, Ω are the trainable parameters for prompt. Θ is the trainable parameters of the MLP. λ represents the strength of regularization, and $||\cdot||$ is L2-norm.

Similarly, the loss for dynamic node classification is as follows:

$$\mathcal{L}_{\text{node}}(\eta, \zeta, \Omega, \Theta) = \text{Cross-Entropy}(\text{MLP}(\overline{h}_u^t; \Theta), l) + \lambda||\Omega||^2 \quad (16)$$

Algorithm 1 DDGPrompt Pretraining and Fine-tuning Framework

Input: Dynamic graph pretraining dataset $G_{pre-train}$ and fine-tuning dataset $G_{fine-tune}$, node features set F_N, edge features set F_E, label set label L, backbone model f_Φ with parameters Φ

// Pretraining Stage:
for each edge $e = (u, v, t) \in G_{pre-train}$ **do**
  for node u, extract the features $\mathbf{F}^t_{u,neigh}, \mathbf{F}^t_{u,edge}, \mathbf{F}^t_{u,time}$ of the node's one-hop neighbors
  Calculate project features $\mathbf{Z}^t_{u,neigh}, \mathbf{Z}^t_{u,edge}, \mathbf{Z}^t_{u,time}$ via Eq. 3
  $\mathbf{Z}^t_u \leftarrow Contact(\mathbf{Z}^t_{u,neigh}, \mathbf{Z}^t_{u,edge}, \mathbf{Z}^t_{u,time})$
  Sample negetive node and get $\mathbf{Z}^t_v, \mathbf{Z}^t_{v-}$ similarity
  Calculate embeddings h^t_u, h^t_v, h^t_{v-} from f_Φ
  Calculate loss $\mathcal{L}_{\text{pre}}(\Phi)$ and optimize model paremeter Φ
end for
// Fine-tuning Stage:
for each edge $e = (u, v, t) \in G_{fine-tune}$ **do**
  $\delta t \leftarrow \text{Linear}(\mathbf{Z}^t_{u,time}; \eta), \overline{\Delta t} \leftarrow \text{ReLu}(\Delta t + \delta t)$
  Calculate time bias prompt $\mathbf{P}_{temp}$ via Eq. 9
  $w^t_u \leftarrow Linear(\mathbf{Z}^t_{u,neigh}||\mathbf{Z}^t_{u,edge}; \zeta)$
  Calculate edge weight prompt $\mathbf{P}_{edge}$ via Eq. 11
  Calculate feature mask prompt $\mathbf{P}_{feat} \leftarrow \text{MLP}(\mathbf{Z}^t_u; \Omega)$ via Eq. 12
  $\overline{Z}^t_u \leftarrow Fine-tuning(\mathbf{P}_{neigh}, \mathbf{P}_{edge}, \mathbf{P}_{feat})$ via Eq. 13
  Calculate embeddings $\overline{h}^t_u, \overline{h}^t_v, \overline{h}^t_{v-}$ from freezed pre-trained model f_Φ
  Calculate loss $\mathcal{L}_{\text{link}}$ or $\mathcal{L}_{\text{node}}$ and optimize prompt paremeter η, ζ, Ω and MLP classifier Θ via Eq. 15 or Eq. 16
end for

5 Experiment

In this section, we conduct extensive experiments to evaluate the performance of our model in the scenario where labeled data is scarce.

5.1 Experimental Setup

5.1.1 Datasets. We evaluate our approach and baselines using four widely used benchmark datasets (Wikipedia, Reddit, MOOC, and a large-scale dataset LastFM) on dynamic graphs. The detailed description of the dataset is as follows:

• **Wikipedia** records the editing history and interaction behavior of users on the platform. The nodes represent different users or pages, and the edges represent the interaction information between users and pages.

• **Reddit** is similar to Wikipedia, recording the interactions between users and different sub-posts on the Reddit website. The timestamp indicates the time of the interaction.

• **MOOC** is derived from the learning activities and interactive behaviors on the online course platform, reflecting the students' participation in MOOC courses.

Table 1: Detailed statistics of Datasets

Datasets	Wikipedia	Reddit	MOOC	LastFM
Nodes	9227	11000	7144	1980
Edges	157474	672447	411749	1293103
Feature dimension	172	172	172	0
Node classes nums	2	2	2	0
Dynamic labels	217	366	4066	0
Timespan	30 days	30 days	30 days	30 days

• **LastFM** records the listening history of users on the music platform within a month, but there are no dynamic labels to indicate the user's status. It is a commonly used large dataset with 1.29 million interactions for link prediction on dynamic graphs.

5.1.2 Task Setting. We evaluate our method on two fundamental tasks in dynamic graphs: dynamic link prediction and dynamic node classification. For link prediction, we consider both transductive and inductive settings. In the transductive setting, the model has access to all nodes in the graph during training. In contrast, the inductive setting requires the model to predict links involving nodes that are entirely unseen during training. Due to the absence of dynamic labels, we only conduct experiments on the dynamic link prediction task using the LastFM dataset.

In our few-shot setting, we adopt a strict and challenging experimental protocol to evaluate the effectiveness of our method with extremely limited labeled samples. Specifically, we split the four datasets in the following ways: (1) First, we select the first 80% of the interaction data in chronological order as the pretraining set $G_{pre-train}$ for self-supervised pretraining. (2) Second, for the remaining 20% of the data, we then select K interactions as K-shot training data to fine-tune in different downstream tasks. Similarly, we continuously select K samples as the validation set for fine-tuning, denoted as $G_{fine-tune}$ and G_{val}. Finally, for all the remaining data, we use it as the test set G_{test} for downstream tasks. We just select K = 70 as the 70-shot scenario for all downstream tasks in the main experiment, which accounts for only 0.05 percent of all interactions for the smallest dataset, Wikipedia.

Besides, for the dynamic node classification task, we first ensure that at least one node is selected in each class when constructing the fine-tuning set and the validation set.

5.1.3 Baselines. We compare DDGPrompt with the following two types of baseline models. (1) **Dynamic graph learning**: JOIDE [14], TGAT [33], TGN [25], GraphMixer [3], TCL [30], DyGFormer [37]. (2) **Dynamic graph prompt**: TIGPrompt [2], DyGPrompt [39].

For TIGPrompt and DyGPrompt, we adopt the best implementation setting reported in their original papers by default, i.e., using TGN as the backbone. Since the code of TIGPrompt and DyGPrompt are not available, we implemented them based on the descriptions provided in their respective papers. Additionally, to ensure a fair comparison, we also integrated the prompts into several recent dynamic graph methods and reported the corresponding results in the main experiments.

Table 2: Experimental results of few-shot scenario

Methods	Transductive Link Prediction								Inductive Link Prediction								Node Classification		
	Wikipedia		Reddit		MOOC		LastFM		Wikipedia		Reddit		MOOC		LastFM		Wikipedia	Reddit	MOOC
Metrics	**AP↑**	**AUC↑**	**AP↑**	**AUC↑**	**AP↑**	**AUC↑**	**AP↑**	**AUC↑**	**AP↑**	**AUC↑**	**AP↑**	**AUC↑**	**AP↑**	**AUC↑**	**AP↑**	**AUC↑**	**AUC↑**	**AUC↑**	**AUC↑**
JODIE	52.26	50.22	56.28	55.08	49.15	49.29	49.09	45.88	52.86	50.52	58.54	55.35	47.66	47.51	48.74	42.84	72.93	57.12	52.42
TGN	58.67	56.63	57.45	54.34	53.92	55.39	54.17	53.81	58.64	56.41	53.36	49.59	54.13	54.39	50.73	49.70	41.54	51.86	49.20
TGAT	73.98	71.32	63.83	64.43	69.19	68.99	59.20	51.88	75.80	73.65	61.31	62.84	67.93	69.12	53.82	52.73	61.92	57.38	49.17
GraphMixer	74.63	73.52	81.13	79.88	72.03	71.19	51.40	50.33	74.43	73.70	78.25	77.78	72.19	72.03	51.40	50.27	54.53	55.01	57.82
TCL	80.37	74.57	87.54	88.77	78.14	79.65	52.81	50.82	80.42	74.90	86.14	87.57	77.30	79.28	52.72	50.81	61.71	54.01	52.56
DyGFormer	85.78	83.03	97.66	97.50	70.94	69.49	64.83	66.44	86.93	83.58	97.97	97.56	68.74	67.17	64.94	66.72	63.67	60.07	54.45
TIGPrompt	58.35	56.64	54.72	51.62	54.76	55.30	53.22	52.77	57.73	55.93	51.57	47.33	54.91	54.65	50.60	49.51	46.13	53.85	56.46
DyGPrompt	50.65	49.56	55.41	52.49	52.92	54.83	48.97	47.05	50.86	49.90	52.03	49.14	54.23	54.34	48.66	46.46	46.16	49.60	48.98
TIGPrompt(T)	81.32	76.08	85.99	86.19	77.66	78.83	54.39	52.32	81.37	76.36	83.73	85.09	76.62	78.15	54.33	52.43	59.34	55.16	59.13
DyGPrompt(T)	79.89	77.53	83.80	86.50	53.30	55.24	51.32	50.67	79.15	76.58	82.33	84.29	53.17	55.76	51.37	50.75	75.03	55.42	42.08
DDGPrompt(T)	88.06	84.23	86.35	87.57	**79.41**	**80.96**	50.59	49.63	87.42	83.49	84.81	86.32	**78.47**	**80.43**	50.53	49.58	66.24	55.99	49.62
TIGPrompt(D)	84.45	81.83	98.45	98.22	69.88	68.49	62.11	63.96	85.78	82.47	98.54	98.32	67.61	65.89	62.14	64.17	65.96	53.82	57.20
DyGPrompt(D)	92.76	90.33	60.19	58.86	67.17	67.08	72.25	69.23	92.85	90.38	60.06	58.72	65.58	66.02	72.46	68.96	67.81	57.35	45.12
DDGPrompt	**97.15**	**96.32**	**98.53**	**98.36**	72.22	71.03	**80.48**	**76.29**	**96.91**	**96.01**	**98.59**	**98.41**	70.76	69.67	**80.53**	**76.28**	**74.32**	**61.30**	**59.60**

Table 3: Experimental results of sparse interaction data scenario

Methods	Transductive Link Prediction								Inductive Link Prediction								Node Classification		
	Wikipedia		Reddit		MOOC		LastFM		Wikipedia		Reddit		MOOC		LastFM		Wikipedia	Reddit	MOOC
Metrics	**AP↑**	**AUC↑**	**AP↑**	**AUC↑**	**AP↑**	**AUC↑**	**AP↑**	**AUC↑**	**AP↑**	**AUC↑**	**AP↑**	**AUC↑**	**AP↑**	**AUC↑**	**AP↑**	**AUC↑**	**AUC↑**	**AUC↑**	**AUC↑**
JODIE	44.90	40.01	52.48	49.38	50.19	50.06	55.14	57.91	48.87	44.86	51.95	48.84	51.89	50.51	40.81	35.76	66.17	55.17	51.91
TGN	54.94	52.40	58.64	56.81	52.82	54.33	55.38	55.32	54.10	51.88	55.70	52.89	52.73	54.06	57.87	56.92	36.11	54.22	50.24
TGAT	46.99	43.72	66.71	65.43	52.06	51.16	51.05	50.25	47.74	44.11	62.90	62.75	51.88	51.07	50.91	49.97	54.03	56.59	45.53
GraphMixer	52.29	49.26	80.62	79.01	58.82	57.45	45.25	42.26	51.47	48.35	76.92	75.39	57.94	56.75	46.60	44.43	45.57	54.32	50.91
TCL	81.00	78.03	54.24	52.03	70.10	71.35	50.84	50.23	80.73	78.07	54.31	52.38	69.40	71.00	51.10	50.46	69.72	55.97	54.25
DyGFormer	80.60	75.26	84.55	84.66	72.93	69.60	85.33	80.55	81.38	75.88	85.27	84.48	71.99	68.24	83.53	78.66	61.68	57.75	55.70
TIGPrompt	52.97	51.46	56.19	52.99	52.15	52.24	53.37	53.51	52.64	51.22	54.58	51.35	51.92	51.86	56.04	55.48	46.49	55.83	56.16
DyGPrompt	55.22	54.51	54.63	53.40	51.26	53.68	54.97	54.31	53.71	52.27	53.43	52.47	53.31	56.15	52.69	50.51	59.24	57.36	48.27
TIGPrompt(D)	70.78	65.22	79.70	79.94	69.69	69.15	85.57	80.70	71.31	65.72	80.71	80.13	68.78	63.84	83.93	78.95	57.73	57.62	56.23
DyGPrompt(D)	76.07	72.80	85.47	83.08	52.33	51.53	86.19	81.38	75.81	72.23	85.54	83.08	52.24	51.59	84.57	79.59	66.53	56.54	47.37
DDGPrompt	**88.35**	**85.04**	**89.82**	**89.60**	**75.13**	**72.64**	**86.52**	**82.17**	**89.33**	**86.04**	**90.76**	**90.12**	**74.23**	**71.55**	**84.88**	**80.38**	**74.64**	**58.13**	**58.62**

5.1.4 Evaluation Metrics. Following the previous work, we use Average Precision (AP) and Area Under the Receiver Operating Characteristic Curve (AUC-ROC) as evaluation metrics for the dynamic link prediction. For the dynamic node classification, we only use AUC-ROC due to label imbalance.

5.1.5 Hyperparameter Settings and Implementation. We build and evaluate all baselines using the DyGLib [37] benchmark. In pretraining, we set all baseline epochs to 100, the learning rate to 0.0001, do not deploy early stopping, and save the best model. The temperature parameter τ of contrastive learning loss in pretraining is set to 0.2. For all downstream tasks, we load and freeze the pre-trained models and use its baseline default parameters in DyGLib (usually optimal) with 20 early stopping to tune the task classifier. We set the learning rate for downstream tasks to 0.0001 for dynamic link prediction and 0.001 for dynamic node classification. We run 5 times with different seeds and calculate the average of all results as the final performance of the model. All experiments were conducted on a Linux server with a single GPU (GeForce RTX 3090).

5.2 Analysis of Main Experiments Performance

We evaluate all methods in four datasets using dynamic link prediction, and in the dynamic node classification task on Wikipedia, Reddit, and MOOC. In addition, for fairness we implement TIGPrompt and DyGPrompt on TCL and DyGFormer respectively, and compare their performance with our DDGprompt. We use the first letter to represent the selected backbone model for prompt. For example, TIGPrompt (D) represents TIGPrompt based on the DyGFormer. We use DyGFormer as the default backbone of our method. The experimental results are shown in Table 2.

We bold the best result and underline the second-best result on each dataset. From the results, we have the following observations.

(1) DDGPrompt achieves the best performance in most datasets for different downstream tasks. Specifically, on dynamic link prediction, DDGPrompt outperforms all baselines in the Wikipedia, Reddit, and LastFM datasets. For the Wikipedia, DDGPrompt improves the backbone by 11.37% and 13.29% on AP and AUC, respectively. On Reddit and MOOC, DDGPrompt still improves the selected backbone. The limited improvement on Reddit is attributed to the overall performance bottleneck. In the dynamic node classification task, DDGPrompt also significantly improves the performance of the selected backbone. The above results prove that DDGPrompt can improve the performance of pre-trained models on different downstream tasks, demonstrating the effectiveness of our proposed method.

(2) DDGprompt outperforms TIGPrompt and DyGPrompt when using the same backbones. We take the AP evaluation indicator as an example. When the backbone is DyGFormer, our DDGPrompt outperforms TIGPrompt and DyGPrompt by 18.37% and 8.23% in link prediction for LastFM. For the node classification, DDGPrompt outperforms TIGPrompt and DyGPrompt by 8.36%/6.48%/2.4% and 6.51%/2.95%/14.48% respectively. The above observations demonstrate that our proposed DDGprompt outperforms existing simple dynamic graph prompting methods [2, 39] by more comprehensively tuning the temporal embeddings of nodes according to downstream tasks.

(3) In addition, different backbones also has a great impact on the final performance. For the dynamic link prediction , we notice that when DDGPrompt selects the best backbone DyGFormer, although it has a certain improvement on MOOC, the final result is inferior to the result when TCL is the backbone. This may be because the performance of the DyGFormer is seriously weaker than TCL in the current experimental setting.

5.3 Analysis of Model Performance on Sparse Interaction Data

In this section, we focus on scenarios with limited interaction data. By filtering and analyzing datasets with sparse interaction nodes, we aim to demonstrate how DDGPrompt can effectively adapt and perform well under the more stringent condition. Specifically, we filter all nodes that have fewer than a specified interaction threshold, retaining only the corresponding interactions for these nodes from the previous datasets, thereby limiting both the sample number and the interaction frequency. For LastFM, we set the interaction threshold to 1000, while for the other three datasets, the threshold is set to 100. This results in the labeled data for each dataset being at most half of the 70-shot scenario in Section 5.2. We show the performance of each baseline under sparse interaction data in Table 3.

We can observe that under this setting, DDGPrompt achieves the best performance on all datasets, highlighting the robustness of our method. It can maintain high performance even when only a small number of interactions are available. This analysis not only emphasizes the advantages of our model but also significantly contributes to enhancing its applicability in real-world scenarios involving cold starts.

5.4 Analysis of Model Components

In this section, we perform ablation experiments on DDGPrompt to verify the effectiveness of each prompt component. We explore three variants based on DDGPrompt and compare them with DDGPrompt. Fig. 3 shows the corresponding results of these variants on the dynamic link prediction task in four datasets. The variable **w/o t.b.** refers to DDGPrompt with the temporal bias prompt removed. Variant **w/o e.w.** and variant **w/o f.m.** represent without edge weight prompt and feature mask prompt, respectively.

We can see that all three proposed prompts are advantageous for dynamic link prediction, and they demonstrate a similar pattern across different datasets. For most datasets, the model performance degrades the most when the temporal bias prompt is absent. Additionally, the effects of the prompts vary across different datasets. In particular, for the Wikipedia and LastFM datasets, each prompt contributes significantly to overall performance improvement. Overall, these observations suggest that the components of our proposed DDGPrompt can adjust node temporal embeddings from different perspectives to better fit specific downstream tasks.

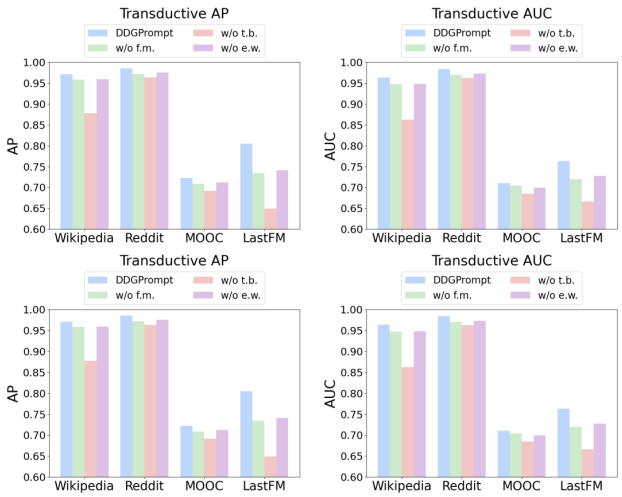

Figure 3: Ablation Studies of prompt components

5.5 Analysis of Model Hyperparameters

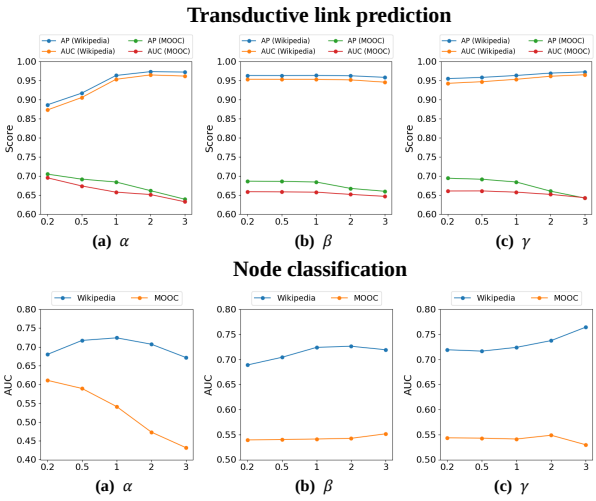

Figure 4: Performance under different hyperparameters on transductive link prediction

Then, we analyze the hyperparameter of DDGPrompt. Specifically, we evaluate the weights of each prompt when added to the original node expression feature matrix under the transductive link prediction, i.e. α, β, γ in Eq. 13. We tested the results of the three hyperparameter values on Wikipedia and MOOC, reflecting the impact of different prompts on the backbone across various datasets. The results for different downstream tasks are presented in Fig. 4.

We can observe that in the transductive link prediction, as we increase the weight α of the temporal bias prompt, the performance of DDGPrompt continuously improves in the Wikipedia dataset while declining in the MOOC. This indicates a larger time gap in the Wikipedia dataset, while the opposite holds for the MOOC. The weight γ of the feature mask prompt shows a similar trend.

5.6 Analysis of Model Scalability

In this section, we analyze the scalability and effectiveness of our proposed DDGPrompt on different backbones. We can notice the performance and improvement of DDGPrompt on different state-of-the-art backbone networks in Table 2. In the previous section, we can notice that although existing prompt works improve the performance of some tasks, their performance is often disappointing when faced with different datasets. In contrast, we observe in Table 2 that although different backbone networks have an impact on the final results, our DDGPrompt improves the backbone that do not use prompt. In particular, for the current best backbone DyGFormer, our DDGPrompt has the greatest improvement. This proves the scalability and effectiveness of our proposed DDGPrompt on different backbones.

5.7 Analysis of Node Feature Distribution

To better demonstrate the effectiveness of our method, we visualize the node feature distribution with and without DDGPrompt on DyGFormer. Specifically, we randomly sample 200 node pairs from the test sets of Wikipedia and LastFM, and perform dimensionality reduction for visualization. Fig. 5 presents the link prediction results, where each point represents a node involved in either a positive or negative sample pair. We expect the node representations of positive pairs to be close to each other, while those of negative pairs should be far apart in dynamic link prediction task. As shown in Fig. 5, after applying DDGPrompt, positive and negative pairs become more clearly separable in the embedding space, and the nodes within positive pairs are more tightly clustered. This indicates that the model, guided by DDGPrompt, is better at distinguishing whether two entities are likely to interact, demonstrating its ability to enhance semantic discrimination in downstream tasks.

5.8 Analysis of Time and Memory Consumption

In addition, we test the training overhead of existing dynamic graph prompt methods, including time and memory consumption in the main experimental setting. We test on Wikipedia and the large dataset LastFM and calculate the average results of five experiments. Table 4 and Table 5 show the experimental data of dynamic link prediction and dynamic node classification tasks respectively.

Table 4: Dynamic graph prompt methods training overhead for dynamic link prediction

	Time(s)		**Memory**(MB)	
	Wikipedia	LastFM	Wikipedia	LastFM
TIGPrompt	98	954	1010	2582
DyGPrompt	126	1096	946	2542
DDGPrompt	102	1098	894	2480

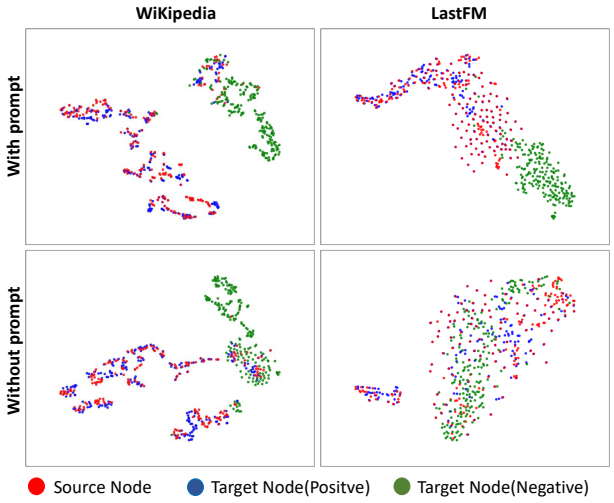

Figure 5: Visualization of node feature distributions with and without DDGPrompt on dynamic link prediction (DyGFormer as the backbone)

Table 5: Dynamic graph prompt methods training overhead for dynamic node classification

	Time(s)	**Memory**(MB)
	Wikipedia	Wikipedia
TIGPrompt	24	992
DyGPrompt	28	932
DDGPrompt	18	888

We can observe that the training costs of the three dynamic image prompting methods are close to each other. Although our proposed DDGPrompt involves a weighted combination of multiple prompt matrices, it still has a small training time overhead because the time complexity is still linear. In terms of memory overhead, TIGPrompt has the largest memory consumption due to its inclusion of transformers, followed by DyGPrompt with a dual MLP network. DDGPrompt has the smallest memory consumption due to the least trainable parameters.

6 Conclusion

In this paper, we propose DDGPrompt, a novel data-centric prompting framework for dynamic graphs. It bridges the gap between pre-trained models and downstream tasks by adaptively refining node embeddings based on interaction patterns through three prompts. Extensive experiments on four dynamic graph datasets under few-shot settings show that DDGPrompt outperforms both traditional baselines and existing prompt-based methods.

Acknowledgments

This work was supported by the National Key Research and Development Program of China (No.2023YFC3303800)

References

[1] Claudio DT Barros, Matheus RF Mendonça, Alex B Vieira, and Artur Ziviani. 2021. A survey on embedding dynamic graphs. *ACM Computing Surveys (CSUR)* 55, 1 (2021), 1–37.

[2] Xi Chen, Siwei Zhang, Yun Xiong, Xixi Wu, Jiawei Zhang, Xiangguo Sun, Yao Zhang, Yinglong Zhao, and Yulin Kang. 2024. Prompt learning on temporal interaction graphs. *arXiv preprint arXiv:2402.06326* (2024).

[3] Weilin Cong, Si Zhang, Jian Kang, Baichuan Yuan, Hao Wu, Xin Zhou, Hanghang Tong, and Mehrdad Mahdavi. 2023. Do we really need complicated model architectures for temporal networks? *arXiv preprint arXiv:2302.11636* (2023).

[4] Taoran Fang, Yunchao Zhang, Yang Yang, Chunping Wang, and Lei Chen. 2024. Universal prompt tuning for graph neural networks. *Advances in Neural Information Processing Systems* 36 (2024).

[5] Yuxin Guo, Cheng Yang, Yuluo Chen, Jixi Liu, Chuan Shi, and Junping Du. 2023. A Data-centric Framework to Endow Graph Neural Networks with Out-Of-Distribution Detection Ability. In *Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*. 638–648.

[6] Yufei He and Bryan Hooi. 2024. UniGraph: Learning a Cross-Domain Graph Foundation Model From Natural Language. *arXiv preprint arXiv:2402.13630* (2024).

[7] Petter Holme and Jari Saramäki. 2012. Temporal networks. *Physics reports* 519, 3 (2012), 97–125.

[8] Weihua Hu, Bowen Liu, Joseph Gomes, Marinka Zitnik, Percy Liang, Vijay Pande, and Jure Leskovec. 2019. Strategies for pre-training graph neural networks. *arXiv preprint arXiv:1905.12265* (2019).

[9] Ziniu Hu, Yuxiao Dong, Kuansan Wang, Kai-Wei Chang, and Yizhou Sun. 2020. Gpt-gnn: Generative pre-training of graph neural networks. In *Proceedings of the 26th ACM SIGKDD international conference on knowledge discovery & data mining*. 1857–1867.

[10] Xunqiang Jiang, Tianrui Jia, Yuan Fang, Chuan Shi, Zhe Lin, and Hui Wang. 2021. Pre-training on large-scale heterogeneous graph. In *Proceedings of the 27th ACM SIGKDD conference on knowledge discovery & data mining*. 756–766.

[11] Seyed Mehran Kazemi, Rishab Goel, Kshitij Jain, Ivan Kobyzev, Akshay Sethi, Peter Forsyth, and Pascal Poupart. 2019. Relational representation learning for dynamic (knowledge) graphs: A survey. *arXiv preprint arXiv:1905.11485* 12 (2019).

[12] Seyed Mehran Kazemi, Rishab Goel, Kshitij Jain, Ivan Kobyzev, Akshay Sethi, Peter Forsyth, and Pascal Poupart. 2020. Representation learning for dynamic graphs: A survey. *Journal of Machine Learning Research* 21, 70 (2020), 1–73.

[13] Elena Kochkina, Maria Liakata, and Arkaitz Zubiaga. 2018. All-in-one: Multi-task learning for rumour verification. *arXiv preprint arXiv:1806.03713* (2018).

[14] Srijan Kumar, Xikun Zhang, and Jure Leskovec. 2019. Predicting dynamic embedding trajectory in temporal interaction networks. In *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining*. 1269–1278.

[15] Hoyeop Lee, Jinbae Im, Seongwon Jang, Hyunsouk Cho, and Sehee Chung. 2019. Melu: Meta-learned user preference estimator for cold-start recommendation. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 1073–1082.

[16] Hao Liu, Jiarui Feng, Lecheng Kong, Ningyue Liang, Dacheng Tao, Yixin Chen, and Muhan Zhang. 2023. One for all: Towards training one graph model for all classification tasks. *arXiv preprint arXiv:2310.00149* (2023).

[17] Jiawei Liu, Cheng Yang, Zhiyuan Lu, Junze Chen, Yibo Li, Mengmei Zhang, Ting Bai, Yuan Fang, Lichao Sun, Philip S Yu, et al. 2023. Towards graph foundation models: A survey and beyond. *arXiv preprint arXiv:2310.11829* (2023).

[18] Zemin Liu, Xingtong Yu, Yuan Fang, and Xinming Zhang. 2023. Graphprompt: Unifying pre-training and downstream tasks for graph neural networks. In *Proceedings of the ACM Web Conference 2023*. 417–428.

[19] Yuanfu Lu, Xunqiang Jiang, Yuan Fang, and Chuan Shi. 2021. Learning to pre-train graph neural networks. In *Proceedings of the AAAI conference on artificial intelligence*, Vol. 35. 4276–4284.

[20] Shuteng Niu, Yongxin Liu, Jian Wang, and Houbing Song. 2020. A decade survey of transfer learning (2010–2020). *IEEE Transactions on Artificial Intelligence* 1, 2 (2020), 151–166.

[21] Aaron van den Oord, Yazhe Li, and Oriol Vinyals. 2018. Representation learning with contrastive predictive coding. *arXiv preprint arXiv:1807.03748* (2018).

[22] Feiyang Pan, Shuokai Li, Xiang Ao, Pingzhong Tang, and Qing He. 2019. Warm up cold-start advertisements: Improving ctr predictions via learning to learn id embeddings. In *Proceedings of the 42nd International ACM SIGIR Conference on Research and Development in Information Retrieval*. 695–704.

[23] Jiezhong Qiu, Qibin Chen, Yuxiao Dong, Jing Zhang, Hongxia Yang, Ming Ding, Kuansan Wang, and Jie Tang. 2020. Gcc: Graph contrastive coding for graph neural network pre-training. In *Proceedings of the 26th ACM SIGKDD international conference on knowledge discovery & data mining*. 1150–1160.

[24] Liang Qu, Ningzhi Tang, Ruiqi Zheng, Quoc Viet Hung Nguyen, Zi Huang, Yuhui Shi, and Hongzhi Yin. 2023. Semi-decentralized federated ego graph learning for recommendation. In *Proceedings of the ACM Web Conference 2023*. 339–348.

[25] Emanuele Rossi, Ben Chamberlain, Fabrizio Frasca, Davide Eynard, Federico Monti, and Michael Bronstein. 2020. Temporal graph networks for deep learning on dynamic graphs. *arXiv preprint arXiv:2006.10637* (2020).

[26] Joakim Skarding, Bogdan Gabrys, and Katarzyna Musial. 2021. Foundations and modeling of dynamic networks using dynamic graph neural networks: A survey. *iEEE Access* 9 (2021), 79143–79168.

[27] Mingchen Sun, Kaixiong Zhou, Xin He, Ying Wang, and Xin Wang. 2022. Gppt: Graph pre-training and prompt tuning to generalize graph neural networks. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*. 1717–1727.

[28] Xiangguo Sun, Jiawen Zhang, Xixi Wu, Hong Cheng, Yun Xiong, and Jia Li. 2023. Graph prompt learning: A comprehensive survey and beyond. *arXiv preprint arXiv:2311.16534* (2023).

[29] Rakshit Trivedi, Mehrdad Farajtabar, Prasenjeet Biswal, and Hongyuan Zha. 2019. Dyrep: Learning representations over dynamic graphs. In *International conference on learning representations*.

[30] Lu Wang, Xiaofu Chang, Shuang Li, Yunfei Chu, Hui Li, Wei Zhang, Xiaofeng He, Le Song, Jingren Zhou, and Hongxia Yang. 2021. Tcl: Transformer-based dynamic graph modelling via contrastive learning. *arXiv preprint arXiv:2105.07944* (2021).

[31] Karl Weiss, Taghi M Khoshgoftaar, and DingDing Wang. 2016. A survey of transfer learning. *Journal of Big data* 3 (2016), 1–40.

[32] Lianghao Xia, Ben Kao, and Chao Huang. 2024. Opengraph: Towards open graph foundation models. *arXiv preprint arXiv:2403.01121* (2024).

[33] Da Xu, Chuanwei Ruan, Evren Korpeoglu, Sushant Kumar, and Kannan Achan. 2020. Inductive representation learning on temporal graphs. *arXiv preprint arXiv:2002.07962* (2020).

[34] Cheng Yang, Deyu Bo, Jixi Liu, Yufei Peng, Boyu Chen, Haoran Dai, Ao Sun, Yue Yu, Yixin Xiao, Qi Zhang, et al. 2023. Data-centric graph learning: A survey. *arXiv preprint arXiv:2310.04987* (2023).

[35] Cheng Yang, Chengdong Yang, Chuan Shi, Yawen Li, Zhiqiang Zhang, and Jun Zhou. 2024. Calibrating Graph Neural Networks from a Data-centric Perspective. In *Proceedings of the ACM on Web Conference 2024*. 745–755.

[36] Huaxiu Yao, Chuxu Zhang, Ying Wei, Meng Jiang, Suhang Wang, Junzhou Huang, Nitesh Chawla, and Zhenhui Li. 2020. Graph few-shot learning via knowledge transfer. In *Proceedings of the AAAI conference on artificial intelligence*, Vol. 34. 6656–6663.

[37] Le Yu, Leilei Sun, Bowen Du, and Weifeng Lv. 2023. Towards better dynamic graph learning: New architecture and unified library. *Advances in Neural Information Processing Systems* 36 (2023), 67686–67700.

[38] Xingtong Yu, Yuan Fang, Zemin Liu, and Xinming Zhang. 2024. Hgprompt: Bridging homogeneous and heterogeneous graphs for few-shot prompt learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 38. 16578–16586.

[39] Xingtong Yu, Zhenghao Liu, Yuan Fang, and Xinming Zhang. 2024. DyG-Prompt: Learning Feature and Time Prompts on Dynamic Graphs. *arXiv preprint arXiv:2405.13937* (2024).

[40] Xingtong Yu, Chang Zhou, Yuan Fang, and Xinming Zhang. 2024. MultiGPrompt for multi-task pre-training and prompting on graphs. In *Proceedings of the ACM on Web Conference 2024*. 515–526.

[41] Fan Zhou, Chengtai Cao, Kunpeng Zhang, Goce Trajcevski, Ting Zhong, and Ji Geng. 2019. Meta-gnn: On few-shot node classification in graph meta-learning. In *Proceedings of the 28th ACM International Conference on Information and Knowledge Management*. 2357–2360.

[42] Zhilun Zhou, Yu Liu, Jingtao Ding, Depeng Jin, and Yong Li. 2023. Hierarchical knowledge graph learning enabled socioeconomic indicator prediction in location-based social network. In *Proceedings of the ACM Web Conference 2023*. 122–132.

[43] Fuzhen Zhuang, Zhiyuan Qi, Keyu Duan, Dongbo Xi, Yongchun Zhu, Hengshu Zhu, Hui Xiong, and Qing He. 2020. A comprehensive survey on transfer learning. *Proc. IEEE* 109, 1 (2020), 43–76.