6-2023

Generalizing graph neural networks across graphs, time, and tasks

Zhihao WEN
Singapore Management University, zhwen.2019@phdcs.smu.edu.sg

Generalizing Graph Neural Networks across Graphs, Time, and Tasks

Zhihao Wen

School of Computing and Information Systems,
Singapore Management University
80 Stamford Rd, Singapore, 178902

Dissertation Committee:

FANG Yuan (Supervisor/Chair)
Assistant Professor
Singapore Management University

JIANG Jing
Professor
Singapore Management University

Hady Wirawan LAUW
Associate Professor
Singapore Management University

Bingsheng He
Professor
National University of Singapore

Submitted to School of Computing and Information Systems
in partial fulfillment of the requirements for the Degree of
Doctor of Philosophy in Computer Science

Abstract

Graph-structured data are ubiquitous across numerous real-world contexts, encompassing social networks, commercial graphs, bibliographic networks, and biological systems. Delving into the analysis of these graphs can yield significant understanding pertaining to their corresponding application fields.Graph representation learning offers a potent solution to graph analytics challenges by transforming a graph into a low-dimensional space while preserving its information to the greatest extent possible. This conversion into low-dimensional vectors enables the efficient computation of subsequent graph algorithms.

The majority of prior research has concentrated on deriving node representations from a single, static graph. However, numerous real-world situations demand rapid generation of representations for previously unencountered nodes, novel edges, or entirely new graphs. This inductive capability is vital for high-performance machine learning systems that operate on ever-changing graphs and consistently encounter unfamiliar nodes.

The inductive graph representation presents considerable difficulty when compared to the transductive setting, as it necessitates the alignment of new subgraphs containing previously unseen nodes with an already trained neural network. We further investigate inductive graph representation learning through three distinct angles: (1) Generalizing Graph Neural Networks (GNNs) across graphs, addressing semi-supervised node classification across multiple graphs; (2) Generalizing GNNs across time, focusing on temporal link prediction; and (3) Generalizing GNNs across tasks, tackling various low-resource text classification tasks.

Contents

Acknowledgements

As I approach the culmination of my doctoral journey, I would like to take this opportunity to express my deepest gratitude to those who have accompanied and supported me throughout this challenging yet rewarding experience.

First and foremost, I am profoundly grateful to my doctoral advisor, Dr. Yuan Fang, for his unwavering support and mentorship. Dr. Fang's extensive knowledge, expertise, and dedication to research have been instrumental in shaping my academic and professional development. His guidance has not only expanded my understanding of the field but also instilled in me a passion for scientific inquiry. I am truly fortunate to have had the opportunity to work under his tutelage, and I look forward to carrying the lessons he has imparted to me throughout my future endeavors.

I would also like to extend my heartfelt appreciation to my committee members, Dr. Jing Jiang, Dr. Hady Wirawan Lauw, and Dr. Bingsheng He, for their invaluable insights, constructive feedback, and persistent encouragement. Their collective wisdom, expertise, and dedication to the pursuit of knowledge have been an inspiration and a driving force behind my accomplishments.

I am deeply grateful to my fellow lab members and junior colleagues, who have enriched my Ph.D. experience in countless ways. I would like to specifically acknowledge Dr. Zemin Liu, Yuanfu Lu, Wentao Zhang, Zhongzhou Liu, Xingtong Yu, Deyu Bo, Pengcheng Wei, Jian Yuan Bo, Dong Viet Hoang, Ran Liu, Nguyen Trung Kien, and Sethupathy Parameswaran, *etc.*, for their support, camaraderie, and shared passion for research. The friendships and professional relationships forged during my time in the lab have been an integral part of my academic journey and I am grateful to have had the opportunity to learn from and work alongside such talented and dedicated individuals.

I would also like to express my gratitude to staff members Boo Chui Ngoh and Caroline Tan, whose assistance and dedication have been invaluable in fostering a supportive and nurturing environment for my research.

My heartfelt appreciation goes to my parents for their unwavering support, encouragement, and understanding throughout this journey. Their belief in my abilities and their constant reassurance have been a source of strength and motivation, enabling me to persevere through the challenges and celebrate the accomplishments that have marked my doctoral experience.

Furthermore, I would like to thank my friends and my girlfriend for their continued support, encouragement, and shared moments of laughter and respite. Their companionship has provided a much-needed balance to the rigors of academic life and has made this journey all the more enjoyable.

In closing, I would like to reiterate my profound appreciation to Dr. Yuan Fang, my committee members, my fellow lab members, and the entire academic community for their unwavering support, guidance, and encouragement. Their collective wisdom, expertise, and dedication to the pursuit of knowledge have been an inspiration and a driving force behind my accomplishments, and I look forward to the opportunity to contribute to the continued advancement of our shared field of research.

Zhihao Wen

June 2023, Singapore

Chapter 1

Introduction

1.1 Motivation

Graph-structured data is prevalent in a multitude of real-world contexts, encompassing social networks, commercial graphs, citation networks, and biological systems. Analyzing these graphs can yield valuable insights into their corresponding application domains, with effective analytics bringing benefits to numerous applications, such as node classification, node recommendation, and link prediction. For instance, an analysis of a social network graph (e.g., Facebook, Twitter, or WeChat) can facilitate user classification, friend recommendations, and predictions of potential interactions between users.

Meanwhile, graph representation learning offers an effective solution to the graph analytics problem by learning low-dimensional embeddings of nodes in a graph, which can be used to perform downstream tasks such as node classification, link prediction, and graph clustering. Compared to manual feature engineering, which involves manually designing and selecting features for a machine learning model, graph representation learning has several advantages: (1) Scalability: Graph representation learning can handle large-scale graphs with millions

of nodes and edges, which would be challenging and time-consuming for manual feature engineering; (2) Flexibility: Graph representation learning can capture complex relationships between nodes in a graph, including both global and local structural patterns, which may be difficult or impossible to capture with manual feature engineering; (3) Generality: Graph representation learning can be applied to a wide range of graph-based tasks, including social network analysis, recommendation systems, and bioinformatics, among others. In contrast, the manual feature engineering may be specific to a particular task or domain. (4) Transferability: Graph representation learning can learn embeddings that can be easily transferred to new graphs or tasks, without the need for re-engineering features. This can save significant time and effort in developing machine learning models. Overall, graph representation learning offers a powerful and flexible approach to graph-based machine learning tasks, with the potential to significantly outperform manual feature engineering in terms of accuracy and efficiency.

However, the majority of prior research has concentrated on deriving node representations from a single, static graph. In contrast, many real-world situations demand rapid generation of representations for previously unencountered nodes, novel edges, or entirely new graphs, *i.e.*, inductive capability. This inductive capability is vital for high-performance machine-learning systems that operate on ever-changing graphs and consistently encounter unfamiliar nodes (e.g., Wikipedia posts, Amazon users, and items). Moreover, an inductive approach can facilitate generalization across graphs sharing similar feature forms, enabling the training of a neural network on protein-protein interaction graphs derived from a model organism and the subsequent generation of node representations for graphs from new organisms using the trained model. The inductive graph representation problem presents considerable difficulty compared to the transductive setting, as it necessitates aligning new subgraphs containing previously unseen nodes with an already trained neural network.

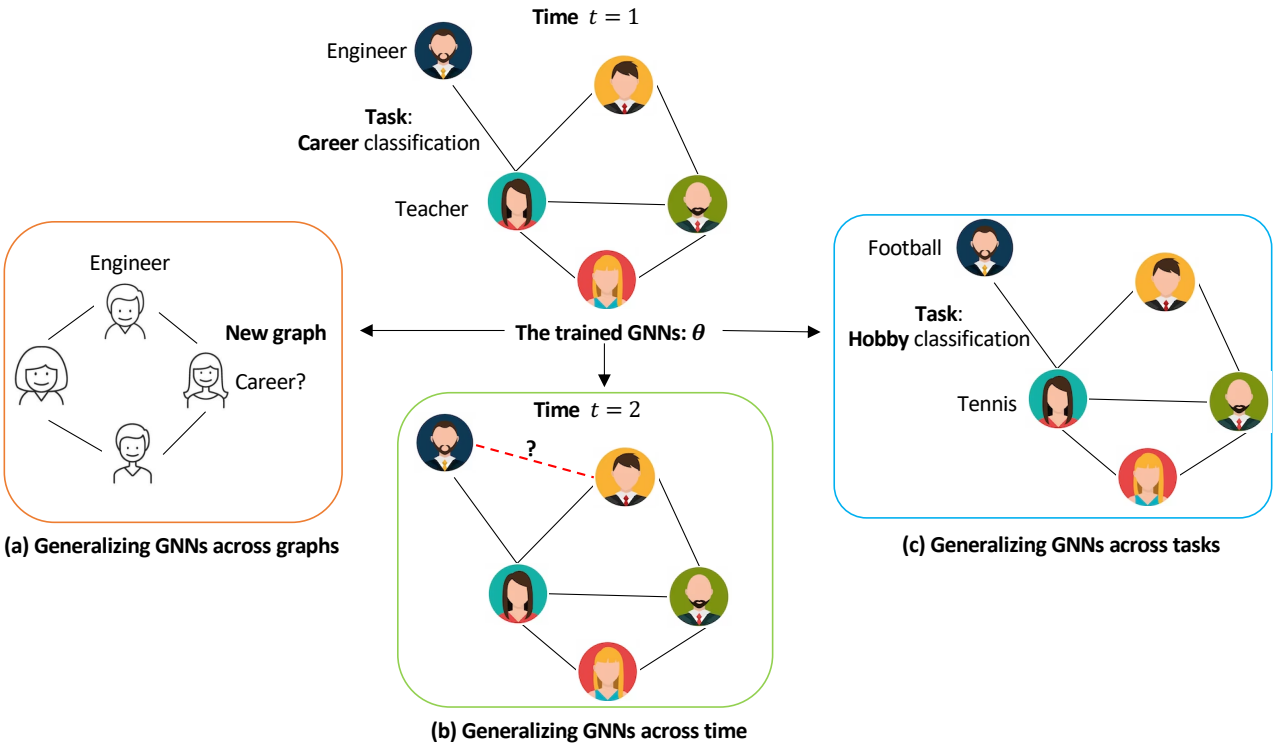


Figure 1.1: Generalizing GNNs across graphs, time, and tasks.

Within the multitude of graph analysis methodologies, Graph Neural Networks (GNNs) are presently the most popular algorithm. This is due to their impressive performance across various tasks and their notable generalization capabilities. GNNs are parameterized by a weight matrix in each layer to map the messages from the neighboring nodes, instead of directly learning the node embedding vectors. In particular, the weight matrices give rise to the inherent inductive power of GNNs, which can be applied to similarly map and aggregate messages in new graphs given the same feature space.

In this dissertation, building upon the existing literature on inductive Graph Neural Networks (GNNs) with robust generalizability, we put forth three perspectives for enhancing the learning of inductive GNNs, as shown in Figure 1.1: (1) Generalizing Graph Neural Networks (GNNs) across graphs, addressing semi-supervised node classification across multiple graphs; (2) Generalizing GNNs across time, focusing on temporal link prediction; and (3) Generalizing GNNs across tasks, tackling various node classification tasks.

Generalizing GNNs across graphs. A key limitation of many GNNs is that they are not easily generalized across different graphs. This means that a model trained on one graph might perform poorly when applied to a different graph, even if the two graphs are structurally similar. This is a significant problem because in many real-world scenarios, we are interested in applying models to new, unseen graphs.

Generalizing GNNs across graphs can address this issue, allowing for models that are more robust and flexible. This has numerous practical applications:

- Drug Discovery: In the field of bioinformatics, molecules can be represented as graphs, where each node represents an atom and each edge represents a bond. A generalized GNN could be trained on a dataset of known molecules and their properties, and then used to predict the properties of unknown molecules, potentially identifying new drug candidates;
- Social Network Analysis: Social networks can also be represented as graphs, where nodes represent individuals and edges represent relationships. A generalized GNN could be used to analyze different social networks, predicting things like the spread of information or the likelihood of new connections forming;
- Transportation and Logistics: In transportation and logistics, networks of roads or shipping routes can be represented as graphs. A generalized GNN could be used to optimize routes and schedules across different networks, improving efficiency and reducing costs;
- Cybersecurity: Networks of computers or other digital assets can be represented as graphs, where nodes represent individual devices and edges represent connections between them. A generalized GNN could be used to detect patterns indicative of cyber attacks or other security issues across different networks;

- Recommendation Systems: In many recommendation systems, items and users can be represented as a graph, where nodes represent users or items, and edges represent user-item interactions. A generalized GNN can be trained on such graphs and then used to provide recommendations in different domains or for different user groups.

In all of these examples, the ability to generalize across graphs allows for models that are more adaptable and capable of handling new, unseen data. This is crucial for creating systems that can handle the complexity and variability of real-world data. Thus, it motivates us to explore a novel framework that can capture the inter-graph difference and customize a trained inductive model to each graph.

Generalizing GNNs across time. Temporal or dynamic graph data is common in many real-world scenarios. Nodes may appear and disappear, edges may be formed and broken, and node or edge attributes may evolve. For such dynamic graphs, it's important to have models that can not only capture the graph structure but also how it changes over time. Generalizing Graph Neural Networks (GNNs) across time, is often referred to as dynamic or temporal GNNs.

Here are some practical examples that highlight the significance of generalizing GNNs across time:

- Social Network Analysis: Social networks are inherently dynamic: new connections are made, old connections are lost, and the strength of connections can change over time. A temporal GNN could track these changes and use them to predict future network states, such as the formation of new connections or the spread of information or trends;

- Financial Fraud Detection: In financial networks, transactions between entities form a dynamic graph. By generalizing GNNs across time, suspicious patterns can be detected not only based on the current transaction graph

but also based on how the graph has evolved over time, which could be crucial for identifying complex fraud schemes;

- Traffic Prediction: In transportation networks, the flow and congestion levels can be different at different times. A temporal GNN can learn these patterns over time and make accurate predictions about future traffic conditions, helping in effective route planning and congestion management;

- Epidemiology: The spread of diseases can be modeled as a dynamic graph, where nodes represent individuals or regions, and edges represent the potential transmission of the disease. Temporal GNNs could be used to model and predict the spread of diseases, informing public health interventions;

- Dynamic Recommendation Systems: User-item interaction data in recommendation systems is inherently temporal. Users' preferences change over time, and items may become more or less popular. Temporal GNNs can capture these dynamics to provide more accurate and timely recommendations;

- Cybersecurity: In computer networks, the patterns of normal and malicious activities change over time. Temporal GNNs can be used to detect anomalous behaviors based on the evolution of the network structure and activities, potentially identifying cyber threats.

In all these examples, the ability to model and understand changes over time can lead to more accurate and insightful predictions, which motivates us to further explore the temporal GNNs.

Generalizing GNNs across tasks. Traditional GNNs are typically trained for a specific node classification task and might not generalize well to other tasks. This is particularly challenging when we have multiple related tasks, and we wish to leverage the shared information between them.

Generalizing GNNs across node classification tasks could potentially im-

prove model performance, transfer learning capabilities, and computational efficiency. Here are a few practical examples to illustrate its significance:

- Social Network Analysis: Suppose we have tasks like predicting a person's occupation, their hobbies, their likelihood of purchasing a product, or their political affiliations. A generalized GNN can leverage commonalities between these tasks to improve prediction accuracy.
- Transportation Networks: We might want to predict various properties, like traffic congestion levels, accident risk, or maintenance needs. A GNN that can generalize across these tasks can make more accurate predictions by learning shared features.
- Citation Networks: We might be interested in multiple node classification tasks like predicting the paper's field of study, its impact factor, or the conference it might get accepted to. A generalized GNN could use the common information across these tasks to enhance its performance.

In all these examples, a GNN that can generalize across node classification tasks can improve the efficiency and effectiveness of the learning process. Therefore, it motivates us to explore a kind of GNN that can identify and leverage shared patterns, making better predictions by exploiting the common structure of the tasks.

The integration of three techniques. Earlier, we separately discussed three techniques for generalizing Graph Neural Networks (GNNs).

Beyond this, it is worth noting that these techniques can be effectively combined, leading to the development of a more robust model capable of tackling more complex real-world problems:

- Social Media Analytics: Social media platforms can also greatly benefit from

this generalized approach. Different tasks could include predicting user behaviors (such as likelihood to click on an ad, share a post, or purchase a product), identifying fake accounts, or spotting harmful content. These tasks could be performed on different social networks (Twitter, Facebook, etc.) and at different times, allowing for a comprehensive analysis of the social media landscape;

- Transportation Networks: Transportation networks can benefit significantly from such a generalized model. For instance, traffic congestion levels, accident risks, and maintenance needs are different tasks that can be predicted using the same transportation graph but at different times. The model can be trained on data from one city (one graph) and then applied to other cities (other graphs), allowing for efficient transfer of learned knowledge;

- Healthcare: In healthcare, patient similarity networks could be used to predict various outcomes like disease progression, medication response, and readmission risk. These tasks could be performed across different hospitals (graphs), across different times (as the patient's health status evolves), and across different tasks (disease prediction, medication response, etc.);

- Cybersecurity: In computer networks, tasks could include detecting various types of cyber threats, predicting system failures, and identifying potential network bottlenecks. These tasks could be performed across different networks (graphs), across different times (as network traffic evolves), and across different tasks (threat detection, system failure prediction, etc.).

In all these cases, a GNN that can generalize across graphs, time, and tasks would be able to handle the complexity and dynamism of real-world data, making more accurate predictions and providing more insightful analysis.

1.2 Contributions

The contributions of this dissertation are as follows:

- **Meta-Inductive Node Classification across Graphs.** We study the problem of inductive node classification across graphs. Unlike existing one-model-fits-all approaches, we propose a novel meta-inductive framework called MI-GNN to customize the inductive model to each graph under a meta-learning paradigm. That is, MI-GNN does not directly learn an inductive model; it learns the general knowledge of how to train a model for semi-supervised node classification on new graphs. To cope with the differences between graphs, MI-GNN employs a dual adaptation mechanism at both the graph and the task levels. More specifically, we learn a graph prior to adapting for the graph-level differences, and a task prior to adapting for the task-level differences conditioned on a graph. Extensive experiments on five real-world graph collections demonstrate the effectiveness of our proposed model.

- **TempoRal Event and Node Dynamics for Graph Representation Learning.** We study the problem of temporal link prediction and propose TREND, a novel framework for temporal graph representation learning, driven by TempoRal Event and Node Dynamics and built upon a Hawkes process-based graph neural network (GNN). TREND presents a few major advantages: (1) it is inductive due to its GNN architecture; (2) it captures the exciting effects between events by the adoption of the Hawkes process; (3) as our main novelty, it captures the individual and collective characteristics of events by integrating both event and node dynamics, driving a more precise modeling of the temporal process. Extensive experiments on four real-world datasets demonstrate the effectiveness of our proposed model.

- **Augmenting Low-Resource Node Classification with Graph-Grounded Pre-training and Prompting.** We propose a novel model called Graph-

Grounded Pre-training and Prompting (G2P2) to address low-resource text classification in a two-pronged approach. During pre-training, we propose three graph interaction-based contrastive strategies to jointly pre-train a graph-text model; during downstream classification, we explore prompting for the jointly pre-trained model to achieve low-resource classification. Extensive experiments on four real-world datasets demonstrate the strength of G2P2 in various zero- and few-shot low-resource text classification tasks.

1.3 Dissertation Structure

The remaining part of this dissertation is as follows. We first review related works in Chapter 2. Chapter 3 presents our work on generalizing GNNs across new tasks. Chapter 4 describes our work on generalizing GNNs across new time. Chapter 5 presents our work on generalizing GNNs to new tasks of low-resource text classification. Finally, we summarize our works and describe future research directions (Chapter 6).

Chapter 2

Related Work

In this chapter, we first review graph neural networks (GNNs) in general. Then we emphasize related work about generalization approaches.

2.1 Graph neural networks

Graph Neural Networks (GNNs) [135] have emerged as a powerful class of deep learning models for handling graph-structured data, addressing the limitations of traditional neural networks in processing irregular data structures. GNNs leverage the inherent relational information present in graphs to learn node and graph representations, making them well-suited for a wide range of applications across diverse domains. By capturing both local and global patterns within the graph, GNNs can effectively tackle tasks such as node classification [46], link prediction [45], graph classification [140], and graph generation [122].

The foundation of GNNs lies in the message-passing mechanism, wherein nodes exchange and aggregate information from their local neighborhood to iteratively refine their representations [130]. The growing interest in GNNs has led to the development of various architectures, such as Graph Convolutional Net-

work (GCN) [46], GraphSAGE [31], Graph Attention Network (GAT) [116], and Graph Isomorphism Network (GIN) [140]. These architectures explore diverse approaches to message-passing and representation learning, further expanding the applicability and impact of GNNs in academia and industry alike.

The versatility of GNNs has led to numerous practical applications, including:

- Social network analysis: GNNs are used to study user behavior [19], community detection [7], and information diffusion [10] in social networks;
- Recommender Systems: GNNs can model the complex relationships between users and items to provide personalized recommendations [126, 123].
- Biological network analysis: GNNs are employed for protein function prediction [25], drug-target interaction prediction [101], and analysis of gene regulatory networks [124].
- Computer vision: GNNs help in scene understanding [141], object detection [76], and point cloud semantic segmentation [54] by modeling the relationships between objects in an image.
- Traffic prediction: GNNs can model the spatial-temporal dependencies in traffic networks to forecast traffic conditions [13].
- Natural language processing: GNNs have also been leveraged to improve text-based tasks through knowledge graphs [9] and heterogeneous graphs [59], or multi-modality learning [66].

2.2 Generalization approaches

Inductive graph representation learning

Inductive Graph Representation Learning [31] refers to the process of learning to generate graph embeddings that can generalize to unseen nodes or graphs. This contrasts with transductive learning, which only learns representations for nodes that are present during training.

Inductive learning is particularly useful in many real-world scenarios where we need to handle dynamic graphs that evolve over time or apply models to entirely new graphs. The learned model can generate embeddings for new nodes or graphs based on their attributes and structural information, which can then be used for various downstream tasks.

Here are a few examples of how inductive graph representation learning can influence real-life problems:

- Social Network Analysis: Social networks are continually evolving, with new users joining and forming new connections. An inductive graph learning model can generate embeddings for new users based on their initial connections and activities, which can be used to predict their future behavior or recommend friends or content [51];
- Recommendation Systems: In many recommendation systems, new users or items are continually appearing. An inductive model can generate embeddings for these new users or items, allowing the system to make recommendations even before they have substantial interaction history [146];
- Bioinformatics: In molecular biology, we often want to predict properties of new molecules, which can be represented as graphs. An inductive model can generate embeddings for these new molecules, enabling predictions about

their properties or behavior [31].

Recent inductive learning on graphs is mainly based on network embedding and GNNs. For the former, some extend classical embedding approaches (*e.g.*, skip-gram) to handle new nodes on dynamic graphs [17, 81, 160], by exploiting structural information from the graph such as co-occurrence between nodes; others employ graph auto-encoders [27, 26] for dynamic graphs, by mining and reconstructing the graph structures. In general, this category of approaches only handle new nodes on the same graph, lacking the ability to extend to entirely new graphs. In the latter category, most GNNs are inherently inductive, and can be applied to new graphs in the same feature space after training [31, 137]. More recently, a few pre-training approaches have also been devised for GNNs [118, 39, 70]. While they also learn some form of transferable knowledge on the training graphs, they are trained in a self-supervised manner, and the main objective is to learn universally good initializations for different downstream tasks. Thus, they address a problem setting that is different from that in Chapter 3.

Meta-learning

Meta-learning, often referred to as "learning to learn," is a subfield of machine learning where models are designed to learn quickly when presented with new tasks. The main idea is to train a model on a variety of learning tasks so that it can learn a new task from a small number of examples in the future.

Meta-learning is particularly useful in scenarios where data is scarce [120], or it is expensive to obtain large labeled datasets. It also has great potential in multi-task learning scenarios where a model needs to generalize across multiple tasks. Here are some examples of how meta-learning can influence real-life problems:

- Few-shot Image Classification: Meta learning algorithms enable rapid adaptation to new image classification tasks with a limited number of labeled examples, improving generalization capabilities [22];

- Personalized Recommendations: Meta-learning can be used to personalize recommendation systems [72] by treating the recommendation process for each user as a separate task. This allows the system to adapt quickly to each user's preferences.

- Medical Diagnosis: In medicine, certain diseases are rare, and therefore, there are few examples to learn from. A meta-learning model can be trained on a variety of disease diagnosis tasks, allowing it to diagnose rare diseases from a small number of examples [151].

Generally, some approaches resort to metric-based protonets [104] which aim to learn a metric space for the class prototypes, and others apply optimization-based techniques such as model-agnostic meta-learning (MAML) [22]. To further enhance task adaptation, the transferable knowledge can be tailored to different clusters of tasks, forming a hierarchical structure of adaptation [144]; feature-specific memories can also guide the adapted model with a further bias [15]; domain-knowledge graphs can also be leveraged to provide task-specific customization [108]. Another subtype of meta-learning called hypernetwork [29, 86] uses a secondary neural network to generate the weights for the target neural network, *i.e.*, it learns to generate different weights conditioned on different input, instead of freezing the weights for all input after training in traditional neural networks.

More recently, meta-learning has also been adopted on graphs for few-shot learning, such as Meta-GNN [155], GFL [145], GPN [14], RALE [68] and meta-tail2vec [69], which is distinct from inductive semi-supervised node classification as further elaborated in Section 3.3. Hypernetwork-based approaches have also emerged, such as LGNN [67] that adapts GNN weights to different local contexts,

and GNN-FiLM [5] that adapts to different relations in a relational graph.

Temporal graph representation learning

To address real-world scenarios in which graphs continuously evolve in time, there have been some efforts in temporal graph representation learning. Intuitively, a temporal graph can be modeled as a series of snapshots. The general idea is to learn node representations for each graph snapshot, and then capture both the graph structures in each snapshot and the sequential effect across the snapshots. The specific techniques vary in different works, such as matrix perturbation theory [55, 158], skip-grams [17] and triadic closure process [157]. To effectively capture the sequential effect, recurrent neural networks (RNNs) have been a popular tool [103, 26, 49, 30], which leverage the chronological sequence of representations across all snapshots. From a different perspective, instead of using RNNs to generate node representations, EvolveGCN [84] uses RNN to evolve GCN parameters. Besides, rather than directly learning the representation, DynGEM [27] incrementally builds the representations of a snapshot from those of the previous snapshot.

However, snapshots are approximations which discretize a continuously evolving graph, inevitably suffering from a fair degree of information loss in the temporal dynamics. To overcome this problem, another line of work aims to model the continuous process of temporal graph evolution, usually by treating each event (typically defined as the formation of a link that can occur continuously in time) as an individual training instance. Among them, some employ temporal random walks to capture the continuous-time network dynamics, including CTDNE [81] based on time-respect random walks, and CAW-N [127] based on causal anonymous walks. Apart from random walks, GNN-based models have also emerged to deal with continuous time, e.g., TGAT [137].

While these methods can deal with a continuously evolving graph, they fail

to explicitly model the exciting effects between sequential events holistically on the entire graph. In view of this, several network embedding methods [160, 71, 42] incorporate temporal point processes such as Hawkes process into their models, being capable of modeling the graph-wide formation process.

Moreover, DyREP [114] is an inductive GNN-based model that also exploits the temporal point processes.

Note that, among existing methods for temporal graph representation learning, those employing an embedding lookup for node representations are usually transductive [17, 157, 103, 81, 160, 71] and thus unable to directly make predictions on new nodes at a future time. In contrast, GNN-based methods [84, 137, 114] are naturally inductive, able to extend to new nodes in the same feature space. However, among them only DyREP [114] leverages temporal point processes, but it is designed to capture association and communication events, which differs from our problem setting to specifically deal with the link formation process. Furthermore, none of existing methods integrates both event- and node-dynamics to capture the individual and collective characteristics of events, respectively.

Pre-training and prompting

Pre-trained language models [32] have become the most popular backbone in NLP. While earlier PLMs such as GPT [90], BERT [43], XLNet [143] and RoBERTa [65] still have affordable model size, recent introductions such as T5 [91] and GPT-3 [6] produce massive models with billions of parameters. To avoid the high fine-tuning cost on these large models, prompting [62] starts to receive more attention in the community. A prompt is a special template to pad the task input, with a goal of extracting useful knowledge from PLMs to flexibly adapt to downstream tasks. Fueled by the success of GPT-3, numerous prompting methods including discrete natural language prompt [102, 98, 23] and continuous prompt [58, 52, 64, 88, 153]

have emerged. The strength of prompting has been validated in a wide range of NLP applications, including text classification [38, 79, 107, 150, 33], machine translation [110] and relation extraction [12, 94]. More recently, prompting has also been applied to GNNs for node classification [106].

Zero- or few-shot paradigms

Broadly speaking, our setting in Chapter 5 is also related to other learning paradigms. For example, in semi-supervised learning [80, 136, 11], each class may only have a few examples, but all classes must be seen in training and they cannot handle any novel class during testing. Meta-learning [22, 147, 34, 2, 3, 155, 125] is another popular paradigm that supports few-shot learning. However, large-scale labeled data are still required in a so-called "meta-training" phase, to support the few-shot learning of novel classes during "meta-testing". In contrast, we only need label-free data for pre-training, without requiring any meta-training phase that would consume large-scale labeled data. Separately, there also exists joint consideration of image and text data using a contrastive pre-training strategy for zero- or few-shot classification [89]. In our work, graph data are significantly different from images, which provide various types of interaction between texts. On graphs, zero-shot node classification has also been done [128]. It relies heavily on the availability of Wikipedia pages or other side information to generate class prototype embeddings. However, it is very labor-intensive to find and curate the right side information, especially when there are a large number of classes and/or novel classes emerge frequently.

2.3 Summary

For generalizing GNNs across graphs, the previous works resort to meta-learning, but they are designed for the few-shot node classification task. While appearing similar, it is distinct from our problem of inductive semi-supervised node classification [130]. As our contribution, we firstly attempt to leverage the meta-learning paradigm for inductive semi-supervised node classification on graphs, which learns to train an inductive model for new graphs. We propose a novel frame-work MI-GNN, which employs a dual-adaptation mechanism to learn the general knowledge of training an inductive model at both the task and graph levels.

For generalizing GNNs across time, most existing works resort to taking discrete snapshots of the temporal graph, or are not inductive to deal with new nodes, or do not model the exciting effects which is the ability of events to influence the occurrence of another event. As our contribution, for the first time in temporal graph representation learning, we recognize the importance of modeling the events at an individual and collective scale, and formulate them as event and node dynamics. We propose a novel framework called TREND [129] with both event and node dynamics to more precisely model events under a Hawkes process-based GNN. On one hand, the event dynamics learns an adaptable event prior to capture the uniqueness of events individually. On the other hand, the node dynamics regularizes the events at the node level to govern their occurrences collectively.

For generalizing GNNs across low-resource node classification tasks, existing works may use pre-trained language models, or pre-trained GNNs. They neither utilize the graph structual information, nor use fine-grained text information. As our contribution, we firstly attempt to pre-train text and graph encoders jointly for low-resource text classification. We propose a novel model called Graph-Grounded Pre-training and Prompting (G2P2), with three graph interaction-based

contrastive strategies in pre-training, and a prompting approach for the jointly pre-trained graph-text model in downstream tasks.

Chapter 3

Generalizing GNNs across Graphs

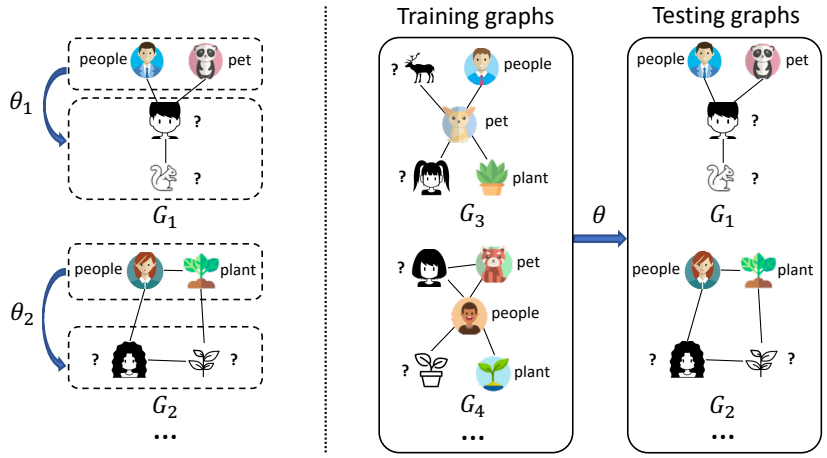
In this chapter, we study the problem of inductive node classification across graphs. Unlike existing one-model-fits-all approaches, we propose a novel meta-inductive framework called MI-GNN to customize the inductive model to each graph under a meta-learning paradigm. That is, MI-GNN does not directly learn an inductive model; it learns the *general knowledge* of how to train a model for semi-supervised node classification on new graphs. To cope with the differences across graphs, MI-GNN employs a *dual adaptation mechanism* at both the graph and task levels. This work is presented in:

- Zhihao Wen, Yuan Fang, Zemin Liu. Meta-Inductive Node Classification across Graphs. In *Proceedings of the 44th International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR '21).*

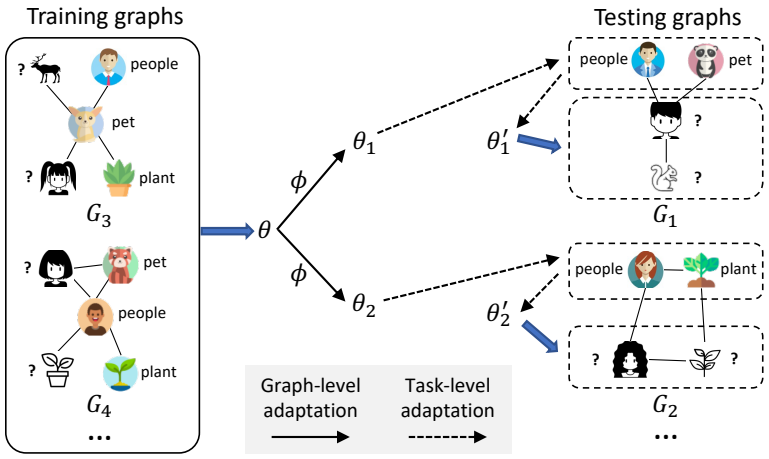
3.1 Introduction

Graph-structured data widely exist in diverse real-world scenarios, such as social networks, e-commerce graphs, citation graphs, and biological networks. Analysis of these graphs can uncover valuable insights about their respective application domain. In particular, semi-supervised node classification on graphs [4] is an important task in information retrieval. For instance, on a content-sharing social network such as Flickr, content classification enables topical filtering and tag-based retrieval for multimedia items [99]; on a query graph for e-commerce, query intent classification enhances the ranking of results by focusing on the intended product category [21]. Such scenarios are semi-supervised, as only some of the nodes on the graph are *labeled* with a category, whilst the remaining nodes are *unlabeled*. The labeled nodes and the intrinsic structures between both labeled and unlabeled nodes (*i.e.*, the graph) can be used for training a model to classify the unlabeled nodes.

Unfortunately, traditional manifold-based semi-supervised approaches on graphs [4, 159, 154, 134, 20] mostly assume a transductive setting. That is, the learned model only works on existing nodes in the same graph, and cannot be applied to new nodes added to the existing graph or entirely new graphs even if they are from the same domain. As Figure 3.1(a) shows, a transductive approach directly trains a model θ_i on the labeled nodes of each graph G_i, and apply the model to classify the unlabeled nodes in the same graph G_i. While some simple inductive extensions exist through nearest neighbors or kernel regression [36], they can only deal with new nodes in a limited manner by processing the local changes, and often cannot generalize to handling new graph structures. The ability to handle new graphs is important, as we often need to deal with a series of ego-networks or subgraphs [149, 56, 16] when the full graph is too large to process or impossible to obtain. Thus, it becomes imperative to equip semi-supervised node classification with the inductive capability of generalizing across graphs.



(a) Transductive approach (b) Conventional inductive approach



(c) Our meta-inductive approach (MI-GNN)

Figure 3.1: Illustrative comparison of transductive, inductive and our meta-inductive approaches for semi-supervised node classification on subgraphs of an image-sharing network. (Colored images: labeled nodes; black & white images: unlabeled nodes.)

Problem setting. In this paper, we study the problem of *inductive semi-supervised node classification across graphs*. Consider a set of training (existing) graphs and a set of testing (new) graphs. In a training graph, some or all of the nodes are labeled with a category; in a testing graph only some of the nodes are labeled and the rest are unlabeled. The nodes in all graphs reside in the same feature space and share a common set of categories. Our goal is to learn an inductive model from the training graphs, which can be applied to the testing graphs to classify their unlabeled nodes.

Prior work. State-of-the-art node classification approaches hinge on graph representation learning, which projects nodes to a latent, low-dimensional vector space. There exist two main factions: network embedding [8] and graph neural networks (GNNs) [135].

On one hand, network embedding methods directly parameterize node embedding vectors and constrain them with various local structures, such as random walks in DeepWalk [87] and node2vec [28], and first- and second-order proximity in LINE [111]. Due to the direct parameterization, network embedding has limited inductive capability like the traditional manifold approaches. For instance, the online version of DeepWalk handles new nodes by incrementally processing the local random walks around them.

On the other hand, GNNs integrate node features and structures into representation learning. They typically follow a message passing framework, in which each node receives, maps and aggregates messages (*i.e.*, features or embeddings) from its neighboring nodes in multiple layers to generate its own embedding vector. The implication is that GNNs are parameterized by a weight matrix in each layer to map the messages from the neighboring nodes, instead of directly learning the node embedding vectors. In particular, the weight matrices give rise to the inherent inductive power of GNNs, which can be applied to similarly map and aggregate messages in a new graph given the same feature space. As Figure 3.1(b) shows, we can train a GNN model θ on a collection of training graphs $\{G_3, G_4\}$, which are image co-occurrence subgraphs of an image-sharing network like Flickr [148]. Specifically, in every subgraph, each node represents an image, and an edge can be formed between two images if they have certain common properties (*e.g.*, submitted to the same gallery or taken by friends). The learned model θ can be deployed to predict the unlabeled nodes on new testing graphs $\{G_1, G_2\}$, which are different subgraphs from the same image-sharing network. In particular, nodes in all subgraphs share the same feature space and belong to a common set

of categories.

Challenges and present work. While most GNNs can be inductive, ultimately they only train a single inductive model to apply on all new graphs. These one-model-fits-all approaches turn out to suffer from a major drawback, as they neglect inter-graph differences that can be crucial to new graphs. Even graphs in the same domain often exhibit a myriad of differences. For instance, social ego-networks for different kind of ego-users (*e.g.*, businesses, celebrities and regular users) show dissimilar structural patterns; image co-occurrence subgraphs in different galleries have varying distributions of node features and categories. To cope with such inter-graph differences, it remains challenging to formulate an inductive approach that not only becomes aware of but also customizes to the differences across graphs. To be more specific, there are two open questions to address.

First, *how do we dynamically adjust the inductive model?* A naïve approach is to perform an additional fine-tuning step on the labeled nodes of the new graph. However, such a fine-tuning on new graphs is decoupled from the training step, which does not learn to deal with inter-graph differences. Thus, the two-step approach cannot adequately customize to different graphs. Instead, the training process must be made aware of inter-graph differences and further adapt to the differences across training graphs. In this paper, we resort to the *meta-learning* paradigm [96, 120], in which we do not directly train an inductive model. Instead, we learn a form of general knowledge that can be quickly utilized to produce a customized inductive model for each new graph. In other words, the general knowledge encodes *how to train* a model for new graphs. While meta-learning has been successfully adopted in various kinds of data including images [61], texts [40] and graphs [155], these approaches mainly address the few-shot learning problem, whereas our work is the first to leverage meta-learning for inductive semi-supervised node classification on graphs.

Second, more concretely, *what form of general knowledge can empower semi-supervised node classification on a new graph?* On one hand, every semi-supervised node classification task is different, which arises from different nodes and labels across tasks. On the other hand, every graph is different, providing a different context to the tasks on different graphs. Thus, the general knowledge should encode how to deal with both task- and graph-level differences. As Figure 3.1(c) illustrates, for task-level differences, we learn a *task prior* θ that can be eventually adapted to the semi-supervised node classification task in a new graph; for graph-level differences, we learn a *graph prior* ϕ that can first transform θ into θ_i conditioned on each graph G_i, before further adapting θ_i to θ_i' for the classification task on G_i. In other words, our general knowledge consists of the task prior and graph prior, amounting to a *dual adaptation mechanism* on both tasks and graphs. Intuitively, the graph-level adaptation exploits the intrinsic relationship between graphs, whereas the task-level adaptation exploits the graph-conditioned relationship between tasks. This is a significant departure from existing task-based meta-learning approaches such as protonets [104] and MAML [22], which assumes that tasks are i.i.d. sampled from a task distribution. In contrast, in our setting tasks are non-i.i.d. as they are sampled from and thus conditioned on different graphs.

Contributions. Given the above challenges and insights for inductive semi-supervised node classification across graphs, we propose a novel Meta-Inductive framework for Graph Neural Networks (MI-GNN). To summarize, we make the following contributions. (1) This is the first attempt to leverage the meta-learning paradigm for inductive semi-supervised node classification on graphs, which learns to train an inductive model for new graphs. (2) We propose a novel framework MI-GNN, which employs a dual-adaptation mechanism to learn the general knowledge of training an inductive model at both the task and graph levels. (3) We conduct extensive experiments on five real-world datasets, and demonstrate the

Table 3.1: List of major notations.

Notation	Description
$G, V, E, \mathbf{X}$	a graph, its node and edge set, and node feature matrix
C	the set of node categories
ℓ	the label mapping function $V \to C$
$\mathcal{N}_v$	the set of neighbors of node v
$\mathcal{G}, \mathcal{G}^{\text{tr}}, \mathcal{G}^{\text{te}}$	the set of all graphs, training graphs and testing graphs
G_i, S_i, Q_i	a graph G_i with support set S_i and query set Q_i
θ, ϕ	general knowledge: task prior θ, graph prior ϕ
γ_i, β_i	scaling and shifting vectors for graph G_i
θ_i	graph G_i-conditioned task prior
θ_i'	graph G_i-conditioned and task adapted model

superior inductive ability of the proposed MI-GNN.

3.2 Preliminaries

In this section, we first formalize the problem of inductive semi-supervised node classification across multiple graphs. We then present a background on graph neural networks as the foundation of our approach. Major notations are summarized in Table 3.1.

3.3 Problem formulation

Graphs. Our setting assumes a set of graphs $\mathcal{G}$ from the same domain. A graph $G \in \mathcal{G}$ is a quintuple $G = (V, E, \mathbf{X}, C, \ell)$ where (1) V denotes the set of nodes; (2) E denotes the set of edges between nodes; (3) $\mathbf{X} \in \mathbb{R}^{|V| \times d}$ is the feature matrix such that $\mathbf{x}_v$ is the d-dimensional feature vector of node v; (4) C is the set of node categories; (5) ℓ is a label function that maps each node to its category, *i.e.*, $\ell : V \to C$. Note that the node feature space and category set are the same across all graphs.

Inductive semi-supervised node classification. The graph set $\mathcal{G}$ comprises two disjoint subsets, namely, training graphs $\mathcal{G}^{\text{tr}}$ and testing graphs $\mathcal{G}^{\text{te}}$. In a training graph, some or all of the nodes are labeled, *i.e.*, the label mapping ℓ is known for these nodes. In contrast, in a testing graph, only some of the nodes are *labeled* and the remaining nodes are *unlabeled*, *i.e.*, their label mapping is unknown. Subsequently, given a set of training graphs $\mathcal{G}^{\text{tr}}$ and a testing graph $G \in \mathcal{G}^{\text{te}}$, our goal is to predict the categories of unlabeled nodes in G. This is known as inductive node classification, as we attempt to distill the training graphs to enable node classification in new testing graphs that have not been seen before.

Distinction from few-shot classification. While we address the semi-supervised node classification problem, it is worth noting that many meta-learning works [61, 145, 155] address the few-shot classification problem. Both problems contain labeled and unlabeled nodes (respectively known as the support and query nodes in the few-shot setting), and thus they may appear similar. However, there are two significant differences. First, in few-shot classification, the query nodes belong to the same category as at least one of the support nodes. This is often unrealistic on a small graph where some categories only contain one node. In contrast, in our setting, the labeled and unlabeled nodes can be randomly split on any graph. Second, few-shot classification typically deals with novel categories on the same graph, but our setting deals with novel graphs with the same set of categories.

Graph neural networks

Our approach is grounded on graph neural networks, which are inductive due to the shared feature space and weights across graphs. We give a brief review of GNNs in the following.

Modern GNNs generally follow a message passing scheme: each node in a graph receives, maps, and aggregates messages from its neighbors recursively in

multiple layers. Specifically, in each layer,

$$\mathbf{h}_v^{l+1} = \mathcal{M}(\mathbf{h}_v^l, \{\mathbf{h}_u^l, \forall u \in \mathcal{N}_v\}; \mathbf{W}^l), \tag{3.1}$$

where $\mathbf{h}_v^l \in \mathbb{R}^{d_l}$ is the message or the d_l-dimensional embedding vector of node v in the l-th layer, $\mathcal{N}_v$ is the set of neighboring nodes of v, $\mathbf{W}^l \in \mathbb{R}^{d_{l+1} \times d_l}$ is a learnable weight matrix to map the node embeddings in the l-th layer, and $\mathcal{M}(\cdot)$ is the message aggregation function. The initial message of node v in the input layer is simply the original node features, *i.e.*, $\mathbf{h}_v^1 \equiv \mathbf{x}_v$. For node classification, the dimension of the output layer is set to the number of node categories and uses a softmax activation.

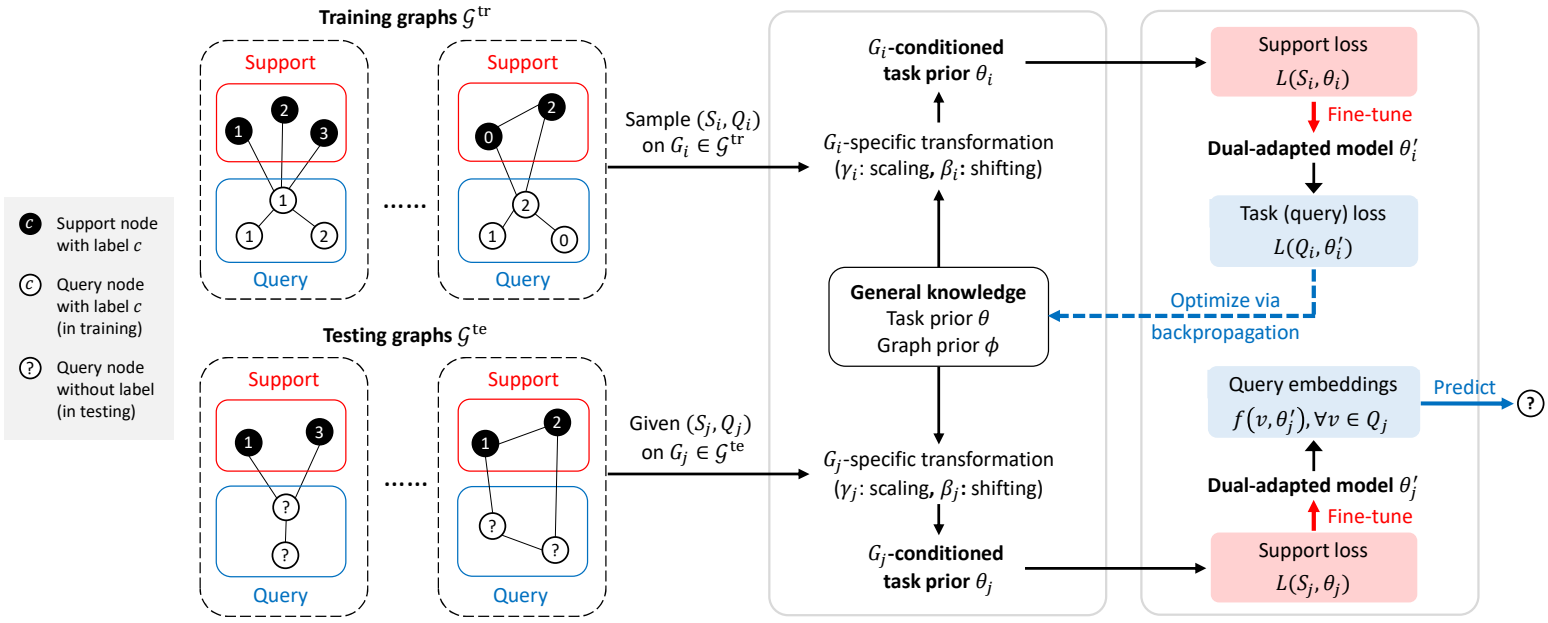
The choice of the message aggregation function $\mathcal{M}$ varies and characterizes different GNN architectures, ranging from a simple mean pooling [46, 31, 132] to more complex mechanisms [117, 31, 140]. Our proposed model is flexible in the aggregation functions.

3.4 Methodology

In this section, we present a novel graph inductive framework called MI-GNN, a meta-inductive model that learns to train a model for every new graph. In the following, we start with an overview of the framework, before we introduce its components in detail.

Overview of MI-GNN

The overarching philosophy of MI-GNN is to design an inductive approach that can dynamically suit to each new graph, in order to cope with the inter-graph differences. A straightforward approach is to train a model on the training graphs,



(a) Training/testing graphs and tasks (b) Graph-level adaptation (c) Task-level adaptation

Figure 3.2: Overall framework of MI-GNN, illustrating the pipeline on a training graph G_i and a testing graph G_j.

and further perform a fine-tuning step on a new graph in the testing phase. However, since the training step is independent of the fine-tuning step, it does not train the model to learn how to fine-tune on unseen graphs. In contrast, MI-GNN, hinging on the meta-learning principle, learns a general training procedure so that it knows how to dynamically generate a model suited to any new graph. We set forth the overall framework of MI-GNN in Figure 3.2.

First of all, MI-GNN exploits each training graph to simulate the semi-supervised node classification task in testing, as shown in Figure 3.2(a). Specifically, we take a training graph and split its nodes with known labels into two subsets: the *support* set and *query* set, following the task-based meta-learning setup [22]. While in a training graph both the support and query nodes have known category labels, we regard the support nodes as the only labeled nodes and the query nodes as the unlabeled nodes to simulate the semi-supervised classification process during training. On a testing graph, the labeled and unlabeled nodes naturally form the support and query sets, respectively, where the ultimate goal is to predict the unknown categories of the query nodes.

Next, on the simulated tasks, we learn a task prior and a graph prior during training. The task prior captures the general knowledge of classifying nodes in a semi-supervised setting, whereas the graph prior captures the general knowledge of transforming the task prior w.r.t. each graph. In other words, our general knowledge allows for dual adaptations at both the graph and task levels. On one hand, the graph prior captures and adapts for macro differences across graphs, as illustrated in Figure 3.2(b). On the other hand, the task prior captures and adapts for micro differences across tasks conditioned on a graph, as illustrated in Figure 3.2(c).

Graphs and tasks

Training tasks. We refer to the upper half of Figure 3.2(a) for illustration. On a training graph $G_i^{\text{tr}} \in \mathcal{G}^{\text{tr}}$, we can sample a semi-supervised node classification task by randomly splitting its nodes with known labels into the support set S_i^{tr} and query set Q_i^{tr} such that $S_i^{\text{tr}} \cap Q_i^{\text{tr}} = \emptyset$. Specifically, without loss of generality, for the node set with known labels $\{v_{i,k} : 1 \leq k \leq m+n\}$ on G_i^{tr}, the support and query sets are given by

$$S_i^{\text{tr}} = \{(v_{i,k}, \ell(v_{i,k})) : 1 \leq k \leq m\}, \tag{3.2}$$

$$Q_i^{\text{tr}} = \{(v_{i,m+k}, \ell(v_{i,m+k})) : 1 \leq k \leq n\}, \tag{3.3}$$

where $m = |S_i^{\text{tr}}|$ and $n = |Q_i^{\text{tr}}|$ denotes the number of nodes in the support and query sets, respectively. Note that for both support and query nodes, their label mapping ℓ is known on training graphs. During training, we mimic the model updating process on the support nodes w.r.t. their classification loss, and further mimic the prediction process on the query nodes. In particular, the labels of the query nodes are hidden from the model updating process on support nodes, but are used to validate the predictions on the query nodes in order to optimize the

general knowledge.

Testing tasks. On the other hand, suppose a testing graph $G_j^{\text{te}} \in \mathcal{G}^{\text{te}}$ has a node set $\{v_{j,k} : 1 \leq k \leq m+n\}$ such that nodes are labeled for $1 \leq k \leq m$ only and unlabeled for $m+1 \leq k \leq m+n$. The support and query sets are then given by

$$S_j^{\text{te}} = \{(v_{j,k}, \ell(v_{j,k})) : 1 \leq k \leq m\}, \tag{3.4}$$

$$Q_j^{\text{te}} = \{v_{j,m+k} : 1 \leq k \leq n\}. \tag{3.5}$$

The main difference from the training setting is that, the support set contains all the labeled nodes and the query set contains all the unlabeled nodes, as illustrated in the lower half of Figure 3.2(a). While the labeled support nodes on a testing graph are used in the same way as in training, the unlabeled query nodes are only used for prediction and evaluation.

In the following, for brevity we will omit the superscripts $^{\text{tr}}$ and $^{\text{te}}$ that distinguishes training and testing counterparts (such as $G_i^{\text{tr}}, S_i^{\text{tr}}, Q_i^{\text{tr}}$ and $G_j^{\text{te}}, S_j^{\text{te}}, Q_j^{\text{te}}$) when there is no ambiguity or we are referring to any graph in general (*i.e.*, regardless of training or testing graphs).

Graph-level adaptation

We first formalize the general knowledge consisting of a task prior and a graph prior, which are the foundation of the graph-level adaptation as illustrated in Figure 3.2(b).

Task and graph priors. The task prior θ is designed for quick adaptation to a new semi-supervised node classification task. Given that GNNs can learn powerful

node representations, our task prior takes the form of a GNN model, *i.e.*,

$$\theta = (\mathbf{W}^1, \mathbf{W}^2, ...), \tag{3.6}$$

where each $\mathbf{W}^l$ is a learnable weight matrix to map the messages from the neighbors in the l-th layer, as introduced in Section 3.3.

Different from most task-based meta learning [104, 22], our tasks are not sampled from an i.i.d. distribution. Instead, tasks are sampled from different graphs, and each task is contextualized and thus conditioned on a graph. We employ a graph prior ϕ to condition the task prior, so that the task prior can be transformed to suit each graph. The transformation model is given by

$$\tau(\theta, \mathbf{g}; \phi), \tag{3.7}$$

which (1) is parameterized by the graph prior ϕ; (2) takes in the task prior θ, and the graph-level representation $\mathbf{g}$ of an input graph G (which can be either a training or testing graph); (3) outputs a transformed, graph G-conditioned task prior. In other words, the graph prior does not directly specify the transformation, but it encodes the rules of how to transform w.r.t. each graph. This is essentially a form of hypernetwork [29, 86], where the task prior is adjusted by a secondary network (parameterized by ϕ) in response to the changing input graph.

In the following, we discuss the concrete formulation of the graph-level representation $\mathbf{g}$, the transformation model τ and its parameters ϕ (*i.e.*, the graph prior).

Graph-conditioned transformation. To transform the task prior conditioned on a given graph G, we need a graph-level representation $\mathbf{g}$ of the graph. A straightforward approach is to perform a mean pooling of the features or embeddings of all nodes. Although simple, mean pooling does not differentiate the

relative importance of each node to the global representation $\mathbf{g}$. Thus, we adopt an attention-based aggregation to compute our graph-level representation [1], which assigns bigger weights to more significant nodes.

Consider a graph G_i (which can be either a training or testing graph) and its graph-level representation vector $\mathbf{g}_i$. We perform feature-wise linear modulations [86] on the task prior in order to adapt to G_i, by conditioning the transformations on $\mathbf{g}_i$. This is more flexible than gating, which can only diminish an input as a function of the same input, instead of a different conditioning input [86]. To be more specific, we use MLPs to generate the scaling vector γ_i and shifting vector β_i given the input $\mathbf{g}_i$, which will be used to transform the task prior in order to suit G_i. Specifically,

$$\gamma_i = \text{MLP}_\gamma(\mathbf{g}_i; \phi_\gamma), \tag{3.8}$$

$$\beta_i = \text{MLP}_\beta(\mathbf{g}_i; \phi_\beta), \tag{3.9}$$

where ϕ_γ and ϕ_β are the learnable parameters of the two MLPs, respectively. Here $\gamma_i \in \mathbb{R}^{d_\theta}$ and $\beta_i \in \mathbb{R}^{d_\theta}$ are d_θ-dimensional vectors, where d_θ is the number of parameters in task prior θ. Note that θ contains all the GNN weight matrices, and we flatten it into a d_θ-dimensional vector in a slight abuse of notation.

Since γ_i and β_i have the same dimension as θ, we can apply the transformation in an element-wise manner, to produce the graph G_i-conditioned task prior as

$$\theta_i = \tau(\theta, \mathbf{g}_i; \phi) = (\gamma_i + \mathbf{1}) \odot \theta + \beta_i, \tag{3.10}$$

where $\odot$ denotes element-wise multiplication, and $\mathbf{1}$ is a vector of ones to ensure that the scaling factors are centered around one. The graph prior ϕ, which forms the parameters of τ, consists of the parameters of the two MLPs, *i.e.*,

$$\phi = \{\phi_\gamma, \phi_\beta\}. \tag{3.11}$$

Note that τ is a function of $\mathbf{g}_i$ as well, since γ_i and β_i are functions of $\mathbf{g}_i$ generated by the two MLPs in Eqs. (3.8)–(3.9) in response to the changing input graph. In particular, the two MLPs play the role of secondary networks in the hypernetwork setting [29].

Task-level adaptation

Given any graph G_i (training or testing), the graph-conditioned task prior θ_i serves as a good initialization of the GNN on G_i, which can be rapidly adapted to different semi-supervised node classification tasks on G_i. Following MAML [22], we perform a few gradient descent updates on the support nodes S_i for rapid adaptation, and finally obtain the dual-adapted model θ_i' as shown in Figure 3.2(c). Too many updates may cause overfitting to the support nodes and thus hurt the generalization to query nodes, especially when the support set is small in the semi-supervised setting.

The following Eq. (3.12) demonstrates one gradient update on the support set S_i w.r.t. the graph G_i-conditioned θ_i, and extension to multiple steps is straightforward.

$$\theta_i' = \theta_i - \alpha \frac{\partial L(S_i, \theta_i)}{\partial \theta_i}, \tag{3.12}$$

where $\alpha \in \mathbb{R}$ is the learning rate of the task-level adaptation, and $L(S_i, \theta_i)$ is the cross-entropy classification loss on the support S_i using the GNN model parameterized by θ_i, as follows.

$$L(S_i, \theta_i) = - \sum_{(v_{i,k}, \ell(v_{i,k})) \in S_i} \sum_{c \in C} I(\ell(v_{i,k}) = c) \log f(v_{i,k}; \theta_i)[c], \tag{3.13}$$

where $I(*)$ is an indicator function, $f(*; \theta_i) \in \mathbb{R}^{|C|}$ is the output layer of the GNN parameterized by θ_i with a softmax activation, and $f(*; \theta_i)[c]$ denotes the

probability of category c.

Overall algorithm

Finally, we present the algorithm for training and testing.

Training. Consider a training graph $G_i \in G^{\text{tr}}$ with graph-level representation $\mathbf{g}_i$, and a corresponding task (S_i, Q_i). The goal is to optimize the general knowledge in terms of the task prior θ and graph prior ϕ via backpropagation w.r.t. the loss on the query nodes after dual adaptions. Specifically, the optimal $\{\theta, \phi\}$ is given by

$$\arg\min_{\theta,\phi} \sum_{G_i \in \mathcal{G}^{\text{tr}}} L(Q_i, \theta_i') + \lambda(\|\gamma_i\|_2 + \|\beta_i\|_2), \tag{3.14}$$

where (1) θ_i' is the dual-adapted prior after performing one gradient update according to Eq. (3.12) on the G_i-condidtioned prior $\theta_i = \tau(\theta, \mathbf{g}_i; \phi)$, implying that θ_i' is a function of θ and ϕ; (2) $L(Q_i, *)$ is the task loss using the same cross-entropy definition shown in Eq. (3.13), but computed on the query set Q_i; (3) the L_2 regularization $\|\gamma_i\|_2 + \|\beta_i\|_2$ ensures that the scaling is close to 1 and the shifting is close to 0 to prevent overfitting to the training graphs, and $\lambda > 0$ is a hyperparameter to control the regularizer. In our rigorous exploration of various popular regularization techniques, we sought to enhance the performance of our model. Extensive experimentation encompassed methodologies such as L1 and L2 regularization, dropout, and batch normalization. However, our empirical analysis revealed that directly regularizing the scaling and shifting factors emerged as the most effective strategy. This finding suggests that imposing explicit constraints on these factors yields superior results in terms of model generalization and mitigating overfitting. The empirical evidence substantiates the superiority of directly regularizing the scaling and shifting factors, underscoring its significance

Algorithm 1 TrainingProcedure

Require: training graph set $\mathcal{G}^{\text{tr}}$.
Ensure: task prior θ, graph prior ϕ.
$\theta, \phi \leftarrow$ parameters initialization;
while not converged **do**
  sample a batch of graphs from $\mathcal{G}^{\text{tr}}$;
  for each graph G_i in the batch **do**
    sample support set S_i, query set Q_i from G_i;
    calculate scaling and shifting factors γ_i, β_i; ▷ Eqs. (3.8), (3.9)
    $\theta_i \leftarrow$ graph-level adaptation on θ; ▷ Eq. (3.10)
    calculate support loss $L(S_i, \theta_i)$ and gradient; ▷ Eq. (3.13)
    $\theta_i' \leftarrow$ task-level adaptation on θ_i; ▷ Eq. (3.12)
    calculate task (query) loss $L(Q_i, \theta_i')$;
  end for
  $\theta, \phi \leftarrow$ backpropagation of total task loss ▷ Eq. (3.14)
end while
return θ, ϕ.

in achieving optimal performance in our study.

In practical implementation, the optimization is performed over batches of training graphs using any gradient-based optimizer. The overall training procedure is outlined in Algorithm 1.

Testing. During testing, we follow the same dual adaption mechanism on each testing graph $G_j \in \mathcal{G}^{\text{te}}$ to generate the dual-adapted prior θ_j'. The only difference from training is that, the query nodes are used for prediction and evaluation, not for backpropagation. That is, for any unlabeled node in the query set $v_{j,k} \in Q_j$, we predict its label as $\arg\max_{c \in C} f(v_{j,k}; \theta_j')[c]$.

3.5 Experiments

In this section, we conduct extensive experiments to evaluate MI-GNN. More specifically, we compare MI-GNN with state-of-the-art baselines, study the effectiveness of our dual adaptations, and further analyze hyperparameter sensitivity and performance patterns.

Table 3.2: Statistics of graph datasets.

Dataset	Flickr	Yelp	Cuneiform	COX2	DHFR
# Graphs	800	800	267	467	756
# Edges (avg.)	13.1	43.5	20.1	44.8	44.5
# Nodes (avg.)	12.5	6.9	21.3	41.2	42.4
# Node features	500	300	3	3	3
# Node classes	7	10	7	8	9
Multi-label?	No	Yes	Yes	No	No

Experimental setup

Datasets. We conduct experiments on five public graph collections, as follows. Their statistics are summarized in Table 3.2.

- Flickr [148] is a collection of 800 ego-networks sampled from an online image-sharing network. Each node is an image, and each edge connects two images that share some common properties (*e.g.*, same geographic location or gallery). Our task is to classify each image into one of the seven categories.

- Yelp [148] is a collection of 800 ego-networks sampled from an online review network. Each node represents a user, and each edge represents the friendship relations between users. Our task is to classify each user node according to the types of business reviewed by the user in a multi-label setting.

- Cuneiform [48] is a collection of 267 cuneiform signs in the form of wedge-shaped marks. Each node is a wedge, and each edge indicates the arrangement of the wedges. Our task is to classify the visual appearance of the wedges in a muli-label setting.

- COX2 and DHFR [109] are two collections of molecular structures. Specifically, COX2 is a set of 467 cyclooxygenase-2 inhibitors; DHFR is a set of 756 dihydrofolate reductase inhibitors. Each node is an atom and each edge is a chemical bond between two atoms. Our task is to predict the node atomic type.

Table 3.3: Performance of MI-GNN and baselines, in percent, with 95% confidence intervals.

In each column, the best result is **bolded** and the runner-up is underlined. Improvement by MI-GNN is calculated relative to the best baseline. ***/**/* denotes the difference between MI-GNN and the best baseline is statistically significant at the 0.01/0.05/0.1 level under the two-tail t-test.

	Flickr	Yelp	Cuneiform	COX2	DHFR
	Accuracy	Accuracy	Accuracy	Accuracy	Accuracy
DeepWalk	39.88±2.42	63.27±2.73	74.61±0.60	37.68±0.73	33.14±0.18
Transduct-GNN	13.61±1.22	24.87±15.4	49.63±0.95	13.23±0.17	11.21±0.33
Planetoid	14.78±8.75	53.12±2.38	53.14±5.49	11.81±7.41	17.35±11.1
Induct-GNN	40.48±1.69	65.95±0.56	74.89±0.35	53.71±0.92	45.23±0.62
K-NN	34.11±1.76	61.70±0.90	70.36±0.27	33.16±0.95	36.32±0.89
AGF	40.58±1.61	65.96±0.54	74.89±0.37	53.97±0.79	44.85±0.56
GFL	30.24±0.68	61.62±0.97	63.72±0.37	29.25±0.73	30.24±0.68
Meta-GNN	39.66±0.92	66.24±0.84	75.12±0.33	53.24±0.77	**45.61**±0.65
MI-GNN	**44.45**±2.18	**67.92**±0.69	**81.48**±0.47	**57.27**±0.80	45.19±0.70
(improv.)	(+9.53%)	(+2.54%)	(+8.47%)	(+6.11%)	(-0.92%)
	**	***	***	***	
	Flickr	Yelp	Cuneiform	COX2	DHFR
	Micro-F1	Micro-F1	Micro-F1	Micro-F1	Micro-F1
DeepWalk	30.01±1.21	57.11±6.29	27.05±2.11	26.16±1.08	29.93±0.58
Transduct-GNN	10.71±1.20	23.85±14.6	34.00±1.15	9.73±0.22	8.65±0.22
Planetoid	8.72±3.07	46.29±3.55	30.22±5.83	10.58±8.79	9.62±9.63
Induct-GNN	29.67±1.77	56.61±1.81	18.03±0.93	41.56±1.90	29.38±6.07
K-NN	26.39±1.39	57.35±1.42	35.66±0.84	32.84±1.00	27.12±1.20
AGF	28.99±2.09	56.64±1.83	18.00±0.94	42.00±1.62	29.08±5.96
GFL	29.51±0.69	58.88±2.03	38.30±0.84	25.53±0.94	29.51±0.69
Meta-GNN	30.02±2.49	56.20±1.81	19.21±1.25	37.36±3.02	28.34±4.46
MI-GNN	**33.79**±1.87	**60.20**±2.23	**43.32**±1.49	**44.66**±2.01	**49.93**±1.62
(improv.)	(+12.57%)	(+2.23%)	(+13.10%)	(+6.34%)	(+66.82%)
	**	***	***	*	***

Table 3.4: Accuracy of MI-GNN and baselines using alternative GNN architectures, in percent, with 95% confidence intervals.

	GCN as the GNN Architecture				
	Flickr	Yelp	Cuneiform	COX2	DHFR
Transduct-GNN	14.89±0.94	50.92±0.95	49.40±2.27	11.89±0.63	10.89±0.43
Induct-GNN	12.08±3.98	55.04±1.77	71.65±0.46	86.06±2.78	90.31±1.03
AGF	11.94±2.45	53.66±3.04	71.66±0.46	86.32±3.08	89.64±1.00
Meta-GNN	22.51±3.05	54.80±1.86	72.24±0.88	86.92±3.66	90.26±0.91
MI-GNN	**29.91**±6.85	**57.22**±1.79	**75.36**±2.07	**86.97**±2.94	**91.39**±0.51
	GraphSAGE as the GNN Architecture				
	Flickr	Yelp	Cuneiform	COX2	DHFR
Transduct-GNN	14.97±1.96	50.14±1.19	50.59±1.37	12.78±0.65	11.19±0.75
Induct-GNN	7.31±1.57	56.48±1.73	84.46±2.68	85.28±1.78	88.65±4.79
AGF	7.45±1.31	56.70±2.04	84.66±2.73	85.21±1.85	88.21±4.45
Meta-GNN	33.88±2.91	61.80±1.81	84.46±2.44	86.05±2.80	88.17±4.71
MI-GNN	**42.37**±3.87	**69.23**±1.18	**91.09**±2.51	**93.24**±0.80	**93.89**±0.83

Training and testing. For each graph collection, we randomly partition the graphs into 60%, 20% and 20% subsets for training, validation and testing, respectively. On each graph, we randomly split its nodes into two equal halves as the support and query sets, respectively. Our goal is to evaluate the performance of node classification on the unlabeled query nodes on the testing graphs, in terms of accuracy and micro-F1. Note that on multi-label graphs with $|C|$ categories, we perform $|C|$ binary classification tasks, one for each category. Each model is trained with 10 random initializations, and we report the average accuracy and micro-F1 over the 10 runs with 95% confidence intervals.

Settings of MI-GNN. First, our approach can work with different GNN architectures. By default, we use simplifying graph convolutional networks (SGC) [132] in all of our experiments, except in Section 3.5 where we also adopt GCN [46] and GraphSAGE [31] to evaluate the flexibility of MI-GNN. For all GNNs, we employ two layers with a hidden dimension of 16. For GraphSAGE, we use the mean aggregator.

Next, for graph-level adaptations, in Eqs. (3.8) and (3.9) we adopt MLPs

with one hidden layer using LeakyReLU as the activation function, and a linear output layer. For task-level adaptations, we set the number of gradient descent updates to two, and the learning rate of task adaptation α in Eq. (3.12) to 0.5 for Flickr, Yelp and Cuneiform or 0.005 for COX2 and DHFR. Lastly, for the overall optimization in Eq. (3.14), we use the Adam optimizer with the learning rate 0.01, and set the regularization co-efficient λ to 1 on Flickr and 0.001 on all other datasets. The settings are tuned using the validation graphs.

Baselines and settings. We compare our proposed MI-GNN with a comprehensive suite of competitive baselines from three categories.

(1) *Transductive* approaches, which do not utilize training graphs. Instead, they directly train the model using the labeled nodes on each testing graph, and we evaluate their classification performance on the unlabeled nodes in the same graph.

- DeepWalk [87]: an unsupervised network embedding method that learns node representations based on the skip-gram model [78] to encode random walk sequences. After obtaining node representations on a testing graph, we further train a logistic regression classifier using the labeled nodes.
- Transduct-GNN: applying a GNN in a transductive setting, where it is directly trained on each testing graph.

(2) *Inductive approaches*, which utilize the training graphs to learn an inductive model that can be applied to new testing graphs. In particular, a fixed inductive model is trained with either no or limited adaptation to the testing graphs.

- Planetoid [142]: Planetoid is a semi-supervised graph embedding approach. We use its inductive variant in our experiments.
- Induct-GNN: applying a GNN in an inductive setting, where it is trained on the training graphs, followed by applying the trained model on each testing graph

to generate the node representations. The labeled nodes on the testing graphs are not utilized to adapt the trained model.

- K-NN [113]: a two-stage process, in which the first stage is the same as Inductive-GNN, and the second stage subsequently employs a K-nearest-neighbor (K-NN) classifier to classify each unlabeled node into the same category as the closest labeled node in terms of their representations.

- AGF [113]: also a two-stage process similar to K-NN, except that in the second stage the K-NN classifier is substituted by a fine-tuning step performed on the labeled nodes.

(3) *Meta-learning approaches*, which "learns to learn" on the training graphs. Instead of learning a fixed model, they learn different forms of general knowledge that can be conveniently adapted to the semi-supervised task on the testing graphs.

- GFL [145]: a few-shot node classification method on graphs, based on protonets [104]. While there are major differences between the few-shot and semi-supervised tasks, GFL can still be used in our setting although its performance may not be ideal.

- Meta-GNN [155]: another few-shot node classification approach on graphs, based on MAML [22].

All methods (except DeepWalk and Planetoid) use the same GNN architecture and corresponding settings in our model. For K-NN, we use the Euclidean distance and set the number of nearest neighbors to 1. For AGF, GFL and Meta-GNN, we use a learning rate of 0.01. For the fine-tuning step in AGF and the task adaptation in Meta-GNN, we use the same setup as the task adaptation in MI-GNN. For DeepWalk and Planetoid, we set their random walk sampling parameters, such as number of walks, walk length and window size according to their recommended settings, respectively.

Performance comparison to baselines

In Table 3.3, we report the performance comparison of our proposed MI-GNN and the baselines. Generally, our method achieves consistently the best performance among all methods, demonstrating its advantages in inductive semi-supervised node classification. More specifically, we make the following observations.

First, in the transductive setting, Transduct-GNN performs worse than DeepWalk, which is not surprising given that GNNs generally require a large training set to learn effective representations. However, in our setting, an individual test graph may be small with a limited number of labeled nodes. In this regard, the unsupervised representation learning in DeepWalk is more advantageous.

Second, the inductive approaches Induct-GNN and AGF generally outperform transductive approaches, as inductive methods can make use of the abundant training graphs. While K-NN is extended from Induct-GNN with an additional K-nearest neighbor step during testing, it actually performs worse than Induct-GNN. Recall that in our problem setting, on a new graph some node categories may not have labeled nodes (although they have some labeled examples in the training graphs), which makes K-NN unable to classify any node into those categories. Another interesting observation is that, AGF with an additional fine-tuning step on top of Inductive-GNN is only comparable to or marginally better than Inductive-GNN. That means fine-tuning can be prone to overfitting especially when the labeled data are scarce, and a better solution is to learn adaptable general knowledge through meta-learning.

Third, the meta-learning approaches achieve competitive results. GFL and Meta-GNN are often better than inductive approaches, but largely trail behind our approach MI-GNN, as they are designed for few-shot classification and lack the dual-level adaptations. In particular, our proposed MI-GNN outperforms all other methods with statistical significance in all but one case. The only excep-

tion is on the highly imbalanced DHFR dataset, where MI-GNN achieves slightly worse accuracy than Meta-GNN at low significance ($p = 0.442$) but significantly better Micro-F1. Note that Micro-F1 is regarded as a more indicative metric than accuracy on imbalanced classes.

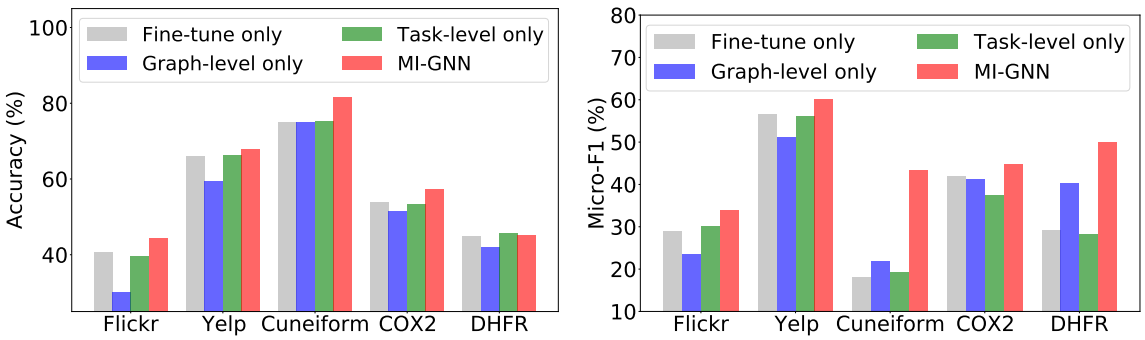
Alternative GNN architectures

As MI-GNN is designed to work with different GNN architectures, we evaluate its flexibility on two other GNN architectures, namely, GCN and GraphSAGE, in addition to SGC as described in the experimental setup. For each architecture, we compare with several representative baselines in Table 3.4. Similar to using SGC, our approach consistently outperforms transductive, inductive and meta-learning baselines alike. The results demonstrate the robustness of our approach across different GNN architectures.

Effect of dual adaptations

The advantage of our approach MI-GNN stems from the dual adaptations at the graph and task levels. To investigate the contribution from each level of adaptation, we perform an ablation study on MI-GNN, comparing with the following variants. (1) *Fine-tune only*: A standard inductive GNN model without any graph- or task-level adaptation, but there is still a simple fine-tuning step on the testing graphs. This is equivalent to the AGF baseline. (2) *Graph-level only*: This can be obtained by removing the task-level adaptation from MI-GNN. (3) *Task-level only*: This can be obtained by removing the graph-level adaptation from MI-GNN.

We present the comparison in Figure 3.3. First of all, MI-GNN outperforms all the ablated models consistently, demonstrating the overall benefit of the dual



(a) Accuracy (b) Micro-F1

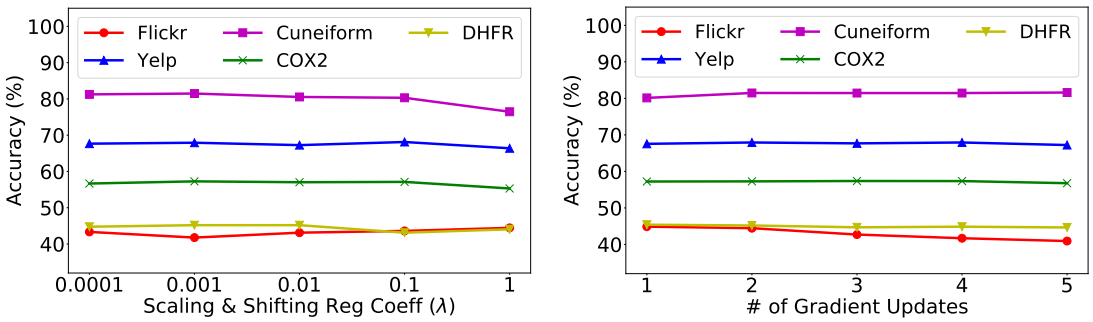
Figure 3.3: Effect of dual adaptations.

adaptations. Among the ablated models, Fine-tune only achieves a surprisingly competitive performance approaching the model with only task-level adaptation, while graph-level adaptation performs rather poorly in majority of the cases. That means in MI-GNN the two levels of adaptations are both crucial and they are well integrated, as each adaptation alone may not give any significant benefit over a simple fine-tuning step but together they work much better.

Hyperparameter sensitivity

We study the effect of regularization in graph-level adaptation, and the number of gradient descent steps in task-level adaptation.

Regularization for graph-level adaptation. To prevent overfitting to the training graphs, we constrain the graph-conditioned transformations to ensure that the scaling is close to 1 and the shifting is close to 0. We study the effect of the regularization in Figure 3.4(a), as controlled by the co-efficient λ in Eq. (3.14). In general, the performance is stable for different values of λ, although smaller values in the range $[0.0001, 0.01]$ tends to perform better. Overly large values will result in very little scaling and shifting, effectively removing the graph-level adaptation and thus suffering from reduced performance.



(a) Regularization (b) Gradient steps

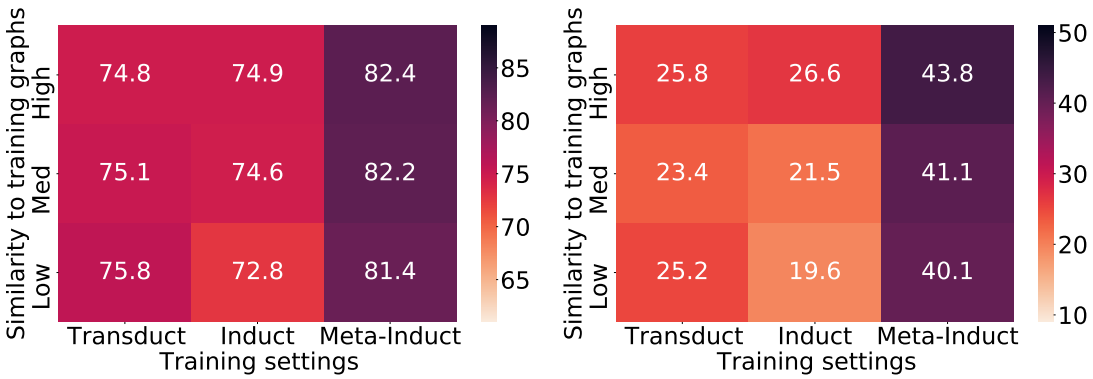
Figure 3.4: Impact of regularization and gradient steps.

Number of gradient steps in task-level adaptation. As discussed in Section 3.4, we achieve task-level adaptation by conducting a few steps of gradient descent on the support set of each graph. To understand the impact of number of steps, we conduct experiments using different number of steps. Results in Figure 3.4(b) reveal that the performance is not sensitive to the number of steps. Thus, it is sufficient to perform just one or two steps for efficiency.

Performance case study

To understand more precisely when our proposed meta-inductive framework can be effective, we conduct a case study on the performance patterns of transductive and inductive methods. On one hand, the performance of inductive models on testing graphs would directly correlate to the similarity between testing and training graphs. Intuitively, the less similar they are, the less effectively knowledge can be transferred from training to testing graphs. On the other hand, transductive methods are not influenced by such similarity, as they do not learn from training graphs at all.

We compute the similarity between two graphs based on the Euclidean distance of their graph-level representations generated by an attention-based model [1]. The similarity between a testing graph and a set of training graphs is then given by the average similarity between the testing graph and each of the training



(a) Accuracy (b) Micro-F1

Figure 3.5: Performance w.r.t. similarity to training graphs.

graph. Subsequently, we split the testing graphs into three groups according to their similarity to the training set, namely, high, medium and low similarity. We report the performance of each group under three settings: transductive (using DeepWalk), inductive (using Induct-GNN) and meta-inductive (using MI-GNN).

We present heatmap visualizations in Figure 3.5 on the Cuneiform dataset. Although the inductive setting can leverage knowledge gained from the training graphs and potentially transfer it to testing graphs, it is not always helpful and can even be harmful when the training data are quite different from the testing data, known as negative transfer [93]. Our heatmaps show that in the transductive setting, the performance remains largely unchanged across the three groups, as transductive methods do not rely on any knowledge transfer from training graphs. In contrast, the conventional inductive setting can suffer from negative transfer, as its performance drops considerably when the testing graphs become less similar to the training graphs. Finally, our meta-inductive approach is generally robust and the effect of negative transfer is much smaller than the conventional inductive method. The underlying reason is that we only learn a form of general knowledge from training graphs, which undergoes a further adaptation process to suit each testing graph. The adaptation process makes our method more robust when dealing with different graphs, which is also our key distinction from conventional inductive methods.

3.6 Conclusion

In this work, we studied the problem of inductive node classification across graphs. Unlike existing one-model-fits-all approaches, we proposed a novel framework called MI-GNN to customize the inductive model to each graph under a meta-learning paradigm. To cope with the differences across graphs, we designed a dual adaptation mechanism at both the graph and task levels. More specifically, we learn a graph prior to adapt for the graph-level differences, and a task prior to further adapt for the task-level differences conditioned on each graph. Extensive experiments on five real-world graph collections demonstrate the effectiveness of MI-GNN.

As for the limitation of MI-GNN, it is that MI-GNN requires a lot of labeled data to do the training. However, in real-world, the acquisition of labeled data is always time cost and labor cost. One possible solution is leveraging self-supervised learning to extract more knowledge and provide a better model initialization. By doing so, MI-GNN can reduce the reliance of labeled data.

3.7 Relation with the other two chapters

The chapter on generalizing Graph Neural Networks (GNNs) across graphs is closely related and integrated with the subsequent chapters on generalizing GNNs across time and generalizing GNNs across node classification tasks. Together, these chapters form a comprehensive exploration of the broader goal of enhancing the generalization capabilities of GNNs in various dimensions.

In the chapter on generalizing GNNs across graphs, the focus is on improving the ability of GNNs to generalize across different graph structures. It investigates techniques and approaches to adapt GNNs to diverse graph topologies, enabling effective learning and inference on unseen graphs. This chapter lays

the foundation for extending the generalization capabilities of GNNs beyond their initial training graph.

Building upon this foundation, the subsequent Chapter 4 on generalizing GNNs across time explores the temporal dimension in graph data. It delves into methods that enable GNNs to capture temporal dependencies and changes over time, facilitating dynamic graph analysis and prediction. By incorporating temporal information, GNNs become more versatile in modeling dynamic graph structures.

In the Chapter 5 on generalizing GNNs across node classification tasks, the focus shifts to the generalization of GNNs across different node classification problems. It investigates techniques that enhance the transferability and adaptation of GNNs to diverse node classification tasks, allowing models to leverage knowledge gained from one task to improve performance on unseen tasks. This chapter expands the applicability of GNNs across a broader range of classification scenarios.

Together, these chapters present a cohesive progression of research, starting with the generalization of GNNs across graph structures, extending to temporal dynamics, and finally encompassing node classification tasks. The integration lies in their shared objective of enhancing the generalization capabilities of GNNs, enabling them to tackle diverse real-world scenarios and facilitating more effective and adaptable graph-based learning systems.

Chapter 4

Generalizing GNNs across Time

The previous chapter explores inductive semi-supervised node classification across graphs where we generalize GNN across graphs. However, in practice, most graphs are temporal and dynamic. In this chapter, we study the temporal graph representation learning and deal with the problem of temporal link prediction. We propose TREND, a novel framework for temporal graph representation learning, driven by TempoRal Event and Node Dynamics and built upon a Hawkes process-based graph neural network (GNN). TREND presents a few major advantages: (1) it is inductive due to its GNN architecture; (2) it captures the exciting effects between events by the adoption of the Hawkes process; (3) as our main novelty, it captures the individual and collective characteristics of events by integrating both event and node dynamics, driving a more precise modeling of the temporal process. This work is presented in:

- Zhihao Wen, Yuan Fang. TREND: TempoRal Event and Node Dynamics for Graph Representation Learning. In *Proceedings of the ACM Web Conference 2022 (WWW '22).*

4.1 Introduction

Graph-structured data widely exist in real-world scenarios, *e.g.*, social networks, citation networks, e-commerce networks, and the World Wide Web. To discover insights from these data, graph representation learning has emerged as a key enabler which can encode graph structures into a low-dimensional latent space. The state of the art has made important progress and many approaches are proposed, which can be mainly divided into two categories: network embedding [8] and graph neural networks (GNNs) [135]. Network embedding approaches are often transductive, which directly learn node embedding vectors using various local structures, like random walks in DeepWalk [87] and node2vec [28], and 1st- and 2nd-order proximity in LINE [111]. In contrast, GNNs do not directly learn node embedding vectors. They instead learn an inductive aggregation function [46, 31, 119, 140, 132] which can be generalized to unseen nodes or even new graphs in the same feature space. Typical GNNs follow a message passing framework, where each node receives and aggregates messages (*i.e.*, node features or embeddings) from its neighboring nodes recursively in multiple layers. In other words, GNNs are capable of not only encoding graph structures, but also preserving node features.

Most of these graph representation methods focus on static graphs with structures frozen in time. However, real-world graphs often present complex dynamics that evolve continuously in time. For instance, in social networks, burst events often rapidly change the short-term social interaction pattern, while on an e-commerce user-item graph, long-term user preferences may drift as new generations of product emerge. More precisely, in a temporal graph [53], the temporal evolution arises from the chronological formation of links between nodes. As illustrated in Fig. 4.1, the toy graph evolving from time t_1 through t_3 can be described by a series of triple $\{(A, B, t_1), (B, C, t_1), (C, D, t_2), (C, E, t_2), (B, C, t_3), \ldots\}$, where each triple (i, j, t) denotes a link formed between nodes i and j at time t. Hence, the prediction of future links depends heavily on the dynamics embodied in the

historical link formation [97]. In this paper, we investigate the important problem of *temporal graph representation learning*, in which we learn representations that evolve with time on a graph. In particular, we treat the formation of a link at a specific time as an *event*, and a graph evolves or grows continuously as more events are accumulated [37].

Prior work. In general, the requirement for temporal graph representation learning is that the learned representations must not only preserve graph structures and node features, but also reflect its topological evolution. However, this goal is non-trivial, and it is not until recently that several works on this problem have emerged. Among them, some [17, 57, 27, 157, 95, 84] discretize the temporal graph into a sequence of static graph snapshots to simplify the model. As a consequence, they cannot fully capture the continuously evolving dynamics, for the fine-grained link formation events "in-between" the snapshots are inevitably lost. For continuous-time methods, CTDNE [81] resort to temporal random walks that respect the chronological sequence of the edges; TGAT [137] employs a GNN framework with functional time encoding to map continuous time and self-attention to aggregate temporal-topological neighborhood. However, these methods often fail to explicitly capture the *exciting effects* [160] between sequential events, particularly the influence of historical events on the current events. Nonetheless, such effects can be well captured by temporal point processes, most notably the *Hawkes process* [35, 75], which assumes that historical events prior to time t can excite the process in the sense that future events become more probable for some time after t. This property is desirable for modeling the graph-wide link formation process, in which each link formation is considered an *event* that can be excited by recent events. For example, in social networks, a celebrity who has attracted a large crowd of followers lately (*e.g.*, due to winning a prestigious award) is likely to attract more followers in the near future. However, for temporal graph representation learning, existing Hawkes process-based network embedding methods [160, 71] are inher-

ently transductive. While DyRep [114] presents an inductive framework based on the temporal point process, it addresses a different problem setting of two-time scale with both association and communication events. In our paper, we focus on learning the dynamics of evolving topology, where each event represents the formation of a link.

Challenges and present work. To effectively model the events of link formation on a temporal graph, we propose a Hawkes process-based GNN framework to reap the benefits of both worlds. Previous methods do not employ Hawkes or similar point processes for modeling the exciting effects between events [137], or not use message-passing GNNs for preserving the structures and features of nodes in an inductive manner [160, 71], or neither [81]. More importantly, while the Hawkes process is well suited for modeling the graph-wide link formation process, prior methods fail to examine two open challenges on modeling the events (*i.e.*, link formation), as follows.

Challenge 1: *How do we capture the uniqueness of the events on an individual scale?* Different links are often formed out of different contexts and time periods, causing subtle differences among events. Taking the research collaboration network in Fig. 4.1 as an example, while links are all collaborations, each collaboration can be unique in its own way. For instance, the collaboration between researchers A and B, and that between F and G, could be formed due to different backgrounds and reasons (*e.g.*, they might simply be students of the same advisor who devised the solution together, or they possess complementary skills required in a large multi-disciplinary project). Multiple collaborations can also be formed at different times, such as between B and C at t_1 and t_3, for potentially different reasons. Conventional methods on temporal graphs train one model to fit all events, where different events tend to pull the model in many opposing directions. The resulting model would be overly diffuse with its center of mass around the most frequent patterns among the events, whilst neglecting

many long-tailed patterns covering their individual characteristics. Hence, in this paper, motivated by hypernetworks [29, 86, 130], we learn an *event prior*, which only encodes the general knowledge of link formation. This event prior can be further specialized in an event-wise manner to fit the individual characteristics of events.

CHALLENGE 2: *How do we govern the occurrence of events on a collective scale?* While events exhibit individual characteristics, they are not formed in isolation and related events often manifest collective characteristics. Events sharing a common node can be "constrained as a collection" due to the common influence from their shared node. That is, the collection of events of each node should match the arrival rate of the node, which we call node dynamics. Of course, for two different nodes, each would have its own event collection, and each collection should match different arrival rates of the two nodes. As shown in Fig. 4.1, researchers A and C have different tendency to form a collaboration with others at time t_3, with C being more active in seeking collaborations. Moreover, as the graph evolves from time t_1 to t_3, researcher C's tendency in collaborating with others also evolves to become higher (*e.g.*, due to C's growing reputation). In other words, the events stemming from a common node are collectively governed by the dynamics of the node as a function of time. Hence, we formulate the notion of *node dynamics* to model the collective characteristics of the events from the same node. Intuitively, integrating the node dynamics provides a regularizing mechanism beyond individual events, to ensure that the events from a node, as a collection, conform to the continuous evolution of the node. Despite the importance of node dynamics, it has not been explored in temporal graph representation learning.

Contributions. We propose TREND, a novel framework for temporal graph representation learning driven by TempoRal Event and Node Dynamics. TREND is built upon a Hawkes process-based GNN, and presents a few major advantages. Firstly, owning to its GNN architecture, it is inductive in nature, *i.e.*, able to

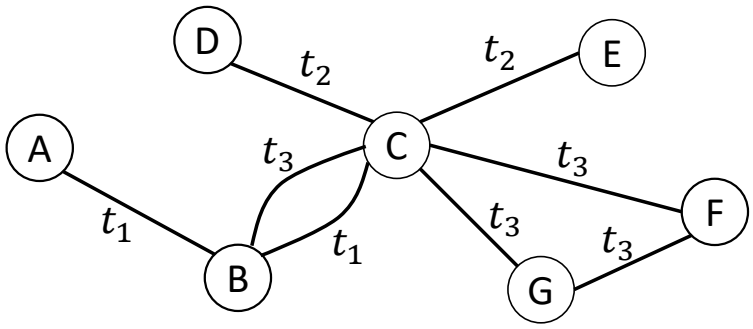


Figure 4.1: Toy temporal graph for research collaborations that evolves through time $t_1, t_2, t_3, \cdots$. Each node is a researcher and each link is a collaboration between researchers formed at a specific time.

handle new nodes at test time. Secondly, owning to the adoption of the Hawkes process, it maps the graph-wide link formation process to capture a holistic view of the temporal evolution. Thirdly, TREND integrates both the event and node dynamics to respectively capture the individual and collective characteristics of events, which drives a more precise modeling of the link formation process.

In summary, our work encompasses the following contributions. (1) For the first time in temporal graph representation learning, we recognize the importance of modeling the events at an individual and collective scale, and formulate them as event and node dynamics. (2) We propose a novel framework called TREND with both event and node dynamics to more precisely model events under a Hawkes process-based GNN. On one hand, the event dynamics learns an adaptable event prior to capture the uniqueness of events individually. On the other hand, the node dynamics regularizes the events at the node level to govern their occurrences collectively. (3) We conduct extensive experiments on four real-world datasets, which demonstrate the advantages of TREND.

4.2 Preliminaries

In this section, we first present the problem of temporal graph representation learning, and then introduce a brief background on the Hawkes process.

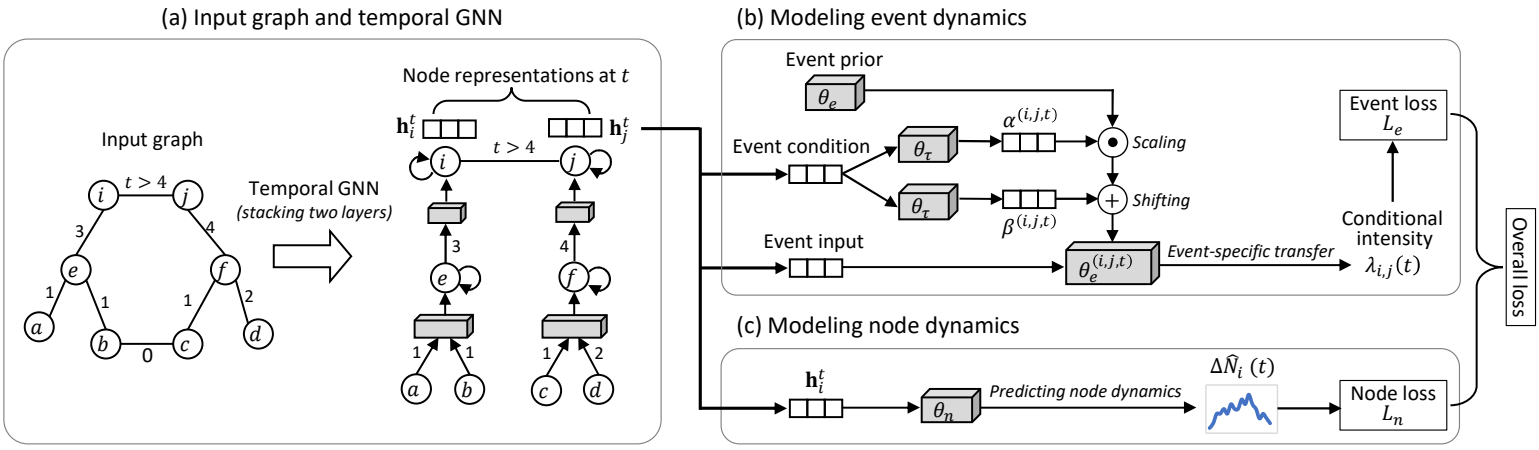


Figure 4.2: Overall framework of TREND, which integrates event and node dynamics in a Hawkes process-based GNN.

Temporal Graph Representation Learning

A *temporal graph* $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathcal{T}, \mathbf{X})$ is defined on a set of nodes $\mathcal{V}$, a set of edges $\mathcal{E}$, a time domain $\mathcal{T}$ and an input feature matrix $\mathbf{X} \in \mathbb{R}^{|\mathcal{V}| \times d_0}$. Each node has a d_0-dimensional input feature vector corresponding to one row in $\mathbf{X}$. An *event* is a triple (i, j, t) denoting the formation of an edge $(i, j) \in \mathcal{E}$ (also called a link) between node $i \in \mathcal{V}$ and node $j \in \mathcal{V}$ at time $t \in \mathcal{T}$. Alternatively, a temporal graph can be defined as a chronological series of events $\mathcal{I} = \{(i, j, t)_m : m = 1, 2, \ldots, |\mathcal{E}|\}$. Note that two nodes may form a link more than once at different times. Thus, there may be two events (i, j, t_1) and (i, j, t_2) such that $t_1 \neq t_2$. Besides, in this work, we only consider the growth of temporal graph, and make deletions of node and edge as future work.

We study the problem of inductive temporal graph representation learning. Specifically, given $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathcal{T}, \mathbf{X})$, we aim to learn a parametric model $\Phi(\cdot; \theta)$ with parameters θ, such that Φ maps any node in the same feature space of $\mathbf{X}$ at any time to a representation vector. That is, $\Phi : \mathcal{V}' \times \mathcal{T} \to \mathbb{R}^d$, where $\mathcal{V} \subset \mathcal{V}'$. The set difference $\mathcal{V}' \setminus \mathcal{V}$ consists of new nodes in the same feature space of $\mathbf{X}$ that may appear at a future time. Such a model Φ is apparently inductive given the ability to handle new nodes.

Hawkes Process

A Hawkes process [35] is a stochastic process that can be understood as counting the number of events up to time t. Its behavior is typically modeled by a *conditional intensity* function $\lambda(t)$, the rate of event occurring at time t given the past events. A common formulation of the conditional intensity [160, 71] is given by

$$\lambda(t) = \mu(t) + \int_{-\infty}^{t} \kappa(t-s) dN(s), \tag{4.1}$$

where $\mu(t)$ is the base intensity at time t, κ is a kernel function to model the time decay effect of historical events on the current event (usually in the shape of an exponential function), and $N(t)$ is the number of events occurred until t. Since the Hawkes process is able to model the exciting effects between events to capture the influence of historical events holistically, it is well suited for modeling the graph-wide link formation process in a temporal graph.

4.3 Proposed Approach

In this section, we present a novel framework for temporal graph representation learning called TREND.

Overview of TREND

Building upon a Hawkes process-based GNN, the proposed TREND is able to model the graph-wide link formation process in an inductive manner. More importantly, it integrates event and node dynamics into the model to fully capture the individual and collective characteristics of events.

The overall framework of TREND is shown in Fig. 4.2. First of all, in

Fig. 4.2(a), an input temporal graph undergoes a temporal GNN aggregation in multiple layers, whose output representations serve as the input for modeling event and node dynamics. The GNN layer aggregates both the self-information and historical neighbors' information, which are building blocks to materialize the conditional intensity in the Hawkes process. Next, we model event dynamics to capture the individual characteristics of events, as shown in Fig. 4.2(b). We perform an event-conditioned, learnable transformation to adapt the event prior to the input event, resulting in an event-specific transfer function to generate the conditional intensity in the Hawkes process. Moreover, we model node dynamics to capture the collective characteristics of events at the node level, as shown in Fig. 4.2(c). We build an estimator to predict the node dynamics across nodes and times, which governs the behavior of events occurring on the same node. At last, we integrate the event and node losses to jointly optimize event and node dynamics.

Hawkes process-based GNN

We first introduce a Hawkes process-based GNN framework, which is to be further integrated with event and node dynamics later.

Hawkes process on temporal graph. In the context of temporal graph, the Hawkes process is able to model the graph-wide link formation process. Specifically, whether nodes i and j form a link at t, can be quantified by the conditional intensity of the event,

$$\lambda_{i,j}(t) = \mu_{i,j}(t) + \sum_{(i,j',t')\in\mathcal{H}_i(t)} \gamma_{j'}(t')\kappa(t-t') + \sum_{(i',j,t')\in\mathcal{H}_j(t)} \gamma_{i'}(t')\kappa(t-t'). \tag{4.2}$$

In particular, $\mu_{i,j}(t)$ is the base rate of the event that i and j form a link at time t,

which is not influenced by historical events on i or j. $\mathcal{H}_i(t)$ is the set of historical events on i w.r.t. time t, *i.e.*, $\mathcal{H}_i(t) = \{(i, j', t') \in \mathcal{I} : t' < t\}$, and we call j' a historical neighbor of i. $\gamma_{j'}(t')$ represents the amount of excitement induced by a historical neighbor j' at t' on the current event. Note that we are treating each link as undirected, and thus the current event is influenced by historical neighbors of both nodes i and j. In the case of directed link, we can modify Eq. (4.2) by keeping only one of the two summation terms. $\kappa(\cdot)$ is a kernel function to capture the time decay effect w.r.t. t, defined in the form of an exponential function as $\kappa(t - t') = \exp(-\delta(t - t'))$, where $\delta > 0$ is a learnable scalar to control the rate of decay.

Next, temporal graph representations are used to materialize the conditional intensity in Eq. (4.2). Given the temporal representations of nodes i, j at time t, denoted $\mathbf{h}_i^t, \mathbf{h}_j^t$ respectively, the conditional intensity can be generated from a *transfer function* f [114, 75], *i.e.*,

$$\lambda_{i,j}(t) = f(\mathbf{h}_i^t, \mathbf{h}_j^t), \tag{4.3}$$

which should meet the following criteria. (1) The input representations $\mathbf{h}_i^t, \mathbf{h}_j^t$ should be derived from not only their inherent self-information, but also their historical neighbors' information. While the self-information is the basis of the base intensity $\mu_{i,j}(t)$, historical neighbors are crucial to model the excitement induced by historical events. (2) The output of the transfer function f must be positive, since it represents an intensity. We will discuss the choice of the transfer function in Sect. 4.3.

Temporal GNN layer. We materialize the temporal representations in Eq. (4.3) using GNNs, owning to their inductive nature and superior performance. Based on the message passing scheme, each node receives, aggregates and maps messages (*e.g.*, features or embeddings) from its neighboring nodes recursively, in multiple

layers. Here we present a temporal formulation of GNN in consideration of the representational criteria listed above, so that the learned temporal representations can be used to materialize the conditional intensity. Formally, let $\mathbf{h}_i^{t,l} \in \mathbb{R}^{d_l}$ be the d_l-dimensional embedding vector of node i at time t in the l-th layer, which is computed by

$$\mathbf{h}_i^{t,l} = \sigma\Big(\underbrace{\mathbf{h}_i^{t,l-1}\mathbf{W}_{\text{self}}^{l}}_{\substack{\text{self-information (for base intensity)}}} + \underbrace{\sum_{(i,j',t')\in\mathcal{H}_i(t)} \mathbf{h}_{j'}^{t',l-1}\mathbf{W}_{\text{hist}}^{l}\tilde{\kappa}_i(t-t')}_{\substack{\text{historical neighbors' information (for excitement by historical events)}}}\Big), \tag{4.4}$$

where σ is an activation function (*e.g.*, ReLU), $\mathbf{W}_{\text{self}}^{l} \in \mathbb{R}^{d_{l-1}\times d_l}$ is a learnable weight matrix to map the embedding of node i itself from the previous layer, $\mathbf{W}_{\text{hist}}^{l} \in \mathbb{R}^{d_{l-1}\times d_l}$ is another learnable weight matrix to map the embeddings of historical neighbors, and $\tilde{\kappa}_i(t-t')$ captures the time decay effect based on the time kernel with softmax, which is given by $\tilde{\kappa}_i(t-t') = \frac{\kappa(t-t')}{\sum_{(i,j'',t'')\in\mathcal{H}_i(t)}\kappa(t-t'')}$.

In other words, the temporal representation of a node is derived by receiving and aggregating messages of itself and the historical neighbors from the previous layer. The self-information is responsible for capturing the base intensity, while the historical neighbors' information is responsible for capturing the excitement induced by historical events. To enhance the representational capacity, we stack multiple temporal GNN layers. In the first layer, the node message can be initialized by the input node features $\mathbf{X}$; in the last layer, the output temporal representation is denoted as $\mathbf{h}_i^t \in \mathbb{R}^d$ for node i at time t. The collection of parameters of all the layers is $\theta_g = \{\mathbf{W}_{\text{self}}^{l}, \mathbf{W}_{\text{hist}}^{l} : l = 1, 2, \ldots\}$.

Connection to conditional intensity. A well chosen transfer function f, taking the temporal representations as input, is equivalent to the conditional intensity of the Hawkes process in Eq. (4.2). We formally show the connection in Ap-

pendix A.2.

Modeling Event Dynamics

The key to materialize the conditional intensity is to fit a transfer function f on top of the temporal GNN layers. Previous studies on Hawkes process employ the softplus function or its variant [75, 114] as the transfer function. To ensure that f is well-fit to the conditional intensity, our first proposal is to instantiate f as a learnable function. More specifically, we use a fully connected layer (FCL). That is,

$$\lambda_{i,j}(t) = f(\mathbf{h}_i^t, \mathbf{h}_j^t) = \text{FCL}_e((\mathbf{h}_i^t - \mathbf{h}_j^t)^{\circ 2}; \theta_e), \tag{4.5}$$

where θ_e denotes the parameters of the fully connected layer FCL_e. Note that the input to FCL_e can be in various forms, such as the concatenation of $\mathbf{h}_i^t$ and $\mathbf{h}_j^t$, or the element-wise square (denoted by $^{\circ 2}$) of the difference between them. We use the latter in our formulation, which tends to achieve better empirical performance. A potential reason is that the differential representation is a good predictor of whether an event occurs between the two nodes. Lastly, FCL_e employs a sigmoid activation, to ensure the transfer function is positive.

Meanwhile, we recognize that each event can be unique in its own way, as different links are often formed out of different contexts and time periods. To precisely capture the uniqueness of events on an individual scale (Challenge 1), a global model in Eq. (4.5)—our first proposal—becomes inadequate. To be more specific, in a conventional one-model-fits-all approach, given the diversity in events, the learned model tends to converge around the most frequent patterns among events, while leaving long-tailed patterns that reflect the individual characteristics of events uncovered. On the other hand, training a large number of models for different kinds of events can easily cause overfiting and scalability issues, not to

mention that it is difficult to categorize events in the first place. Inspired by meta-learning, particularly the line of work on *hypernetworks* [29, 86], we address the dilemma by learning an *event prior*, which can be quickly adapted to a unique model for each event, without the need to train a large number of models.

Event prior and adaptation. In our first proposal in Eq. (4.5), we learn a global model for all events, *i.e.*, the same θ_e parameterizes a global FCL_e as the transfer function for all events. To deal with the diversity of events, we propose to learn an event prior θ_e that aims to encode the general knowledge of link formation, such that it can be quickly specialized to fit the individual characteristics of each event. In other words, θ_e does not directly parameterize FCL_e used as the transfer function; instead, it will be adapted to each event via a learnable transformation model first, and the adapted parameters will instantiate an event-specific FCL_e as the transfer function for each event. This approach is a form of hypernetwork [29], in which a secondary neural network is used to generate the parameters of the primary network. This means the parameters of the primary network can flexibly adapt to its input, as opposed to conventional models whose parameters are frozen once training is completed. In our context, the primary network is FCL_e for the transfer function, and the secondary network is the learnable transformation model.

Particularly, during the adaptation, the event prior θ_e will transform into event (i, j, t)-specific parameters $\theta_e^{(i,j,t)}$ as follows.

$$\theta_e^{(i,j,t)} = \tau(\theta_e, \mathbf{h}_i^t \| \mathbf{h}_j^t; \theta_\tau), \tag{4.6}$$

which (1) is parameterized by θ_τ; (2) is conditioned on event-specific temporal representations of nodes i, j, namely, $\mathbf{h}_i^t \| \mathbf{h}_j^t$ where $\|$ is the concatenation operator; (3) outputs adapted parameters $\theta_e^{(i,j,t)}$ by transforming the event prior θ_e conditioned on $\mathbf{h}_i^t \| \mathbf{h}_j^t$. The transformed $\theta_e^{(i,j,t)}$ will further parameterize FCL_e as the

transfer function, and materialize the conditional intensity below.

$$\lambda_{i,j}(t) = \mathrm{FCL_e}((\mathbf{h}_i^t - \mathbf{h}_j^t)^{\circ 2}; \theta_e^{(i,j,t)}). \tag{4.7}$$

In the following, we will materialize the transformation model τ and its parameters θ_τ in detail.

Learnable transformation. We consider Feature-wise Linear Modulation (FiLM) [86], which employs affine transformations including scaling and shifting on the event prior, conditioned on event-specific temporal representations. Compared with gating [139] which can only adjust the parameters in a diminishing way, FiLM is more flexible in adjusting the parameters and can be conditioned on arbitrary input. Specifically, we employ fully connected layers to generate the scaling operator $\alpha^{(i,j,t)}$ and shifting operator $\beta^{(i,j,t)}$, conditioned on the event-specific input $\mathbf{h}_i^t \| \mathbf{h}_j^t$, as follows.

$$\alpha^{(i,j,t)} = \sigma\big((\mathbf{h}_i^t \| \mathbf{h}_j^t)\mathbf{W}_\alpha + \mathbf{b}_\alpha\big), \tag{4.8}$$

$$\beta^{(i,j,t)} = \sigma\big((\mathbf{h}_i^t \| \mathbf{h}_j^t)\mathbf{W}_\beta + \mathbf{b}_\beta\big), \tag{4.9}$$

where $\mathbf{W}_\alpha, \mathbf{W}_\beta \in \mathbb{R}^{2d \times d_{\theta_e}}$ and $\mathbf{b}_\alpha, \mathbf{b}_\beta \in \mathbb{R}^{d_{\theta_e}}$ are learnable weight matrices and bias vectors of the fully connected layers, in which d is the dimension of node representation and d_{θ_e} is the total number of parameters in the event prior θ_e. The output $\alpha^{(i,j,t)}, \beta^{(i,j,t)} \in \mathbb{R}^{d_{\theta_e}}$ are both d_{θ_e}-dimensional vectors, which represent the scaling and shifting operations of the transformation model τ. They are used to transform the event prior into event (i, j, t)-specific parameters by element-wise scaling and shifting, given by

$$\theta_e^{(i,j,t)} = \tau(\theta_e, \mathbf{h}_i^t \| \mathbf{h}_j^t; \theta_\tau) = (\alpha^{(i,j,t)} + \mathbf{1}) \odot \theta_e + \beta^{(i,j,t)}, \tag{4.10}$$

where $\odot$ stands for element-wise multiplication, and $\mathbf{1}$ is a vector of ones to ensure

that the scaling factors are centered around one. Note that θ_e contains all the weights and biases of FCL_e, and we flatten it into a d_{θ_e}-dimensional vector.

In summary, the learnable transformation model τ is parameterized by $\theta_\tau = \{\mathbf{W}_\alpha, \mathbf{b}_\alpha, \mathbf{W}_\beta, \mathbf{b}_\beta\}$, *i.e.*, the collection of parameters of the fully connected layers that generate the scaling and shifting operators. Furthermore, τ is also a function of the event condition $\mathbf{h}_i^t \| \mathbf{h}_j^t$, for $\alpha^{(i,j,t)}$ and $\beta^{(i,j,t)}$ are functions of the event condition.

Event loss. Given an event $(i, j, t) \in \mathcal{I}$ that has occurred on the graph, we expect a higher conditional intensity $\lambda_{i,j}(t)$. On the contrary, given an event $(i, j, t) \notin \mathcal{I}$ that does not happen, we expect a lower conditional intensity. Thus, we formulate the event loss based on negative log-likelihood, the optimization of which encourages the conditional intensity of an event to match its occurrence or non-occurrence. Given any event $(i, j, t) \in \mathcal{I}$ that has occurred, its loss is defined as

$$L_e(i, j, t) = -\log(\lambda_{i,j}(t)) - Q \cdot \mathbb{E}_{k \sim P_n} \log(1 - \lambda_{i,k}(t)), \tag{4.11}$$

where we sample a negative node k according to the distribution P_n, so that $(i, k, t) \notin \mathcal{I}$ does not occur, and Q is the number of negative samples for each positive event. As a common practice, P_n is defined on the node degrees, namely, $P_n(v) \propto \deg(v)^{\frac{3}{4}}$ where $\deg(v)$ is the degree of node v.

Modeling Node Dynamics

Different from the event dynamics that captures the individual characteristics of events, node dynamics aims to govern the collective characteristics of events. While events can be individually different, they do not occur in isolation. Particularly, links are formed to connect nodes, which means their behaviors are

collectively bounded by their common nodes. Thus, we propose to govern the collective characteristics of nodes at the node level, to capture the "event tendency" of nodes (CHALLENGE 2)—different nodes have varying tendency to form new links with others, and even the same node would manifest different tendency at different times.

Estimator of node dynamics. More specifically, the node dynamics or the event tendency of a node at time t can be quantified by the number of new events occurring on the node at t, denoted $\Delta N_i(t)$. We build an estimator for node dynamics with a fully connected layer, trying to fit the number of new events on a given node:

$$\Delta \hat{N}_i(t) = \mathrm{FCL}_n(\mathbf{h}_i^t; \theta_n), \tag{4.12}$$

where the input is the temporal representation $\mathbf{h}_i^t$, the output $\Delta \hat{N}_i(t)$ is the predicted number of new events occurring on node i at time t, and θ_n contains the parameters of FCL_n.

Node loss. To ensure that the occurrence of events are consistent with the node dynamics evolving continuously on a temporal graph, we formulate a node loss such that the estimator $\Delta \hat{N}_i(t)$ can accurately reflect the groundtruth dynamics $\Delta N_i(t)$ across all nodes and times. In particular, we adopt the following smooth L_1 loss [24]:

$$L_n(i,t) = \begin{cases} 0.5(\Delta \hat{N}_i(t) - \Delta N_i(t))^2, & |\Delta \hat{N}_i(t) - \Delta N_i(t)| < 1 \\ |\Delta \hat{N}_i(t) - \Delta N_i(t)| - 0.5. & \text{otherwise} \end{cases} \tag{4.13}$$

The smooth L_1 loss can be viewed as a combination of both L_1 loss and L_2 loss. It is less sensitive to outliers than the L_2 loss when the input is large, and it suffers from less oscillations than the L_1 loss when the input is small. In our scenario,

there exist some nodes with a very large number of new links at certain times (*e.g.*, due to burst topics on social networks). To prevent the models from being overly skewed to these nodes, and to simultaneously cater to nodes with only a few links, the smooth L_1 loss is an ideal choice.

Overall Model: TREND

Finally, we integrate both event and node dynamics into a Hawkes process-based GNN model, resulting in our proposed model TREND. Consider the set of training events $\mathcal{I}^{\text{tr}} = \{(i, j, t) \in \mathcal{I} : t \leq t^{\text{tr}}\}$, *i.e.*, all events on the graph up to time t^{tr}. (New events after time t^{tr} can be reserved for testing.) We optimize all parameters $\Theta = (\theta_g, \theta_e, \theta_\tau, \theta_n)$ jointly, including those of the temporal GNN layers θ_g, the event prior θ_e, the transformation model θ_τ and the estimator of node dynamics θ_n, based on the following loss:

$$\arg\min_{\Theta} \sum_{(i,j,t)\in\mathcal{I}^{\text{tr}}} L_e + \eta_1 L_n + \eta_2(\|\alpha^{(i,j,t)}\|_2^2 + \|\beta^{(i,j,t)}\|_2^2), \tag{4.14}$$

where (1) $\eta_1 > 0$ is a hyper-parameter controlling the contribution of node dynamics to our model TREND; (2) the L_2 regularization on $\alpha^{(i,j,t)}$ and $\beta^{(i,j,t)}$ constrains the scaling and shifting operators, as it is preferred that the scaling is close to 1 and the shifting is close to zero, in order to avoid overfitting to individual events; (3) $\eta_2 > 0$ is a hyperparameter controlling the effect of the L_2 regularizor.

For implementation, we perform optimization over batches of training events using a gradient-based optimizer. The overall training procedure of TREND is outlined in Appendix A.3. It can be seen that the training time complexity is $O(K|\mathcal{I}^{\text{tr}}|h^l Q))$, where K is the number of epochs, $|\mathcal{I}^{\text{tr}}|$ is the number of training events, h is the number of historical neighbors in temporal GNN aggregation, l is the number of temporal GNN layers, and Q is the number of negative samples per training event. Note that Q and l are small constants (typically 5 or less), and h

Table 4.1: Statistics of datasets.

Dataset	CollegeMsg	cit-HepTh	Wikipedia	Taobao
# Events	59,835	51,315	157,474	4,294,000
# Nodes	1,899	7,577	8,227	1,818,851
# Node features	–	128	172	128
Multi-edge?	Yes	No	Yes	Yes
New nodes in testing	22.79%	100%	7.26%	23.46%

can also be a constant when employing a commonly used neighborhood sampling approach [31]. Hence, the complexity can be regarded as linear in the number of events or temporal edges on the graph.

4.4 Experiments

We conduct extensive experiments to evaluate TREND, with comparison to state-of-the-art baselines and in-depth model analysis.

Experimental Setup

Datasets. Four public temporal networks are used in our experiments, as summarized in Tab. 4.1. Note that "new nodes in testing" refers to the ratio of testing events containing at least one new node not seen during training. (1) **CollegeMsg** [83]: an online social network in which an event is a user sending another user a private message. (2) **cit-HepTh** [50]: a citation graph about high energy physics theory in which an event is a paper citation. (3) **Wikipedia** [49]: a Wikipedia graph in which an event is a user editing a page. (4) **Taobao** [18]: an e-commerce platform in which an event is a user purchasing an item. More dataset details are presented in Appendix A.4.

Prediction tasks. We adopt *temporal link prediction* as our main task. We

Table 4.2: Performance of temporal link prediction by TREND and the baselines, in percent, with 95% confidence intervals.

In each column, the best result is **bolded** and the runner-up is underlined. Improvement by TREND is calculated relative to the best baseline. "-" indicates no result obtained due to out of memory issue; * indicates that our model significantly outperforms the best baseline based on two-tail t-test ($p < 0.05$).

	CollegeMsg	cit-HepTh	Wikipedia	Taobao
	Accuracy	Accuracy	Accuracy	Accuracy
DeepWalk	66.54±5.36	51.55±0.90	65.12±0.94	53.59±0.18
Node2vec	65.82±4.12	65.68±1.90	75.52±0.58	52.74±0.33
VGAE	65.82±5.68	66.79±2.58	66.35±1.48	55.97±0.22
GAE	62.54±5.11	69.52±1.10	68.70±1.34	58.13±0.15
GraphSAGE	58.91±3.67	70.72±1.96	72.32±1.25	60.74±0.18
CTDNE	62.55±3.67	49.42±1.86	60.99±1.26	51.64±0.32
EvolveGCN	63.27±4.42	61.57±1.53	71.20±0.88	-
GraphSAGE+T	69.09±4.91	67.80±1.27	57.93±0.53	67.05±0.23
TGAT	58.18±4.78	78.02±1.93	76.45±0.91	70.07±0.59
HTNE	73.82±5.36	66.70±1.80	77.88±1.56	59.03±0.17
MMDNE	73.82±5.36	66.28±3.87	79.76±0.89	58.24±0.10
TREND	**74.55**±1.95	**80.37***±2.08	**83.75***±1.19	**78.56***±0.17
(improv.)	(+0.99%)	(+3.01%)	(+5.00%)	(+12.11%)
	CollegeMsg	cit-HepTh	Wikipedia	Taobao
	F1	F1	F1	F1
DeepWalk	67.86±5.86	50.39±0.98	64.25±1.32	56.67±0.12
Node2vec	69.10±3.50	66.13±2.15	75.61±0.52	54.86±0.32
VGAE	68.73±4.49	67.27±2.84	68.04±1.18	59.80±0.16
GAE	66.97±3.22	70.28±1.33	69.74±1.43	61.40±0.07
GraphSAGE	60.45±4.22	71.27±2.41	73.39±1.25	61.61±0.20
CTDNE	65.56±2.34	44.23±3.92	62.71±1.49	43.99±0.38
EvolveGCN	65.44±4.72	62.42±1.54	73.43±0.51	-
GraphSAGE+T	69.41±5.45	69.12±1.12	63.41±0.91	67.69±0.17
TGAT	57.23±7.57	78.52±1.61	76.99±1.16	71.31±0.18
HTNE	74.24±5.36	67.47±1.16	78.09±1.40	60.34±0.19
MMDNE	74.10±3.70	66.70±3.39	79.87±0.95	59.04±0.16
TREND	**75.64**±2.09	**81.13***±1.92	**83.86***±1.24	**80.67***±0.15
(improv.)	(+1.89%)	(+3.32%)	(+4.99%)	(+13.12%)

evaluate a model by predicting future links based on historical links [97]. Given a temporal graph, we split the events into training and testing. Specifically, the set of training events $\mathcal{I}^{\text{tr}} = \{(i, j, t) \in \mathcal{I} : t \leq t^{\text{tr}}\}$ consists of all events up to time t^{tr}. The remaining events after time t^{tr}, denoted by the set $\mathcal{I}^{\text{te}} = \mathcal{I} \setminus \mathcal{I}^{\text{tr}}$, is reserved for testing. Given a candidate triple (i, j, t) for some $t > t^{\text{tr}}$, the objective is to predict whether a link between nodes i and j is formed at the given future time t, *i.e.*, if $(i, j, t) \in \mathcal{I}^{\text{te}}$. Note that our model can perform temporal link prediction between all nodes, including new nodes not seen during training, due to its inductive nature. Specifically, in testing, we first generate temporal node representations based on the trained model, and feed them to a downstream logistic regression classifier to predict if a candidate triple is positive or negative. The classifier is trained using a 80%/20% train/test split on the testing events, and repeated for five different splits. More details are given in Appendix A.5.

We further adopt a secondary task of *temporal node dynamics prediction*. While the training and testing events follow the same setup of the main task, we aim to predict the number of new neighbors of a node i at a specific future time $t > t^{\text{tr}}$. Similarly, the first step in testing is to generate temporal node representations based on the trained model, which are then fed into a downstream linear regression model. To train the regression model, we randomly split the nodes of the testing events into 80%/20% train/test split.

Settings of TREND. For the temporal GNN, we employ two layers with a ReLU activation. The hidden layer dimension is set to 16 on all datasets. The output dimension is set to 32 on CollegeMsg, 16 on cit-HepTh and 128 on Wikipedia and Taobao, based on the size and complexity of the graph. The transfer function FCL_e employs a sigmoid activation, the estimator of node dynamics FCL_n uses a ReLU activation, and $\alpha^{(i,j,t)}, \beta^{(i,j,t)}$ both employ a LeakyReLU. The number of negative samples per positive event is set to $Q = 1$. For the final loss function in Eq. (4.14), the coefficient of node loss is set to $\eta_1 = 0.1$ on Taobao and $\eta_1 = 0.01$

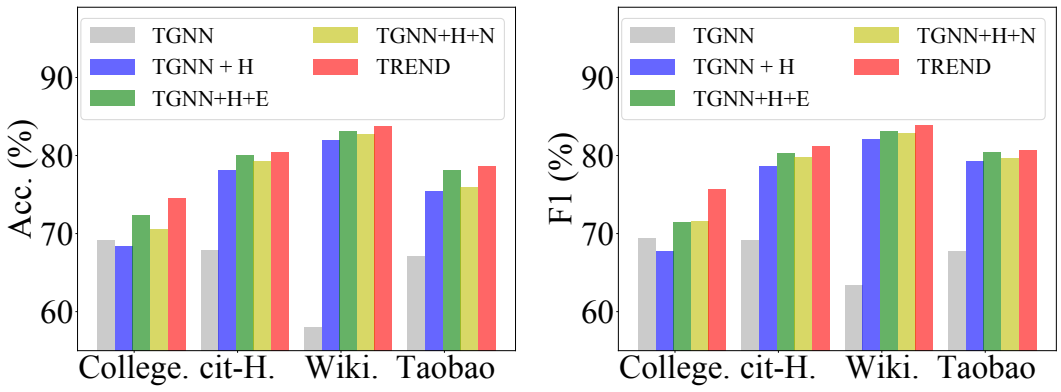
on other datasets, whereas the coefficient of L_2 regularizer is set to $\eta_2 = 0.001$ on CollegeMsg and cit-HepTh, $\eta_2 = 0.01$ on Taobao, and $\eta_2 = 1$ on Wikipedia. Note that we will present an analysis on the impact of the hyperparameters Q, η_1 and η_2 in Sect. 4.4. Lastly, we use the Adam optimizer with the learning rate 0.001.

Baselines. We compare TREND with a competitive suit of baselines from three categories. (1) *Static approaches*: DeepWalk [87], Node2vec [28], VGAE [45], GAE [45] and GraphSAGE [31]. They train a model or node embedding vectors on the static graph formed from the training events, without considering any temporal information. (2) *Temporal approaches*: CTDNE [81], EvolveGCN [84], GraphSAGE+T [31] and TGAT [137]. They train a model or node embedding vectors on the temporal graph formed from the training events. Note that GraphSAGE+T is a temporal extension of GraphSAGE implemented by us, in which the time decay effect is incorporated into the aggregation function. (3) *Hawkes process-based approaches*: HTNE [160] and MMDNE [71]. They similarly train node embedding vectors on the temporal graph formed from the training events. However, they leverage the node representations to model the conditional intensity of events based on the Hawkes process. More baseline descriptions are in Appendix A.6.

Temporal Link Prediction

In Tab. 4.2, we compare the performance of TREND with the baselines on the main task. In general, our method performs the best among all methods, demonstrating the benefits of event and node dynamics. We make two further observations.

First, among static methods, we can see that GNN-based methods (VGAE, GAE and GraphSAGE) tend to perform better, as they are inductive in nature, and their message passing scheme is able to integrate both node features and graph structures. On the other hand, DeepWalk and Node2vec are transductive, which

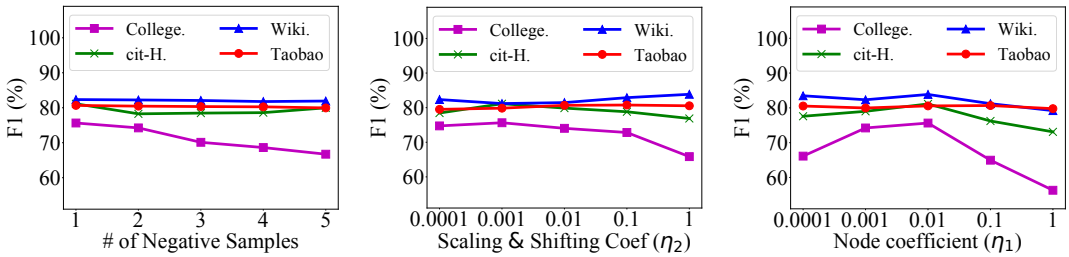


(a) Accuracy (b) F1

Figure 4.3: Effect of main components.

cannot directly extend to new nodes in testing. In our experiments, the embedding vector of new nodes are randomly initialized for tranductive methods, and thus their performance can be poor when dealing with new nodes. One exception is on the CollegeMsg dataset, where there is no node features and one-hot encoding of node IDs are used instead. In this case, GNN-based methods lose the inductive capability and do not outperform transductive methods.

Second, temporal approaches are generally superior to static approaches, showing the importance of temporal information. Among the three GNN-based approaches (EvolveGCN, GraphSAGE+T and TGAT), EvolveGCN often performs the worst. The reason is that EvolveGCN is based on discrete snapshots, which inevitably suffers from some loss in the temporal evolution. Moreover, the Hawkes process-based approaches (HTNE and MMDNE) achieve strong performance, demonstrating that the Hawkess process is ideal for modeling the temporal evolution on graphs. Unfortunately, they are transductive and thus do not outperform GraphSAGE-T and TGAT on cit-HepTh and Taobao where there are a large proportion of new nodes in testing. Besides, we can see that TREND performs much better on Taobao than on other datasets. A potential reason is that Taobao is the biggest graph having more "diversity" in events, such that the adaptation of event prior becomes more crucial and can lead to larger performance gain.



(a) Negative samples (b) Regularization (c) Node coefficient

Figure 4.4: Hyperparameter sensitivity.

Ablation Study

To understand the contribution of each component in TREND, we study the following ablated models on the task of temporal link prediction. (1) *TGNN*, which only stacks two temporal GNN layers and optimizes the inner product of node pairs; (2) *TGNN+H*, which adds the global transfer function for the Hawkess process in Eq. (4.5) to *TGNN*; (3) *TGNN+H+E* and *TGNN+H+N*, which further model the event and node dynamics on top of *TGNN+H*, respectively. Note that *TGNN+H+E* uses the event-specific transfer function in Eq. (4.7).

As shown in Fig. 4.3, the performance generally increases when we gradually add more components to *TGNN*. This shows that every component is useful for modeling temporal graphs. Note that *TGNN+H+E* typically outperforms *TGNN+H+N*, since the event dynamics directly deals with individual events while the node dynamics only works at the node level. Nevertheless, when integrating both event and node dynamics, the full model TREND achieves the best performance, showing that it is important to jointly model both event and node dynamics.

Hyperparameter Study

Here we present a sensitivity analysis of the hyperparameters.

Negative sampling size. As shown in Fig. 4.4(a), generally speaking, the performance of TREND does not improve with more negative samples. On all datasets, it is robust to choose just one negative sample for each positive event for efficiency.

Regularization on scaling and shifting. To prevent overfitting, the event-conditioned transformations are regularized to prevent excessive scaling or shifting. The regularization is controlled by the coefficient η_2, and we study its effect in Fig. 4.4(b). The performance is quite stable over different values of η_2, although smaller values in the range [0.0001, 0.01] tend to perform better. Worse performance can be observed on larger values, which implies very little scaling and shifting similar to removing the event-conditioned transformation.

Coefficient for node loss. We vary η_1, which controls the weight of the node loss, and study its impact in Fig. 4.4(c). We observe that the performance is suboptimal if η_1 is too small or large, and the performance of TREND is generally robust when η_1 is round 0.01. This shows that a well balanced node and event loss can improve the stability and performance.

Temporal Node Dynamics Prediction

Finally, we evaluate the task of temporal node dynamics prediction.

We report the mean absolute error (MAE) between the predicted value $\Delta\hat{N}_i(t)$ and the groundtruth $\Delta N_i(t)$ in Tab. 4.3. The results show that TREND consistently achieves the smallest MAE on all four datasets, which demonstrate its versatility beyond temporal link prediction, and that the estimator of node dynamics works well as intended.

Table 4.3: Performance of node dynamics prediction.

Model	CollegeMsg	cit-HepTh	Wikipedia	Taobao
CTDNE	10.0636	3.0173	7.3265	0.5789
EvolveGCN	3.1964	2.5610	6.8651	-
GraphSAGE+T	21.9444	<u>2.2421</u>	<u>5.9231</u>	<u>0.5505</u>
TGAT	<u>2.6903</u>	2.8094	7.7737	0.5550
HTNE	12.3587	3.2781	6.8860	0.5749
MMDNE	8.0555	2.7456	6.9552	0.5643
TREND	**2.3549**	**2.2066**	**5.9140**	**0.5491**

4.5 Conclusion

In this work, we studied the problem of temporal graph representation learning. Specifically, we proposed TREND, a novel framework for temporal graph representation learning, driven by event and node dynamics on a Hawkes process-based GNN. TREND is inductive and able to capture a holistic view of the link formation process. More importantly, it integrates both the event and node dynamics to respectively capture the individual and collective characteristics of events, for a more precise modeling of the temporal evolution. Finally, we conducted extensive experiments on four real-world datasets and demonstrated the superior performance of TREND.

As for the limitation of TREND is that TREND only models the process of new edge addition in the future, but ignores the process of edge deletion in the future. However, it is common that there are edge deletions in the future, in temporal graphs. One possible solution is to introduce one more transfer function to models the process of edge deletion.

Chapter 5

Generalizing GNNs across Tasks

The previous chapter studies temporal graph representation learning and solves the temporal link prediction problem. In this chapter, we study the problem of low-resource text classification, with no or few labeled samples, which presents a serious concern for supervised learning. Besides, when there are lots of downstream tasks, developing parameter and time efficient tuning method holds significant practical implications. Motivated by this, we leverage self-supervised learning, which extracts informative knowledge through well-designed pretext tasks, to jointly pre-train a graph-text model, augmenting the performance under the low-resource setting. For downstream node (text) classification tasks, we resort to parameter and time efficient prompting to achieve good performance on various classification tasks. This work is presented in :

- Zhihao Wen, Yuan Fang. Augmenting Low-Resource Text Classification with Graph-Grounded Pre-training and Prompting. Accepted by *the 46th International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR '23).*

5.1 Introduction

Text classification is a fundamental research problem with many important applications in information retrieval. For example, predicting the topics of online articles can help readers easily search and navigate within the website or portal [74], and classifying the category of e-commerce product descriptions enables businesses to structure their inventory efficiently and improve users' search experience [138]. Recent advances in natural language processing (NLP) have achieved remarkable success for text classification, especially when there are large-scale and high-quality labeled data. However, data labeling is often costly and time-consuming, making low-resource classification, in which no or few labeled samples are available, an appealing alternative.

To address low-resource text classification, one approach is to utilize pre-trained language models (PLM) [43, 90], many of which are based on the transformer architecture [115] due to its powerful ability of encoding texts. A PLM can be adapted to different tasks by *fine-tuning* the model parameters to task-specific objectives. While the "pre-train, fine-tune" paradigm requires fewer labeled data than traditional supervised learning, it suffers from two drawbacks. Firstly, state-of-the-art PLMs typically have huge model size, *e.g.*, GPT-3 has 175 billion parameters [6], which makes fine-tuning prohibitively expensive [58]. Secondly, fine-tuning still needs a reasonable amount of labeled data due to the gap between pre-training and fine-tuning objectives, and thus struggles with low-resource scenarios including zero- and few-shot classification.

To overcome the problem of pre-training and fine-tuning, *prompting* [6] has been proposed. It uses a natural language instruction or "prompt" to give a hint of the downstream task, whilst freezing the parameters of a large PLM. In other words, no fine-tuning or additional training is required at all for a new task. However, discrete natural language prompts can be difficult to design and may

result in suboptimal performance compared to fine-tuning [52]. More recently, *prompt tuning* [64, 52] formulates a continuous prompt as a learnable embedding, which is optimized during task adaptation without updating the PLM.

Meanwhile, text data are often grounded on some network structure. For instance, online articles are related through a hyperlink/citation network, and e-commerce products are related through a user-item interaction graph. These graph structures reveal rich relationships between article contents or product descriptions, which can be used to augment low-resource text classification. While existing PLMs and prompting do not exploit such relationships, graph neural networks (GNNs) [135] have been effective in learning with graph data. Based on a message-passing architecture, GNNs has shown powerful performance on graphs owing to the ability to integrate both node features and topological structures. Nevertheless, traditional end-to-end training of GNNs heavily relies on abundant task-specific labels, which motivates self-supervised GNNs [133] using well-designed pretext tasks derived from a label-free graph in a contrastive [118] or generative [39, 41] manner. However, the treatment of text features in GNNs remains rudimentary. A simple bag-of-words representation [142] or aggregation of shallow word embeddings [77] is fed into GNNs as the "initial message", which are further propagated along graph structures. Hence, the modeling of texts is coarse-grained, unable to fully capture the subtle semantic differences and similarities within texts.

Challenges and present work. To overcome the limitations of existing text- and graph-based solutions, we must address two open questions.

Firstly, *how do we capture fine-grained textual semantics, while leveraging graph structure information jointly?* A naïve approach is to use a language model to generate features from raw texts as input, and then train a GNN. However, in this way the texts and graph are only loosely coupled, lacking an explicit pairing

to complement each other. In this paper, we propose graph-grounded contrastive pre-training, to maximize the alignment between text and graph representations based on three types of graph interaction, namely, text-node, text-summary, and node-summary interactions.

Secondly, *how do we augment low-resource text classification given a jointly pre-trained graph-text model?* Instead of following the traditional fine-tuning paradigm, we try to "prompt" our jointly pre-trained graph-text model, from which the most relevant structural and semantic information can be located to improve low-resource classification. Without the need to update a large pre-trained model, prompting is also more efficient than fine-tuning. Specifically, we employ discrete prompts in zero-shot classification and continuous prompts in few-shot settings. While discrete prompts are manually crafted in the absence of class labels, continuous prompts can be automatically learned from the few-shot labels through a prompt-tuning process. Prompt-tuning is both data- and computation-efficient owing to the much fewer parameters in a continuous prompt than in the pre-trained model. Furthermore, considering the graph structures between texts, we propose a context-based initialization for prompt tuning, which could provide a more informative starting point than random initialization.

Contributions. To summarize, we make the following contributions to low-resource text classification.

- This is the first attempt to pre-train text and graph encoders jointly for low-resource text classification.
- We propose a novel model G2P2, with three graph interaction-based contrastive strategies in pre-training, and a prompting approach for the jointly pre-trained graph-text model in downstream tasks.
- We conduct extensive experiments on four real-world datasets to demonstrate the strength of G2P2 in zero- and few-shot text classification.

5.2 Proposed Approach

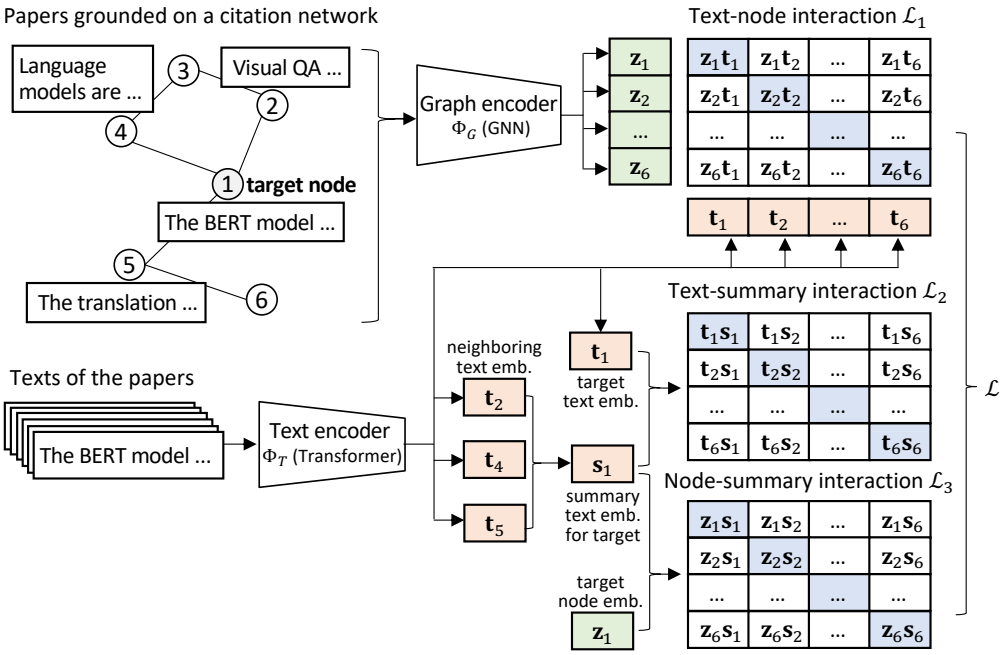
In this section, we introduce our approach G2P2 for low-resource text classification. We start with some preliminaries and an overview, and then present the details of the proposed approach.

Preliminaries

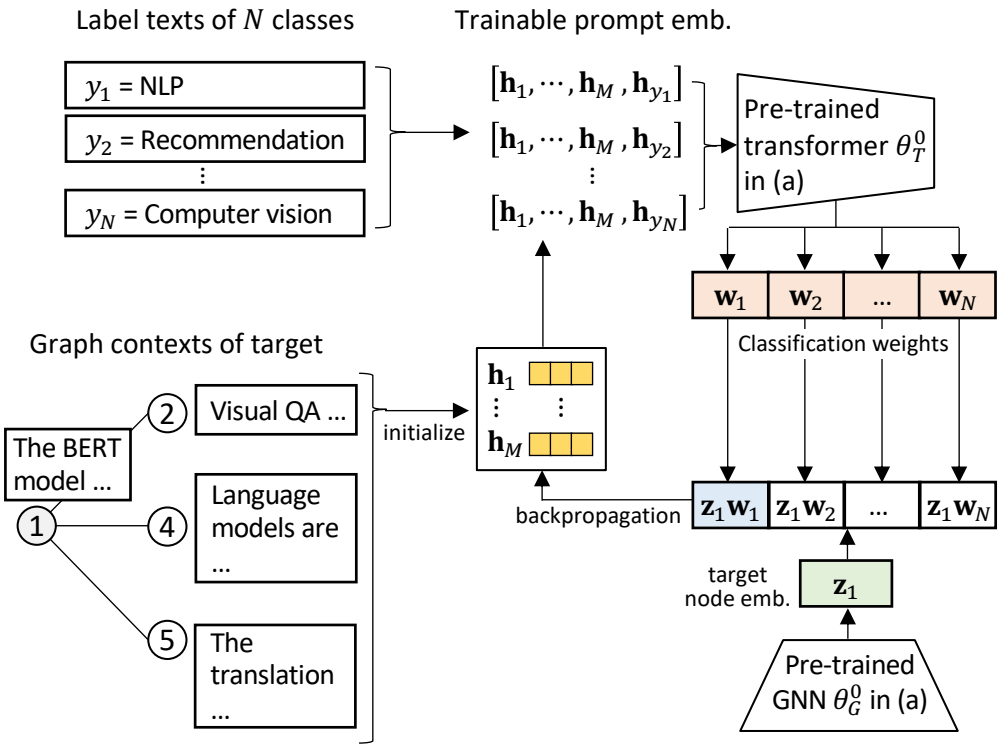
Graph-grounded text corpus. Consider a set of documents $\mathcal{D}$, which is grounded on a graph $\mathcal{G} = (\mathcal{D}, \mathcal{E}, \mathbf{X})$ such that each document $d_i \in \mathcal{D}$ is a node v_i in the graph. The documents are linked via edges in $\mathcal{E}$, which are formed based on the application (*e.g.*, if each document represents an article, the edges could be citations between articles). Each node v_i is also associated with a feature vector $\mathbf{x}_i$, given by the input feature matrix $\mathbf{X}$. Finally, each document/node[1] has a class label (*e.g.*, the topic of the article).

Low-resource classification. A low-resource task consists of a support set $\mathcal{S}$ and a query set $\mathcal{Q}$. The support set $\mathcal{S}$ contains N classes, and each class has K labeled examples where K is a small number (*e.g.*, 1 or 5), known as N-way K-shot classification. The query set $\mathcal{Q}$ contains one or more unlabeled instances belonging to the N classes in the support set. Our goal is to classify the instances in the query set based on the labeled examples in the support set. Unlike episodic few-shot meta-learning [22] which has both training tasks and testing tasks, we only have testing tasks; in the training stage, we perform self-supervised pre-training on label-free data only. As a special case, tasks with $K = 0$ are known as zero-shot classification, which means that there is no labeled example at all and we can only rely on class metadata (*e.g.*, class label text).

[1] We will use "node" and "document" interchangeably given their one-one correspondence in our context.



(a) Graph-grounded contrastive pre-training



(b) Graph-grounded prompt tuning (few-shot classification)

Figure 5.1: Overall framework of G2P2. (a) During pre-training, it jointly trains a text and a graph encoder through three contrastive strategies. (b) During testing, it performs prompt-assisted zero- or few-shot classification (the figure only shows prompt-tuning for few-shot classification, while zero-shot inference adopts a simplified scheme).

Overview of G2P2

As shown in Fig. 5.1, our model consists of two stages: (a) graph-grounded contrastive pre-training, and (b) graph-grounded prompt-tuning for low-resource classification.

During pre-training, we learn a dual-modal embedding space by jointly training a text encoder and graph encoder in a self-supervised fashion, since a document also exists as a node on the graph. More specifically, we use a transformer-based text encoder and a GNN-based graph encoder. The transformer takes the text on each node (*i.e.*, document) as the input, and outputs a text embedding vector $\mathbf{t}_i$ for node v_i. On the other hand, the GNN takes the graph and node features as input, and generates a node embedding vector $\mathbf{z}_i$ for node v_i. Subsequently, in the dual-modal embedding space, we align the text and graph representations on the same or related nodes through three contrastive strategies based on different types of interaction on the graph.

In downstream testing, we employ prompting on our jointly pre-trained graph-text model for zero- or few-shot classification. For zero-shot classification, we use handcrafted discrete prompts together with the label text. For few-shot classification, we use continuous prompts to pad the label text. In particular, for prompt-tuning, we initialize the continuous prompt embeddings based on graph contexts.

Graph-grounded contrastive pre-training

The graph-grounded pre-training learns a dual-modal embedding space by jointly training a text encoder and a graph encoder, based on three types of interaction on the underlying graph.

Dual encoders. The text encoder is a transformer [115], which we denote Φ_T. Given a document d_i, the text encoder[2] outputs the d-dimensional embedding vector of d_i, denoted $\mathbf{t}_i \in \mathbb{R}^d$:

$$\mathbf{t}_i = \Phi_T(d_i; \theta_T), \tag{5.1}$$

where θ_T represents the parameter set of the transformer. Correspondingly, let $\mathbf{T} \in \mathbb{R}^{|\mathcal{D}| \times d}$ represent the text embedding matrix for all documents.

At the same time, a document d_i is also a node v_i in the graph. We choose a classic GNN called graph convolutional network (GCN) [47] as the graph encoder, denoted Φ_Z. It similarly outputs an embedding vector $\mathbf{z}_i \in \mathbb{R}^d$ for a given node v_i:

$$\mathbf{z}_i = \Phi_Z(v_i; \theta_G), \tag{5.2}$$

where θ_G represents the parameter set of the GCN. Likewise, let $\mathbf{Z} \in \mathbb{R}^{|\mathcal{D}| \times d}$ represent the graph embedding matrix for all nodes.

Text-node interaction. Our graph-grounded texts naturally implies a bijection between nodes and texts, where each document d_i corresponds to the node v_i in the graph. Inspired by the pairing of image and its caption text [89] and the mapping of content and node sequences [60], we design a pre-training strategy to predict which text document matches which node in the graph.

Specifically, given n documents and the corresponding n nodes, there are n^2 possible document-node pairs $\{(d_i, v_j) \mid i, j = 1, \ldots, n\}$. Among them, only n pairs with $i = j$ are true matching, whereas the remaining $n^2 - n$ pairs are false matching. As our first contrastive strategy, we exploit the bijective interaction between texts and nodes on the graph, to maximize the cosine similarity of the n

[2]Technically, the input to the text encoder is a sequence of continuous embeddings; the tokens in a document are first converted to word embeddings.

matching pairs, while minimizing the cosine similarity of the $n^2 - n$ unmatching pairs. To compute the cosine similarity for the n^2 pairs, we first perform a row-wise L2 normalization on embedding matrices $\mathbf{T}$ and $\mathbf{Z}$ to obtain $\tilde{\mathbf{T}}$ and $\tilde{\mathbf{Z}}$, respectively. We then compute a node-text similarity matrix $\mathbf{\Lambda}_1 \in \mathbb{R}^{n \times n}$ to capture pairwise cosine similarity, as follows.

$$\mathbf{\Lambda}_1 = \left(\tilde{\mathbf{Z}}\tilde{\mathbf{T}}^\top\right) \cdot \exp(\tau), \tag{5.3}$$

where $\tau \in \mathbb{R}$ is a trainable temperature parameter to scale the similarity values [89].

Remark. Although $\mathbf{\Lambda}_1 \in \mathbb{R}^{n \times n}$ is a dense matrix, it is constructed batch-wise for practical implementation. That is, n is not the total number of documents, but the relatively small batch size, and thus the overhead is negligible. $\mathbf{\Lambda}_2$ and $\mathbf{\Lambda}_3$ will be introduced later following the same treatment. □

To formulate the contrastive loss based on the text-node bijective interaction, we adapt the *multi-class N-pair loss* [105, 152], by considering both the row-wise and column-wise cross entropy loss w.r.t. the row or column index. For example, the i-th row of $\mathbf{\Lambda}_1$ represents the similarity scores between node v_i and every document, in which the row index i indicates the ground truth document d_i that matches v_i.

$$\mathcal{L}_1 = \tfrac{1}{2}\left(\mathrm{CE}(\mathbf{\Lambda}_1, \mathbf{y}) + \mathrm{CE}(\mathbf{\Lambda}_1^\top, \mathbf{y})\right), \tag{5.4}$$

where $\mathbf{y} = (1, 2, \ldots, n)^\top$ is the label vector for contrastive training, and CE denotes the cross entropy loss applied to the input matrix $\mathbf{\Lambda}_1$ or $\mathbf{\Lambda}_1^\top$ in a row-wise manner.

Text-summary interaction. Apart from the bijective text-node interaction, we further exploit higher-order interactions on the graph. In particular, each

document has a set of neighboring documents defined by graph topology. The neighboring documents can be understood as a summary of the target document given the semantic relatedness between them. For example, on an e-commerce network, the products purchased by a user naturally portray a summary of the user and vice versa. Without loss of generality, we employ a simple mean pooling to generate the summary embedding $\mathbf{s}_i \in \mathbb{R}^d$ as follows.

$$\mathbf{s}_i = \frac{1}{|\mathcal{N}_i|} \sum_{j \in \mathcal{N}_i} \mathbf{t}_j. \tag{5.5}$$

For efficiency, we only sample a fixed number of neighboring documents to generate the summary. Then, let $\mathbf{S} \in \mathbb{R}^{n \times d}$ denote the summary text embedding matrix for all documents.

Hence, as our second contrastive strategy, we seek to align the text embedding of each document and its corresponding summary text embedding, based on the text-summary interaction derived from graph neighborhood. In other words, we maximize the cosine similarity of the n matching pairs of document and its neighborhood-based summary, while minimizing the cosine similarity of the $n^2 - n$ unmatching pairs. Specifically, we first follow Eq. (5.3) to construct a text-summary similarity matrix $\mathbf{\Lambda}_2 \in \mathbb{R}^{n \times n}$:

$$\mathbf{\Lambda}_2 = \left(\tilde{\mathbf{T}} \tilde{\mathbf{S}}^\top \right) \cdot \exp(\tau). \tag{5.6}$$

Subsequently, we apply the same contrastive loss following Eq. (5.4), as follows.

$$\mathcal{L}_2 = \frac{1}{2} \left(\mathrm{CE}(\mathbf{\Lambda}_2, \mathbf{y}) + \mathrm{CE}(\mathbf{\Lambda}_2^\top, \mathbf{y}) \right), \tag{5.7}$$

Node-summary interaction. The neighborhood-based summary for document d_i also serves as a semantic description of node v_i. Mirroring the text-summary interaction, as our third contrastive strategy, we seek to align the node embed-

ding and its neighborhood-based summary text embedding. In the following, we similarly compute a node-summary similarity matrix $\mathbf{\Lambda}_3 \in \mathbb{R}^{n \times n}$, and formulate the corresponding contrastive loss $\mathcal{L}_3$.

$$\mathbf{\Lambda}_3 = \left(\tilde{\mathbf{Z}}\tilde{\mathbf{S}}^\top\right) \cdot \exp(\tau), \tag{5.8}$$

$$\mathcal{L}_3 = \tfrac{1}{2}\left(\mathrm{CE}(\mathbf{\Lambda}_3, \mathbf{y}) + \mathrm{CE}(\mathbf{\Lambda}_3^\top, \mathbf{y})\right). \tag{5.9}$$

Overall pre-training objective. Finally, we integrate the three contrastive losses based on the text-node, text-summary and node-summary interactions. We obtain a pre-trained model $\theta^0 = (\theta_T^0, \theta_G^0)$ consisting of the parameters of the dual encoders, given by

$$\theta^0 = \arg\min_{\theta_T, \theta_G} \mathcal{L}_1 + \lambda(\mathcal{L}_2 + \mathcal{L}_3), \tag{5.10}$$

where $\lambda \in \mathbb{R}^+$ is a hyperparameter to balance the contribution from summary-based interactions.

The pre-training procedure is outlined in Algorithm 2, which has the following complexity per epoch. Let $|\mathcal{D}|$ be the number of documents, η be the number of neighbors sampled to generate the summary embedding in Eq. (5.5), and β be the batch size. First, the cost of generating the three types of embeddings (lines 5–8) per epoch is $O(|\mathcal{D}|\eta)$, given that calculating the summary embedding needs go through η neighbors. Second, the cost of calculating the three similarity matrices in each batch is $O(\beta^2)$, and the total cost per epoch is $O\left(\frac{|\mathcal{D}|}{\beta}\beta^2\right) = O(|\mathcal{D}|\beta)$ given $\frac{|\mathcal{D}|}{\beta}$ batches in an epoch. Thus, the overall complexity is $O(|\mathcal{D}|(\eta+\beta))$, which is linear in the number of documents, since η and β are small constants. In our implementation, we set $\eta = 3$ and $\beta = 64$.

Algorithm 2 Pre-training Procedure of G2P2

Require: A graph-grounded text corpus $\mathcal{G} = (\mathcal{D}, \mathcal{E}, \mathbf{X})$.
Ensure: Pre-trained weights of text encoder θ_T^0, graph encoder θ_G^0.

$\theta_T^0, \theta_G^0 \leftarrow$ parameters initialization;
while not converged **do**
  sample batches of documents from $\mathcal{D}$;
  for each batch **do**
    for each node v_i/document d_i in the batch **do**
      calculate d_i's text embedding $\mathbf{t}_i$; ▷ Eq. (5.1)
      calculate v_i's node embedding $\mathbf{z}_i$; ▷ Eq. (5.2)
      calculate v_i's summary embedding $\mathbf{s}_i$; ▷ Eq. (5.5)
    end for
    calculate the similatity matrices $\mathbf{\Lambda}_1, \mathbf{\Lambda}_2, \mathbf{\Lambda}_3$; ▷ Eqs. (5.3), (5.6), (5.8)
    calculate the contrastive losses $\mathcal{L}_1$, $\mathcal{L}_2$, $\mathcal{L}_3$; ▷ Eqs. (5.4), (5.7), (5.9)
    update the overall loss $\mathcal{L}$; ▷ Eq. (5.10)
    $\theta_T^0, \theta_G^0 \leftarrow$ update via backpropagation
  end for
end while
return θ_T^0, θ_G^0.

Prompting joint graph-text model

After pre-training our graph-text model, it is non-trivial to apply it to low-resource classification. To narrow the gap between pre-training and downstream tasks, the traditional "pre-train, fine-tune" paradigm typically introduces a new projection head for the downstream task, which will be fine-tuned together with the whole pre-trained model. However, in a low-resource setting, it is neither effective nor efficient to update the entire model with a huge number of parameters. Without updating massive PLMs, prompting has recently emerged as a powerful alternative to fine-tuning in NLP [62]. However, prompting has not been explored for graph-text models, where structural and textual information have been jointly pre-trained. In the following, we elaborate on our prompting strategies for zero- and few-shot classification.

Zero-shot classification. In the zero-shot setting, we can only use handcrafted discrete prompts, as the absence of labeled data in zero-shot tasks cannot support learnable prompts.

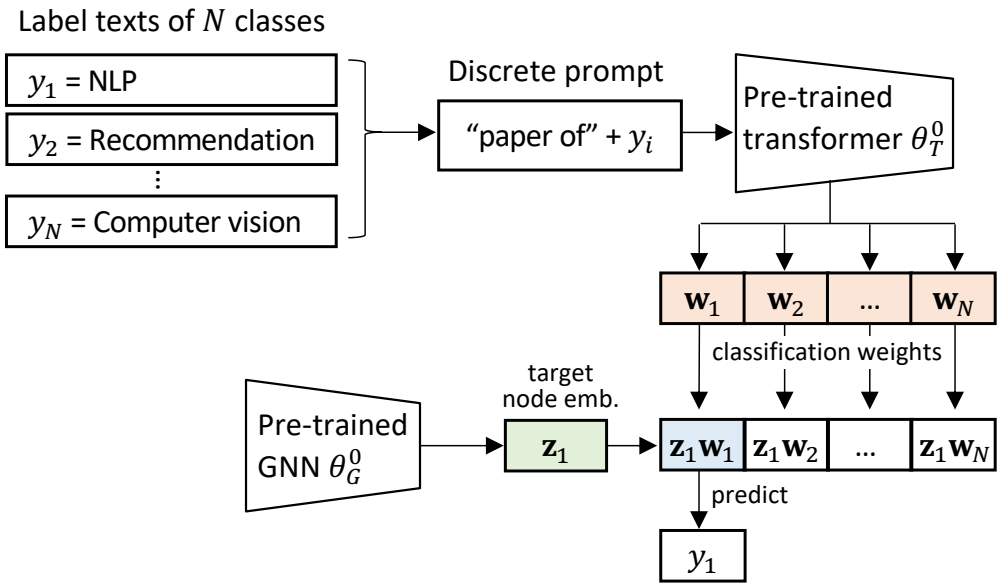


Figure 5.2: Schematic diagram for zero-shot classification. The pre-trained models θ_G^0 and θ_T^0 are obtained from Fig. 5.1(a).

In N-way zero-shot classification, out of N classes, we predict the class which has the highest similarity to the given node. As illustrated by the diagram in Fig. 5.2, the classification weights can be generated by the text encoder based on the class label texts [121], without requiring any labeled sample for the classification task. Specifically, the weight vector $\mathbf{w}_y$ for class $y \in \{1, 2, \ldots, N\}$ is the output of the pre-trained text encoder, *i.e.*,

$$\mathbf{w}_y = \phi_T(\text{“}\texttt{prompt [CLASS]}\text{”}; \theta_T^0). \tag{5.11}$$

Here “`prompt [CLASS]`” is a prompt template, where `[CLASS]` refers to the label text of the target class y (*e.g.*, “`NLP`” for paper area classification), and `prompt` is a manually engineered sequence of natural language tokens to signal the relevance of the label text (*e.g.*, “`paper of NLP`” helps focus on the topic of the paper). In the simplest case, “`prompt`” can be an empty string so that we only rely on the label text. Then, the class distribution given node representation $\mathbf{z}_i$ is predicted as

$$p(y \mid \mathbf{z}_i) = \frac{\exp\left(\langle \mathbf{z}_i, \mathbf{w}_y \rangle\right)}{\sum_{y=1}^{N} \exp\left(\langle \mathbf{z}_i, \mathbf{w}_y \rangle\right)}, \tag{5.12}$$

where $\langle \cdot, \cdot \rangle$ is the cosine similarity.

Few-shot classification. The problem with discrete prompts is that they are difficult to optimize, given that PLMs are intrinsically continuous. Substituting discrete natural language prompts with learnable continuous prompts, prompt tuning [52, 64, 63] can automate the optimization of prompts when some labeled data are available. Hence, in the few-shot setting, we explore prompt tuning to cue in the relevant structural and semantic information from our jointly pre-trained graph-text model.

Specicifally, instead of a sequence of discrete tokens, we take a sequence of continuous embeddings $[\mathbf{h}_1, \cdots, \mathbf{h}_M, \mathbf{h}_{\texttt{CLASS}}]$ as the prompt, where M is a hyperparameter indicating the number of context tokens, each $\mathbf{h}_m$ $(m \leq M)$ is a trainable vector, and $\mathbf{h}_{\texttt{CLASS}}$ is the word embedding sequence of the target class label. The continuous prompt is fed as input to the text encoder to generate the classification weights for each class y:

$$\mathbf{w}_y = \phi_T([\mathbf{h}_1, \cdots, \mathbf{h}_M, \mathbf{h}_{\texttt{CLASS}}]; \theta_T^0), \tag{5.13}$$

where each $\mathbf{h}_m$ $(m \leq M)$ has the same dimension as the input word embeddings to the text encoder.

Using the same softmax layer in Eq. (5.12), we further update the continuous prompt embeddings using the labeled support set of the few-shot task by minimizing a cross entropy loss, whilst freezing the parameters of the dual encoders. This prompt tuning process is both data- and computation-efficient, given the small number of learnable parameters in the prompt.

Furthermore, existing prompt tuning methods either initialize the prompt embeddings randomly [52, 63] or using the word embeddings of handcrafted discrete prompts [156]. While random initialization is non-informative and more

prone to local optimum, it is still difficult to pick the right discrete prompts for initialization. Therefore, we take the advantage of graph structures to initialize the prompt embeddings.

Specifically, given a node v_i, we define its *graph contexts* as its neighbor set $\{v_j \mid j \in \mathcal{N}_i\}$. Due to the underlying semantic relatedness, the graph contexts of the few-shot examples carry strong signals about the task, which can be exploited to improve the initialization. For each document/node v_i in the task support set, we sample η nodes from its graph contexts. For v_i itself and each context node sampled, we truncate its corresponding document to M words, and convert it to a sequence of M word embedding vectors, each having the same dimension as the vector $\mathbf{h}_m$ $(m \leq M)$ in our continuous prompt. Hence, for each support node, we would obtain $\eta + 1$ such sequences; in an N-way K-shot task, there is a total of $NK(\eta + 1)$ sequences. We take the average of these embedding sequences to initialize the learnable prompt vectors $\mathbf{h}_1, \ldots, \mathbf{h}_M$, which is derived from graph contexts and thus could provide a more informative starting point than random initialization.

5.3 Experiments

We conduct extensive experiments to evaluate G2P2, with comparison to state-of-the-art baselines and model analyses.

Experimental setup

Datasets. Four public graph-grounded text corpora are used, as summarized in Tab. 5.1.

- **Cora**[3] [73]: known as the "Cora Research Paper Classification" dataset, it is a collection of research papers that are linked to each other through citations. The abstract of a paper is deemed a text document. The papers are classified into a topic hierarchy with 73 leaves. After removing papers with no content or label, the resulting hierarchy has 70 leaf topics. Note that we are using a more comprehensive version of the Cora dataset, which is larger and has more classes than the version used elsewhere [47].

- **Art**, **Industrial** and **Music Instruments (M.I.)**[4] are three Amazon review datasets [82], respectively from three broad areas, namely, arts, crafts and sewing (Art), industrial and scientific (Industrial), and musical instruments (M.I.). The description of each product is deemed a text document, whereas the review texts of a user are combined into a document to reflect his/her shopping preferences. If a user has reviewed a product, a link is constructed between them. The product subcategories within a broad area represent the classes, which are fine-grained and can involves thousands of classes with subtle differences. The classification is only performed on product description documents, whereas the user review documents only serve to enrich the text semantics in relation to the products.

For all datasets, we employ the word2vec algorithm [77] to obtain the 128-dimensional word embeddings of each word in the text documents. Then, for each node, we average the word embedding vectors of all the words in its document, and the averaged vector is used as the node's input features for the GNN-based methods.

Task construction. We perform zero- or few-shot text classification. We adopt a *5-way* setting, *i.e.*, we sample five classes from all the classes to construct a task. In each task, we construct a K-shot support set by further sampling K

[3] https://people.cs.umass.edu/~mccallum/data.html
[4] http://deepyeti.ucsd.edu/jianmo/amazon/index.html

Table 5.1: Statistics of datasets.

Dataset	Cora	Art	Industrial	M.I.
# Documents	25,120	1,615,902	1,260,053	905,453
# Links	182,280	4,898,218	3,101,670	2,692,734
# Avg. doc length	141.26	54.23	52.15	84.66
# Avg. node deg	7.26	3.03	2.46	2.97
# Total classes	70	3,347	2,462	1,191

examples from each class for $K \in \{0, 1, \ldots, 5\}$, and a validation set of the same size as the support set. The remaining examples form the query set. Note that the support set is labeled and serve as the task training data, whereas the query set is unlabeled and used for evaluation. Note that in our experiment all the classes are used—it is only that each task involves 5 classes, and we have multiple tasks during testing to cover all the classes. This is a typical task setup [22], allowing for a comprehensive evaluation under different class combinations. The reported results are averaged over all the tasks on each dataset.

Baselines for few-shot classification. We consider competitive baselines from four categories.

(1) *End-to-end GNNs*, which are graph neural networks trained in a supervised, end-to-end manner from random initialization.

- GCN [47]: an variant of convolutional neural network and operates on the graph only.
- SAGE_{sup} [31]: the supervised version of GraphSAGE, an inductive GNN which generates node embeddings by sampling and aggregating features from a node's local neighborhood.
- TextGCN [145]: a GCN-based model on a text graph constructed from word co-occurrence and document-word relations, which jointly learns the embeddings of both words and documents.

Table 5.2: *Five-shot* classification performance (percent) with 95% confidence intervals.

In each column, the best result among all methods is **bolded** and the best among the baselines is underlined. Improvement by G2P2 is calculated relative to the best baseline. * indicates that our model significantly outperforms the best baseline based on two-tail t-test ($p < 0.05$).

	Cora	Art	Industrial	M.I.
	Accuracy	Accuracy	Accuracy	Accuracy
GCN	41.15±2.41	22.47±1.78	21.08±0.45	22.54±0.82
SAGE_{sup}	41.42±2.90	22.60±0.56	20.74±0.91	22.14±0.80
TextGCN	59.78±1.88	43.47±1.02	53.60±0.70	46.26±0.91
GPT-GNN	76.72±2.02	65.15±1.37	62.13±0.65	67.97±2.49
DGI	78.42±1.39	65.41±0.86	52.29±0.66	68.06±0.73
$\text{SAGE}_{\text{self}}$	77.59±1.71	76.13±0.94	71.87±0.61	77.70±0.48
BERT	37.86±5.31	46.39±1.05	54.00±0.20	50.14±0.68
BERT*	27.22±1.22	45.31±0.96	49.60±0.27	40.19±0.74
RoBERTa	62.10±2.77	72.95±1.75	76.35±0.65	70.67±0.87
RoBERTa*	67.42±4.35	74.47±1.00	77.08±1.02	74.61±1.08
P-Tuning v2	71.00±2.03	76.86±0.59	79.65±0.38	72.08±0.51
G2P2-p	79.16±1.23	79.59±0.31	80.86±0.40	81.26±0.36
G2P2	**80.08***±1.33	**81.03***±0.43	**82.46***±0.29	**82.77***±0.32
(improv.)	(+2.12%)	(+5.43%)	(+3.53%)	(+6.53%)
	Cora	Art	Industrial	M.I.
	F1	F1	F1	F1
GCN	34.50±2.23	15.45±1.14	15.23±0.29	16.26±0.72
SAGE_{sup}	35.14±2.14	16.01±0.28	15.31±0.37	16.69±0.62
TextGCN	55.85±1.50	32.20±1.30	45.97±0.49	38.75±0.78
GPT-GNN	72.23±1.17	52.79±0.83	54.47±0.67	59.89±2.51
DGI	74.58±1.24	53.57±0.75	45.26±0.51	60.64±0.61
$\text{SAGE}_{\text{self}}$	73.47±1.53	65.25±0.31	65.09±0.47	70.87±0.59
BERT	32.78±5.01	37.07± 0.68	47.57±0.50	42.96±1.02
BERT*	23.34±1.11	36.28±0.71	43.36±0.27	33.69±0.72
RoBERTa	57.21±2.51	62.25±1.33	70.49±0.59	63.50±1.11
RoBERTa*	62.72±3.02	63.35±1.09	71.44±0.87	67.78±0.95
P-Tuning v2	66.76±1.95	66.89±1.14	74.33±0.37	65.44±0.63
G2P2-p	74.99±1.35	68.26±0.43	74.44±0.29	74.82±0.45
G2P2	**75.91***±1.39	**69.86***±0.67	**76.36***±0.25	**76.48***±0.52
(improv.)	(+1.78%)	(+4.44%)	(+2.7%)	(+7.92%)

(2) *Pre-trained/self-supervised GNNs*, these GNNs are pre-trained using pretext tasks without labeled data, followed by fine-tuning or fitting a classification head while freezing the model parameters.

- GPT-GNN [41]: a GNN pre-training approach by a self-supervised graph generation task, including node attribute generation and edge generation. It follows the "pre-train, fine-tune" paradigm.
- DGI [118]: a GNN pre-training approach which maximizes the mutual information between local- and global-level representations. As an unsupervised method, it also freezes the model parameters and fits a simple logistic regression model for the downstream few-shot classification, after pre-training.
- $\text{SAGE}_{\text{self}}$ [31]: the self-supervised version of GraphSAGE, encouraging similar embeddings for neighboring nodes and distinct embeddings for non-adjacent nodes. After pre-training, it follows the same approach of DGI for the downstream classification.

(3) *Pre-trained transformers*, which are pre-trained using masked language modeling, and then are fine-tuned together with a randomly initialized classification head (*e.g.*, a fully connected layer), for the downstream few-shot classification task.

- BERT [43]: a pre-trained transformer which pre-trains the transformer using masked language modeling, *i.e.*, during training, output deep bidirectional representations from unlabeled text by jointly conditioned on both left and right context in all layers.
- RoBERTa [65]: a replicaition of BERT that carefully measures the impact of many key hyperparameters and training data size during training.
- BERT* and RoBERTa*: variants of BERT and RoBERTa, which are obtained by fine-tuning the pre-trained BERT and RoBERTa, respectively, using masked

language modeling on our datasets, to mitigate the domain gap between our datasets and the datasets used for pre-training BERT and RoBERTa.

(4) *Prompt tuning*: P-Tuning v2 [63], is a version of prefix-tuning [58] optimized and adapted for natural language. It uses deep prompt tuning, which applies continuous prompts for every layer of the pre-trained language model.

Note that our setting is distinct from few-shot learning under the meta-learning paradigm [22], as there is no few-shot tasks for the meta-training phase. Hence, we cannot use state-of-the-art meta-learning models for comparison. Besides, two of the baselines we compared, DGI and $\text{SAGE}_{\text{self}}$, have adopted a form of linear probe which is known to be a strong few-shot learner [112].

Baselines for zero-shot classification. We only compare with PLMs, as all the other methods require at least one shot to work. For each method, we use the discrete prompt `[CLASS]` (*i.e.*, the label text alone). We also evaluate hand-crafted prompts "`prompt [CLASS]`", where `prompt` is a sequence of tokens found by prompt engineering, and annotate the model name with "+d". Essentially, we compute the similarity between the target document and the label text of each class (with or without additional tokens), and predict the most similar class following Fig. 5.2.

Settings of G2P2 and baselines. For G2P2, the text encoder is a transformer [115]. Following CLIP [89], we use a 63M-parameter, 12-layer 512-wide model with 8 attention heads. It operates on a lower-cased byte pair encoding (BPE) representation of the texts with a 49,152 vocabulary size [100]. The max sequence length is capped at 128. The graph encoder employs a GCN [47], using two layers [31] with a LeakyReLU activation, each with 128 dimensions [87]. The pre-training of our model starts from scratch without initializing the graph and text encoders with previously pre-trained weights. λ in Eq. (5.10) is set to 0.1 on Cora, and set to 10 on the three amazon review datasets, which were chosen from {0.01,

0.1, 1, 10, 100} according to the accuracy on validation data. The number of learnable prompt tokens, M in Eq. (5.13), is set to 4, which was chosen from {2, 4, 8, 16, 32} according to the accuracy on validation data. We use the Adam optimizer with the learning rate 2×10^{-5} with 2 training epochs, and a batch size of 64 in pre-training, referring to Hugging Face's [131] example settings. The text embedding size is 128, same to the output from the graph encoder. To generate the summary embedding and the context-based prompt initialization, the number of neighboring nodes sampled is 3. For prompt tuning, we set the learning rate as 0.01, which was chosen from {0.0001,0.001,0.01,0.1} according to the accuracy on validation data.

For all the GNN methods, including the GNN component in G2P2, we use the 128-dimensional word2vec embeddings [77] of raw texts as the input node features. We use a two-layer architecture, and set the hidden dimension to be 128, except for GCN and SAGE_{sup} whose hidden dimension is set to 32 [47] which gives better empirical performance. For all GNN pre-training baselines, we use 0.01 as the learning rate. For BERT, RoBERTa and G2P2, we adopt 0.00002 as the learning rate. Our implementations of BERT, RoBERTa and their masked language modeling are based on Hugging Face's transformers [131]. For both BERT and RoBERTa, we use their base versions, given that our model G2P2 uses just a small 63M-parameter model, following previous work [89]. For P-Tuning v2, we use the original code on the RoBERTa backbone, and take the recommended 0.005 as the learning rate for prompt tuning. For G2P2, the learning rate for prompt tuning is set to 0.01.

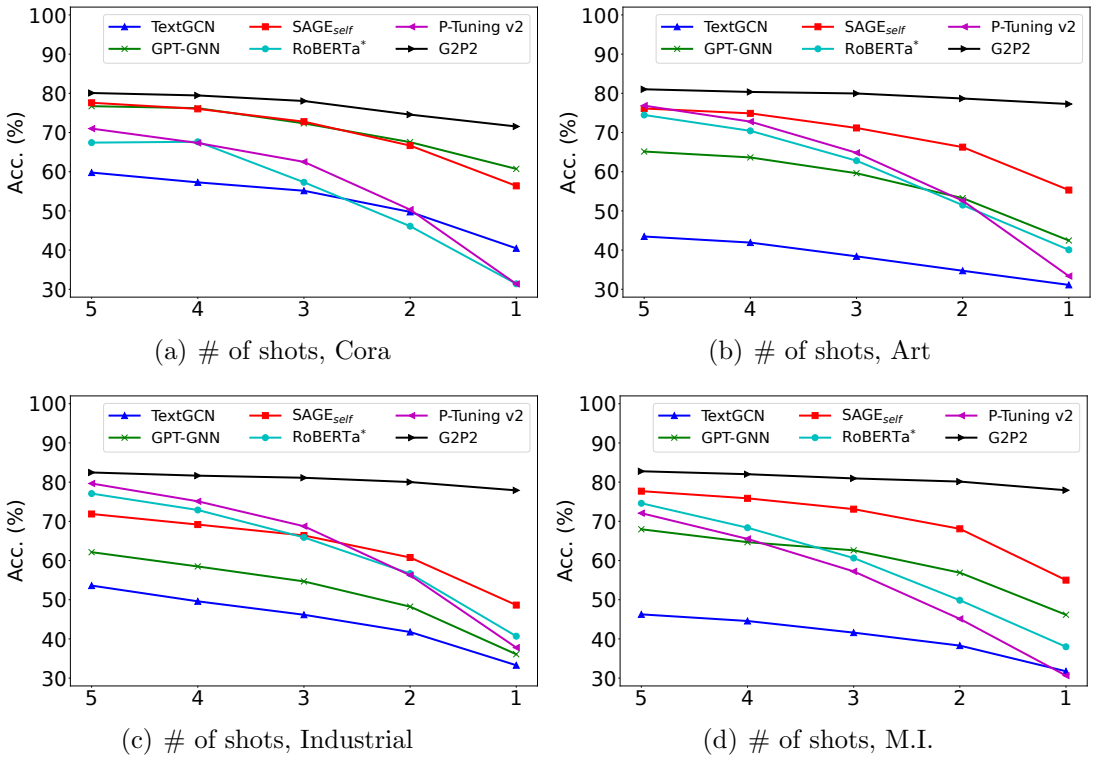
We conduct all experiments on a server with 4 units of GeForce RTX 3090 GPU. Pre-training G2P2 takes about 0.5/6/9/10 hours on Cora/M.I./Industrial/Art, respectively, on a single GPU. The inference (with prompt tuning) is conducted with five different splits generated from five random seeds {1, 2, 4, 8, 16}.

Performance of low-resource classification

We evaluate the classification performance under various shots.

Five shots. In Tab. 5.2, we first compare the performance of G2P2 with baselines under the *5-shot* setting. G2P2 emerges as the winner consistently, outperforming the best baseline by around 2–8% with statistical significance.

We also make a few more observations. Firstly, among the GNNs, pre-trained/self-supervised models tend to perform better than the end-to-end approaches, since the latter heavily rely on labeled data. Among the former, DGI and $\text{SAGE}_{\text{self}}$ perform better as they are a form of linear probe, known to be a strong few-shot learner [112]. Note that, instead of using word2vec embeddings [77] of raw texts as node features, we also tired using the pre-trained RoBERTa [65] to generate the node features for DGI and $\text{SAGE}_{\text{self}}$. However, doing so does not bring any improvement, showing that it is ineffective to simply combine a language model and GNN in a decoupled manner. In contrast, our proposed model jointly learns the text and graph encoders through three graph-grounded contrastive strategies. Secondly, PLMs are generally superior to GNNs, illustrating the importance of leveraging texts in a fine-grained way. Additionally, RoBERTa outperforms BERT owing to an improved pre-training procedure [65]. However, further training PLMs on our texts gives mixed results: RoBERTa* slightly outperforms RoBERTa but BERT* is much worse than BERT. That means it is not straightforward to mitigate the domain gap by simply continuing training on the domain texts. Thirdly, the continuous prompt approach P-Tuning v2 achieves competitive results compared to fine-tuning, while having the advantage of being much cheaper than fine-tuning. Nevertheless, it is still significantly outperformed by our model G2P2. Furthermore, G2P2-p without prompt tuning is inferior to G2P2, showing the benefit of continuous prompts.



(a) # of shots, Cora

(b) # of shots, Art

(c) # of shots, Industrial

(d) # of shots, M.I.

Figure 5.3: Performance on different shots.

Fewer shots. In addition to the 5-shot setting, in Fig. 5.3 we also study the impact of fewer shots on G2P2 and several representative baselines. G2P2 generally performs the best across different shots. In general, the performances of all approaches degrade as fewer shots become available. However, the baselines suffer significantly under extreme low-resource (*e.g.*, 1- or 2-shot) settings. In contrast, G2P2 remains robust, reporting a relatively small decrease in performance even with just 1 or 2 shots.

The results demonstrate the practical value of our proposed model especially when labeled data are difficult or costly to obtain in time. On the other hand, traditional approaches constantly face the challenge of the inability to keep up with the rapid growth of emerging classes in dynamic and open environments [128]. For example, labeling a large volume of texts for novel topics in online articles, or new product categories in open-ended e-commerce platforms, can suffer a substantial time lag.

Zero shot. Finally, we report the zero-shot performance in Tab. 5.3, where our models G2P2 and G2P2+d significantly outperforms the baselines. The results particularly demonstrate the effectiveness of our graph-grounded contrastive pre-training in the absence of labeled data, which is crucial to handling evolving classes without any labeled sample in many real-world scenarios. Moreover, handcrafted discrete prompts (*i.e.*, BERT*+d and G2P2+d) can be superior to using label text only (*i.e.*, BERT* and G2P2), showing the effectiveness of additional prompt tokens.

However, finding the optimal discrete prompts often requires significant engineering work. Specifically, for the three approaches with discrete prompts, namely, RoBERTa*+d, BERT*+d and G2P2+d, we explored more than 10 handcrafted prompt templates on each dataset, which are typically relevant to the corresponding dataset and require some domain knowledge to devise. While discrete prompts are generally helpful to zero-shot classification, their effectiveness varies. In Tab. 5.3, we simply report the performance of the best handcrafted template for each approach and each dataset. Besides, it is worth noting that the same prompt can sometimes generate opposite results on different models. For instance, in Cora dataset, while "`a model of [CLASS]`" is the best prompt for RoBERTa*+d, it is a bad choice for G2P2+d. Moreover, some prompts without any semantic meaning, like "`a [CLASS]`", can be the best choice sometimes. The observations imply that prompt engineering involves labor-intensive work, and the outcomes contain much uncertainty on what would be the optimal discrete prompt. Hence, using the label text only is still a reasonably good choice.

Model analyses

We conduct more in-depth studies on G2P2. Unless otherwise stated, we report the classification *accuracy* under the *5-shot* setting.

Table 5.3: *Zero-shot* classification accuracy (percent).

(See Table 5.2 for explanations on entry styles.)

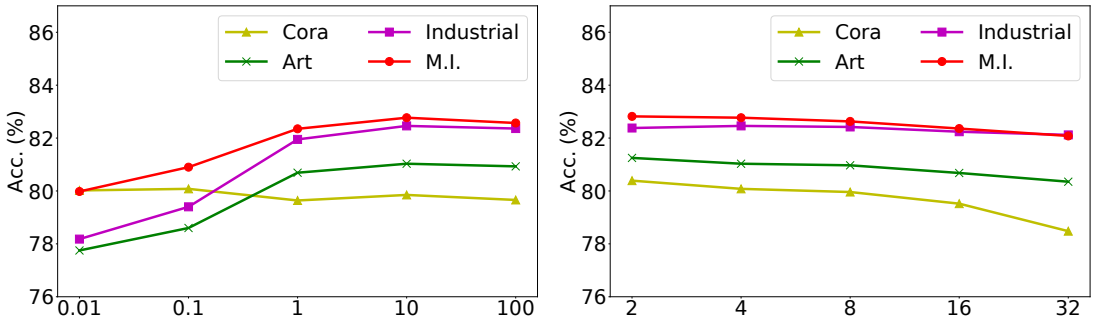
	Cora	Art	Industrial	M.I.
RoBERTa	30.46±2.01	42.80±0.94	42.89±0.97	36.40±1.20
RoBERTa*	39.58±1.26	34.77±0.65	37.78±0.32	32.17±0.68
RoBERTa*+d	45.53±1.33	36.11±0.66	39.40±1.22	37.65±0.33
BERT	23.58±1.88	35.88±1.44	37.32±0.85	37.42±0.80
BERT*	23.38±1.96	54.27±1.85	56.02±1.22	50.19±0.72
BERT*+d	26.65±1.71	56.61±1.76	55.93±0.96	52.13±0.88
G2P2	63.52±2.89	76.52±0.59	76.66±0.31	74.60±0.62
G2P2+d	**65.28***±3.12	**76.99***±0.60	**77.43***±0.27	**75.86***±0.69
(improv.)	(+45.38%)	(+36.00%)	(+38.22%)	(+45.52%)

Ablation study. We first evaluate the contribution from each of the three graph interaction-based contrastive strategies, by employing different combinations of the proposed loss terms $\mathcal{L}_1, \mathcal{L}_2$ and $\mathcal{L}_3$. As shown in Tab. 5.4, strategies without $\mathcal{L}_1$ have performed quite poorly, demonstrating that the bijective text-node interaction is the fundamental component of our pre-training. That being said, when further adding $\mathcal{L}_2$ or $\mathcal{L}_3$ to $\mathcal{L}_1$, we still observe a noticeable performance improvement, showing the benefit of incorporating additional graph-based interactions for text data. Lastly, G2P2 with all three loss terms outperforms all 1- or 2-combinations of the losses, demonstrating that the three contrastive strategies are all useful and they are well integrated. Overall, the results reveal that graph information is vital to low-resource text classification, since graph structures reveal rich relationships between documents.

Next, we evaluate the contribution from our prompt-tuning approach. Specifically, we compare G2P2 with two ablated variants: using label text only without trainable prompt vectors, and randomly initializing the prompt vectors. As reported in Tab. 5.4, only using label text clearly degrades the classification performance, implying the importance of learning continuous prompts via prompt tuning. Furthermore, our approach G2P2 with context-based initialization for prompt vectors shows a small but consistent advantage over random initializa-

Table 5.4: Ablation study.

	Cora	Art	Industrial	M.I.
Only $\mathcal{L}_3$	74.66±1.80	52.56±1.09	45.97±0.81	49.05±0.54
Only $\mathcal{L}_2$	77.01±1.30	58.90±0.55	52.99±0.46	59.41±0.85
Only $\mathcal{L}_1$	79.50±1.19	77.37±0.72	78.10±0.34	79.70±0.56
$\mathcal{L}_2$+$\mathcal{L}_3$	70.04±2.89	49.91±1.57	50.07±0.50	56.14±1.01
$\mathcal{L}_1$+$\mathcal{L}_3$	79.73±0.89	78.60±0.40	79.97±0.43	80.42±0.45
$\mathcal{L}_1$+$\mathcal{L}_2$	79.42±1.04	80.55±0.52	81.06±0.33	82.39±0.41
Only label text	79.16±1.23	79.59±0.31	80.86±0.40	81.26±0.36
Random init.	80.03±0.99	80.85±0.43	82.43±0.33	82.64±0.21
G2P2	**80.08**±1.33	**81.03**±0.43	**82.46**±0.35	**82.77**±0.32



(a) Interaction coefficient, λ (b) Prompt length, M

Figure 5.4: Hyperparameter study.

tion, implying the usefulness of considering graph structures in prompt tuning.

Hyperparameter study. We first investigate the impact of the interaction coefficient λ in Fig. 5.4(a), which balances the high-order contrastive losses ($\mathcal{L}_2, \mathcal{L}_3$). The performance is generally better and stable when λ is slightly bigger (*e.g.*, $\geq$ 10), indicating the significance of the high-order text-summary and node-summary interactions. Next, we study the prompt length M in Fig. 5.4(b), which refers to the number of trainable prompt vectors in Sect. 5.2. The performance is relatively unaffected by the prompt length, and thus it is robust to choose a small M (*e.g.*, 4) for efficiency.

Efficiency of prompt tuning. In our work, the continuous prompts are optimized by prompt tuning [63, 156] without updating the pre-trained model. In this

Table 5.5: Tuning time and parameter size.

	Tuning time per task (in seconds)				Param. size
	Cora	Art	Industrial	M.I.	
RoBERTa	45.47±2.38	64.22±3.62	43.46±2.99	44.99±2.58	123 M
RoBERTa*	39.38±2.01	59.56±3.55	35.10±2.75	38.84±2.39	123 M
BERT	32.23±1.71	51.77±2.00	31.72±1.77	33.55±2.39	110 M
BERT*	34.82±1.68	55.16±2.32	31.11±1.74	29.00±2.23	110 M
G2P2	**2.42**±0.41	**22.03**±1.39	**14.63**±1.26	**12.72**±1.17	**2048**

experiment, we investigate the prompt tuning efficiency of G2P2 in comparison to the efficiency of traditional fine-tuning. As G2P2 has a transformer component, we compare it with four transformer based models, all of which follow the classical "pre-train, fine-tune" paradigm [43].

As shown in Tab. 5.5, "Tuning time per task" refers to the average time required per task for prompt tuning by G2P2 or fine-tuning by the baselines, while "Param. size" refers to the number of parameters that require updating. The results demonstrate that prompt tuning in G2P2 is much more efficient than fine-tuning in the baselines, achieving 2.1∼18.8x speedups. The reason is that prompt tuning updates far fewer parameters. In G2P2, we used 4 trainable 512-dimensional prompt vectors, totalling to 2048 parameters only, while fine-tuning in the baselines needs to update the whole pre-trained model with more than 100M parameters. Note that the speedup is not linear w.r.t. the parameter size, due to overheads in the data loader and the optimizer. Overall, our prompt tuning is not only effective under low-resource settings, but also parameter- and computation-efficient.

Generalization study. Our previous experiments can be deemed "transductive" as both the pre-training of text encoder and downstream text classification are conducted on the whole corpus. To further evaluate the generalization ability of our model, we adopt an "inductive" setting, whereby we pre-train the text encoder only on a subset of the corpus and perform downstream classification on a disjoint

Table 5.6: "Inductive" performance on text classification.

	Art	Industrial	M.I.
BERT*	43.66±0.90	48.35±0.25	39.24±0.88
RoBERTa*	69.55±1.14	73.65±0.86	71.96±1.44
G2P2	**79.81**±0.22	**81.29**±0.32	**81.85**±0.33

subset. Particularly, in the three Amazon datasets, since user texts have no labels and item texts have labels, it is natural for us to pre-train with only user texts and classify only item texts downstream. We also employ masked language modeling on only the user texts for BERT and RoBERTa, to get BERT* and RoBERTa*. As shown in Tab. 5.6, G2P2 still performs very well in the inductive setting, illustrating the strong generalization ability of our pre-trained model.

5.4 Conclusion

In this paper, we studied the problem of low-resource text classification. Given that many text documents are related through an underlying network, we proposed a novel model called Graph-Grounded Pre-training and Prompting (G2P2). It consists of three graph interaction-based contrastive strategies in pre-training, and a prompting mechanism for the jointly pre-trained graph-text model in downstream classification. We conducted extensive experiments and showed the advantages of G2P2 in zero- and few-shot text classification.

A limitation of this work is the need of a graph to complement the texts. Although graphs are ubiquitous in information retrieval applications, in the case that an organic graph is unavailable, a potential solution is to construct synthetic graphs based on word co-occurrences or other relations, *e.g.*, linking up news articles in close time periods and locations. We leave further explorations to future work.

Chapter 6

Conclusion and Future work

6.1 Conclusion

In this dissertation, we explore three distinct avenues for augmenting the learning capabilities of inductive Graph Neural Networks (GNNs):

- To generalize GNNs across multiple graphs, we examined the issue of inductive node classification spanning various graphs. In contrast to prevailing one-size-fits-all methodologies, we introduced a pioneering framework called MI-GNN, which adapts the inductive model to individual graphs within a meta-learning paradigm. To address graph discrepancies, we devised a dual adaptation mechanism operating at both graph and task levels. Specifically, we employ a graph prior for graph-level differences and a task prior to accommodate task-level variations contingent on each graph.
- In the pursuit of generalizing GNNs across temporal dimensions, we investigated temporal graph representation learning. We proposed TREND, an innovative framework for temporal graph representation learning that relies on event and node dynamics within a Hawkes process-based GNN. TREND's inductive nature captures a comprehensive perspective of the link formation process. Crucially,

it integrates both event and node dynamics to accurately model the temporal evolution by accounting for individual and collective event characteristics.

- Lastly, to generalize GNNs across tasks, we delved into the problem of low-resource text classification. Given the interconnected nature of numerous text documents through underlying networks, we put forth a novel model called Graph-Grounded Pre-training and Prompting (G2P2). This model comprises three graph interaction-based contrastive strategies during pre-training and a prompting mechanism for the jointly pre-trained graph-text model in subsequent classification tasks.

6.2 Future work

The three proposed techniques of generalizing Graph Neural Networks (GNNs) across graphs, time, and node classification tasks offer promising avenues for their application in real-world scenarios. To further advance the practical utility and impact of these approaches, future work will focus on their application in specific domains and real-world applications.

Several potential directions for future work include:

- Social Network Analysis: The generalization of GNNs across graphs can be applied to real-world social network analysis scenarios. For example, applying these techniques to social media platforms can help identify influential users, detect communities, and analyze information diffusion patterns. This could aid in understanding social dynamics, predicting trends, and enhancing targeted marketing strategies;

- Financial Markets: The extension of GNNs across time presents opportunities for applications in financial markets. By capturing temporal dependencies and patterns, these techniques can be employed to predict stock market

trends, identify anomalies, and assist in portfolio management. This can enable investors to make more informed decisions, manage risks, and optimize their investment strategies;

- Healthcare and Disease Progression: Applying the generalized GNNs across time to healthcare data can facilitate the analysis of disease progression and patient monitoring. By capturing temporal dynamics, these techniques can be utilized for predicting disease progression, identifying risk factors, and designing personalized treatment plans. This has the potential to enhance healthcare decision-making, improve patient outcomes, and optimize resource allocation;

- Bioinformatics: The generalization of GNNs across node classification tasks can be valuable in bioinformatics. For instance, these techniques can aid in protein function prediction, gene expression analysis, and drug discovery. By leveraging the rich graph structures inherent in biological data, these approaches can improve our understanding of complex biological systems and support advancements in personalized medicine;

- Recommendation Systems: The generalization of GNNs across graphs can be applied to recommendation systems in various domains, such as e-commerce, media streaming, and content platforms. By incorporating graph-based representations and capturing temporal dynamics, these techniques can provide more accurate and personalized recommendations, enhancing user experiences and driving customer satisfaction.

Besides, in Chapter 3, our proposed MI-GNN primarily focuses on the critical problem of node classification in graph learning. However, beyond node classification, there are other intriguing problems that warrant exploration. For instance, cross-graph link prediction and cross-graph clustering present additional avenues for investigation.

In addition, in Chapter 4, the TREND approach, as put forth in this study, demonstrates its capability to effectively manage dynamic graphs that undergo alterations in their topologies, particularly with regard to the addition of nodes and links. However, the issue of handling nodes and links deletion remains unaddressed in the present work, serving as a potential avenue for future research and development.

Bibliography

[1] Yunsheng Bai, Hao Ding, Yang Qiao, Agustin Marinovic, Ken Gu, Ting Chen, Yizhou Sun, and Wei Wang. Unsupervised inductive graph-level representation learning via graph-graph proximity. In *IJCAI*, pages 1988–1994, 2019.

[2] Trapit Bansal, Rishikesh Jha, Tsendsuren Munkhdalai, and Andrew McCallum. Self-supervised meta-learning for few-shot natural language classification tasks. In *EMNLP*, pages 522–534, 2020.

[3] Yujia Bao, Menghua Wu, Shiyu Chang, and Regina Barzilay. Few-shot text classification with distributional signatures. In *ICLR*, 2020.

[4] Avrim Blum and Shuchi Chawla. Learning from labeled and unlabeled data using graph mincuts. In *ICML*, pages 19–26, 2001.

[5] Marc Brockschmidt. Gnn-film: Graph neural networks with feature-wise linear modulation. In *ICML*, pages 1144–1152, 2020.

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

[7] Joan Bruna and X Li. Community detection with graph neural networks. *stat*, 1050:27, 2017.

[8] Hongyun Cai, Vincent W Zheng, and Kevin Chen-Chuan Chang. A comprehensive survey of graph embedding: Problems, techniques, and applications. *TKDE*, 30(9):1616–1637, 2018.

[9] Xianshuai Cao, Yuliang Shi, Han Yu, Jihu Wang, Xinjun Wang, Zhongmin Yan, and Zhiyong Chen. Dekr: description enhanced knowledge graph for machine learning method recommendation. In *SIGIR*, pages 203–212, 2021.

[10] Zongmai Cao, Kai Han, and Jianfu Zhu. Information diffusion prediction via dynamic graph neural networks. In *2021 IEEE 24th international conference on computer supported cooperative work in design (CSCWD)*, pages 1099–1104. IEEE, 2021.

[11] Jiaao Chen, Zichao Yang, and Diyi Yang. Mixtext: Linguistically-informed interpolation of hidden space for semi-supervised text classification. In *ACL*, pages 2147–2157, 2020.

[12] Xiang Chen, Ningyu Zhang, Xin Xie, Shumin Deng, Yunzhi Yao, Chuanqi Tan, Fei Huang, Luo Si, and Huajun Chen. Knowprompt: Knowledge-aware prompt-tuning with synergistic optimization for relation extraction. In *The Web Conference*, pages 2778–2788, 2022.

[13] Zulong Diao, Xin Wang, Dafang Zhang, Yingru Liu, Kun Xie, and Shaoyao He. Dynamic spatial-temporal graph convolutional neural networks for traffic forecasting. In *AAAI*, volume 33, pages 890–897, 2019.

[14] Kaize Ding, Jianling Wang, Jundong Li, Kai Shu, Chenghao Liu, and Huan Liu. Graph prototypical networks for few-shot learning on attributed networks. In *CIKM*, pages 295–304, 2020.

[15] Manqing Dong, Feng Yuan, Lina Yao, Xiwei Xu, and Liming Zhu. Mamo: Memory-augmented meta-optimization for cold-start recommendation. In *KDD*, pages 688–697, 2020.

[16] Raïssa Yapan Dougnon, Philippe Fournier-Viger, Jerry Chun-Wei Lin, and Roger Nkambou. Inferring social network user profiles using a partial social graph. *Journal of Intelligent Information Systems*, 47(2):313–344, 2016.

[17] Lun Du, Yun Wang, Guojie Song, Zhicong Lu, and Junshan Wang. Dynamic network embedding: an extended approach for skip-gram based network embedding. In *IJCAI*, pages 2086–2092, 2018.

[18] Zhengxiao Du, Xiaowei Wang, Hongxia Yang, Jingren Zhou, and Jie Tang. Sequential scenario-specific meta learner for online recommendation. In *KDD*, pages 2895–2904, 2019.

[19] Lu Fan, Qimai Li, Bo Liu, Xiao-Ming Wu, Xiaotong Zhang, Fuyu Lv, Guli Lin, Sen Li, Taiwei Jin, and Keping Yang. Modeling user behavior with graph convolution for personalized product search. In *The Web Conference*, pages 203–212, 2022.

[20] Yuan Fang, Kevin Chen-Chuan Chang, and Hady Wirawan Lauw. Graph-based semi-supervised learning: Realizing pointwise smoothness probabilistically. In *ICML*, pages 406–414, 2014.

[21] Yuan Fang, Bo-June Paul Hsu, and Kevin Chen-Chuan Chang. Confidence-aware graph regularization with heterogeneous pairwise features. In *SIGIR*, pages 951–960, 2012.

[22] Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In *ICML*, pages 1126–1135, 2017.

[23] Tianyu Gao, Adam Fisch, and Danqi Chen. Making pre-trained language models better few-shot learners. In *ACL*, pages 3816–3830, 2021.

[24] Ross Girshick. Fast R-CNN. In *ICCV*, pages 1440–1448, 2015.

[25] Vladimir Gligorijević, P Douglas Renfrew, Tomasz Kosciolek, Julia Koehler Leman, Daniel Berenberg, Tommi Vatanen, Chris Chandler, Bryn C Taylor, Ian M Fisk, Hera Vlamakis, et al. Structure-based protein function prediction using graph convolutional networks. *Nature communications*, 12(1):3168, 2021.

[26] Palash Goyal, Sujit Rokka Chhetri, and Arquimedes Canedo. dyngraph2vec: Capturing network dynamics using dynamic graph representation learning. *Knowledge-Based Systems*, 187:104816, 2020.

[27] Palash Goyal, Nitin Kamra, Xinran He, and Yan Liu. DynGEM: Deep embedding method for dynamic graphs. *arXiv*, 2018.

[28] Aditya Grover and Jure Leskovec. node2vec: Scalable feature learning for networks. In *KDD*, pages 855–864, 2016.

[29] David Ha, Andrew Dai, and Quoc V Le. Hypernetworks. In *ICLR*, 2017.

[30] Ehsan Hajiramezanali, Arman Hasanzadeh, Krishna Narayanan, Nick Duffield, Mingyuan Zhou, and Xiaoning Qian. Variational graph recurrent neural networks. In *NeurIPS*, pages 10701–10711, 2019.

[31] Will Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. In *NeurIPS*, 2017.

[32] Xu Han, Zhengyan Zhang, Ning Ding, Yuxian Gu, Xiao Liu, Yuqi Huo, Jiezhong Qiu, Yuan Yao, Ao Zhang, Liang Zhang, et al. Pre-trained models: Past, present and future. *AI Open*, 2:225–250, 2021.

[33] Xu Han, Weilin Zhao, Ning Ding, Zhiyuan Liu, and Maosong Sun. Ptr: Prompt tuning with rules for text classification. *arXiv*, 2021.

[34] Xu Han, Hao Zhu, Pengfei Yu, Ziyun Wang, Yuan Yao, Zhiyuan Liu, and Maosong Sun. Fewrel: A large-scale supervised few-shot relation classifica-

tion dataset with state-of-the-art evaluation. In *EMNLP*, pages 4803–4809, 2018.

[35] Alan G Hawkes. Spectra of some self-exciting and mutually exciting point processes. *Biometrika*, 58(1):83–90, 1971.

[36] Jingrui He, Jaime Carbonell, and Yan Liu. Graph-based semi-supervised learning as a generative model. In *IJCAI*, pages 2492–2497, 2007.

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

[38] Shengding Hu, Ning Ding, Huadong Wang, Zhiyuan Liu, Jingang Wang, Juanzi Li, Wei Wu, and Maosong Sun. Knowledgeable prompt-tuning: Incorporating knowledge into prompt verbalizer for text classification. In *ACL*, pages 2225–2240, 2022.

[39] Weihua Hu, Bowen Liu, Joseph Gomes, Marinka Zitnik, Percy Liang, Vijay Pande, and Jure Leskovec. Strategies for pre-training graph neural networks. In *ICLR*, 2019.

[40] Ziniu Hu, Ting Chen, Kai-Wei Chang, and Yizhou Sun. Few-shot representation learning for out-of-vocabulary words. In *ACL*, pages 4102–4112, 2019.

[41] Ziniu Hu, Yuxiao Dong, Kuansan Wang, Kai-Wei Chang, and Yizhou Sun. Gpt-gnn: Generative pre-training of graph neural networks. In *KDD*, pages 1857–1867, 2020.

[42] Yugang Ji, Tianrui Jia, Yuan Fang, and Chuan Shi. Dynamic heterogeneous graph embedding via heterogeneous hawkes process. In *ECML-PKDD*, pages 388–403, 2021.

[43] Jacob Devlin Ming-Wei Chang Kenton and Lee Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. In *NAACL*, pages 4171–4186, 2019.

[44] Diederik P. Kingma and Max Welling. Auto-encoding variational bayes. In *ICLR*, 2014.

[45] Thomas N Kipf and Max Welling. Variational graph auto-encoders. *arXiv*, 2016.

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

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

[48] Nils M Kriege, Matthias Fey, Denis Fisseler, Petra Mutzel, and Frank Weichert. Recognizing cuneiform signs using graph based methods. In *International Workshop on Cost-Sensitive Learning*, pages 31–44, 2018.

[49] Srijan Kumar, Xikun Zhang, and Jure Leskovec. Predicting dynamic embedding trajectory in temporal interaction networks. In *KDD*, pages 1269–1278, 2019.

[50] Jure Leskovec, Jon Kleinberg, and Christos Faloutsos. Graphs over time: densification laws, shrinking diameters and possible explanations. In *KDD*, pages 177–187, 2005.

[51] Jure Leskovec, Kevin J Lang, Anirban Dasgupta, and Michael W Mahoney. Community structure in large networks: Natural cluster sizes and the absence of large well-defined clusters. *Internet Mathematics*, 6(1):29–123, 2009.

[52] Brian Lester, Rami Al-Rfou, and Noah Constant. The power of scale for parameter-efficient prompt tuning. In *EMNLP*, pages 3045–3059, 2021.

[53] Aming Li, Sean P Cornelius, Y-Y Liu, Long Wang, and A-L Barabási. The fundamental advantages of temporal networks. *Science*, 358(6366):1042–1046, 2017.

[54] Guohao Li, Matthias Muller, Ali Thabet, and Bernard Ghanem. Deepgcns: Can gcns go as deep as cnns? In *ICCV*, pages 9267–9276, 2019.

[55] Jundong Li, Harsh Dani, Xia Hu, Jiliang Tang, Yi Chang, and Huan Liu. Attributed network embedding for learning in a dynamic environment. In *CIKM*, pages 387–396, 2017.

[56] Rui Li, Chi Wang, and Kevin Chen-Chuan Chang. User profiling in an ego network: co-profiling attributes and relationships. In *WWW*, pages 819–830, 2014.

[57] Taisong Li, Jiawei Zhang, S Yu Philip, Yan Zhang, and Yonghong Yan. Deep dynamic network embedding for link prediction. *IEEE Access*, 6:29219–29230, 2018.

[58] Xiang Lisa Li and Percy Liang. Prefix-tuning: Optimizing continuous prompts for generation. In *ACL*, pages 4582–4597, 2021.

[59] Hu Linmei, Tianchi Yang, Chuan Shi, Houye Ji, and Xiaoli Li. Heterogeneous graph attention networks for semi-supervised short text classification. In *EMNLP*, pages 4821–4830, 2019.

[60] Jie Liu, Zhicheng He, Lai Wei, and Yalou Huang. Content to node: Self-translation network embedding. In *KDD*, pages 1794–1802, 2018.

[61] Lu Liu, Tianyi Zhou, Guodong Long, Jing Jiang, and Chengqi Zhang. Learning to propagate for graph meta-learning. In *NeurIPS*, pages 1039–1050, 2019.

[62] Pengfei Liu, Weizhe Yuan, Jinlan Fu, Zhengbao Jiang, Hiroaki Hayashi, and Graham Neubig. Pre-train, prompt, and predict: A systematic survey of prompting methods in natural language processing. *arXiv*, 2021.

[63] Xiao Liu, Kaixuan Ji, Yicheng Fu, Weng Tam, Zhengxiao Du, Zhilin Yang, and Jie Tang. P-tuning: Prompt tuning can be comparable to fine-tuning across scales and tasks. In *ACL*, pages 61–68, May 2022.

[64] Xiao Liu, Yanan Zheng, Zhengxiao Du, Ming Ding, Yujie Qian, Zhilin Yang, and Jie Tang. Gpt understands, too. *arXiv*, 2021.

[65] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. Roberta: A robustly optimized bert pretraining approach. *arXiv*, 2019.

[66] Yong Liu, Susen Yang, Chenyi Lei, Guoxin Wang, Haihong Tang, Juyong Zhang, Aixin Sun, and Chunyan Miao. Pre-training graph transformer with multimodal side information for recommendation. In *ACM MM*, pages 2853–2861, 2021b.

[67] Zemin Liu, Yuan Fang, Chenghao Liu, and Steven C.H. Hoi. Node-wise localization of graph neural networks. In *IJCAI*, 2021.

[68] Zemin Liu, Yuan Fang, Chenghao Liu, and Steven C.H. Hoi. Relative and absolute location embedding for few-shot node classification on graph. In *AAAI*, 2021.

[69] Zemin Liu, Wentao Zhang, Yuan Fang, Xinming Zhang, and Steven C.H. Hoi. Towards locality-aware meta-learning of tail node embeddings on networks. In *CIKM*, pages 975–984, 2020.

[70] Yuanfu Lu, Xunqiang Jiang, Yuan Fang, and Chuan Shi. Learning to pre-train graph neural networks. In *AAAI*, volume 35, pages 4276–4284, 2021.

[71] Yuanfu Lu, Xiao Wang, Chuan Shi, Philip S Yu, and Yanfang Ye. Temporal network embedding with micro-and macro-dynamics. In *CIKM*, pages 469–478, 2019.

[72] Mi Luo, Fei Chen, Pengxiang Cheng, Zhenhua Dong, Xiuqiang He, Jiashi Feng, and Zhenguo Li. Metaselector: Meta-learning for recommendation with user-level adaptive model selection. In *The Web Conference*, pages 2507–2513, 2020.

[73] Andrew McCallum, Kamal Nigam, Jason Rennie, and Kristie Seymore. Automating the construction of internet portals with machine learning. *Information Retrieval Journal*, 3:127–163, 2000. www.research.whizbang.com/data.

[74] Andrew Kachites McCallum, Kamal Nigam, Jason Rennie, and Kristie Seymore. Automating the construction of internet portals with machine learning. *Information Retrieval*, 3(2):127–163, 2000.

[75] Hongyuan Mei and Jason M Eisner. The neural hawkes process: A neurally self-modulating multivariate point process. In *KDD*, pages 6754–6764, 2017.

[76] Michael Meyer, Georg Kuschk, and Sven Tomforde. Graph convolutional networks for 3d object detection on radar data. In *ICCV*, pages 3060–3069, 2021.

[77] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *arXiv*, 2013.

[78] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. In *NeurIPS*, pages 3111–3119, 2013.

[79] Sewon Min, Mike Lewis, Hannaneh Hajishirzi, and Luke Zettlemoyer. Noisy channel language model prompting for few-shot text classification. In *ACL*, pages 5316–5330, 2022.

[80] Takeru Miyato, Andrew M. Dai, and Ian J. Goodfellow. Adversarial training methods for semi-supervised text classification. In *ICLR*. OpenReview.net, 2017.

[81] Giang Hoang Nguyen, John Boaz Lee, Ryan A Rossi, Nesreen K Ahmed, Eunyee Koh, and Sungchul Kim. Continuous-time dynamic network embeddings. In *WWW*, pages 969–976, 2018.

[82] Jianmo Ni, Jiacheng Li, and Julian McAuley. Justifying recommendations using distantly-labeled reviews and fine-grained aspects. In *EMNLP*, pages 188–197, 2019.

[83] Pietro Panzarasa, Tore Opsahl, and Kathleen M Carley. Patterns and dynamics of users' behavior and interaction: Network analysis of an online community. *Journal of the American Society for Information Science and Technology*, 60(5):911–932, 2009.

[84] Aldo Pareja, Giacomo Domeniconi, Jie Chen, Tengfei Ma, Toyotaro Suzumura, Hiroki Kanezashi, Tim Kaler, Tao Schardl, and Charles Leiserson. Evolvegcn: Evolving graph convolutional networks for dynamic graphs. In *AAAI*, pages 5363–5370, 2020.

[85] James W Pennebaker, Martha E Francis, and Roger J Booth. *Linguistic inquiry and word count: LIWC 2001*. Mahway: Lawrence Erlbaum Associates, 2001.

[86] Ethan Perez, Florian Strub, Harm de Vries, Vincent Dumoulin, and Aaron C Courville. Film: Visual reasoning with a general conditioning layer. In *AAAI*, pages 3942–3951, 2018.

[87] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. Deepwalk: Online learning of social representations. In *KDD*, pages 701–710, 2014.

[88] Guanghui Qin and Jason Eisner. Learning how to ask: Querying lms with mixtures of soft prompts. In *NAACL*, pages 5203–5212, 2021.

[89] Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, et al. Learning transferable visual models from natural language supervision. In *ICML*, pages 8748–8763. PMLR, 2021.

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

[91] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, Peter J Liu, et al. Exploring the limits of transfer learning with a unified text-to-text transformer. *J. Mach. Learn. Res.*, 21(140):1–67, 2020.

[92] Danilo Jimenez Rezende, Shakir Mohamed, and Daan Wierstra. Stochastic backpropagation and approximate inference in deep generative models. In *ICML*, pages 1278–1286, 2014.

[93] Michael T Rosenstein, Zvika Marx, Leslie Pack Kaelbling, and Thomas G Dietterich. To transfer or not to transfer. In *NeurIPS*, pages 1–4, 2005.

[94] Oscar Sainz, Oier Lopez de Lacalle, Gorka Labaka, Ander Barrena, and Eneko Agirre. Label verbalization and entailment for effective zero and few-shot relation extraction. In *EMNLP*, pages 1199–1212, 2021.

[95] Aravind Sankar, Yanhong Wu, Liang Gou, Wei Zhang, and Hao Yang. DySAT: Deep neural representation learning on dynamic graphs via self-attention networks. In *WSDM*, pages 519–527, 2020.

[96] Adam Santoro, Sergey Bartunov, Matthew Botvinick, Daan Wierstra, and Timothy Lillicrap. Meta-learning with memory-augmented neural networks. In *ICML*, pages 1842–1850, 2016.

[97] Purnamrita Sarkar, Deepayan Chakrabarti, and Michael I Jordan. Nonparametric link prediction in dynamic networks. In *ICML*, pages 1897–1904, 2012.

[98] Timo Schick and Hinrich Schütze. Exploiting cloze-questions for few-shot text classification and natural language inference. In *EACL*, pages 255–269, 2021.

[99] Boon-Siew Seah, Aixin Sun, and Sourav S Bhowmick. Killing two birds with one stone: Concurrent ranking of tags and comments of social images. In *SIGIR*, pages 937–940, 2018.

[100] Rico Sennrich, Barry Haddow, and Alexandra Birch. Neural machine translation of rare words with subword units. In *ACL*, pages 1715–1725. Association for Computational Linguistics (ACL), 2016.

[101] Kanghao Shao, Yunhao Zhang, Yuqi Wen, Zhongnan Zhang, Song He, and Xiaochen Bo. Dti-heta: prediction of drug–target interactions based on gcn and gat on heterogeneous graph. *Briefings in Bioinformatics*, 23(3), 2022.

[102] Taylor Shin, Yasaman Razeghi, Robert L Logan IV, Eric Wallace, and Sameer Singh. Autoprompt: Eliciting knowledge from language models with automatically generated prompts. In *EMNLP*, pages 4222–4235, 2020.

[103] Uriel Singer, Ido Guy, and Kira Radinsky. Node embedding over temporal graphs. In *IJCAI*, pages 4605–4612, 2019.

[104] Jake Snell, Kevin Swersky, and Richard Zemel. Prototypical networks for few-shot learning. In *NeurIPS*, pages 4077–4087, 2017.

[105] Kihyuk Sohn. Improved deep metric learning with multi-class n-pair loss objective. *NeurIPS*, 29, 2016.

[106] Mingchen Sun, Kaixiong Zhou, Xin He, Ying Wang, and Xin Wang. Gppt: Graph pre-training and prompt tuning to generalize graph neural networks. In *KDD*, pages 1717–1727, 2022.

[107] Yi Sun, Yu Zheng, Chao Hao, and Hangping Qiu. Nsp-bert: A prompt-based zero-shot learner through an original pre-training task–next sentence prediction. *arXiv*, 2021.

[108] Qiuling Suo, Jingyuan Chou, Weida Zhong, and Aidong Zhang. Tadanet: Task-adaptive network for graph-enriched meta-learning. In *KDD*, pages 1789–1799, 2020.

[109] Jeffrey J Sutherland, Lee A O'brien, and Donald F Weaver. Spline-fitting with a genetic algorithm: A method for developing classification structure-activity relationships. *Journal of Chemical Information and Computer Sciences*, 43(6):1906–1915, 2003.

[110] Zhixing Tan, Xiangwen Zhang, Shuo Wang, and Yang Liu. Msp: Multi-stage prompting for making pre-trained language models better translators. In *EMNLP*, pages 6131–6142, 2022.

[111] Jian Tang, Meng Qu, Mingzhe Wang, Ming Zhang, Jun Yan, and Qiaozhu Mei. Line: Large-scale information network embedding. In *WWW*, pages 1067–1077, 2015.

[112] Yonglong Tian, Yue Wang, Dilip Krishnan, Joshua B Tenenbaum, and Phillip Isola. Rethinking few-shot image classification: a good embedding is all you need? In *ECCV*, pages 266–282. Springer, 2020.

[113] Eleni Triantafillou, Tyler Zhu, Vincent Dumoulin, Pascal Lamblin, Utku Evci, Kelvin Xu, Ross Goroshin, Carles Gelada, Kevin Swersky, Pierre-Antoine Manzagol, et al. Meta-dataset: a dataset of datasets for learning to learn from few examples. In *ICLR*, 2019.

[114] Rakshit Trivedi, Mehrdad Farajtabar, Prasenjeet Biswal, and Hongyuan Zha. DyRep: Learning representations over dynamic graphs. In *ICLR*, 2019.

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

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

[117] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. Graph attention networks. In *ICLR*, 2018.

[118] Petar Velickovic, William Fedus, William L Hamilton, Pietro Liò, Yoshua Bengio, and R Devon Hjelm. Deep graph infomax. *ICLR*, 2(3):4, 2019.

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

[120] Oriol Vinyals, Charles Blundell, Timothy Lillicrap, Daan Wierstra, et al. Matching networks for one shot learning. *NeurIPS*, 29, 2016.

[121] Guoyin Wang, Chunyuan Li, Wenlin Wang, Yizhe Zhang, Dinghan Shen, Xinyuan Zhang, Ricardo Henao, and Lawrence Carin. Joint embedding of words and labels for text classification. In *ACL*, pages 2321–2331, 2018.

[122] Hongwei Wang, Jia Wang, Jialin Wang, Miao Zhao, Weinan Zhang, Fuzheng Zhang, Xing Xie, and Minyi Guo. GraphGAN: Graph representation learning with generative adversarial nets. In *AAAI*, pages 2508–2515, 2018.

[123] Hongwei Wang, Miao Zhao, Xing Xie, Wenjie Li, and Minyi Guo. Knowledge graph convolutional networks for recommender systems. In *The Web Conference*, pages 3307–3313, 2019.

[124] Juexin Wang, Anjun Ma, Qin Ma, Dong Xu, and Trupti Joshi. Inductive inference of gene regulatory network using supervised and semi-supervised graph neural networks. *Computational and structural biotechnology journal*, 18:3335–3343, 2020.

[125] Ning Wang, Minnan Luo, Kaize Ding, Lingling Zhang, Jundong Li, and Qinghua Zheng. Graph few-shot learning with attribute matching. In *CIKM*, pages 1545–1554, 2020.

[126] Xiang Wang, Xiangnan He, Meng Wang, Fuli Feng, and Tat-Seng Chua. Neural graph collaborative filtering. In *SIGIR*, pages 165–174, 2019.

[127] Yanbang Wang, Yen-Yu Chang, Yunyu Liu, Jure Leskovec, and Pan Li. Inductive representation learning in temporal networks via causal anonymous walks. In *ICLR*, 2021.

[128] Zheng Wang, Jialong Wang, Yuchen Guo, and Zhiguo Gong. Zero-shot node classification with decomposed graph prototype network. In *KDD*, pages 1769–1779, 2021.

[129] Zhihao Wen and Yuan Fang. Trend: Temporal event and node dynamics for graph representation learning. In *The Web Conference*, pages 1159–1169, 2022.

[130] Zhihao Wen, Yuan Fang, and Zemin Liu. Meta-inductive node classification across graphs. In *SIGIR*, pages 1219–1228, 2021.

[131] Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, et al. Transformers: State-of-the-art natural language processing. In *EMNLP*, pages 38–45, 2020.

[132] Felix Wu, Amauri H Souza Jr, Tianyi Zhang, Christopher Fifty, Tao Yu, and Kilian Q Weinberger. Simplifying graph convolutional networks. In *ICML*, pages 6861–6871, 2019.

[133] Lirong Wu, Haitao Lin, Cheng Tan, Zhangyang Gao, and Stan Z Li. Self-supervised learning on graphs: Contrastive, generative, or predictive. *TKDE*, 2021.

[134] Xiao-Ming Wu, Zhenguo Li, Anthony M So, John Wright, and Shih-Fu Chang. Learning with partially absorbing random walks. In *NeurIPS*, pages 3086–3094, 2012.

[135] Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and S Yu Philip. A comprehensive survey on graph neural networks. *TNNLS*, 32(1):4–24, 2020.

[136] Qizhe Xie, Zihang Dai, Eduard Hovy, Thang Luong, and Quoc Le. Unsupervised data augmentation for consistency training. *NeurIPS*, 33:6256–6268, 2020.

[137] Da Xu, Chuanwei Ruan, Evren Korpeoglu, Sushant Kumar, and Kannan Achan. Inductive representation learning on temporal graphs. In *ICLR*, 2020.

[138] Hu Xu, Bing Liu, Lei Shu, and P Yu. Open-world learning and application to product classification. In *The Web Conference*, pages 3413–3419, 2019.

[139] Kelvin Xu, Jimmy Ba, Ryan Kiros, Kyunghyun Cho, Aaron Courville, Ruslan Salakhudinov, Rich Zemel, and Yoshua Bengio. Show, attend and tell: Neural image caption generation with visual attention. In *ICML*, pages 2048–2057, 2015.

[140] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? In *ICLR*, 2019.

[141] Jianwei Yang, Jiasen Lu, Stefan Lee, Dhruv Batra, and Devi Parikh. Graph r-cnn for scene graph generation. In *ECCV*, pages 670–685, 2018.

[142] Zhilin Yang, William Cohen, and Ruslan Salakhudinov. Revisiting semi-supervised learning with graph embeddings. In *ICML*, pages 40–48. PMLR, 2016.

[143] Zhilin Yang, Zihang Dai, Yiming Yang, Jaime Carbonell, Russ R Salakhutdinov, and Quoc V Le. Xlnet: Generalized autoregressive pretraining for language understanding. *NeurIPS*, 32, 2019.

[144] Huaxiu Yao, Ying Wei, Junzhou Huang, and Zhenhui Li. Hierarchically structured meta-learning. In *ICML*, pages 7045–7054, 2019.

[145] Liang Yao, Chengsheng Mao, and Yuan Luo. Graph convolutional networks for text classification. In *AAAI*, volume 33, pages 7370–7377, 2019.

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

[147] Mo Yu, Xiaoxiao Guo, Jinfeng Yi, Shiyu Chang, Saloni Potdar Yu Cheng Gerald Tesauro, Haoyu Wang Bowen Zhou, and AI Foundations-Learning. Diverse few-shot text classification with multiple metrics. In *NAACL*, pages 1206–1215, 2018.

[148] Hanqing Zeng, Hongkuan Zhou, Ajitesh Srivastava, Rajgopal Kannan, and Viktor Prasanna. Graphsaint: Graph sampling based inductive learning method. In *ICLR*, 2020.

[149] Jing Zhang, Biao Liu, Jie Tang, Ting Chen, and Juanzi Li. Social influence locality for modeling retweeting behaviors. In *IJCAI*, volume 13, pages 2761–2767, 2013.

[150] Wenxuan Zhang, Yang Deng, Xin Li, Yifei Yuan, Lidong Bing, and Wai Lam. Aspect sentiment quad prediction as paraphrase generation. In *EMNLP*, pages 9209–9219, 2021.

[151] Xi Sheryl Zhang, Fengyi Tang, Hiroko H Dodge, Jiayu Zhou, and Fei Wang. Metapred: Meta-learning for clinical risk prediction with limited patient electronic health records. In *KDD*, pages 2487–2495, 2019.

[152] Yuhao Zhang, Hang Jiang, Yasuhide Miura, Christopher D Manning, and Curtis P Langlotz. Contrastive learning of medical visual representations from paired images and text. *arXiv*, 2020.

[153] Zexuan Zhong, Dan Friedman, and Danqi Chen. Factual probing is [mask]: Learning vs. learning to recall. In *NAACL*, pages 5017–5033, 2021.

[154] Dengyong Zhou, Olivier Bousquet, Thomas Navin Lal, Jason Weston, and Bernhard Schölkopf. Learning with local and global consistency. In *NeurIPS*, pages 321–328, 2003.

[155] Fan Zhou, Chengtai Cao, Kunpeng Zhang, Goce Trajcevski, Ting Zhong, and Ji Geng. Meta-gnn: On few-shot node classification in graph meta-learning. In *CIKM*, pages 2357–2360, 2019.

[156] Kaiyang Zhou, Jingkang Yang, Chen Change Loy, and Ziwei Liu. Learning to prompt for vision-language models. *IJCV*, 130(9):2337–2348, 2022.

[157] Lekui Zhou, Yang Yang, Xiang Ren, Fei Wu, and Yueting Zhuang. Dynamic network embedding by modeling triadic closure process. In *AAAI*.

[158] Dingyuan Zhu, Peng Cui, Ziwei Zhang, Jian Pei, and Wenwu Zhu. High-order proximity preserved embedding for dynamic networks. *IEEE Transactions on Knowledge and Data Engineering*, 30(11):2134–2144, 2018.

[159] Xiaojin Zhu, Zoubin Ghahramani, and John Lafferty. Semi-supervised learning using Gaussian fields and harmonic functions. In *ICML*, pages 912–919, 2003.

[160] Yuan Zuo, Guannan Liu, Hao Lin, Jia Guo, Xiaoqian Hu, and Junjie Wu. Embedding temporal network via neighborhood formation. In *KDD*, pages 2857–2866, 2018.

Appendix A

Appendices

A.1 Links to the three proposed models

The three proposed models are all open-sourced, and their links are as follows:

- MI-GNN: https://github.com/WenZhihao666/MI-GNN
- TREND: https://github.com/WenZhihao666/TREND
- G2P2: https://github.com/WenZhihao666/G2P2

A.2 Connection between transfer function and conditional intensity of TREND

In the following, we show that a well chosen transfer function f, taking the temporal representations as input, is equivalent to the conditional intensity of the Hawkes process in Eq. (4.2). Suppose the temporal representations are generated through stacking l temporal GNN layers. First, let us define the base intensity as

a function of the self-information:

$$\mu_{i,j}(t) = f_\mu(\mathbf{h}_i^{t,l-1}\mathbf{W}_{\text{self}}^l, \mathbf{h}_j^{t,l-1}\mathbf{W}_{\text{self}}^l). \tag{A.1}$$

Next, we define the amount of excitement induced by a historical neighbor as a function of the historical neighbors' information:

$$\gamma_{j'}(t') = f_\gamma(\mathbf{h}_{j'}^{t',l-1}\mathbf{W}_{\text{hist}}^l), \quad \gamma_{i'}(t') = f_\gamma(\mathbf{h}_{i'}^{t',l-1}\mathbf{W}_{\text{hist}}^l). \tag{A.2}$$

Given these building blocks, we rewrite the conditional intensity in Eq. (4.2) as

$$\begin{aligned}\lambda_{i,j}(t) = f_\lambda\Big(&\mathbf{h}_i^{t,l-1}\mathbf{W}_{\text{self}}^l, \{\mathbf{h}_{j'}^{t',l-1}\mathbf{W}_{\text{hist}}^l : (i,j',t') \in \mathcal{H}_i(t)\},\\ &\mathbf{h}_j^{t,l-1}\mathbf{W}_{\text{self}}^l, \{\mathbf{h}_{i'}^{t',l-1}\mathbf{W}_{\text{hist}}^l : (i',j,t') \in \mathcal{H}_j(t)\}\Big),\end{aligned} \tag{A.3}$$

where f_λ is a composite function of f_μ, f_γ and the summation. By choosing the right transfer function f, we further rewrite f_λ as the composition of f and the temporal GNN layer f_g given in Eq. (4.4), *i.e.*, $f_\lambda = f \circ f_g$. Subsequently, the conditional intensity is given by

$$\begin{aligned}\lambda_{i,j}(t) &= (f \circ f_g)\Big(\mathbf{h}_i^{t,l-1}\mathbf{W}_{\text{self}}^l, \{\mathbf{h}_{j'}^{t',l-1}\mathbf{W}_{\text{hist}}^l : (i,j',t') \in \mathcal{H}_i(t)\},\\ &\qquad \mathbf{h}_j^{t,l-1}\mathbf{W}_{\text{self}}^l, \{\mathbf{h}_{i'}^{t',l-1}\mathbf{W}_{\text{hist}}^l : (i',j,t') \in \mathcal{H}_j(t)\}\Big)\\ &= f(\mathbf{h}_i^t, \mathbf{h}_j^t).\end{aligned} \tag{A.4}$$

Thus, a well-fit transfer function f, such as a neural network, can approximate the conditional intensity.

A.3 Pseudocode of TREND

We outline the training procedure of TREND in Algorithm 3.

Algorithm 3 Training Procedure of TREND

Input: Training graph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathcal{T}, \mathbf{X})$, training events $\mathcal{I}^{\text{tr}}$.
Output: Temporal GNN θ_g, event prior θ_e, transformation model θ_τ, estimator of node dynamics θ_n.

$\theta_g, \theta_e, \theta_\tau, \theta_n \leftarrow$ parameters initialization;
while not converged **do**
  sample a batch of temporal events (i, j, t) from $\mathcal{I}^{\text{tr}}$;
  for each event (i, j, t) in the batch **do**
    calculate node representations $\mathbf{h}_i^t$, $\mathbf{h}_j^t$for nodes i,j; ▷ Eq. (4.4)
    $\theta_e^{(i,j,t)} \leftarrow$ event-conditioned adaptation on θ_e; ▷ Eq. (4.10)
    calculate event intensity $\lambda_{i,j}(t)$; ▷ Eq. (4.7)
    calculate the overall loss; ▷ Eqs. (4.11), (4.13), (4.14)
  end for
  $\theta_g, \theta_e, \theta_\tau, \theta_n \leftarrow$ backpropagation of overall loss ▷ Eq. (4.14)
end while
return $\theta_g, \theta_e, \theta_\tau, \theta_n$.

A.4 Additional Description of Datasets of TREND

We include more details of the datasets below.

- CollegeMsg [83] is an online social network where private messages were sent and received at the University of California, Irvine. If user i sent a private message to user j at time t, there is a temporal edge (i, j, t). Since the nodes have no feature, we use the one-hot encoding of the node ID as node features.

- cit-HepTh [50] is a citation graph about high energy physics theory from the e-print arXiv, in the period from January 1993 to April 2003. A temporal edge (i, j, t) here means a paper i cites paper j at time t. We use word2vec [78] to convert the text of paper abstract (*i.e.*, the raw node features) into node embedding as the node feature.

- Wikipedia [49] is a graph in which temporal edges are interactions induced by

users' editing on the Wikipedia pages in one month. User edits consist of textual features, which are converted into 172-dimensional LIWC [85] feature vectors. The edit vectors of each user are added and normalized to serve as the node feature.

- Taobao [18] is a quite large online purchase network on the e-commerce platform taobao.com. If a user i purchased an item j at time t, there is a temporal edge (i, j, t). Node features are preprocessed embeddings of textual features.

A.5 Details of Task Setup of TREND

We describe more details of our main task, namely, temporal link prediction. For each temporal graph, node representations are learnt on the graph consisting of events before time t^{tr}, and we try to predict events on or after t^{tr}. In our experiments, we use all the events before the last time step for training, and test on the events at the last time step. For instance, on the graph cit-HepTh, we train the model only using events before the 78^{th} time step, and we predict the links formed at the 78^{th} time step. At test time, a logistic regression classifier is trained for the downstream task of temporal link prediction. While links formed at the last time step are our positive examples, we further randomly sample an equal number of negative examples (*i.e.*, node pairs which do not form a link at the last time step). We define the feature vector of a candidate triple (i, j, t) as $|\mathbf{h}_i^t - \mathbf{h}_j^t|$ [71].

A.6 Additional Description of Baselines of TREND

We include more details of the baselines below.

(1) *Static approaches*, in which models or node embedding vectors are

trained on the static graph formed before the testing time, regardless of the temporal information.

- DeepWalk [87]: a static network embedding method, which regards the random walk sequences as sentences and leverages skip-grams [78] to learn node embeddings.
- Node2vec [28]: another static network embedding method, which generalizes DeepWalk with biased random walks.
- VGAE [45]: based on variational auto-encoder (VAE) [44, 92], it is a classical GNN-based link prediction model, using a GCN [46] encoder and an inner product decoder.
- GAE [45]: a non-probabilistic variant of the VGAE model.
- GraphSAGE [31]: a GNN model on static graphs, which supports inductive representation learning on large graphs.

(2) *Temporal approaches*, which train models or node embedding vectors on the temporal graph formed before the testing time.

- CTDNE [81]: based on random walks, it is a network embedding method which learns time-respecting embedding from continuous-time dynamic networks.
- EvolveGCN [84]: using RNN to evolve GCN parameters to capture the dynamic information of sequences of static graph snapshots.
- GraphSAGE+T: our implementation based on GraphSAGE. Specifically, when aggregating neighbors' information, it will consider the time decay effect, *i.e.*, earlier neighbors will get smaller weights while more recent neighbors will get larger weights during aggregation.

- TGAT [137]: it uses the self-attention mechanism to aggregate temporal-topological neighborhood features. Besides, based on Bochner's theorem, it encodes the event time as part of the node embedding vector.

(3) *Hawkes process-based approaches*, which train node embedding vectors using the temporal graph formed before the testing time, based on Hawkes process.

- HTNE [160]: a network embedding method which integrates the Hawkes process into network embedding so as to capture the influence of historical neighbors on the current neighbors.
- MMDNE [71]: a network embedding method with micro- and macro-dynamics. Specifically, the micro-dynamics describe the link formation process, while the macro-dynamics refer to the evolution pattern of the network scale.

For DeepWalk, Node2vec and CTDNE, we set their random walk sampling parameters, such as number of walks, walk length and window size according to their recommended settings, respectively. For all network embedding methods, the node embedding dimension is 128, which tends to perform well empirically. For all GNN-based methods, we set the number of layers, the node embedding dimensions and the learning rates to the same with our model TREND. For HTNE, MMDNE and EvolveGCN, we set their historical window size to 5 (*i.e.*, only use the 5 most recent neighbors), given the empirical performance and efficiency considerations. For efficiency reasons, we perform random neighborhood sampling on GraphSAGE and GraphSAGE+T, setting the sample size to 10 on CollegeMsg and cit-HepTh, 20 on Wikipedia, and 5 on Taobao (which is the most sparse graph), the same with TREND; we use the same sample size on TGAT, but sample the more recent neighbors with higher probability following its original design. For EvolveGCN, we use the EvolveGCN-O version. For TGAT, we set the number of attention heads to 3. Other hyperparameters are chosen empirically, following guidance from literature.

A.7 Scalability Study of TREND

On Taobao (with a total of more than 4 million events), we extract different ratios (20%–100%) of training events to form 5 subgraphs, and record the training time per epoch on each subgraph. In Fig. A.1, the training time grows linearly in the number of training events, which is consistent with our complexity analysis, and implies that the proposed model is scalable.

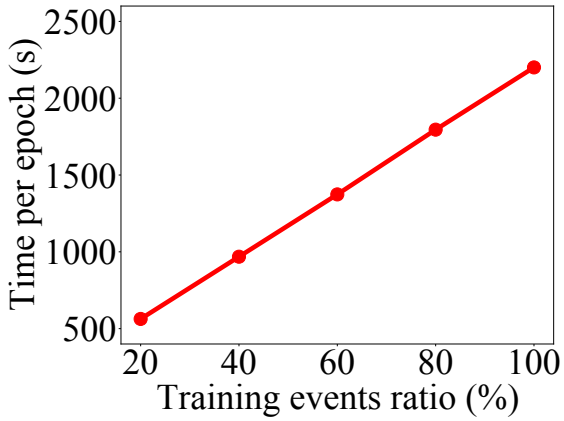


Figure A.1: Time complexity.