Meta-Inductive Node Classification across Graphs

Zhihao Wen
Singapore Management University
Singapore
zhwen.2019@smu.edu.sg

Yuan Fang
Singapore Management University
Singapore
yfang@smu.edu.sg

Zemin Liu
Singapore Management University
Singapore
zmliu@smu.edu.sg

ABSTRACT

Semi-supervised node classification on graphs is an important research problem, with many real-world applications in information retrieval such as content classification on a social network and query intent classification on an e-commerce query graph. While traditional approaches are largely transductive, recent graph neural networks (GNNs) integrate node features with network structures, thus enabling inductive node classification models that can be applied to new nodes or even new graphs in the same feature space. However, inter-graph differences still exist across graphs within the same domain. Thus, training just one global model (*e.g.*, a state-of-the-art GNN) to handle all new graphs, whilst ignoring the inter-graph differences, can lead to suboptimal performance.

In this paper, 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. More specifically, we learn a *graph prior* to adapt for the graph-level differences, and a *task prior* to adapt for the task-level differences conditioned on a graph. Extensive experiments on five real-world graph collections demonstrate the effectiveness of our proposed model.

CCS CONCEPTS

• **Computing methodologies** → **Learning latent representations**; • **Information systems** → **Data mining**.

KEYWORDS

Graph neural networks, semi-supervised node classification, inductive graph model, meta-learning

ACM Reference Format:
Zhihao Wen, Yuan Fang, and Zemin Liu. 2021. Meta-Inductive Node Classification across Graphs. In *Proceedings of the 44th International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR '21), July 11–15, 2021, Virtual Event, Canada.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3404835.3462915

SIGIR '21, July 11–15, 2021, Virtual Event, Canada

ACM ISBN 978-1-4503-8037-9/21/07…$15.00
https://doi.org/10.1145/3404835.3462915

1 INTRODUCTION

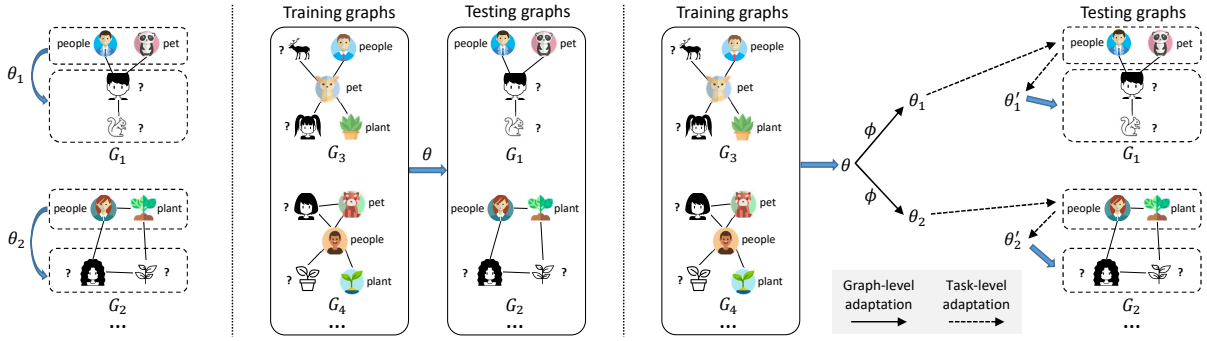
Graph-structured data widely exist in diverse real-world scenarios, such as social networks, e-commerce graphs, citation graphs, and biological networks. Analysis of these graphs can uncover valuable insights about their respective application domain. In particular, semi-supervised node classification on graphs [2] 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 [34]; on a query graph for e-commerce, query intent classification enhances the ranking of results by focusing on the intended product category [10]. 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 [2, 9, 44, 54, 56] 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 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 [17], 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 [7, 22, 53] 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.

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 [4] and graph neural networks (GNNs) [45].

On one hand, network embedding methods directly parameterize node embedding vectors and constrain them with various local



(a) Transductive approach (b) Conventional inductive approach (c) Our meta-inductive approach (MI-GNN)

Figure 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.)

structures, such as random walks in DeepWalk [31] and node2vec [14], and first- and second-order proximity in LINE [38]. 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 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 [52]. 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 [33, 42], 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 [23], texts [19] and graphs [55], 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 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 [35] and MAML

[11], 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 superior inductive ability of the proposed MI-GNN.

2 RELATED WORK

We investigate related work in three groups: graph neural networks, inductive graph representation learning and meta-learning.

Graph neural networks. A surge of attention has been attracted to graph neural networks (GNNs) [45]. Based on they key operation of neighborhood aggregation in a message passing framework, they exploit the underlying graph structure and node features simultaneously. In particular, different message aggregation functions materialize different GNNs, *e.g.*, GCN [20] employs an aggregation roughly equivalent to mean pooling, and GAT [40] employs self-attention to aggregate neighbors in a weighted manner. Recent works often exploit more structural information on graphs, such as graph isomorphism [47] and node positions [51].

Inductive graph representation learning. Recent inductive learning on graphs are 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 [8, 29, 57], by exploiting structural information from the graph such as co-occurrence between nodes; others employ graph auto-encoders [12, 13] 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 [16, 46]. In this paper, we follow the line of inductive learning based on GNNs. More recently, a few pre-training approaches have also been devised for GNNs [18, 27, 41]. 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 ours.

Meta-learning. Also known as "learning to learn," meta-learning [11, 35] aims to simulate the learning strategy of humans, by learning some general knowledge across a set of learning tasks and adapting the knowledge to novel tasks. Generally, some approaches resort to protonets [35] which aim to learn a metric space for the class prototypes, and others apply optimization-based techniques such as model-agnostic meta-learning (MAML) [11]. To further enhance task adaptation, the transferable knowledge can be tailored to different clusters of tasks, forming a hierarchical structure of adaptation [49]; feature-specific memories can also guide the adapted model with a further bias [6]; domain-knowledge graphs can also be leveraged to provide task-specific customization [36]. Another subtype of meta-learning called hypernetwork [15, 30] 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 [55], GFL [50], GPN [5], RALE [25] and meta-tail2vec [26], which is distinct from inductive semi-supervised node classification as further elaborated in Section 3.1. Hypernetwork-based approaches have also emerged, such as LGNN [24] that adapts GNN weights to different local contexts, and GNN-FiLM [3] that adapts to different relations in a relational graph.

Table 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 $V \rightarrow C$
$\mathcal{N}_v$	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

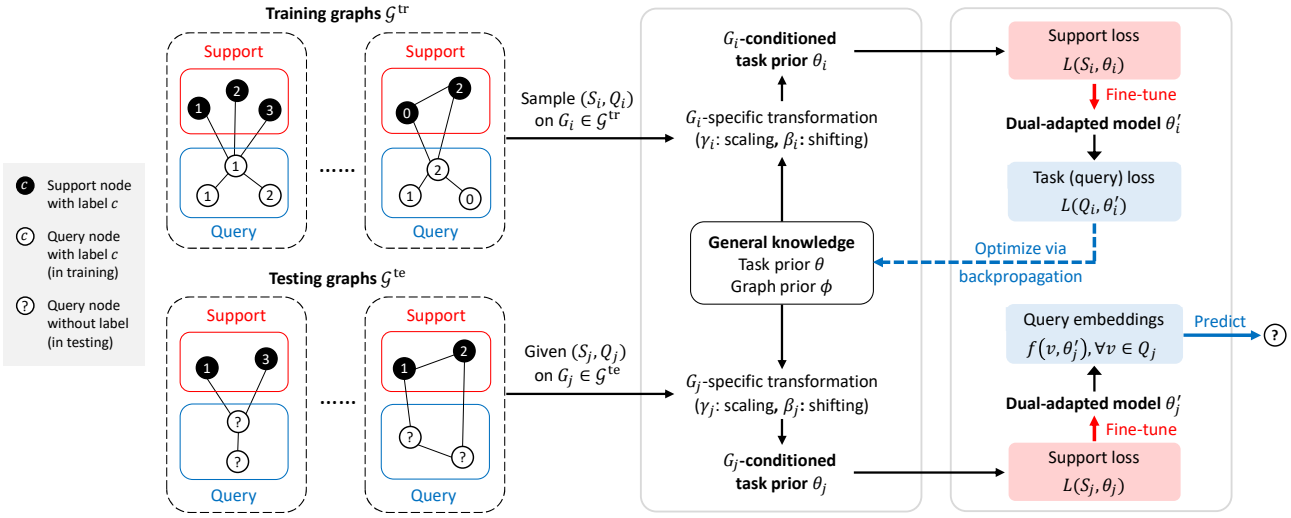
3 PRELIMINARIES

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 1.

3.1 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.*, $\ell : V \rightarrow C$. 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



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

Figure 2: Overall framework of MI-GNN, illustrating the pipeline on a training graph G_i and a testing graph G_j.

$G \in \mathcal{G}^{te}$, 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 [23, 50, 55] 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.

3.2 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,

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

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 [16, 20, 43] to more complex mechanisms [16, 40, 47]. Our proposed model is flexible in the aggregation functions.

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.

4.1 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, 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 2.

First of all, MI-GNN exploits each training graph to simulate the semi-supervised node classification task in testing, as shown in Figure 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 [11]. 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 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 2(c).

4.2 Graphs and tasks

Training tasks. We refer to the upper half of Figure 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 $\{v_{i,k} : 1 \le k \le m+n\}$ on G_i^{tr}, the support and query sets are given by

$$S_i^{\text{tr}} = \{(v_{i,k}, \ell(v_{i,k})) : 1 \le k \le m\}, \tag{2}$$

$$Q_i^{\text{tr}} = \{(v_{i,m+k}, \ell(v_{i,m+k})) : 1 \le k \le n\}, \tag{3}$$

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 $\{v_{j,k} : 1 \le k \le m+n\}$ such that nodes are labeled for $1 \le k \le m$ only and unlabeled for $m+1 \le k \le m+n$. The support and query sets are then given by

$$S_j^{\text{te}} = \{(v_{j,k}, \ell(v_{j,k})) : 1 \le k \le m\}, \tag{4}$$

$$Q_j^{\text{te}} = \{v_{j,m+k} : 1 \le k \le n\}. \tag{5}$$

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 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).

4.3 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 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.*,

$$\theta = (\mathbf{W}^1, \mathbf{W}^2, ...), \tag{6}$$

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 3.2.

Different from most task-based meta learning [11, 35], 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

$$\tau(\theta, \mathbf{g}; \phi), \tag{7}$$

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 [15, 30], 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 [30] 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 [30]. 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,

$$\gamma_i = \text{MLP}_\gamma(\mathbf{g}_i; \phi_\gamma), \tag{8}$$

$$\beta_i = \text{MLP}_\beta(\mathbf{g}_i; \phi_\beta), \tag{9}$$

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

$$\theta_i = \tau(\theta, \mathbf{g}_i; \phi) = (\gamma_i + \mathbf{1}) \odot \theta + \beta_i, \tag{10}$$

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.*,

$$\phi = \{\phi_\gamma, \phi_\beta\}. \tag{11}$$

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. (8)–(9) in response to the changing input graph. In particular, the two MLPs play the role of secondary networks in the hypernetwork setting [15].

4.4 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 [11], 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 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. (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.

$$\theta_i' = \theta_i - \alpha \frac{\partial L(S_i, \theta_i)}{\partial \theta_i}, \tag{12}$$

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.

$$L(S_i, \theta_i) = -\sum_{(v_{i,k}, \ell(v_{i,k})) \in S_i} \sum_{c \in C} I(\ell(v_{i,k}) = c) \log f(v_{i,k}; \theta_i)[c], \tag{13}$$

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.

4.5 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

$$\arg\min_{\theta,\phi} \sum_{G_i \in \mathcal{G}^{\text{tr}}} L(Q_i, \theta_i') + \lambda(\|\gamma_i\|_2 + \|\beta_i\|_2), \tag{14}$$

where (1) θ_i' is the dual-adapted prior after performing one gradient update according to Eq. (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. (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 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.

Algorithm 1 TrainingProcedure

Input: training graph set $\mathcal{G}^{\text{tr}}$.
Output: 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. (8), (9)
    $\theta_i \leftarrow$ graph-level adaptation on θ; ▷ Eq. (10)
    calculate support loss $L(S_i, \theta_i)$ and gradient; ▷ Eq. (13)
    $\theta_i' \leftarrow$ task-level adaptation on θ_i; ▷ Eq. (12)
    calculate task (query) loss $L(Q_i, \theta_i')$;
  end for
  $\theta, \phi \leftarrow$ backpropagation of total task loss ▷ Eq. (14)
end while
return θ, ϕ.

Table 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

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 $\arg\max_{c \in C} f(v_{j,k}; \theta_j')[c]$.

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.

5.1 Experimental setup

Datasets. We conduct experiments on five public graph collections, as follows. Their statistics are summarized in Table 2.

- Flickr [52] 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 [52] 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 [21] 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 [37] 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.

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) [43] in all of our experiments, except in Section 5.3 where we also adopt GCN [20] and GraphSAGE [16] 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. (8) and (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. (12) to 0.5 for Flickr, Yelp and Cuneiform or 0.005 for COX2 and DHFR. Lastly, for the overall optimization in Eq. (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 [31]: an unsupervised network embedding method that learns node representations based on the skip-gram model [28] 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 [48]: 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 [39]: 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 [39]: 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 [50]: a few-shot node classification method on graphs, based on protonets [35]. 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 [55]: another few-shot node classification approach on graphs, based on MAML [11].

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.

5.2 Performance comparison to baselines

In Table 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

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

Table 4: Accuracy of MI-GNN and baselines using alternative GNN architectures, in percent, with 95% confidence intervals.

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

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 exception 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.

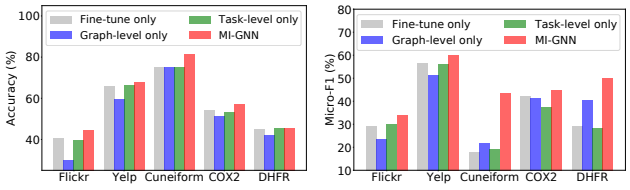
5.3 Alternative GNN architectures

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 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.

5.4 Effect of dual adaptations

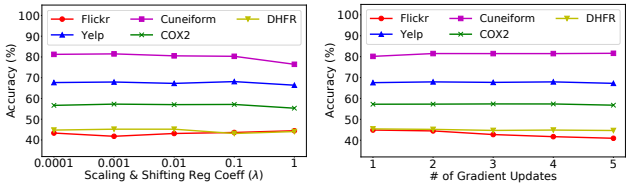
The advantage of our approach MI-GNN stems from the dual adaptations at the graph and task levels. To investigate the contribution from each level of adaptation, we perform an ablation study on MI-GNN, comparing with the following variants. (1) *Fine-tune only*: A standard inductive GNN model without 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. First of all, MI-GNN outperforms all the ablated models consistently, demonstrating the overall benefit of the dual 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



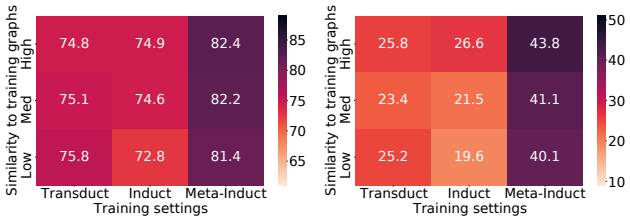
(a) Accuracy (b) Micro-F1

Figure 3: Effect of dual adaptations.



(a) Regularization (b) Gradient steps

Figure 4: Impact of regularization and gradient steps.



(a) Accuracy (b) Micro-F1

Figure 5: Performance w.r.t. similarity to training graphs.

not give any significant benefit over a simple fine-tuning step but together they work much better.

5.5 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 4(a), as controlled by the co-efficient λ in Eq. (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.

Number of gradient steps in task-level adaptation. As discussed in Section 4.4, 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 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.

5.6 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 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 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 [32]. 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.

6 CONCLUSION

In this paper, 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.

ACKNOWLEDGMENTS

This research is supported by the Agency for Science, Technology and Research (A*STAR) under its AME Programmatic Funds (Grant No. A20H6b0151).

REFERENCES

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

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

[3] Marc Brockschmidt. 2020. GNN-FiLM: Graph neural networks with feature-wise linear modulation. In *ICML*. 1144–1152.

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

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

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

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

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

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

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

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

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

[13] Palash Goyal, Nitin Kamra, Xinran He, and Yan Liu. 2018. DynGEM: Deep embedding method for dynamic graphs. *arXiv preprint arXiv:1805.11273* (2018).

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

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

[16] Will Hamilton, Zhitao Ying, and Jure Leskovec. 2017. Inductive representation learning on large graphs. In *NeurIPS*. 1024–1034.

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

[18] Weihua Hu, Bowen Liu, Joseph Gomes, Marinka Zitnik, Percy Liang, Vijay Pande, and Jure Leskovec. 2020. Strategies for pre-training graph neural networks. In *ICLR*.

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

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

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

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

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

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

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

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

[27] Yuanfu Lu, Xunqiang Jiang, Yuan Fang, and Chuan Shi. 2021. Learning to pre-train graph neural networks. In *AAAI*.

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

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

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

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

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

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

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

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

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

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

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

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

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

[41] Petar Velickovic, William Fedus, William L Hamilton, Pietro Lio, Yoshua Bengio, and R Devon Hjelm. 2019. Deep graph infomax. In *ICLR*.

[42] Oriol Vinyals, Charles Blundell, Timothy Lillicrap, Daan Wierstra, et al. 2016. Matching networks for one shot learning. In *NIPS*. 3630–3638.

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

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

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

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

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

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

[49] Huaxiu Yao, Ying Wei, Junzhou Huang, and Zhenhui Li. 2019. Hierarchically Structured Meta-learning. In *ICML*. 7045–7054.

[50] Huaxiu Yao, Chuxu Zhang, Ying Wei, Meng Jiang, Suhang Wang, Junzhou Huang, Nitesh V Chawla, and Zhenhui Li. 2020. Graph few-shot learning via knowledge transfer. In *AAAI*. 6656–6663.

[51] Jiaxuan You, Rex Ying, and Jure Leskovec. 2019. Position-aware graph neural networks. In *ICML*. 7134–7143.

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

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

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

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

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

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