Task-Adaptive Few-shot Node Classification

Song Wang
University of Virginia
sw3wv@virginia.edu

Kaize Ding
Arizona State University
kding9@asu.edu

Chuxu Zhang
Brandeis University
chuxuzhang@brandeis.edu

Chen Chen
University of Virginia
zrh6du@virginia.edu

Jundong Li
University of Virginia
jundong@virginia.edu

ABSTRACT

Node classification is of great importance among various graph mining tasks. In practice, real-world graphs generally follow the long-tail distribution, where a large number of classes only consist of limited labeled nodes. Although Graph Neural Networks (GNNs) have achieved significant improvements in node classification, their performance decreases substantially in such a few-shot scenario. The main reason can be attributed to the vast generalization gap between meta-training and meta-test due to the task variance caused by different node/class distributions in meta-tasks (i.e., node-level and class-level variance). Therefore, to effectively alleviate the impact of task variance, we propose a task-adaptive node classification framework under the few-shot learning setting. Specifically, we first accumulate meta-knowledge across classes with abundant labeled nodes. Then we transfer such knowledge to the classes with limited labeled nodes via our proposed task-adaptive modules. In particular, to accommodate the different node/class distributions among meta-tasks, we propose three essential modules to perform *node-level*, *class-level*, and *task-level* adaptations in each meta-task, respectively. In this way, our framework can conduct adaptations to different meta-tasks and thus advance the model generalization performance on meta-test tasks. Extensive experiments on four prevalent node classification datasets demonstrate the superiority of our framework over the state-of-the-art baselines. Our code is provided at https://github.com/SongW-SW/TENT.

CCS CONCEPTS

• **Computing methodologies** → **Transfer learning**.

KEYWORDS

node classification; few-shot learning; graph neural networks

ACM Reference Format:
Song Wang, Kaize Ding, Chuxu Zhang, Chen Chen, and Jundong Li. 2022. Task-Adaptive Few-shot Node Classification. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD '22), August 14–18, 2022, Washington, DC, USA.* , 10 pages. https://doi.org/10.1145/3534678.3539265

KDD '22, August 14–18, 2022, Washington, DC, USA

ACM ISBN 978-1-4503-9385-0/22/08.
https://doi.org/10.1145/3534678.3539265

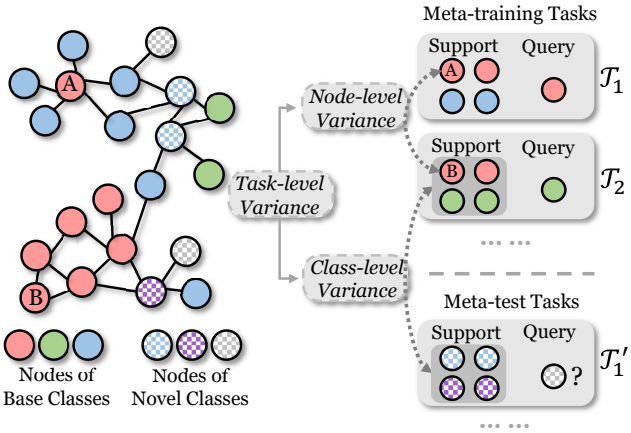

Figure 1: Issues of task variance of existing few-shot node classification frameworks.

1 INTRODUCTION

Recently, extensive research efforts have been devoted to the node classification task, which aims at predicting class labels for unlabeled nodes in a graph. In real-world scenarios, the task of node classification yields an expansive variety of practical applications [21, 32]. For example, predicting chemical properties for proteins in a protein network is an important problem in bioinformatics [31], which can be formulated as the node classification problem. In recent years, the state-of-the-art approaches for node classification often utilize Graph Neural Networks (GNNs) [33, 39, 41] in a semi-supervised manner [15]. Specifically, for each node, GNNs aim to learn a vector representation for each node by transforming and aggregating information from its neighbors. The learned representations will be further utilized for the classification task in an end-to-end manner. Nevertheless, these approaches typically require sufficient labeled nodes for all classes in achieving a decent classification performance [45]. In practice, although we can access a large number of labeled nodes for certain classes, many other classes may only contain a limited number of labeled nodes. Here we refer to the former classes as *base classes* and the latter as *novel classes*. For example, in a protein network [11], newly discovered chemical properties with limited protein nodes are considered as novel classes, while common properties with abundant protein nodes are considered as base classes. Due to the widespread existence of novel classes in real-world graphs, many recent studies [6, 18, 36] focus on the problem of classifying nodes in novel classes, known as the *few-shot node classification* problem.

To tackle the few-shot node classification problem, recent works typically strive to extract transferable knowledge from base classes and then generalize such knowledge to novel classes [5, 43, 45]. More specifically, these works learn from base classes across a series of *meta-training* tasks and evaluate the model on *meta-test* tasks sampled from novel classes (we refer to both meta-training and meta-test tasks as meta-tasks). In fact, each meta-task contains a small number of *support nodes* as references and several *query nodes* to be classified. Since support nodes and query nodes in each meta-task are sampled from nodes on the entire graph, there could exist large variance among different meta-tasks (i.e., task variance) [12]. Therefore, the crucial part of few-shot node classification is to ensure that the underlying model has the generalization capability to handle a variety of meta-tasks in the presence of massive task variance [16, 30]. However, despite much progress has been made in few-shot node classification, recent studies ignore the task variance and treat each meta-task identically [6, 18, 36]. As a result, the task variance significantly jeopardizes the model generalization capability to meta-test tasks even when the performance on meta-training tasks is satisfactory [24].

Despite the importance of considering task variance, reducing its adverse impact remains non-trivial. In essence, there are two main factors that constitute such task variance. First, the *Node-level Variance* widely exists among meta-tasks and can lead to task variance. Specifically, node-level variance represents the differences of node features and local structures of nodes across different meta-tasks. For example, in addition to the common difference in node features, the red class nodes A and B in Fig. 1 also have different connectivity patterns in terms of neighboring nodes (node A is surrounded by blue nodes, while node B is only connected to red nodes). It should be noted that few-shot node classification models generally learn crucial information from the support nodes within each meta-task to perform classification on the query nodes. Therefore, if the variance among the support nodes is too large, it will become difficult to extract decisive information for classification. In other words, it is vital to consider node-level variance for the purpose of handling task variance. Second, *Class-level Variance* may also cause task variance. Class-level variance denotes the difference in class distributions among meta-tasks. In practice, since many real-world graphs contain a large number of node classes, the distribution of classes in each meta-task varies greatly [36, 45]. For example, in Fig. 1, different meta-tasks consist of a variety of classes (e.g., red and blue classes in $\mathcal{T}_1$ and dotted blue and green classes in $\mathcal{T}_1'$). Since the model evaluation is conducted on a vast number of meta-test tasks, the model will encounter many distinct classes during meta-test. That being said, in the presence of massive class-level variance, the resulting task variance will substantially deteriorate the generalization performance on meta-test tasks.

To alleviate the adverse impact of task variance resulting from the above two factors (i.e., node-level and class-level variance), we propose a novel **T**ask-adaptiv**E** few-shot **N**ode classifica**T**ion framework, named as TENT. Specifically, we aim to alleviate task variance via performing task adaptations from three perspectives. First, to handle node-level variance, we perform node-level adaptations via constructing a class-ego subgraph for each class in each meta-task. Specifically, such a subgraph explicitly connects nodes in the same class and their neighbors with a virtual class node. In this way, the neighbors of nodes in the same class are aggregated in this subgraph to reduce the influence of node-level variance. Second, to deal with class-level variance, we design a class-specific GNN to leverage information from different classes and perform class-level adaptations. Third, to reduce the adverse impact of task variance during classification on query nodes, we propose to perform task-level adaptations via maximally preserving the mutual information between query nodes and support nodes in each meta-task. As a result, our proposed framework can conduct classification in a task-adaptive manner to alleviate the adverse impact of task variance. In summary, our main contributions are three-folds:

- **Problem.** We investigate the limitations of existing few-shot node classification methods from the lens of task variance and discuss the importance and necessity of task adaptations for few-shot node classification.
- **Method.** We develop a novel task-adaptive few-shot node classification framework with three essential modules: (1) *node-level adaptation* to mitigate node-level variance; (2) *class-level adaptation* to alleviate the problem of class-level variance; and (3) *task-level* adaptation to consider task variance during classification on query nodes.
- **Experiments.** We conduct experiments on four benchmark node classification datasets under the few-shot setting and demonstrate the superiority of our proposed framework.

2 PRELIMINARIES

2.1 Problem Statement

Formally, let $G = (\mathcal{V}, \mathcal{E}, \mathbf{X})$ denote an attributed graph, where $\mathcal{V}$ is the set of nodes, $\mathcal{E}$ is the set of edges, and $\mathbf{X} \in \mathbb{R}^{|\mathcal{V}| \times d}$ is the feature matrix of nodes with d denoting the feature dimension. Moreover, we denote the entire set of node classes as C, which can be further divided into two categories: C_b and C_n, where $C = C_b \cup C_n$ and $C_b \cap C_n = \emptyset$. Here C_b and C_n denote the sets of base and novel classes, respectively. It is worth mentioning that the number of labeled nodes in C_b is sufficient, while it is typically small in C_n [6, 18, 45]. Then we can formulate the studied problem of few-shot node classification as follows:

Definition 1. ***Few-shot Node Classification:*** *Given an attributed graph $G = (\mathcal{V}, \mathcal{E}, \mathbf{X})$, our goal is to develop a machine learning model such that after training on labeled nodes in C_b, the model can accurately predict labels for the nodes (i.e., query set $\mathcal{Q}$) in C_n with only a limited number of labeled nodes (i.e., support set $\mathcal{S}$).*

More specifically, if the support set $\mathcal{S}$ contains exactly K nodes for each of N classes from C_n, and the query set $\mathcal{Q}$ are sampled from these N classes, the problem is called N-way K-shot node classification. Essentially, the objective of few-shot node classification is to learn a classifier that can be fast adapted to C_n with only limited labeled nodes. Thus, the crucial part is to learn transferable knowledge from C_b and generalize it to C_n.

2.2 Episodic Learning

In practice, we adopt the episodic learning framework for both meta-training and meta-test, which has proven to be effective in many areas [6, 7, 28, 34, 40]. Specifically, the meta-training and meta-test

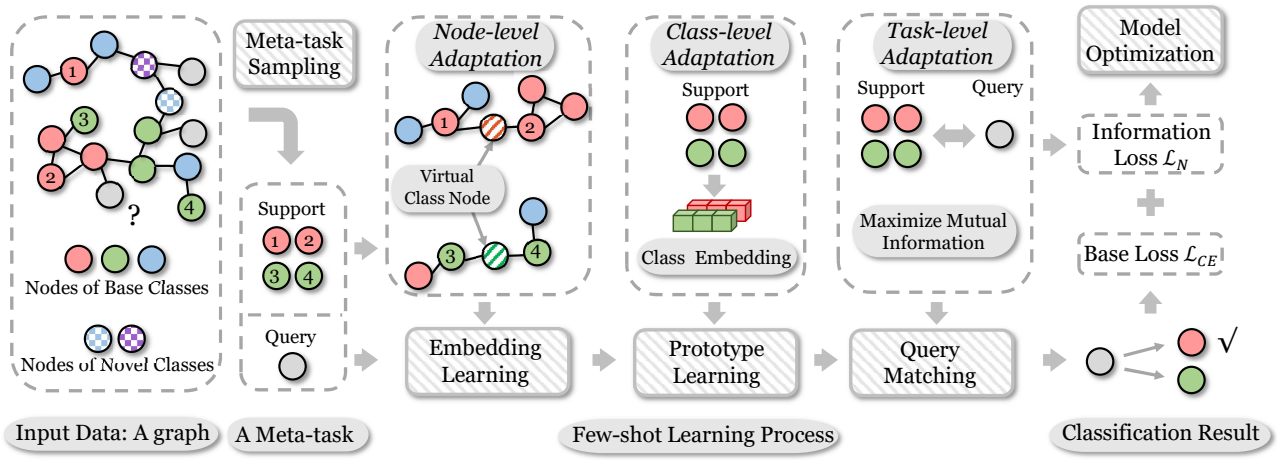

Figure 2: An illustration of the overall process of TENT. We first sample a meta-task from the given graph. Then we construct subgraphs for node-level adaptions and utilize node embeddings in each class for class-level adaptations. We further maximize the mutual information between the support set and the query set during query matching for task-level adaptations.

processes are conducted on a certain number of *meta-training tasks* and *meta-test tasks*, respectively. These meta-tasks share a similar structure, except that meta-training tasks are sampled from C_b, while meta-test tasks are sampled from C_n. The main idea of few-shot node classification is to keep the consistency between meta-training and meta-test to improve the generalization performance.

To construct a meta-training (or meta-test) task $\mathcal{T}_t$, we first randomly sample N classes from C_b (or C_n). Then we randomly sample K nodes from each of the N classes (i.e., N-way K-shot) to establish the support set $\mathcal{S}_t$. Similarly, the query set $\mathcal{Q}_t$ consists of Q different nodes (distinct from $\mathcal{S}_t$) from the same N classes. The components of the sampled meta-task $\mathcal{T}_t$ can be denoted as follows:

$$\begin{aligned} \mathcal{S}_t &= \{(v_1, y_1), (v_2, y_2), \ldots, (v_{N\times K}, y_{N\times K})\}, \\ \mathcal{Q}_t &= \{(q_1, y'_1), (q_2, y'_2), \ldots, (q_Q, y'_Q)\}, \\ \mathcal{T}_t &= \{\mathcal{S}_t, \mathcal{Q}_t\}, \end{aligned} \tag{1}$$

where v_i (or q_i) is a node in $\mathcal{V}$, and y_i (or y'_i) is the corresponding label. In this way, the whole training process is conducted on a set of T meta-training tasks $\mathcal{T}_{train} = \{\mathcal{T}_t\}_{t=1}^{T}$. After training, the model has learned the transferable knowledge from $\mathcal{T}_{train}$ and will generalize it to meta-test tasks $\mathcal{T}_{test} = \{\mathcal{T}'_t\}_{t=1}^{T_{test}}$ sampled from C_n.

3 OUR PROPOSED FRAMEWORK

In this section, we introduce the overall structure of our proposed framework TENT in detail. As illustrated in Fig. 2, we formulate the *few-shot node classification* problem under the prevailing N-way K-shot learning framework, which means a meta-task contains K nodes for each of N classes as the support set. In addition, the query set consists of Q unlabeled nodes to be classified from these N classes. Specifically, our framework follows the prevalent three phases for few-shot learning: embedding learning, prototype learning, and query matching. Generally, in each meta-task, we learn embeddings for its nodes and then learn a prototype (i.e., embedding of a class in the support set) based on the node embeddings. Finally, the model matches query nodes with these prototypes via specific matching functions to output classification results. Nevertheless, these three steps ignore the task variance that widely exists among meta-tasks. Therefore, as illustrated in Fig. 2, we propose to perform three levels of adaptations (node-level, class-level, and task-level adaptations) in these three phases, respectively, to alleviate the adverse impact of task variance.

3.1 Node-level Adaptation

During the embedding learning phase, existing methods learn embeddings for nodes in each meta-task from the entire graph [6, 36, 45]. Since nodes are distributed across the entire graph, the learned node representations can be easily influenced by node-level variance (i.e., have different connectivity patterns in terms of neighboring nodes). Instead, we perform node-level adaptations in each meta-task, which aims to modify the neighbors of support nodes to reduce node-level variance caused by different connectivity patterns. Toward this goal, we explicitly construct a subgraph for each class in each meta-task via a virtual class node, which connects to the K support nodes in that class. In addition, we also include the one-hop neighbors of these K nodes in this subgraph. By doing the above, we can aggregate local structures of support nodes in the same class into this subgraph, which contains the virtual class node, K support nodes, and one-hop neighbors of these K nodes. Here the virtual class node acts as a bridge to explicitly connect these K support nodes and their one-hop neighbors that can be originally far from each other on the graph. Moreover, its embedding will be used as the prototype of this class since it is the centroid node of the subgraph. As a result, in this subgraph, support nodes will share a similar neighbor node set because the neighboring nodes of support nodes are explicitly connected. Since the neighbors of support nodes become more similar in this subgraph, we can effectively reduce the node-level variance. We denote the constructed subgraphs in each meta-task as *class-ego subgraphs*.

Specifically, given a meta-task $\mathcal{T}$ and its support set $\mathcal{S}$ ($|\mathcal{S}| = N \times K$) on a graph $G = (\mathcal{V}, \mathcal{E}, \mathbf{X})$, we aim to construct a class-ego subgraph for each of the N classes in $\mathcal{T}$. Before the construction of class-ego subgraphs, we employ a GNN [15, 39] parameterized

by ϕ to perform message propagation on the entire graph G and generate first-step node representations for nodes in $\mathcal{V}$ as follows:

$$\mathbf{H} = \text{GNN}_\phi(\mathcal{V}, \mathcal{E}, \mathbf{X}), \tag{2}$$

where $\mathbf{H} \in \mathbb{R}^{|\mathcal{V}| \times d_h}$ denotes the first-step representations of nodes in $\mathcal{V}$ and d_h is the output dimension of GNN_ϕ. In this way, $\mathbf{H}$ will act as the input node representations for the class-ego subgraphs.

Let $\mathcal{S}_i$ denote the set of nodes belonging to the i-th class in $\mathcal{S}$, which means $|\mathcal{S}_i| = K$, $i = 1, 2, \ldots, N$. To construct the class-ego subgraph from these nodes, we first create a virtual class node c_i and connect it to all nodes in $\mathcal{S}_i$. Then to incorporate the local graph structures, we also extract all one-hop neighbors of nodes in $\mathcal{S}_i$ to establish a neighbor node set $\mathcal{N}_i = \bigcup_{j=1}^{K} \mathcal{N}_i^j$, where $\mathcal{N}_i^j$ denotes the set of neighbors of the j-th node in $\mathcal{S}_i$. In this way, the final node set of the class-ego subgraph is aggregated as $\mathcal{V}_i = \{c_i\} \cup \mathcal{S}_i \cup \mathcal{N}_i$. After that, we accordingly denote the extracted edge set of nodes in $\mathcal{V}_i$ as $\mathcal{E}_i$. To obtain the input node features for $\mathcal{V}_i$, we utilize the corresponding first-step representations from $\mathbf{H}$. However, we still need to compute the representation of c_i since it is newly created. Here we propose to initiate its representation $\mathbf{h}_{c_i}$ as follows:

$$\mathbf{h}_{c_i} = \text{MEAN}(\mathbf{h}_v | v \in \mathcal{S}_i), \tag{3}$$

where $\mathbf{h}_{c_i} \in \mathbb{R}^{d_h}$ and $\mathbf{h}_v$ is the first-step node representation of node v. MEAN denotes the averaging operation. In this way, the input node features for $\mathcal{V}_i$ can be obtained as $\mathbf{X}_i$. Then the class-ego subgraph can be constructed and denoted as $G_i = (\mathcal{V}_i, \mathcal{E}_i, \mathbf{X}_i)$. As a result, we can achieve node-level adaptations by learning node representations on the subgraphs with reduced node-level variance.

3.2 Class-level Adaptation

Typically, after learning the node representations in $\mathcal{T}$, existing models learn prototypes for classes in $\mathcal{T}$ by aggregating node representations in the same class [6, 28]. However, this strategy can be easily influenced by class-level variance and thus renders suboptimal generalization performance since it treats each class identically. Instead, we propose to perform class-level adaptations, which aim to obtain prototypes for classes in a class-adaptive manner. In particular, we design a class-specific adapter to adjust GNN parameters regarding different classes in $\mathcal{T}$. In this way, our framework can leverage the discriminative information in each class for class-level adaptations and reduce the adverse impact of class-level variance.

Specifically, we use a new GNN_θ parameterized by θ on the class-ego subgraphs to learn prototypes. Then we adapt θ according to the first-step representations of nodes in each class (i.e., $\{\mathbf{h}_v | v \in \mathcal{S}_i\}, i = 1, 2, \ldots, N$) in meta-task $\mathcal{T}$. To comprehensively incorporate the information in each class, we leverage the feature-wise linear modulations [26, 38] to perform class-level adaptations:

$$\alpha_i = \text{MLP}_\alpha\left(\text{MEAN}\left(\{\mathbf{h}_v | v \in \mathcal{S}_i\}\right)\right), \tag{4}$$

$$\beta_i = \text{MLP}_\beta\left(\text{MEAN}\left(\{\mathbf{h}_v | v \in \mathcal{S}_i\}\right)\right), \tag{5}$$

where $\mathcal{S}_i$ is the set of nodes belonging to the i-th class in $\mathcal{S}$. $\alpha_i \in \mathbb{R}^{d_\theta}$ and $\beta_i \in \mathbb{R}^{d_\theta}$ are d_θ-dimensional adaptation parameters, where d_θ is the total number of parameters in GNN_θ. With the adaptation parameters α_i and β_i, we can perform an adaptation based on each class to obtain class-specific GNN parameters as follows:

$$\theta_i = (\alpha_i + \mathbf{1}) \circ \theta + \beta_i, \tag{6}$$

where $\circ$ denotes the element-wise multiplication and $\mathbf{1}$ is a vector of ones to limit the scaling range around one. θ_i denotes the adapted GNN parameters for the i-th class in $\mathcal{S}$. Then we perform message propagation on each class-ego subgraph with the adapted θ_i:

$$\mathbf{s}_i = \text{Centroid}\left(\text{GNN}_{\theta_i}(\mathcal{V}_i, \mathcal{E}_i, \mathbf{X}_i)\right), \tag{7}$$

where $\mathbf{s}_i \in \mathbb{R}^{d_s}$ denotes the learned embedding of the virtual class node (i.e., the centroid node) and acts as the prototype of the i-th class. Centroid$(\cdot)$ denotes the operation of extracting the centroid node representation from the GNN output. d_s is the output dimension of GNN_θ. As a result, the GNN parameters can absorb the information from each class to reduce the adverse impact of class-level variance. Similarly, for the representations of query nodes, we also apply the proposed class-specific adapter. Since labels of query nodes are unknown during meta-training, we utilize the entire support set $\mathcal{S}$ to conduct adaptations for query nodes:

$$\alpha_q = \text{MLP}_\alpha\left(\text{MEAN}\left(\{\mathbf{h}_v | v \in \mathcal{S}\}\right)\right), \tag{8}$$

$$\beta_q = \text{MLP}_\beta\left(\text{MEAN}\left(\{\mathbf{h}_v | v \in \mathcal{S}\}\right)\right), \tag{9}$$

$$\theta_q = (\alpha_q + \mathbf{1}) \circ \theta + \beta_q, \tag{10}$$

where θ_q is the adapted GNN parameters for query nodes. To obtain the representation $\mathbf{q}_i$ for the i-th query node q_i in the query set $\mathcal{Q}$, we extract the 2-hop neighbors of q_i and obtain the corresponding node set $\mathcal{V}_i^q$, edge set $\mathcal{E}_i^q$, and input node features $\mathbf{X}_i^q$. The reason is that using subgraph structures (e.g., considering 2-hop neighbors) to learn node representations provides a more robust generalization ability [12]. Then we utilize the adapted GNN_{θ_q} to compute $\mathbf{q}_i$:

$$\mathbf{q}_i = \text{Centroid}\left(\text{GNN}_{\theta_q}(\mathcal{V}_i^q, \mathcal{E}_i^q, \mathbf{X}_i^q)\right), \tag{11}$$

where $\mathbf{q}_i \in \mathbb{R}^{d_s}$ is the embedding of q_i (i.e., the centroid node).

3.3 Task-level Adaptation

Although we have achieved node-level and class-level adaptations in the embedding learning and prototype learning phases, respectively, the task variance caused by differences in the support set among meta-tasks still exist in the final query matching phase [12]. Nevertheless, existing methods typically leverage the Euclidean distance metric [6] or an MLP layer [18] to classify query nodes, which ignores the task variance. Instead, we propose to perform task-level adaptations in each meta-task, which aim to further reduce the adverse impact of task variance in this phase. Specifically, we propose a task-adaptive matching strategy to maximally preserve the mutual information between learned representations of nodes in the query set $\mathcal{Q}$ and the support set $\mathcal{S}$. As a result, the matching phase on query nodes can incorporate information from the entire support set $\mathcal{S}$ for task-level adaptations.

The optimization problem of maximizing the mutual information can be formulated as follows:

$$\max_{\widetilde{\theta}} I(\mathbf{Q}; \mathbf{S}) = \max_{\widetilde{\theta}} \sum_{i=1}^{Q} \sum_{j=1}^{N} p(q_i, s_j; \widetilde{\theta}) \log \frac{p(q_i | s_j; \widetilde{\theta})}{p(q_i; \widetilde{\theta})}, \tag{12}$$

where $\mathbf{Q} \in \mathbb{R}^{Q \times d_s}$ and $\mathbf{S} \in \mathbb{R}^{N \times d_s}$ are learned representations of query nodes in $\mathcal{Q}$ and classes (i.e., prototypes) in $\mathcal{S}$, respectively. $Q = |\mathcal{Q}|$ and N is the number of classes in $\mathcal{S}$. $\widetilde{\theta}$ denotes the parameters of our framework to be optimized. q_i is the i-th query node

in Q and s_j is the j-th class in $\mathcal{S}$. Since the mutual information $I(\mathbf{Q};\mathbf{S})$ is difficult to obtain and thus infeasible to maximize [23], we re-write the function to obtain an accessible form:

$$I(\mathbf{Q};\mathbf{S}) = \sum_{i=1}^{Q}\sum_{j=1}^{N} p(q_i|s_j;\widetilde{\theta})p(s_j;\widetilde{\theta}) \log \frac{p(q_i|s_j;\widetilde{\theta})}{p(q_i;\widetilde{\theta})}. \tag{13}$$

Since each meta-task contains N classes, we may assume that the prior probability of $p(s_j;\widetilde{\theta})$ follows a uniform distribution and set it as $p(s_j;\widetilde{\theta}) = 1/N$. Since $p(s_j;\widetilde{\theta})$ is a constant, according to the Bayes' theorem, the objective function becomes:

$$\begin{aligned} I(\mathbf{Q};\mathbf{S}) &= \frac{1}{N}\sum_{i=1}^{Q}\sum_{j=1}^{N} p(q_i|s_j;\widetilde{\theta}) \log \frac{p(s_j|q_i;\widetilde{\theta})}{p(s_j;\widetilde{\theta})} \\ &= \frac{1}{N}\sum_{i=1}^{Q}\sum_{j=1}^{N} p(q_i|s_j;\widetilde{\theta}) \left(\log(p(s_j|q_i;\widetilde{\theta})) - \log\left(\frac{1}{N}\right)\right). \end{aligned} \tag{14}$$

To further estimate $p(q_i|s_j;\widetilde{\theta})$, we compute it by $p(q_i|s_j;\widetilde{\theta}) = \mathbb{1}(q_i \in s_j)$, where $\mathbb{1}(q_i \in s_j) = 1$ if q_i belongs to the class represented by s_j; otherwise $\mathbb{1}(q_i \in s_j) = 0$. In this way, the above objective function is simplified as follows:

$$I(\mathbf{Q};\mathbf{S}) = \frac{1}{N}\sum_{i=1}^{Q}\sum_{j=1}^{N} \mathbb{1}(q_i \in s_j) \left(\log(p(s_j|q_i;\widetilde{\theta})) - \log\left(\frac{1}{N}\right)\right). \tag{15}$$

Since each q_i can only belong to one s_j (i.e., one class), we can further simplify the objective function:

$$\sum_{i=1}^{Q}\sum_{j=1}^{N} \mathbb{1}(q_i \in s_j) \log(p(s_j|q_i;\widetilde{\theta})) = \sum_{i=1}^{Q} \log(p(s_i'|q_i;\widetilde{\theta})), \tag{16}$$

where s_i' denotes the specific s_j that q_i belongs to (i.e., $q_i \in s_i'$). Moreover, since $\log(1/N)$ is also a constant, we can simplify the objective function as follows:

$$I(\mathbf{Q};\mathbf{S}) = \sum_{i=1}^{Q} \log(p(s_i'|q_i;\widetilde{\theta})). \tag{17}$$

To estimate $p(s_i'|q_i;\widetilde{\theta})$, we can define the probability of q_i belonging to s_j according to the squared ℓ_2 norm of the embedding distance. Specifically, we further assign a weight parameter τ_i to each class and normalize the probability with a softmax function:

$$p(s_i'|q_i;\widetilde{\theta}) = \frac{\exp\left(-(\mathbf{q}_i - \mathbf{s}_i')^2/\tau_i'\right)}{\sum_{j=1}^{N}\exp\left(-(\mathbf{q}_i - \mathbf{s}_j)^2/\tau_j\right)}, \tag{18}$$

where $\mathbf{q}_i$ and $\mathbf{s}_j$ denote the representations of the i-th query node x_i in Q and the j-th class-ego subgraph in $\mathcal{S}$, respectively. τ_i is the adaptation parameter of the i-th class, and $\mathbf{s}_i'$ and τ_i' denote the specific class-ego subgraph representation and the adaptation parameter of the class that $\mathbf{q}_i$ belongs to, respectively. Then if we further apply the ℓ_2 normalization to both $\mathbf{q}_i$ and $\mathbf{s}_i$, we obtain $(\mathbf{q}_i - \mathbf{s}_i)^2 = 2 - 2\mathbf{q}_i \cdot \mathbf{s}_i$. Combining the above equations, we can present the final optimization problem as follows:

$$\max_{\widetilde{\theta}} I(Q;\mathcal{S}) = \min_{\widetilde{\theta}} \sum_{i=1}^{Q} -\log \frac{\exp(\mathbf{q}_i \cdot \mathbf{s}_i'/\tau_i')}{\sum_{j=1}^{N}\exp(\mathbf{q}_i \cdot \mathbf{s}_j/\tau_j)}. \tag{19}$$

Algorithm 1 Detailed learning process of our framework.

Input: A graph $G = (\mathcal{V}, \mathcal{E}, \mathbf{X})$, a meta-test task $\mathcal{T}_{test} = \{\mathcal{S}, \mathcal{Q}\}$, base classes C_b, meta-training epochs T, the number of classes N, and the number of labeled nodes for each class K.
Output: Predicted labels of the query nodes in $\mathcal{Q}$.

// Meta-training phase
for $i = 1, 2, \ldots, T$ **do**
  Sample a meta-training task $\mathcal{T}_i = \{\mathcal{S}_i, \mathcal{Q}_i\}$ from C_b;
  Compute first-step node representations with GNN_ϕ;
  Construct a class-ego subgraphs for each of N classes in $\mathcal{S}_i$;
  Adapt GNN_θ to $\mathcal{T}_i$ according to Eq. (6) and (10);
  Compute the representations for class-ego subgraphs and query nodes with the adapted GNN_{θ_i} and GNN_{θ_q};
  Update model parameters with the meta-training loss of $\mathcal{T}_i$ according to Eq. (24) by one gradient descent step;
end for
// Meta-test phase
Compute first-step node representations with GNN_ϕ;
Construct a class-ego subgraphs for each of N classes in $\mathcal{S}$;
Adapt GNN_θ to $\mathcal{T}_{test}$ according to Eq. (6) and (10);
Compute the representations for class-ego subgraphs and query nodes with the adapted GNN_{θ_i} and GNN_{θ_q};
Predict labels for query nodes in $\mathcal{Q}$;

Since the distribution of node embeddings in each class $\mathcal{S}_i$ differs around the class representation $\mathbf{s}_i$, the node embeddings of specific classes can be scattered in a large distance from $\mathbf{s}_i$. In this case, the classification process should incorporate the distribution of node embeddings in each class while considering the entire support set $\mathcal{S}$. Thus, we propose to obtain τ_i' as follows:

$$\tau_i = \frac{N\sum_k^K \|\mathbf{s}_i^k - \mathbf{s}_i\|_2}{\sum_j^N \sum_k^K \|\mathbf{s}_j^k - \mathbf{s}_j\|_2}, \tag{20}$$

where $\{\mathbf{s}_i^k\}_{i=1}^K$ denotes the node embeddings in the i-th class processed by GNN_{θ_i}. In this way, the classification process is adapted regrading the entire support set to reduce the adverse impact of task variance. Hence, the model is able to incorporate information in the entire support set to achieve task-level adaptations. Then according to Eq. (19), we can optimize the following information loss to preserve the mutual information $I(\mathcal{Q};\mathcal{S})$:

$$\mathcal{L}_N = -\sum_{i=1}^{Q} \log \frac{\exp(\mathbf{q}_i \cdot \mathbf{s}_i'/\tau_i')}{\sum_{j=1}^{N}\exp(\mathbf{q}_i \cdot \mathbf{s}_j/\tau_j)}. \tag{21}$$

It is worth mentioning that $\mathcal{L}_N$ shares a similar expression with the InfoNCE loss [10, 23]. Thus, InfoNCE can be considered as a special case where positive and negative pairs are defined by different views of nodes, while in our case, they are defined according to the labels of query nodes. $\mathcal{L}_N$ also differs from the supervised contrastive loss [13], which utilizes various views of images and supervised information, as our loss aims at maximally preserving the mutual information between query nodes and support nodes within each meta-task. Moreover, the adaptation parameter acts similarly with the temperature parameter in InfoNCE, while in our framework, it is adjustable and provides task-level adaptations.

3.4 Few-shot Node Classification

So far, we can train GNN_ϕ and GNN_θ with the proposed loss $\mathcal{L}_N$ in Eq. (21). However, since GNN_ϕ provides the first-step representations for nodes from an overview of the entire graph G, the supervised information within each meta-task could be insufficient for the optimization of GNN_ϕ. Thus, we propose to classify query nodes from base classes C_b to optimize GNN_ϕ. Specifically, we utilize an MLP layer followed by a softmax function to calculate the cross-entropy classification loss over C_b:

$$\mathbf{p}_i = \text{Softmax}\left(\text{MLP}(\mathbf{h}_i)\right), \tag{22}$$

$$\mathcal{L}_{CE} = -\sum_{i=1}^{Q}\sum_{j=1}^{|C_b|} y_{i,j}\log p_{i,j}, \tag{23}$$

where $\mathbf{p}_i \in \mathbb{R}^{|C_b|}$ is the probability that the i-th query node in Q belongs to each class in C_b. $y_{i,j} = 1$ if the i-th node belongs to the j-th class, and $y_{i,j} = 0$, otherwise. $p_{i,j}$ is the j-th element in $\mathbf{p}_i$. In this way, instead of classifying nodes only from classes in a meta-task, we can utilize the supervised information in C_b from a global perspective. Then the meta-training loss is defined as follows:

$$\mathcal{L} = \mathcal{L}_N + \gamma\mathcal{L}_{CE}, \tag{24}$$

where γ is an adjustable weight hyper-parameter.

After meta-training, the meta-test process is the same as the meta-training process, except that meta-test tasks are sampled from novel classes C_n. The labels of query nodes are obtained by $\hat{y}_i = \text{argmax}_j\{\mathbf{q}_i \cdot \mathbf{s}_j/\tau_i | j = 1, 2, \ldots, N\}$, where $\mathbf{q}_i$ and $\mathbf{s}_j$ are representations of the i-th query node and the j-th class-ego subgraph, respectively. τ_i is the adaptation parameter of the i-th class. The detailed process of our framework is demonstrated in Algorithm 1.

4 EXPERIMENTS

In this section, we conduct experiments to evaluate our framework TENT on four prevalent few-shot node classification datasets. Furthermore, we conduct experiments to verify the effectiveness of different modules in our framework with ablation study and demonstrate the parameter sensitivity.

4.1 Datasets

To evaluate our framework on few-shot node classification tasks, we conduct experiments on four prevalent real-world graph datasets: `Amazon-E` [20], `DBLP` [32], `Cora-full` [1], and `OGBN-arxiv` [11]. We summarize the detailed statistics of these datasets in Table 1. Specifically, # Nodes and # Edges denote the number of nodes and edges in the graph, respectively. # Features denotes the dimension of node features. Class Split denotes the number of classes used for meta-training/validation/meta-test. More details are provided in Appendix A.2.4.

Table 1: Statistics of four node classification datasets.

Dataset	# Nodes	# Edges	# Features	Class Split
`Amazon-E`	42,318	43,556	8,669	90/37/40
`DBLP`	40,672	288,270	7,202	80/27/30
`Cora-full`	19,793	65,311	8,710	25/20/25
`OGBN-arxiv`	169,343	1,166,243	128	15/5/20

4.2 Experimental Settings

To validate the effectiveness of our proposed framework TENT, we conduct experiments with the following baseline methods to compare performance:

- **Prototypical Networks** [28]: Prototypical Networks learn prototypes for classes for query matching.
- **MAML** [7]: MAML proposes to optimize model parameters based on gradients of support instances across meta-tasks.
- **GCN** [15]: GCN performs information propagation based on local structures.
- **G-Meta** [12]: G-Meta utilizes representations of subgraphs as node embeddings for few-shot learning on graphs.
- **GPN** [6]: GPN leverages node importance and Prototypical Networks to improve performance.
- **RALE** [18]: RALE proposes to learn node dependencies according to node locations on the graph.

During training, we sample a certain number of meta-training tasks from training classes (i.e., base classes) and train the model with these meta-tasks. Then we evaluate the model based on a series of randomly sampled meta-test tasks from test classes (i.e., novel classes). For consistency, the class splitting is identical for all baseline methods. Then the final result of the average classification accuracy is obtained based on these meta-test tasks. More detailed parameter settings can be found in Appendix A.2.1.

4.3 Overall Evaluation Results

We first present the performance comparison of our framework and baseline methods on few-shot node classification in Table 2. Specifically, to better demonstrate the efficacy of our framework under different few-shot settings, we conduct the experiments under four different settings: 5-way 3-shot, 5-way 5-shot, 10-way 3-shot, and 10-way 5-shot. Moreover, the evaluation metric is the average classification accuracy over ten repetitions. From the overall results, we can obtain the following observations:

- Our proposed framework TENT outperforms all other baselines in all datasets under different few-shot settings, which validates the effectiveness of our task-adaptive framework on few-shot node classification.
- Conventional few-shot methods such as Prototypical Network [28] and MAML [7] exhibit inferior performance compared with other baselines. The reason is that such methods are proposed in other domains and thus result in unsatisfactory performance on graphs.
- When increasing the value of K (i.e., more support nodes in each class), all methods gain considerable performance improvements. Moreover, our framework achieves better results due to that the node-level and class-level adaptations benefit more from a larger size of nodes in each class.
- The performance of all methods significantly decreases when the value of N increases (i.e., more classes in each meta-task). The main reason is that the variety of classes in each meta-task leads to a more complex class distribution and results in classification difficulties. However, by incorporating the class-level and task-level adaptations, our framework is capable of alleviating this problem when a larger N is presented.

Table 2: The overall few-shot node classification results (accuracy in %) of various models under different few-shot settings.

Dataset	DBLP				Amazon-E			
Setting	5-way 3-shot	5-way 5-shot	10-way 3-shot	10-way 5-shot	5-way 3-shot	5-way 5-shot	10-way 3-shot	10-way 5-shot
PN [28]	41.51 ± 3.60	46.17 ± 3.55	28.98 ± 3.87	36.71 ± 3.35	56.80 ± 3.60	62.53 ± 2.80	44.26 ± 2.64	48.20 ± 3.89
MAML [7]	43.06 ± 2.92	49.93 ± 2.57	34.63 ± 3.91	38.44 ± 3.25	56.03 ± 2.11	63.40 ± 3.33	40.80 ± 2.75	47.06 ± 3.15
GCN [15]	62.87 ± 1.44	70.51 ± 1.37	47.22 ± 2.97	53.95 ± 2.49	55.33 ± 1.23	62.96 ± 2.61	45.18 ± 2.61	50.89 ± 2.95
G-Meta [12]	73.49 ± 2.82	78.56 ± 2.86	60.77 ± 3.03	66.26 ± 3.47	64.56 ± 4.23	68.36 ± 4.10	59.75 ± 4.90	63.02 ± 4.11
GPN [6]	76.42 ± 3.11	80.85 ± 3.68	63.14 ± 2.25	69.55 ± 2.56	65.16 ± 3.17	71.89 ± 3.94	62.52 ± 3.12	63.98 ± 2.04
RALE [18]	75.38 ± 4.94	79.85 ± 4.69	62.81 ± 3.48	67.61 ± 3.99	69.55 ± 4.24	74.97 ± 4.66	63.27 ± 3.31	64.85 ± 3.04
TENT	**79.04 ± 3.14**	**82.84 ± 3.97**	**65.47 ± 4.21**	**72.38 ± 4.14**	**75.76 ± 3.63**	**79.38 ± 4.98**	**67.59 ± 4.16**	**69.77 ± 3.76**

Dataset	Cora-full				OGBN-arxiv			
Setting	5-way 3-shot	5-way 5-shot	10-way 3-shot	10-way 5-shot	5-way 3-shot	5-way 5-shot	10-way 3-shot	10-way 5-shot
PN [28]	42.62 ± 3.78	56.66 ± 2.91	35.95 ± 3.95	38.69 ± 3.09	37.99 ± 3.98	49.71 ± 4.20	31.44 ± 3.00	35.79 ± 3.63
MAML [7]	47.10 ± 4.32	54.89 ± 3.09	30.68 ± 3.08	42.22 ± 2.76	41.83 ± 2.54	42.14 ± 3.86	33.15 ± 2.92	36.82 ± 3.03
GCN [15]	49.05 ± 2.04	58.03 ± 3.50	34.27 ± 3.98	39.85 ± 3.50	44.80 ± 2.56	47.29 ± 3.58	35.80 ± 2.21	37.78 ± 2.90
G-Meta [12]	57.93 ± 3.79	60.30 ± 2.93	45.67 ± 3.35	47.76 ± 3.25	47.66 ± 3.27	49.81 ± 4.01	35.93 ± 3.04	40.13 ± 4.35
GPN [6]	58.38 ± 3.49	63.82 ± 2.93	41.65 ± 2.20	45.63 ± 3.17	49.16 ± 3.43	53.06 ± 3.13	37.28 ± 3.99	43.33 ± 3.27
RALE [18]	62.83 ± 3.12	65.93 ± 3.24	48.05 ± 3.09	51.67 ± 3.21	53.90 ± 3.45	56.99 ± 4.43	37.60 ± 4.12	41.42 ± 3.03
TENT	**64.80 ± 4.10**	**69.24 ± 4.49**	**51.73 ± 4.34**	**56.00 ± 3.53**	**55.62 ± 3.13**	**62.96 ± 3.74**	**41.13 ± 4.26**	**44.73 ± 3.42**

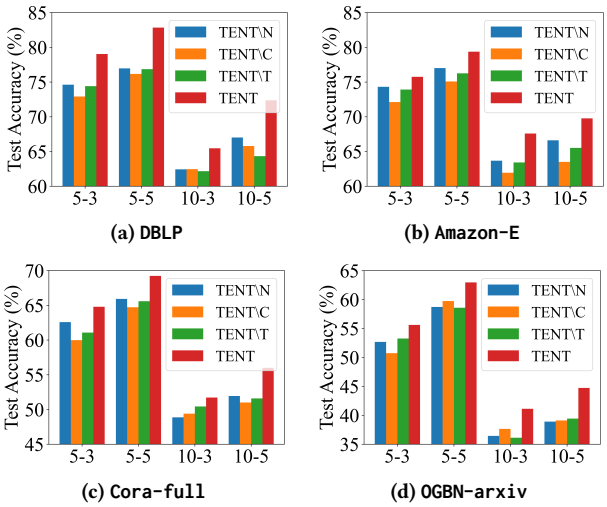

Figure 3: Ablation study on our framework in the N-way K-shot setting.

4.4 Ablation Study

In this part, we conduct an ablation study on four datasets to verify the importance of three crucial components in TENT. First, we remove the node-level adaptation and directly utilize the original graph instead of class-ego subgraphs to learn representations for each class in meta-tasks. In this way, the support nodes of different classes are distributed over the entire graph and thus lack node-level adaptations. We refer to this variant as *TENT\N*. Second, we remove the class-specific adapter so that the framework identically learns class representations and lacks class-level adaptations, and we refer to this variant as *TENT\C*. The final variant is to replace the task-level adaptation module with a common Euclidean distance classifier, which means during training, the framework fails to learn task-level adaptations across meta-training tasks, and we refer it to as *TENT\T*. The overall ablation study results are presented in Fig. 3. From the results, we can observe that TENT outperforms all variants, which demonstrates the effectiveness of all three types of adaptations. Specifically, removing node-level adaptations results in a large decrease in few-shot node classification performance. Furthermore, integrating class-level adaptations provides a considerable performance improvement, especially when the number of classes increases, which introduces larger class variance. More significantly, without the task-level adaptations, the performance decreases rapidly when the support set size increases. Therefore, the result further demonstrates the importance of task-level adaptations in the presence of a more complex few-shot setting with a large support set.

4.5 Effect of Meta-training Support Set Size

In this section, we conduct experiments to study the sensitivity of several parameters in TENT. Since TENT provides task adaptations for both meta-training and meta-test tasks, the values of N (i.e., number of classes in a support set) and K (i.e., number of support nodes in each class) are unnecessary to be consistent during meta-training and meta-test. In other words, it differs from the general few-shot learning setting, where the parameters of N and K are consistent during meta-training and meta-test. Therefore, we can adjust these two parameters during meta-training to analyze their effects for better performance. Fig. 4 reports the classification accuracy of TENT when varying the parameters of N and K during meta-training on four datasets, denoted as N_t and K_t, respectively. Specifically, we vary the values of N_t and K_t as 3, 5, 10, and 20. Note

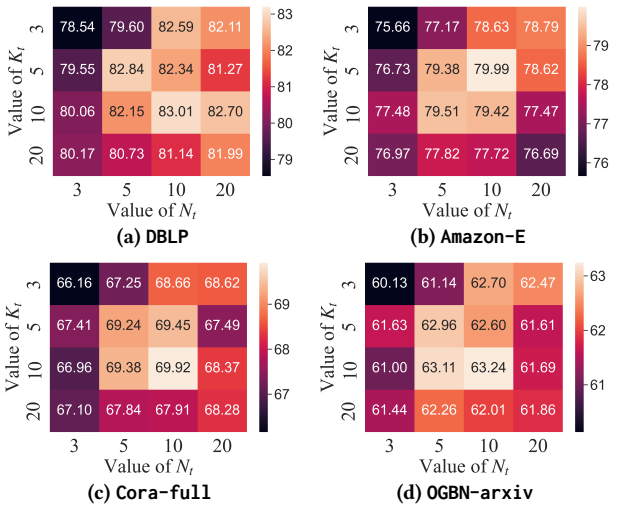

Figure 4: Results of TENT with different N_t and K_t.

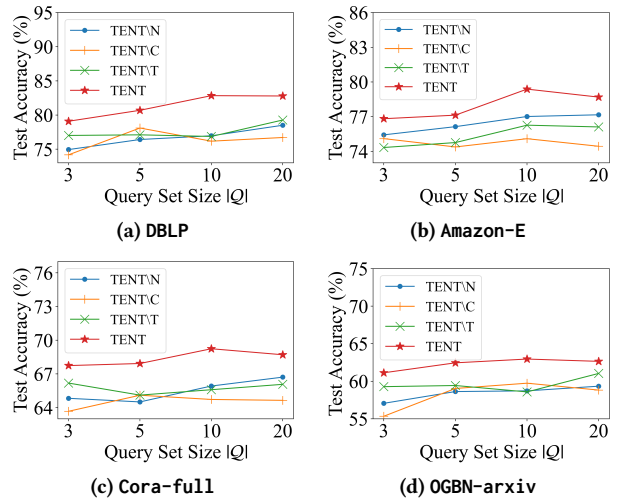

Figure 5: Results of TENT with different values of $|Q|$.

that during meta-test, the values of N and K are kept invariant as 5 and 5, respectively (i.e., 5-way 5-shot). From the results, we observe that increasing N_t and K_t both provide better results on few-shot node classification. The reason is that TENT learns the three types of adaptations from a larger support set during meta-training and thus is more capable of handling node-level and class-level variance. More specifically, increasing the value of N results in a more significant improvement. The main reason is that the class-level and task-level adaptations benefit from more classes in each meta-task. In addition, incorporating more support nodes in each class (i.e., larger K_t) also enhances the interactions among nodes in each class-ego subgraph for more comprehensive node-level adaptations.

4.6 Effect of Query Set Size

In this part, we conduct experiments to present how the query set size $|Q|$ in each meta-task during meta-training affects the performance of our proposed framework TENT. Fig. 5 reports the results of TENT when varying the value of $|Q|$ on four datasets under the 5-way 5-shot setting. Specifically, $|Q|$ during meta-training is changed from 3 to 20, while it remains 10 during meta-test for a fair comparison. From the results, we can observe that the few-shot node classification performance increases when $|Q|$ becomes larger. The reason is mainly attributed to the fact that involving more query nodes during meta-training (i.e., increasing the value of $|Q|$) helps alleviate the over-fitting problem. However, as the results suggest, an excessively large query set size may result in a performance drop. The reason is that the optimization process may be more difficult on a large query set.

5 RELATED WORK

5.1 Graph Neural Networks

Recently, many researchers focus on studying Graph Neural Networks (GNNs) to learn comprehensive node representations in graphs [2, 3, 42]. In general, GNNs aim at learning node representations through a certain number of information propagation steps in a recurrent manner [9, 44, 46]. In this way, GNNs can aggregate information from neighboring nodes to generate node representations based on local structures. For example, Graph Convolutional Networks (GCNs) [15] perform convolution operations on graphs based on the graph spectral theory. Graph Attention Networks (GATs) [33] leverage the attention mechanism to select more important neighboring nodes for aggregation. Moreover, Graph Isomorphism Networks (GINs) [41] develop an expressive architecture, which is as powerful as the Weisfeiler-Lehman graph isomorphism test. Nonetheless, GNNs typically render sub-optimal performance when there are limited labeled nodes for each class [6, 43], which further indicates the necessity of few-shot learning on graphs.

5.2 Few-shot Learning on Graphs

Few-shot Learning (FSL) aims to learn transferable knowledge from tasks with abundant supervised information and generalize it to novel tasks with a limited number of labeled instances. In general, few-shot learning methods can be divided into two categories: *metric-based* approaches and *meta-optimizer-based* approaches. Specifically, the metric-based approaches aim at learning generalizable metric functions to match the query set with the support set for classification [17, 29]. For example, Matching Networks [34] conduct predictions based on the similarity between a query instance and each support instance learned by attention networks. Prototypical Networks [28] learn a prototype as the representation for each class and perform classification based on the Euclidean distances between query instances and prototypes. On the other hand, meta-optimizer-based approaches aim at optimizing model parameters according to gradients calculated from few-shot instances [22, 27]. For example, MAML [7] optimizes model parameters based on gradients on support instances for fast generalization. Moreover, LSTM-based meta-learner [27] proposes to adjust the step size for updating parameters during meta-training.

In the field of graphs, several recent works propose to conduct graph-based tasks under the few-shot learning scenario [4, 19, 37]. Among them, GPN [6] proposes to leverage node importance based on Prototypical Networks [28] for better performance, where nodes are classified via finding the nearest class prototype. G-Meta [12]

leverages local subgraphs to learn node representations while combining meta-learning [7] for model generalization. More recently, RALE [18] learns to model node dependencies within each meta-task by assigning relative and absolute locations for nodes with task-level and graph-level dependencies, respectively.

6 CONCLUSION

In this paper, we study the problem of few-shot node classification, which aims at predicting labels for nodes in novel classes with limited labeled nodes. Furthermore, to address the associated challenges caused by insufficient labeled nodes and the variety of novel classes, we propose a novel framework TENT to perform task adaptations for each meta-task from three perspectives: node-level, class-level, and task-level. As a result, our framework can perform these adaptions to each meta-task and advance classification performance with respect to a variety of novel classes during meta-test. Moreover, extensive experiments are conducted on four prevalent few-shot node classification datasets. The experimental results further validate that TENT outperforms other state-of-the-art baselines. In addition, the ablation study also verifies the effectiveness of three different levels of adaptations in our framework. Nevertheless, there still exists a considerable number of difficulties in few-shot node classification. For example, the inductive setting for few-shot node classification is still challenging. Future work may incorporate more sophisticated adaptation methods to handle the novel classes on graphs unseen during meta-training.

ACKNOWLEDGEMENT

This material is supported by the National Science Foundation (NSF) under grant #2006844.

REFERENCES

[1] Aleksandar Bojchevski and Stephan Günnemann. 2018. Deep Gaussian Embedding of Graphs: Unsupervised Inductive Learning via Ranking. In *ICLR*.
[2] Shaosheng Cao, Wei Lu, and Qiongkai Xu. 2016. Deep neural networks for learning graph representations. In *AAAI*.
[3] Shiyu Chang, Wei Han, Jiliang Tang, Guo-Jun Qi, Charu C Aggarwal, and Thomas S Huang. 2015. Heterogeneous network embedding via deep architectures. In *SIGKDD*.
[4] Jatin Chauhan, Deepak Nathani, and Manohar Kaul. 2020. Few-Shot Learning on Graphs via Super-Classes based on Graph Spectral Measures. In *ICLR*.
[5] Kaize Ding, Jundong Li, Nitin Agarwal, and Huan Liu. 2020. Inductive anomaly detection on attributed networks. In *IJCAI*.
[6] 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*.
[7] Chelsea Finn, Pieter Abbeel, and Sergey Levine. 2017. Model-agnostic meta-learning for fast adaptation of deep networks. In *ICML*.
[8] Xavier Glorot and Yoshua Bengio. 2010. Understanding the difficulty of training deep feedforward neural networks. In *AISTATS*.
[9] William L Hamilton, Rex Ying, and Jure Leskovec. 2017. Inductive representation learning on large graphs. In *NeurIPS*.
[10] Kaiming He, Haoqi Fan, Yuxin Wu, Saining Xie, and Ross Girshick. 2020. Momentum contrast for unsupervised visual representation learning. In *CVPR*.
[11] Weihua Hu, Matthias Fey, Marinka Zitnik, Yuxiao Dong, Hongyu Ren, Bowen Liu, Michele Catasta, and Jure Leskovec. 2020. Open Graph Benchmark: Datasets for Machine Learning on Graphs. In *NeurIPS*.
[12] Kexin Huang and Marinka Zitnik. 2020. Graph meta learning via local subgraphs. In *NeurIPS*.
[13] Prannay Khosla, Piotr Teterwak, Chen Wang, Aaron Sarna, Yonglong Tian, Phillip Isola, Aaron Maschinot, Ce Liu, and Dilip Krishnan. 2020. Supervised contrastive learning. In *NeurIPS*.
[14] Diederik P Kingma and Jimmy Ba. 2015. Adam: A method for stochastic optimization. In *ICLR*.
[15] Thomas N Kipf and Max Welling. 2017. Semi-supervised classification with graph convolutional networks. In *ICLR*.
[16] Moshe Lichtenstein, Prasanna Sattigeri, Rogerio Feris, Raja Giryes, and Leonid Karlinsky. 2020. Tafssl: Task-adaptive feature sub-space learning for few-shot classification. In *ECCV*.
[17] Lu Liu, Tianyi Zhou, Guodong Long, Jing Jiang, and Chengqi Zhang. 2019. Learning to propagate for graph meta-learning. In *NeurIPS*.
[18] Zemin Liu, Yuan Fang, Chenghao Liu, and Steven CH Hoi. 2021. Relative and absolute location embedding for few-shot node classification on graph. In *AAAI*.
[19] Ning Ma, Jiajun Bu, Jieyu Yang, Zhen Zhang, Chengwei Yao, Zhi Yu, Sheng Zhou, and Xifeng Yan. 2020. Adaptive-Step Graph Meta-Learner for Few-Shot Graph Classification. In *CIKM*.
[20] Julian McAuley, Rahul Pandey, and Jure Leskovec. 2015. Inferring networks of substitutable and complementary products. In *SIGKDD*.
[21] Julian J McAuley and Jure Leskovec. 2012. Learning to discover social circles in ego networks.. In *NeurIPS*.
[22] Nikhil Mishra, Mostafa Rohaninejad, Xi Chen, and Pieter Abbeel. 2018. A Simple Neural Attentive Meta-Learner. In *ICLR*.
[23] Aaron van den Oord, Yazhe Li, and Oriol Vinyals. 2018. Representation learning with contrastive predictive coding. In *arXiv:1807.03748*.
[24] Boris N Oreshkin, Pau Rodriguez, and Alexandre Lacoste. 2018. Tadam: Task dependent adaptive metric for improved few-shot learning. In *NeurIPS*.
[25] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. 2017. Automatic differentiation in pytorch. In *NeurIPS*.
[26] Ethan Perez, Florian Strub, Harm De Vries, Vincent Dumoulin, and Aaron Courville. 2018. Film: Visual reasoning with a general conditioning layer. In *AAAI*.
[27] Sachin Ravi and Hugo Larochelle. 2016. Optimization as a model for few-shot learning. In *ICLR*.
[28] Jake Snell, Kevin Swersky, and Richard Zemel. 2017. Prototypical networks for few-shot learning. In *NeurIPS*.
[29] Flood Sung, Yongxin Yang, Li Zhang, Tao Xiang, Philip HS Torr, and Timothy M Hospedales. 2018. Learning to compare: relation network for few-shot learning. In *CVPR*.
[30] Qiuling Suo, Jingyuan Chou, Weida Zhong, and Aidong Zhang. 2020. Tadanet: Task-adaptive network for graph-enriched meta-learning. In *SIGKDD*.
[31] Damian Szklarczyk, Annika L Gable, David Lyon, Alexander Junge, Stefan Wyder, Jaime Huerta-Cepas, Milan Simonovic, Nadezhda T Doncheva, John H Morris, Peer Bork, et al. 2019. STRING v11: protein–protein association networks with increased coverage, supporting functional discovery in genome-wide experimental datasets. In *Nucleic Acids research*.
[32] Jie Tang, Jing Zhang, Limin Yao, Juanzi Li, Li Zhang, and Zhong Su. 2008. Arnetminer: extraction and mining of academic social networks. In *SIGKDD*.
[33] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. 2018. Graph attention networks. *ICLR*.
[34] Oriol Vinyals, Charles Blundell, Timothy Lillicrap, Daan Wierstra, et al. 2016. Matching networks for one shot learning. In *NeurIPS*.
[35] Kuansan Wang, Zhihong Shen, Chiyuan Huang, Chieh-Han Wu, Yuxiao Dong, and Anshul Kanakia. 2020. Microsoft academic graph: When experts are not enough. In *Quantitative Science Studie*.
[36] Ning Wang, Minnan Luo, Kaize Ding, Lingling Zhang, Jundong Li, and Qinghua Zheng. 2020. Graph Few-Shot Learning with Attribute Matching. In *CIKM*.
[37] Song Wang, Xiao Huang, Chen Chen, Liang Wu, and Jundong Li. 2021. REFORM: Error-Aware Few-Shot Knowledge Graph Completion. In *CIKM*.
[38] Zhihao Wen, Yuan Fang, and Zemin Liu. 2021. Meta-inductive node classification across graphs. In *SIGIR*.
[39] Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and S Yu Philip. 2020. A comprehensive survey on graph neural networks. In *IEEE TNNLS*.
[40] Wenhan Xiong, Mo Yu, Shiyu Chang, Xiaoxiao Guo, and William Yang Wang. 2018. One-shot relational learning for knowledge graphs. In *EMNLP*.
[41] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. 2019. How powerful are graph neural networks?. In *ICLR*.
[42] Keyulu Xu, Chengtao Li, Yonglong Tian, Tomohiro Sonobe, Ken-ichi Kawarabayashi, and Stefanie Jegelka. 2018. Representation learning on graphs with jumping knowledge networks. In *ICML*.
[43] Huaxiu Yao, Chuxu Zhang, Ying Wei, Meng Jiang, Suhang Wang, Junzhou Huang, Nitesh Chawla, and Zhenhui Li. 2020. Graph few-shot learning via knowledge transfer. In *AAAI*.
[44] Ziwei Zhang, Peng Cui, and Wenwu Zhu. 2020. Deep learning on graphs: A survey. In *IEEE TKDE*.
[45] 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*.
[46] Jie Zhou, Ganqu Cui, Shengding Hu, Zhengyan Zhang, Cheng Yang, Zhiyuan Liu, Lifeng Wang, Changcheng Li, and Maosong Sun. 2020. Graph neural networks: A review of methods and applications. In *AI Open*.

A APPENDIX

A.1 Notations

To provide better understandings, we present the utilized notations in this paper and the corresponding descriptions.

Table 3: Notations used in this paper.

Notations	Definitions or Descriptions
G	the input graph
$\mathcal{V}$, $\mathcal{E}$	the node set and the edge set of G
$\mathbf{X}$	the input node features of G
C_b,C_n	the base class set and the novel class set
$\mathcal{T}_i$, $\mathcal{S}_i$, $\mathcal{Q}_i$	the i-th meta-task and its support set and query set
α_i, β_i, τ_i	adaptation parameters for the i-th class
N	the number of support classes in each meta-task
K	the number of labeled nodes in each class
N_t, K_t	the value of N and K during meta-training
$\mathbf{s}_i$	the embedding of the i-th class in each meta-task
$\mathbf{q}_i$	the embedding of the i-th query node in each meta-task
$\mathbf{p}_i$	the classification probabilities of the i-th query node over C_b

A.2 Reproducibility

In this section, we present the details on the reproducibility of our experiments. More specifically, we first elaborate on the implementation setting of our experiments. Then we introduce the required packages with the corresponding versions, followed by the experimental settings of baselines used in our main experiments. Finally, we provide details of datasets used in this paper.

A.2.1 Implementation of TENT. Our framework TENT is implemented based on PyTorch [25]. We train our model on a single 16GB Nvidia V100 GPU. For the specific implementation setting, we set the number of training epochs T as 500. We implement GNN_θ and GNN_ϕ using two-layer GINs [41] with the hidden sizes d_h and d_s both set as 16. To effectively initialize GNNs in our experiments, we utilize the Xavier initialization [8]. The READOUT function is implemented as mean-pooling. For the model optimization, we adopt Adam [14] with the learning rate of 0.05 and a dropout rate of 0.2. The weight decay rate is set as 10^{-4} and the loss weight γ is set as 1. Finally, the model that achieves the best result on the validation dataset will be saved and used for test. In addition, we randomly sample 500 tasks from novel classes C_n (i.e., T_{test}=500) for test with a query set size $|Q|$ of 10. Furthermore, to keep consistency, the test tasks are identical for all baselines. Our code is provided at https://github.com/SongW-SW/TENT.

A.2.2 Required Packages. The more detailed package requirements are listed as below.

- Python == 3.7.10
- torch == 1.8.1
- torch-cluster == 1.5.9
- torch-scatter == 2.0.6
- torch-sparse == 0.6.9
- torch-geometric == 1.4.1
- torch-spline-conv==1.2.1
- numpy == 1.18.5
- scipy == 1.5.3
- cuda == 11.0
- tensorboard == 2.2.2
- networkx == 2.5.1
- scikit-learn == 0.24.1
- pandas==1.2.3

A.2.3 Baseline Setting. Here, we present the detailed parameter setting of baselines. We mainly follow the original setting in the corresponding source code while adopting specific selections of parameters for better performance.

- **Prototypical Network (PN)** [28]: For PN, we set the learning rate as 0.005 with a weight decay of 0.0005.
- **MAML** [7]: The meta-learning rate is set as 0.001 and the number of update step is 10 with a learning rate of 0.01.
- **GCN** [15]: The learning rate is set as 0.001 and the hidden size of GCN is set as 32.
- **G-Meta** [12]: For G-Meta, we set the meta-learning rate as 0.001. The number of update step is 10 and the update learning rate is 0.01. The dimension size of GNN is 128.
- **GPN** [6]: For GPN, we follow the setting in the source code and set the learning rate as 0.005 with a weight decay of 0.0005. The dimension sizes of two GNNs used in GPN are set as 32 and 16, respectively.
- **RALE** [18]: We follow the setting in the source code and set the learning rates for training and fine-tuning as 0.001 and 0.01, respectively. The dropout rate is set as 0.6. The hidden size of used GNNs is 32.

A.2.4 Dataset Description. In this section, we describe the detailed dataset settings. Specifically, among the four prevalent datasets used in our experiments, `Amazon-E` [20] and `DBLP` [32] datasets are obtained from [6], while `Cora-full` [1] and `OGBN-arxiv` [11] are obtained from the corresponding sources and processed by us. The statistics and details are as follows:

- **Amazon-E** [20] is a product network, where nodes represent different "Electronics" products on Amazon. Moreover, edges are created according to the "viewed" relationship and class labels are assigned from the low-level product categories. For this dataset, we use 90/37/40 node classes for training/validation/test.
- **DBLP** [32] is a citation network. More specifically, each node represents a paper, and links are created according to the citation relations. The attributes are obtained via the paper abstract, and the class labels denote the paper venues. For this dataset, we use 80/27/30 node classes for training/validation/test.
- **Cora-full** [1] is a prevalent citation network, where nodes are labeled based on the paper topic. This dataset extends the prevalent small dataset via extracting original data from the entire network. For this dataset, we use 25/20/25 node classes for training/validation/test.
- **OGBN-arxiv** [11] is a directed citation network of all CS arXiv papers indexed by MAG [35], where nodes represent arXiv papers and edges indicate citations. The feature of each node is a 128-dimensional feature vector obtained by averaging the embeddings of words in its title and abstract. The labels are assigned according to 40 subject areas of arXiv CS papers. We use 15/5/20 node classes for training/validation/test.