Non-Homophilic Graph Pre-Training and Prompt Learning

Xingtong Yu*‡
Singapore Management University
Singapore
xingtongyu@smu.edu.sg

Jie Zhang*
National University of Singapore
Singapore
jiezhang_jz@u.nus.edu

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

Renhe Jiang†
The University of Tokyo
Japan
jiangrh@csis.u-tokyo.ac.jp

Abstract

Graphs are ubiquitous for modeling complex relationships between objects across various fields. Graph neural networks (GNNs) have become a mainstream technique for graph-based applications, but their performance heavily relies on abundant labeled data. To reduce labeling requirement, pre-training and prompt learning has become a popular alternative. However, most existing prompt methods do not distinguish between homophilic and heterophilic characteristics in graphs. In particular, many real-world graphs are *non-homophilic*—neither strictly nor uniformly homophilic—as they exhibit varying homophilic and heterophilic patterns across graphs and nodes. In this paper, we propose ProNoG, a novel pre-training and prompt learning framework for such non-homophilic graphs. First, we examine existing graph pre-training methods, providing insights into the choice of pre-training tasks. Second, recognizing that each node exhibits unique non-homophilic characteristics, we propose a conditional network to characterize node-specific patterns in downstream tasks. Finally, we thoroughly evaluate and analyze ProNoG through extensive experiments on ten public datasets.

CCS Concepts

• **Information systems → Data mining**; • **Computing methodologies → Machine learning**.

Keywords

Non-homophilic graph, prompt learning, graph pre-training

ACM Reference Format:
Xingtong Yu*‡, Jie Zhang*, Yuan Fang†, and Renhe Jiang†. 2025. Non-Homophilic Graph Pre-Training and Prompt Learning. In *Proceedings of the 31st ACM SIGKDD Conference on Knowledge Discovery and Data Mining V.1 (KDD '25), August 3–7, 2025, Toronto, ON, Canada.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3690624.3709219

1 Introduction

Graph data are pervasive in real-world applications, such as citation networks, social networks, transportation systems, and molecular graphs. Traditional methods typically train graph neural networks (GNNs) [18, 41] or graph transformers [51, 58] in a supervised manner. However, they require substantial labeled data and retraining for each specific task.

To mitigate the limitations of supervised methods, pre-training methods have gained significant traction [15, 42, 52]. They first learn universal, task-independent properties from unlabeled graphs, and then fine-tune the pre-trained models to various downstream tasks using task-specific labels [42, 52]. However, a significant gap occurs between the pre-training objectives and downstream tasks, resulting in suboptimal performance [39, 53]. Moreover, fine-tuning large pre-trained models is costly and still requires sufficient task-specific labels to prevent overfitting. As an alternative to fine-tuning, prompt learning has emerged as a popular parameter-efficient technique for adaptation to downstream tasks [7, 23, 37, 54]. They first utilize a universal template to unify pre-training and downstream tasks. Then, a learnable prompt is employed to modify the input features or hidden embeddings of the pre-trained model to align with the downstream task without updating the pre-trained weights. Since a prompt has far fewer parameters than the pre-trained model, prompt learning can be especially effective in low-resource settings [53].

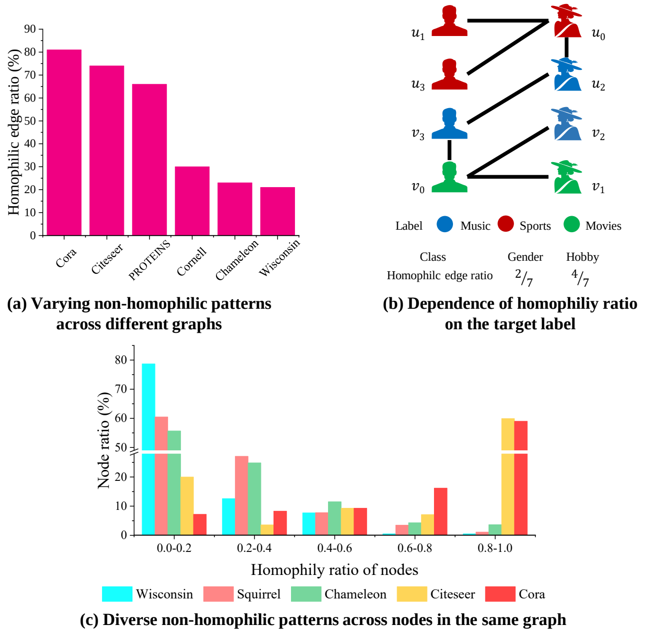
However, current graph "pre-train, prompt" approaches rely on the homophily assumption or overlook the presence of heterophilic edges. Specifically, the homophily assumption [26, 65] states that neighboring nodes should share the same labels, whereas heterophily refers to the opposite scenario where two neighboring nodes have different labels. We observe that real-world graphs are typically *non-homophilic*, meaning they are *neither strictly nor uniformly homophilic* and mix *both homophilic and heterophilic patterns* [47, 48]. In this work, we investigate the pre-training and prompt learning methodology for non-homophilic graphs. We first revisit existing graph pre-training methods, and then propose a **Pro**mpt learning framework for **N**on-h**o**mophilic **G**raphs (or ProNoG in short). The solution is non-trivial, as the notion of homophily encompasses two key aspects, each with its own unique challenge.

First, different graphs exhibit varying degrees of non-homophily. As shown in Fig. 1(a), the *Cora* citation network is typically considered largely homophilic with 81% homophilic edges[1], whereas

[*] Co-first authors. [‡] Part of the work was done at the University of Tokyo.
[†] Corresponding authors.

KDD '25, August 3–7, 2025, Toronto, ON, Canada.
© 2025 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ACM ISBN 979-8-4007-1245-6/25/08
https://doi.org/10.1145/3690624.3709219

[1] Defined as edges connecting two nodes of the same label; see Eq. (1) in Sect. 3.



(a) Varying non-homophilic patterns across different graphs

(b) Dependence of homophiliy ratio on the target label

(c) Diverse non-homophilic patterns across nodes in the same graph

Figure 1: Non-homophilic characteristics of graphs.

the *Wisconsin* web graph links different kinds of webpages, which is highly heterophilic with only 21% homophilic edges. Moreover, the non-homophilic characteristics of a graph also depends on the target label. For example, in a dating network shown in Fig. 1(b), taking gender as the node label, the graph is more heterophilic with 2/7 homophilic edges. However, taking hobbies as the node label, the graph becomes more homophilic with 4/7 homophilic edges. Hence, *how do we pre-train a graph model irrespective of the graph's homophily characteristics?* In this work, we propose definitions for *homophily tasks* and *homophily samples.* We show that pre-training with non-homophily samples increases the loss of any homophily task. Meanwhile, a less homophilic graph results in a higher number of non-homophily samples, subsequently increasing the pre-training loss for homophily tasks. This motivates us to move away from homophily tasks for graph pre-training [23, 37] and instead choose a non-homophily task [47, 52].

Second, different nodes within the same graph are distributed differently in terms of their non-homophilic characteristics. As shown in Fig. 1(c), on each dataset, different nodes within the same graph generally exhibit diverse homophily ratios[2]. Hence, *how do we capture the fine-grained, node-specific non-homophilic characteristics?* Due to the diverse characteristics across nodes, a one-size-fits-all solution for all nodes would be inadequate. However, existing approaches generally apply a single prompt to all nodes [23, 37], treating all nodes uniformly. Thus, these methods overlook the fine-grained node-wise non-homophilic characteristics, leading to suboptimal performance. Although some recent works [5, 40] have proposed node-specific prompts, they are not designed to account for the variation in nodes' non-homophilic charactersitics. Inspired by conditional prompt learning [62], we propose generating a unique prompt from each node with a conditional network (condition-net) to capture the distinct characteristics of each node. We first capture the non-homophilic patterns of each node by reading out its multi-hop neighborhood. Then, conditioned on these non-homophilic patterns, the condition-net produces a series of prompts, one for each node that reflects its varying non-homophilic characteristics. These prompts can adjust the node embeddings to better align them with the downstream task.

In summary, the contributions of this work are threefold: (1) We observe varying degrees of homophily across graphs, which motivates us to revisit graph pre-training tasks. We provide theoretical insights which guide us to choose non-homophily tasks for graph pre-training. (2) We further observe that, within the same graph, different nodes have diverse distributions of non-homophilic characteristics. To adapt to the unique non-homophilic patterns of each node, we propose the ProNoG framework for non-homophilic prompt learning, which is equipped with a condition-net to generate a series of prompts conditioned on each node. The node-specific prompts enables fine-grained, node-wise adaptation for the downstream tasks. (3) We perform extensive experiments on ten benchmark datasets, demonstrating the superior performance of ProNoG compared to a suite of state-of-the-art methods.

2 Related Work

In the following, we briefly review the literature on general and non-homophilic graph learning, as well as graph prompt learning.

Graph representation learning. GNNs [18, 41] are mainstream approaches for graph representation learning. They typically operate on a message-passing framework, where nodes iteratively update their representations by aggregating messages received from their neighboring nodes [11, 49, 55]. However, their effectiveness relies on abundant task-specific labeled data and requires re-training for each task. Inspired by the success of the pre-training paradigm in the language [4, 6, 9, 35] and vision [1, 59, 62, 63] domains, pre-training methods [14, 15, 17, 24] have been widely explored for graphs. These methods first pre-train a graph encoder based on self-supervised tasks, and subsequently transfer the pre-trained prior knowledge to downstream tasks. However, these pre-training methods often make the homophily assumption, overlooking that real-world graphs are generally non-homophilic.

Non-homophilic graph learning. Many GNNs [25, 26, 64] have been proposed for non-homophilic graphs, employing techniques such as capturing high-frequency signals [2], discovering potential neighbors [16, 30], and high-order message passing [65]. Moreover, recent works have explored pre-training on non-homophilic graphs [13, 47, 48] by capturing neighborhood information to construct unsupervised tasks for pre-training a graph encoder, and then transferring prior non-homophilic knowledge to downstream tasks through fine-tuning with task-specific supervision. However, a significant gap exists between the objectives of pre-training and fine-tuning [20, 39, 53]. While pre-training focuses on learning inherent graph properties without supervision, fine-tuning adapts these properties to downstream tasks based on task-specific supervision. This discrepancy hinders effective knowledge transfer and negatively impacts downstream performance.

[2] Defined as the proportion of a node's neighbors that share the same label as the node; refer to Eq. (2) in Sect. 3.

Graph prompt learning. Originally developed for the language domain, prompt learning effectively unifies pre-training and downstream objectives [4, 19, 21]. Recently, graph prompt learning has emerged as a popular alternation to fine-tuning methods [23, 37, 54, 56]. These methods first propose a unified template, then design prompts specifically tailored to each downstream task, allowing them to better align with the pre-trained model while keeping the pre-trained parameters frozen. However, current graph prompt learning methods [38, 53] typically do not consider the fact that real-world graphs are generally non-homophilic, exhibiting a mixture of diverse homophilic and heterophilic patterns across nodes. Hence, these methods usually apply a single prompt to all nodes, overlooking the unique non-homophilic pattern of each node.

3 Preliminaries

Graph. A graph is defined as $G = (V, E)$, where V represents the set of nodes and E represents the set of edges. The nodes are also associated with a feature matrix $\mathbf{X} \in \mathbb{R}^{|V|\times d}$, such that $\mathbf{x}_v \in \mathbb{R}^d$ is a row of $\mathbf{X}$ representing the feature vector for node $v \in V$. For a collection of multiple graphs, we use the notation $\mathcal{G} = \{G_1, G_2, \dots, G_N\}$.

Homophily ratio. Given a mapping between the nodes of a graph and a predefined set of labels, let y_v denote the label mapped to node v. The homophily ratio $\mathcal{H}(G)$ evaluates the relationships between the labels and the graph structure [26, 65], measuring the fraction of homophilic edges whose two end nodes share the same label. More concretely,

$$\mathcal{H}(G) = \frac{|\{(u, v) \in E : y_u = y_v\}|}{|E|}. \tag{1}$$

Additionally, the homophily ratio can be defined for each node based on its local structure [27, 48], measuring the fraction of a node's neighbors that share the same label. This node-specific ratio can be defined as

$$\mathcal{H}(v) = \frac{|\{u \in \mathcal{N}(v) : y_u = y_v\}|}{|\mathcal{N}(v)|}, \tag{2}$$

where $\mathcal{N}(v)$ is the set of neighboring nodes of v. Note that both $\mathcal{H}(G)$ and $\mathcal{H}(v)$ fall in $[0, 1]$. Graphs or nodes with a larger proportion of homophilic edges have a higher homophily ratio.

Graph encoder. Graph encoders learn latent representations of graphs, embedding their nodes into some feature space. A widely used family of graph encoders is GNNs, which typically utilize a message-passing mechanism [46, 61]. Specifically, each node aggregates messages from its neighbors to generate its own representation. By stacking multiple layers, GNNs enables recursive message passing throughout the graph. Formally, the embedding of a node v in the l-th GNN layer, denoted as $\mathbf{h}_v^l$, is computed as follows.

$$\mathbf{h}_v^l = \texttt{Aggr}(\mathbf{h}_v^{l-1}, \{\mathbf{h}_u^{l-1} : u \in \mathcal{N}(v)\}; \theta^l), \tag{3}$$

where θ^l are the learnable parameters in the l-th layer, and $\texttt{Aggr}(\cdot)$ is the aggregation function, which can take various forms [11, 18, 41, 49, 55]. In the first layer, the input node embedding $\mathbf{h}_v^0$ is typically initialized from the node feature vector $\mathbf{x}_v$. The full set of learnable parameters is denoted as $\Theta = \{\theta^1, \theta^2, \dots\}$. For simplicity, we define the output node representations of the final layer as $\mathbf{h}_v$, which can then be fed into the loss function for a specific task.

Problem statement. In this work, we aim to pre-train a graph encoder and develop a prompt learning framework for non-homophilic graphs. More specifically, both the pre-training and prompt learning are not sensitive to the diverse non-homophilic characteristics of the graph and its nodes.

To evaluate our non-homophilic pre-training and prompt learning, we focus on two common tasks on graph data: node classification and graph classification, in few-shot settings. For node classification within a graph $G = (V, E)$ over a set of node classes Y, each node $v_i \in V$ has a class label $y_i \in Y$. Similarly, for graph classification across a graph collection $\mathcal{G}$ with class labels Y, each graph $G_i \in \mathcal{G}$ has a class label $Y_i \in Y$. In the few-shot setting, there are only k labeled samples per class, where k is a small number (e.g., $k \leq 10$). This scenario is known as k-shot classification [23, 54, 57]. Note that the homophily ratio is defined with respect to some predefined set of labels, which may or may not be related to the class labels in downstream tasks.

4 Revisiting Graph Pre-training

In this section, we revisit graph pre-training tasks to cope with non-homophilic graphs. We first propose the definition of *homophily tasks* and reveal its connection to the training loss. The theoretical insights further guide us in choosing graph pre-training tasks.

4.1 Theoretical Insights

We focus on contrastive graph pre-training tasks. Consider a mainstream contrastive task [12, 23, 32, 52, 54, 66], $T = (\{\mathcal{A}_u : u \in V\}, \{\mathcal{B}_u : u \in V\})$, where $\mathcal{A}_u$ is the set of positive instances for node u, and $\mathcal{B}_u$ is the set of negative instances for u. Its loss function $\mathcal{L}_T$ can be standardized [54] to a similar form as follows.

$$\mathcal{L}_T = -\sum_{u\in V} \ln P(u, \mathcal{A}_u, \mathcal{B}_u), \tag{4}$$

$$P(u, \mathcal{A}_u, \mathcal{B}_u) \triangleq \frac{\sum_{a\in\mathcal{A}_u} \texttt{sim}(\mathbf{h}_u, \mathbf{h}_a)}{\sum_{a\in\mathcal{A}_u} \texttt{sim}(\mathbf{h}_u, \mathbf{h}_a) + \sum_{b\in\mathcal{B}_u} \texttt{sim}(\mathbf{h}_u, \mathbf{h}_b)}, \tag{5}$$

where $\texttt{sim}(\cdot, \cdot)$ represents a similarity function such as cosine similarity in our experiments. The optimization objective of task T in Eq. (4) is to maximize the similarity between u and its positive instances while minimizing the similarity between u and its negative instances. Based on this loss, we further propose the definitions of *homophily tasks* and *homophily samples*.

Definition 1 (Homophily Task). *On a graph $G = (V, E)$, a pre-training task $T = (\{\mathcal{A}_u : u \in V\}, \{\mathcal{B}_u : u \in V\})$ is a homophily task if and only if, $\forall u \in V, \forall a \in \mathcal{A}_u, \forall b \in \mathcal{B}_u, (u, a) \in E \wedge (u, b) \notin E$. A task that is not a homophily task is called a non-homophily task.* □

In particular, the widely used link prediction task [23, 29, 31, 54, 56, 57] is a homophily task, where $\mathcal{A}_u$ is a subset of nodes linked to u and $\mathcal{B}_u$ is a subset of nodes not linked to u.

Definition 2 (Homophily Sample). *On a graph $G = (V, E)$, consider a triplet (u, a, b) where $u \in V$, $(u, a) \in E$ and $(u, b) \notin E$. The triplet (u, a, b) is a homophily sample if and only if $\texttt{sim}(\mathbf{h}_u, \mathbf{h}_a) > \texttt{sim}(\mathbf{h}_u, \mathbf{h}_b)$, and it is a non-homophily sample otherwise.* □

Subsequently, we can establish the following theorems.

Theorem 1. *For a homophily task T, adding a homophily sample always results in a smaller loss than adding a non-homophily sample.*

Proof. Consider a homophily sample (u, a, b) for some $(u, a) \in E$ and $(u, b) \notin E$, as well as a non-homophily sample (u, a', b') for some $(u, a') \in E$ and $(u, b') \notin E$. Let the overall loss with (u, a, b) be L_T, and that with (u, a', b') be L'_T. Since (u, a, b) is a homophily sample, we have $\texttt{sim}(\mathbf{h}_u, \mathbf{h}_a) > \texttt{sim}(\mathbf{h}_u, \mathbf{h}_b)$, and thus

$$p(u, a, b) = \frac{\texttt{sim}(\mathbf{h}_u,\mathbf{h}_a)}{\texttt{sim}(\mathbf{h}_u,\mathbf{h}_a)+\texttt{sim}(\mathbf{h}_u,\mathbf{h}_b)} > 0.5.$$

Moreover, since (u, a', b') is a non-homophily sample, we have $\texttt{sim}(\mathbf{h}_u, \mathbf{h}'_a) \leq \texttt{sim}(\mathbf{h}_u, \mathbf{h}'_b)$, and thus $p(u, a', b') \leq 0.5$. Hence, $p(u, a, b) > p(u, a', b')$, implying that $L_T < L'_T$. □

Theorem 2. *Consider a graph $G = (V, E)$ with a label mapping function $V \longrightarrow Y$, where $y_v \in Y$ is the label mapped to $v \in V$. Suppose that the label mapping and node similarity are consistent, i.e.,*

$$\forall u, a, b \in V, y_u = y_a \wedge y_u \neq y_b \Rightarrow \texttt{sim}(\mathbf{h}_u, \mathbf{h}_a) > \texttt{sim}(\mathbf{h}_u, \mathbf{h}_b).$$

Let $\mathbb{E}_T$ denote the expected number of homophily samples for a homophily task T on the graph G. Then, $\mathbb{E}_T$ increases monotonically as the homophily ratio $\mathcal{H}(G)$ defined w.r.t. Y increases.

Proof. For a homophily task $T = (\{\mathcal{A}_u : u \in V\}, \{\mathcal{B}_u : u \in V\})$, a triplet (u, a, b) for some $u \in V$, $a \in \mathcal{A}_u$ and $b \in \mathcal{B}_u$ is a homophily sample with a probability of $P(y_u = y_a)(1 - P(y_u = y_b))$, since $y_u = y_a \wedge y_u \neq y_b$ implies $\texttt{sim}(\mathbf{h}_u, \mathbf{h}_a) > \texttt{sim}(\mathbf{h}_u, \mathbf{h}_b)$. Hence, the expected number of homophily samples for T is

$$\mathbb{E}_T = \sum_{u \in V} |\mathcal{A}_u||\mathcal{B}_u| P(y_u = y_a)(1 - P(y_u = y_b)). \tag{6}$$

For a constant number of nodes with label y_u, as $\mathcal{H}(G)$ increases, $P(y_u = y_a)$ increases while $P(y_u = y_b)$ decreases, leading to a larger $\mathbb{E}_T$. □

In the next part, the theorems will guide us in choosing the appropriate pre-training tasks for non-homophilic graphs.

4.2 Non-homophilic Graph Pre-training

Consider a homophily task T. Following Theorem 2, non-homophilic graphs with lower homophily ratios are expected to have fewer homophily samples and more non-homophily samples for T. Furthermore, based on Theorem 1, adding a non-homophily sample results in a larger loss than adding a homophily sample. Consequently, for non-homophilic graphs, especially those with low homophily ratio, homophily tasks are not optimal for working with standard contrastive training losses, whereas non-homophily tasks may offer a better alternative.

We revisit mainstream graph pre-training methods and categorize them into two categories: homophily methods that employ homophily tasks, and non-homophily methods that do not. Specifically, GPPT [37], GraphPrompt [23] and HGPrompt [56] are all homophily methods, since their pre-training tasks utilizes a form of link prediction, where $\mathcal{A}_u$ is a set of nodes linked to u, and $\mathcal{B}_u$ is a set of nodes not linked from u. In contrast, DGI [42], GraphCL [52], and GraphACL [48] are non-homophily methods, since $\mathcal{A}_u$ or $\mathcal{B}_u$ in their pre-training tasks is not related to the connectivity with u. Further details of these methods are shown in Appendix B.

In our experiments, we find that GraphCL [52], a non-homophily pre-training method, performs well across most graphs, including non-homophilic ones. We also experiment with link prediction [23, 37] and GraphACL [48] to study their effects on different graphs.

5 Non-homophilic Prompt Learning

In this section, we propose ProNoG, a prompt learning framework for non-homophilic graphs. We first introduce the overall framework, and then develop the prompt generation and tuning process. Finally, we analyze the complexity of the proposed algorithm.

5.1 Overall Framework

We illustrate the overall framework of ProNoG in Fig. 2. It involves two stages: (a) graph pre-training and (b) downstream adaptation. In graph pre-training, we pre-train a graph encoder using a non-homophilic pre-training task, as shown in Fig. 2(a). Subsequently, to adapt the pre-trained model to downstream tasks, we propose a conditional network (condition-net) that generates a series of prompts, as depicted in Fig. 2(b). As a result, each node is equipped with its own prompt, which can be used to modify its features to align with the downstream task. More specifically, the prompt generation is conditioned on the unique patterns of each node, in order to achieve fine-grained adaptation catering to the diverse non-homophilic characteristics of each node, as detailed in Fig. 2(c).

5.2 Prompt Generation and Tuning

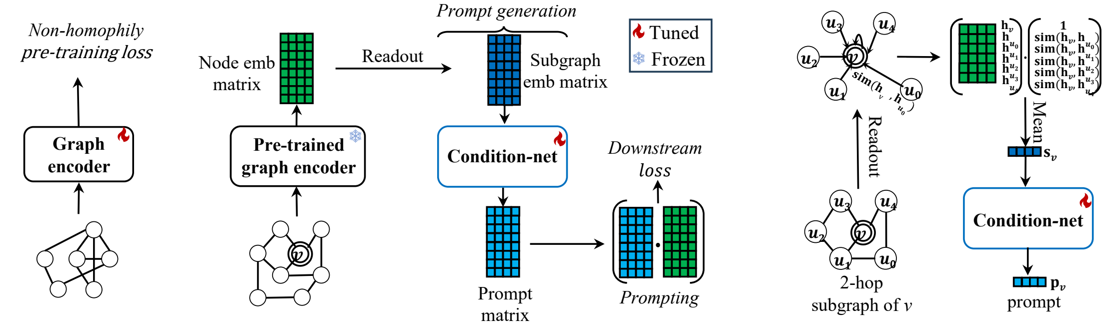
Prompt generation. In non-homophilic graphs, different nodes are characterized by unique non-homophilic patterns. Specifically, different nodes typically have diverse homophily ratios $\mathcal{H}(v)$, indicating distinct topological structures linking to their neighboring node. Moreover, even nodes with similar homophily ratios may have different neighborhood distributions in terms of the varying homophily ratios of the neighboring nodes. Therefore, instead of learning a single prompt for all nodes as in standard graph prompt learning [23, 37, 38, 54], we design a condition-net [62] to generate a series of non-homophilic pattern-conditioned prompts. Consequently, each node is equipped with its own unique prompt, aiming to adapt to its distinct non-homophilic characteristics.

First, the non-homophilic patterns of a node can be characterized by considering a multi-hop neighborhood around the node. Specifically, given a node v, we readout their δ-hop ego-network S_v, which is an induced subgraph containing the node v and nodes reachable from v in at most δ steps. Inspired by GGCN [50], the readout is weighted by the similarity between v and their neighbors, as shown in Fig. 2(c) , obtaining a representation of the subgraph S_v given by

$$\mathbf{s}_v = \frac{1}{|V(S_v)|} \sum_{u \in V(S_v)} \mathbf{h}_u \cdot \texttt{sim}(\mathbf{h}_u, \mathbf{h}_v), \tag{7}$$

where $V(S_v)$ denotes the set of nodes in S_v. In our experiment, we set $\delta = 2$ to balance between efficiency and capturing more unique non-homophilic patterns in the neighborhood of v.

Next, for each downstream task, our goal is to assign a unique prompt vector to each node. However, directly parameterizing these prompt vectors would significantly increase the number of learnable parameters, which may overfit to the lightweight supervision in few-shot settings. To cater to the unique non-homophilic characteristics of each node with minimal parameters, we propose to



(a) Pre-training **(b) Downstream adaptation with conditional prompting** **(c) Details of prompt generation**

Figure 2: Overall framework of ProNoG.

employ a condition-net [62] to generate node-specific prompt vectors. Specifically, conditioned on the subgraph readout $\mathbf{s}_v$ of a node v, the condition-net generates a unique prompt vector for v w.r.t. a task t, denoted by $\mathbf{p}_{t,v}$, as follows.

$$\mathbf{p}_{t,v} = \texttt{CondNet}(\mathbf{s}_v; \phi_t), \tag{8}$$

where CondNet is the condition-net parameterized by ϕ_t. It outputs a unique prompt vector $\mathbf{p}_{t,v}$, which varies based on the input s_v that characterizes the non-homophily patterns of node v. Note that this is a form of hypernetworks [10], which employs a secondary network to generate the parameters for the main network conditioned on the input feature. In our context, the condition-net is the secondary network, generating prompt parameters without expanding the number of learnable parameters in the main network. The secondary network CondNet can be any learnable function, such as a fully-connected layer or a multi-layer perceptron (MLP). We employ an MLP with a compact bottleneck architecture [45].

Subsequently, we perform fine-grained, node-wise adaptation to task t. Concretely, the prompt $\mathbf{p}_{t,v}$ for node v is employed to adjust v's features or its embeddings in the hidden or output layers [54]. In our experiments, we choose a simple yet effective implementation that modifies the nodes' output embeddings through an element-wise product, as follows.

$$\tilde{\mathbf{h}}_{t,v} = \mathbf{p}_{t,v} \odot \mathbf{h}_v, \tag{9}$$

where the prompt $\mathbf{p}_{t,v}$ is generated with an equal dimension as $\mathbf{h}_v$.

Prompt tuning. In this work, we focus on two common types of downstream task: node classification and graph classification. The prompt tuning process does not directly optimize the prompt vectors; instead it optimizes the condition-net, which subsequently generates the prompt vectors, for a given downstream task.

We utilize a loss function based on node/graph similarity following previous work [23, 54]. Formally, for a task t with a labeled training set $\mathcal{D}_t = \{(x_1, y_1), (x_2, y_2), \dots\}$, where x_i can be either a node or a graph, and $y_i \in Y$ is x_i's class label from a set of classes Y. The downstream loss function is

$$\mathcal{L}_{\text{down}}(\phi_t) = -\sum_{(x_i,y_i)\in\mathcal{D}_t} \ln \frac{\exp\left(\frac{1}{\tau}\text{sim}(\tilde{\mathbf{h}}_{t,x_i}, \bar{\mathbf{h}}_{t,y_i})\right)}{\sum_{c\in Y}\exp\left(\frac{1}{\tau}\text{sim}(\tilde{\mathbf{h}}_{t,x_i}, \bar{\mathbf{h}}_{t,c})\right)}, \tag{10}$$

where $\tilde{\mathbf{h}}_{t,x_i}$ denotes the output embedding of node v/graph G for task t. Specifically, for node classification $\tilde{\mathbf{h}}_{t,v}$ is the output embedding in Eq. (9); for graph classification, $\tilde{\mathbf{h}}_{t,G} = \sum_{u\in V} \tilde{\mathbf{h}}_{t,u}$, involving an additional graph readout. The prototype embedding for class c, $\bar{\mathbf{h}}_{t,c}$, is the average of the embedding of all labeled nodes/graphs belonging to class c.

During prompt tuning, we update only the lightweight parameters of the condition-net (ϕ_t), while freezing the pre-trained GNN weights. Thus, our approach is parameter-efficient and amenable to few-shot settings, where $\mathcal{D}_t$ contains only a small number of training examples for task t.

5.3 Algorithm and Complexity Analysis

Algorithm. We detail the main steps for the conditional prompt generation and tuning in Algorithm 1, Appendix A.

Complexity analysis. For a downstream graph G, the computational process of ProNoG involves two main parts: encoding nodes via a pre-trained GNN, and conditional prompt learning. The first part's complexity is determined by the GNN architecture, akin to other methods employing a pre-trained GNN. In a standard GNN, each node aggregates features from up to D neighbors per layer. Thus, the complexity of calculating node embeddings over L layers is $O(D^L \cdot |V|)$, where $|V|$ denotes the number of nodes. The second part, conditional prompt learning, has two stages: prompt generation and prompt tuning. In prompt generation, each subgraph embedding is fed into the condition-net. The subgraph embedding of each node involves a readout from the δ-hop neighborhood, resulting in a complexity of $O(D^\delta \cdot |V|)$ with at most D neighbors per hop. During prompt tuning, each node in G is adjusted using a prompt vector, with a complexity of $O(|V|)$. Therefore, the total complexity for conditional prompt learning is $O(D^\delta \cdot |V|)$.

In conclusion, the overall complexity of ProNoG is $O((D^L + D^\delta) \cdot |V|)$. As both L and δ are small constants, the two parts have comparable complexity. That is, the proposed conditional prompt learning does not increase the order of complexity relative to the pre-trained GNN encoder, if δ is chosen to be no larger than L.

6 Experiments

In this section, we conduct experiments to evaluate ProNoG, and analyze the empirical results.

Table 1: Accuracy evaluation on one-shot node classification.

Methods	Wisconsin	Squirrel	Chameleon	Cornell	PROTEINS	ENZYMES	Citeseer	Cora
GCN	21.39 ± 6.56	20.00 ± 0.29	25.11 ± 4.19	21.81 ± 4.71	43.32 ± 9.35	48.08 ± 4.71	31.27 ± 4.53	28.57 ± 5.07
GAT	28.01 ± 5.40	21.55 ± 2.30	24.82 ± 4.35	23.03 ± 13.19	31.79 ± 20.11	35.32 ± 18.72	30.76 ± 5.40	28.40 ± 6.25
H2GCN	23.60 ± 4.64	21.90 ± 2.15	25.89 ± 4.96	32.77 ± 14.88	29.60 ± 6.99	37.27 ± 8.73	26.98 ± 6.25	34.58 ± 9.43
FAGCN	35.03 ± 17.92	20.91 ± 1.79	22.71 ± 3.74	28.67 ± 17.64	32.63 ± 9.94	35.87 ± 13.47	26.46 ± 6.34	28.28 ± 9.57
DGI	28.04 ± 6.47	20.00 ± 1.86	19.33 ± 4.57	32.54 ± 15.66	45.22 ± 11.09	48.05 ± 14.83	45.00 ± 9.19	54.11 ± 9.60
GraphCL	29.85 ± 8.46	21.42 ± 2.22	27.16 ± 4.31	24.69 ± 14.06	46.15 ± 10.94	48.88 ± 15.98	43.12 ± 9.61	51.96 ± 9.43
DSSL	28.46 ± 10.31	20.94 ± 1.88	27.92 ± 3.93	20.36 ± 5.38	40.42 ± 10.08	66.59 ± 19.28	39.86 ± 8.60	40.79 ± 7.31
GraphACL	34.57 ± 10.46	24.44 ± 3.94	26.72 ± 4.67	33.17 ± 16.06	42.16 ± 13.50	47.57 ± 14.36	35.91 ± 7.87	46.65 ± 9.54
GPPT	27.39 ± 6.67	20.09 ± 0.91	24.53 ± 2.55	25.09 ± 2.92	35.15 ± 11.40	35.37 ± 9.37	21.45 ± 3.45	15.37 ± 4.51
GraphPrompt	31.48 ± 5.18	21.22 ± 1.80	25.36 ± 3.99	31.00 ± 13.88	47.22 ± 11.05	53.54 ± 15.46	45.34 ± 10.53	54.25 ± 9.38
GraphPrompt+	31.54 ± 4.54	21.24 ± 1.82	25.73 ± 4.50	31.65 ± 14.48	46.08 ± 9.96	57.68 ± 13.12	45.23 ± 10.01	52.51 ± 9.73
ProNoG	**44.72** ± 11.93	**24.59** ± 3.41	**30.67** ± 3.73	**37.90** ± 9.31	**48.95** ± 10.85	**72.94** ± 20.23	**49.02** ± 10.66	**57.92** ± 11.50

Results are reported in percent. The best method is bolded and the runner-up is underlined.

Table 2: Accuracy evaluation on one-shot graph classification.

Methods	Wisconsin	Squirrel	Chameleon	Cornell	PROTEINS	ENZYMES	BZR	COX2
GCN	21.39 ± 6.56	11.77 ± 3.10	17.21 ± 4.80	26.36 ± 4.35	51.66 ± 10.87	19.30 ± 6.36	45.06 ± 16.30	43.84 ± 13.94
GAT	24.93 ± 7.59	20.70 ± 1.51	25.71 ± 3.32	22.66 ± 12.46	51.33 ± 11.02	20.24 ± 6.39	46.28 ± 15.26	51.72 ± 13.70
H2GCN	22.23 ± 6.38	20.69 ± 1.42	26.76 ± 3.98	23.11 ± 11.78	53.81 ± 8.85	19.40 ± 5.57	50.28 ± 12.13	53.70 ± 11.73
FAGCN	23.81 ± 9.50	20.83 ± 1.43	25.93 ± 4.03	25.71 ± 13.12	55.45 ± 11.57	19.95 ± 5.94	50.93 ± 12.41	50.22 ± 11.50
DGI	29.77 ± 6.22	20.50 ± 1.52	24.29 ± 4.33	18.60 ± 12.79	50.32 ± 13.47	21.57 ± 5.37	49.97 ± 12.63	54.84 ± 14.76
GraphCL	27.93 ± 5.27	21.01 ± 1.86	26.45 ± 4.30	20.03 ± 10.05	54.81 ± 11.44	19.93 ± 5.65	50.50 ± 18.62	47.64 ± 22.42
DSSL	22.05 ± 3.90	20.74 ± 1.61	26.19 ± 3.72	18.38 ± 10.63	52.73 ± 10.98	**23.14** ± 6.71	49.04 ± 8.75	54.23 ± 14.17
GraphACL	22.98 ± 5.89	20.80 ± 1.28	26.28 ± 3.93	26.50 ± 17.18	**56.11** ± 13.95	20.28 ± 5.60	49.24 ± 17.87	49.59 ± 23.93
GraphPrompt	28.34 ± 3.89	**21.22** ± 1.80	26.51 ± 4.67	24.06 ± 13.71	53.61 ± 8.90	21.85 ± 6.17	50.46 ± 11.46	55.01 ± 15.23
GraphPrompt+	26.95 ± 7.42	20.80 ± 1.45	26.03 ± 4.17	25.31 ± 7.65	54.55 ± 12.61	21.85 ± 5.15	**53.26** ± 14.99	54.73 ± 14.58
ProNoG	**31.54** ± 5.30	20.92 ± 1.37	**28.50** ± 5.30	**27.17** ± 9.58	**56.11** ± 10.19	22.55 ± 6.70	51.62 ± 14.27	**56.46** ± 14.57

6.1 Experimental Setup

Datasets. We conduct experiments on ten benchmark datasets. *Wisconsin* [30], *Cornell* [30], *Chameleon* [34], and *Squirrel* [34] are all web graphs. Each dataset features a single graph, where nodes correspond to web pages and edges represent hyperlinks connecting these pages. *Cora* [28] and *Citeseer* [36] are citation networks. They consist of a single graph each, with nodes representing academic papers and edges indicating citation relationships. *PROTEINS* [3] consists of a series of protein graphs. Nodes in these graphs denote secondary structures, while edges depict neighboring relationships either within the amino acid sequence or in three-dimensional space. *ENZYMES* [44], *BZR* [33], and *COX2* [33] are collections of molecular graphs. These datasets describe enzyme structures from the BRENDA enzyme database, ligands related to benzodiazepine receptors, and cyclooxygenase-2 inhibitors, respectively. We summarize these datasets in Table 6, Appendix C.

Baselines. We evaluate ProNoG against a series of state-of-the-art methods from the following three categories. (1) *End-to-end GNNs*: GCN [18], GAT [41], H2GCN [65], and FAGCN [2] are trained in a supervised manner directly using downstream labels. Specifically, GCN and GAT are designed for homophilic graphs, whereas H2GCN is developed for heterophilic graphs, and FAGCN for non-homophilic graphs. (2) *Graph pre-training models*: DGI [42], GraphCL [52], DSSL [47], and GraphACL [48] follow the "pre-train, fine-tune" paradigm. (3) *Graph prompt learning models*: GPPT [37], GraphPrompt [23], and GraphPrompt+ [54] employ self-supervised pre-training tasks, and use the same prompts for all nodes in downstream adaptation. Note that GPPT is specifically designed for node classification and cannot be directly used for graph classification. Therefore, we evaluate GPPT on node classification tasks only.

Note that some graph few-shot learning methods, such as Meta-GNN [60], AMM-GNN [43], RALE [22], VNT [40], and ProG [38], are based on the meta-learning paradigm [8]. They require a set of labeled base classes in addition to the few-shot classes, and thus are not compared here.

Parameter settings. For all baselines, we use the original authors' code and reference their recommended settings, while further tuning their hyperparameters to ensure optimal performance. Detailed descriptions of the implementations and settings for both the baselines and our ProNoG are provided in Appendix C.

Downstream tasks and evaluation. We conduct two types of downstream task: *node classification*, and *graph classification*. These tasks are set up as k-shot classification problems, meaning that for each class, k instances (nodes or graphs) are randomly selected for supervision. The low-homophily datasets, *i.e.*, *Wisconsin*, *Squirrel*, *Chameleon* and *Cornell*, only comprise a single graph and cannot be directly used for graph classification. Thus, following previous research [24, 56], we generate multiple graphs by constructing ego-networks centered on the labeled nodes in each dataset. We then perform graph classification on these ego-networks, each labeled according to its ego node. Among datasets with high homophily ratios, *PROTEINS*, *ENZYMES*, *BZR* and *COX2* have ground-truth graph labels, which we employ directly for graph classification.

Since the k-shot tasks are balanced classification problems, we use accuracy to evaluate the performance, in line with prior studies [22, 23, 43, 54]. We pre-train the graph encoder once for each dataset and then use the same pre-trained model for all downstream tasks. We generate 100 k-shot tasks for both node classification and graph classification by repeating the sampling process 100 times. Each task is executed with five different random seeds, leading to a total of 500 results per task type. We report the mean and standard deviation of these 500 outcomes.

6.2 Performance Evaluation

We first evaluate one-shot classification tasks. Then, we vary the number of shots to investigate their impact on performance.

One-shot performance. We present the results of one-shot node and graph classification tasks on non-homophilic graphs in Tables 1 and 2, respectively. We make the following observations: (1) ProNoG surpasses the vast majority of baseline methods, outperforming the best competitor by up to 21.49% on node classification and 6.50% on graph classification. These results demonstrate its effectiveness in learning prior knowledge from non-homophilic graphs and capturing node-specific patterns. (2) Other graph prompt learning methods, *i.e.*, GPPT, GraphPrompt, and GraphPrompt+, significantly lag behind ProNoG. Their suboptimal performance can be attributed to their inability to account for a variety of node-specific patterns. These results underscore the importance of our conditional prompting in characterizing node embeddings to capture the unique pattern of each node. (3) GPPT is at best comparable to, and often performs worse than other baselines because it is not well suited to few-shot learning.

Few-shot performance. To assess the performance of ProNoG with different amounts of labeled data, we vary the number of shots in the node classification tasks. We present the results in Fig. 3 with several competitive baselines for selected datasets. Note that given the limited number of nodes in *Wisconsin*, we only conduct tasks up to 3 shots. We observe that ProNoG generally outperforms these baselines in low-shot scenarios ($k \leq 5$) by a significant margin, showcasing the effectiveness of our approach with limited labeled data. Furthermore, as the number of shots increases, while all methods generally show improved performance, ProNoG remains competitive and often surpasses the other methods. We focus on the one-shot setting for the remaining experimental results.

6.3 Ablation Study

To comprehensively understand the influence of conditional prompt learning in ProNoG, we perform an ablation study comparing

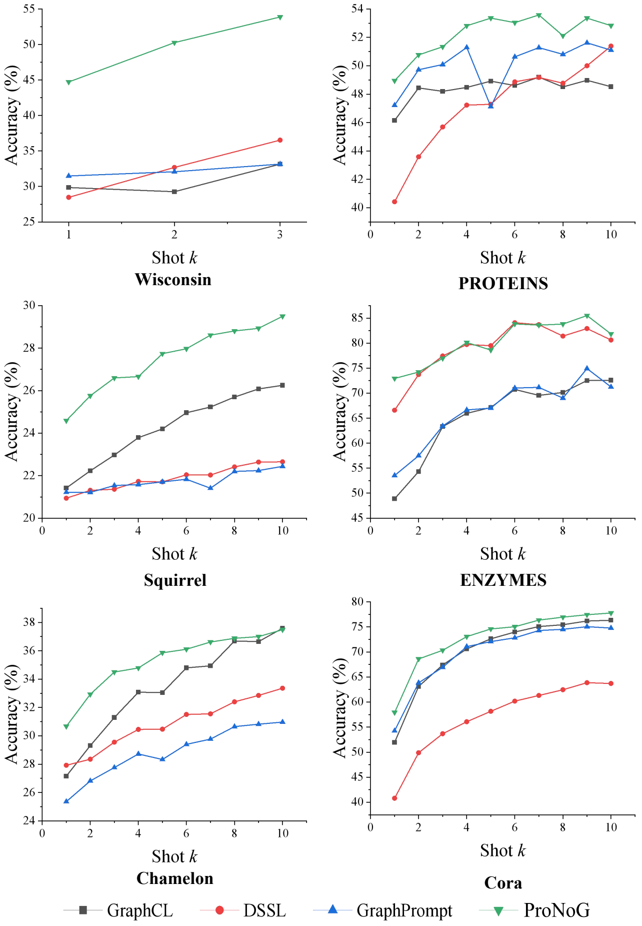


Figure 3: Impacts of different shots on node classification.

ProNoG with four of its variants: (1) NoPrompt replaces conditional prompt learning with a classifier for downstream tasks; (2) SinglePrompt uses a single prompt instead of conditional prompts to modify all nodes; (3) NodeCond directly uses the output embedding of a node from the pre-trained graph encoder as input to the condition-net to generate the prompt, without reading out the subgraph in Eq. (7); (4) ProNoG\sim reads out the subgraph via a mean pooling without similarity weighting between the ego nodes and their neighbors as in Eq. (7).

As shown in Table 3, ProNoG consistently outperforms these variants in all but one instance, in which its performance is still competitive. This highlights the necessity of reading out subgraphs with similarity weighting in order to capture the characteristics of each node, and the advantage of using conditional prompt learning to adapt to each node.

6.4 Analysis on Pre-Training Methods

To investigate the effect of homophily and non-homophily tasks on pre-training, we employ two forms of link prediction from GPPT [37] and GraphPrompt [23] as the homophily tasks, as well as GraphCL [52] and GraphACL [48] as non-homophily tasks. To

Table 3: Ablation study on the effects of key components.

Methods	Node classification						Graph classification					
	Wisconsin	Squirrel	Chameleon	PROTEINS	ENZYMES	Citeseer	Wisconsin	Squirrel	Chameleon	PROTEINS	ENZYMES	COX2
NoPrompt	25.41± 3.13	20.60±1.30	22.71±3.54	47.22±11.05	66.59±19.28	43.12± 9.61	20.85±6.74	20.18±1.30	22.34±4.15	53.61± 8.90	21.85±6.17	54.29±17.31
SinglePrompt	32.76± 5.21	20.85±1.32	22.78±3.35	30.33±19.59	65.32±21.67	48.64±10.09	25.77±6.24	20.68±0.91	27.03±3.98	56.35±10.59	19.38±7.12	47.24±15.53
NodeCond	35.56± 4.65	21.26±3.95	21.13±2.23	36.01±19.70	68.54±19.31	48.30±10.22	25.30±4.62	20.98±1.56	27.24±5.24	**56.61**±10.03	20.70±6.67	55.92±14.66
ProNoG\sim	30.65± 4.05	20.05±0.59	20.96±4.21	33.73±17.82	36.02±20.64	48.74± 2.66	22.05±5.86	19.93±0.42	20.20±1.11	52.30±10.94	16.70±1.28	50.05±17.67
ProNoG	**44.72**±11.93	**24.59**±3.41	**30.67**±3.73	**48.95**±10.85	**72.94**±20.23	**49.02**±10.66	**31.54**±5.30	**20.92**±1.37	**28.50**±5.30	56.11±10.19	**22.55**±6.70	**56.46**±14.57

Table 4: Comparison between homophily and non-homophily tasks in pre-training.

Pre-training task	Node classification				Graph classification			
	Wisconsin	Cornell	PROTEINS	ENZYMES	Wisconsin	Cornell	PROTEINS	ENZYMES
	0.21	0.30	0.66	0.67	0.21	0.30	0.66	0.67
Link Prediction [37]	23.01±11.40	26.27± 7.61	35.88± 5.41	36.74± 2.61	20.96± 4.21	25.38± 2.50	51.50± 6.02	17.47± 4.04
Link Prediction [23]	28.93±11.74	16.29± 7.93	**48.95**±10.85	**52.87**±14.73	23.15± 5.67	22.05±13.80	**55.83**±10.87	**22.23**± 5.51
GraphACL [48]	33.91± 9.04	29.55±12.30	44.08±10.03	50.57±13.11	26.42± 7.25	26.15± 3.87	54.15±10.58	21.64± 5.88
GraphCL [52]	**44.72**±11.93	**37.90**± 9.31	48.28±11.09	51.46±13.93	**31.54**± 1.37	**27.17**± 5.30	53.91± 5.51	21.78±12.12

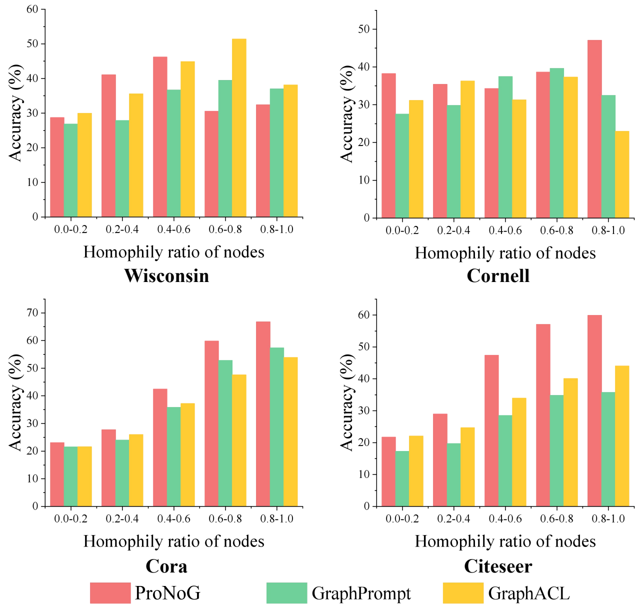


Figure 4: Results across nodes with varying homophily ratios.

isolate their effects, we apply the same conditional prompt learning from ProNoG for downstream adaptation.

We present the comparison in Table 4. It can be observed that, for graphs with lower homophily ratios (*i.e.*, *Wisconsin* and *Cornell*), non-homophily tasks significantly outperform the homophily tasks. Conversely, for graphs with higher homophily ratios (*i.e.*, *PROTEINS* and *ENZYMES*), the performance of homophily and non-homophily tasks becomes more comparable. While the homophily tasks with link prediction may have a slight advantage on highly homophilic graphs, non-homophily tasks are competitive across both homophilic and non-homophilic graphs. Hence, non-homophily tasks provide a more robust solution overall.

6.5 Analysis on Diverse Node Patterns

To evaluate the ability of ProNoG in capturing node-specific patterns, we investigate the classification accuracy of different node groups with varying homophily ratios, *i.e.*, $[0.0, 0.2)$, $[0.2, 0.4)$, $[0.4, 0.6)$, $[0.6, 0.8)$, $[0.8, 1.0]$. In each node group, we compare the performance of ProNoG with several competitive baselines.

As shown in Fig. 4, ProNoG outperforms the baselines across various node groups, regardless of their unique characteristics reflected in different homophily ratios. These consistent improvements in all groups further demonstrate the effectiveness of ProNoG in capturing diverse node patterns and highlight the advantage of our proposed conditional prompt learning.

7 Conclusions

In this paper, we explored pre-training and prompt learning on non-homophilic graphs. The goals are twofold: learning comprehensive knowledge irrespective of the varying non-homophilic patterns of graphs, and adapting the nodes with diverse distributions of non-homophily patterns to downstream tasks in a fine-grained, node-wise manner. We first revisit graph pre-training on non-homophilic graphs, providing theoretical insights into the choice of pre-training tasks. Then, for downstream adaptation, we proposed a condition-net to generate a series of prompts conditioned on node-specific non-homophilic patterns. Finally, we conducted extensive experiments on ten public datasets, showing that ProNoG significantly outperforms diverse state-of-the-art baselines.

Acknowledgments

This research / project is supported by the Ministry of Education, Singapore, under its Academic Research Fund Tier 2 (Proposal ID: T2EP20122-0041). Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not reflect the views of the Ministry of Education, Singapore.

References

[1] Hangbo Bao, Li Dong, Songhao Piao, and Furu Wei. 2022. BEiT: BERT Pre-Training of Image Transformers. In *ICLR*.

[2] Deyu Bo, Xiao Wang, Chuan Shi, and Huawei Shen. 2021. Beyond low-frequency information in graph convolutional networks. In *AAAI*. 3950–3957.

[3] Karsten M Borgwardt, Cheng Soon Ong, Stefan Schönauer, SVN Vishwanathan, Alex J Smola, and Hans-Peter Kriegel. 2005. Protein function prediction via graph kernels. *Bioinformatics* 21, suppl_1 (2005), i47–i56.

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

[5] Mouxiang Chen, Zemin Liu, Chenghao Liu, Jundong Li, Qiheng Mao, and Jianling Sun. 2023. Ultra-dp: Unifying graph pre-training with multi-task graph dual prompt. *arXiv preprint arXiv:2310.14845* (2023).

[6] Li Dong, Nan Yang, Wenhui Wang, Furu Wei, Xiaodong Liu, Yu Wang, Jianfeng Gao, Ming Zhou, and Hsiao-Wuen Hon. 2019. Unified language model pre-training for natural language understanding and generation. *NeurIPS* 32 (2019).

[7] Taoran Fang, Yunchao Zhang, Yang Yang, Chunping Wang, and Lei Chen. 2024. Universal prompt tuning for graph neural networks. *NeurIPS* (2024).

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

[9] Tianyu Gao, Adam Fisch, and Danqi Chen. 2021. Making Pre-trained Language Models Better Few-shot Learners. In *ACL*. 3816–3830.

[10] David Ha, Andrew Dai, and Quoc V Le. 2016. Hypernetworks. *arXiv preprint arXiv:1609.09106* (2016).

[11] Will Hamilton, Zhitao Ying, and Jure Leskovec. 2017. Inductive representation learning on large graphs. *NeurIPS* (2017), 1025–1035.

[12] Kaveh Hassani and Amir Hosein Khasahmadi. 2020. Contrastive multi-view representation learning on graphs. In *ICML*. 4116–4126.

[13] Dongxiao He, Jitao Zhao, Rui Guo, Zhiyong Feng, Di Jin, Yuxiao Huang, Zhen Wang, and Weixiong Zhang. 2023. Contrastive learning meets homophily: two birds with one stone. In *International Conference on Machine Learning*. 12775–12789.

[14] 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*.

[15] Ziniu Hu, Yuxiao Dong, Kuansan Wang, Kai-Wei Chang, and Yizhou Sun. 2020. GPT-GNN: Generative pre-training of graph neural networks. In *SIGKDD*. 1857–1867.

[16] Wei Jin, Tyler Derr, Yiqi Wang, Yao Ma, Zitao Liu, and Jiliang Tang. 2021. Node similarity preserving graph convolutional networks. In *WSDM*. 148–156.

[17] Thomas N Kipf and Max Welling. 2016. Variational graph auto-encoders. In *Bayesian Deep Learning Workshop*.

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

[19] Brian Lester, Rami Al-Rfou, and Noah Constant. 2021. The Power of Scale for Parameter-Efficient Prompt Tuning. In *EMNLP*. 3045–3059.

[20] Pengfei Liu, Weizhe Yuan, Jinlan Fu, Zhengbao Jiang, Hiroaki Hayashi, and Graham Neubig. 2023. Pre-train, prompt, and predict: A systematic survey of prompting methods in natural language processing. *Comput. Surveys* (2023), 1–35.

[21] Xiao Liu, Yanan Zheng, Zhengxiao Du, Ming Ding, Yujie Qian, Zhilin Yang, and Jie Tang. 2021. GPT understands, too. *arXiv preprint arXiv:2103.10385* (2021).

[22] 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*. 4267–4275.

[23] Zemin Liu, Xingtong Yu, Yuan Fang, and Xinming Zhang. 2023. GraphPrompt: Unifying pre-training and downstream tasks for graph neural networks. In *WWW*. 417–428.

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

[25] Sitao Luan, Chenqing Hua, Qincheng Lu, Jiaqi Zhu, Mingde Zhao, Shuyuan Zhang, Xiao-Wen Chang, and Doina Precup. 2022. Revisiting heterophily for graph neural networks. *Advances in neural information processing systems* (2022), 1362–1375.

[26] Yao Ma, Xiaorui Liu, Neil Shah, and Jiliang Tang. 2022. Is homophily a necessity for graph neural networks?. In *ICLR*.

[27] Haitao Mao, Zhikai Chen, Wei Jin, Haoyu Han, Yao Ma, Tong Zhao, Neil Shah, and Jiliang Tang. 2023. Demystifying structural disparity in graph neural networks: Can one size fit all?. In *NeurIPS*.

[28] Andrew Kachites McCallum, Kamal Nigam, Jason Rennie, and Kristie Seymore. 2000. Automating the construction of internet portals with machine learning. *Information Retrieval* (2000).

[29] Trung-Kien Nguyen and Yuan Fang. 2024. Diffusion-based Negative Sampling on Graphs for Link Prediction. In *WWW*. 948–958.

[30] Hongbin Pei, Bingzhe Wei, Kevin Chen-Chuan Chang, Yu Lei, and Bo Yang. 2020. Geom-gcn: Geometric graph convolutional networks. *arXiv preprint arXiv:2002.05287* (2020).

[31] Zhen Peng, Wenbing Huang, Minnan Luo, Qinghua Zheng, Yu Rong, Tingyang Xu, and Junzhou Huang. 2020. Graph representation learning via graphical mutual information maximization. In *WWW*. 259–270.

[32] Jiezhong Qiu, Qibin Chen, Yuxiao Dong, Jing Zhang, Hongxia Yang, Ming Ding, Kuansan Wang, and Jie Tang. 2020. GCC: Graph contrastive coding for graph neural network pre-training. In *SIGKDD*. 1150–1160.

[33] Ryan A. Rossi and Nesreen K. Ahmed. 2015. The Network Data Repository with Interactive Graph Analytics and Visualization. In *AAAI*. 4292–4293.

[34] Benedek Rozemberczki, Carl Allen, and Rik Sarkar. 2021. Multi-scale attributed node embedding. *Journal of Complex Networks* (2021), cnab014.

[35] Timo Schick and Hinrich Schütze. 2021. It's Not Just Size That Matters: Small Language Models Are Also Few-Shot Learners. In *NAACL*. 2339–2352.

[36] Prithviraj Sen, Galileo Namata, Mustafa Bilgic, Lise Getoor, Brian Galligher, and Tina Eliassi-Rad. 2008. Collective classification in network data. *AI magazine* (2008).

[37] Mingchen Sun, Kaixiong Zhou, Xin He, Ying Wang, and Xin Wang. 2022. GPPT: Graph Pre-training and Prompt Tuning to Generalize Graph Neural Networks. In *SIGKDD*. 1717–1727.

[38] Xiangguo Sun, Hong Cheng, Jia Li, Bo Liu, and Jihong Guan. 2023. All in One: Multi-Task Prompting for Graph Neural Networks. In *SIGKDD*.

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

[40] Zhen Tan, Ruocheng Guo, Kaize Ding, and Huan Liu. 2023. Virtual Node Tuning for Few-shot Node Classification. *arXiv preprint arXiv:2306.06063* (2023).

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

[42] Petar Velickovic, William Fedus, William L Hamilton, Pietro Liò, Yoshua Bengio, and R Devon Hjelm. 2019. Deep Graph Infomax. In *ICLR*.

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

[44] Song Wang, Yushun Dong, Xiao Huang, Chen Chen, and Jundong Li. 2022. FAITH: Few-Shot Graph Classification with Hierarchical Task Graphs. In *IJCAI*.

[45] Yuzhong Wu and Tan Lee. 2018. Reducing model complexity for DNN based large-scale audio classification. In *ICASSP*. 331–335.

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

[47] Teng Xiao, Zhengyu Chen, Zhimeng Guo, Zeyang Zhuang, and Suhang Wang. 2022. Decoupled self-supervised learning for graphs. *NeurIPS* (2022), 620–634.

[48] Teng Xiao, Huaisheng Zhu, Zhengyu Chen, and Suhang Wang. 2023. Simple and asymmetric graph contrastive learning without augmentations. *Advances in Neural Information Processing Systems* (2023).

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

[50] Yujun Yan, Milad Hashemi, Kevin Swersky, Yaoqing Yang, and Danai Koutra. 2022. Two sides of the same coin: Heterophily and oversmoothing in graph convolutional neural networks. In *ICDM*. 1287–1292.

[51] Chengxuan Ying, Tianle Cai, Shengjie Luo, Shuxin Zheng, Guolin Ke, Di He, Yanming Shen, and Tie-Yan Liu. 2021. Do transformers really perform badly for graph representation?. In *NeurIPS*. 28877–28888.

[52] Yuning You, Tianlong Chen, Yongduo Sui, Ting Chen, Zhangyang Wang, and Yang Shen. 2020. Graph contrastive learning with augmentations. *NeurIPS* 33 (2020), 5812–5823.

[53] Xingtong Yu, Yuan Fang, Zemin Liu, Yuxia Wu, Zhihao Wen, Jianyuan Bo, Xinming Zhang, and Steven CH Hoi. 2024. Few-Shot Learning on Graphs: from Meta-learning to Pre-training and Prompting. *arXiv preprint arXiv:2402.01440* (2024).

[54] Xingtong Yu, Zhenghao Liu, Yuan Fang, Zemin Liu, Sihong Chen, and Xinming Zhang. 2024. Generalized graph prompt: Toward a unification of pre-training and downstream tasks on graphs. *IEEE Transactions on Knowledge and Data Engineering* (2024).

[55] Xingtong Yu, Zemin Liu, Yuan Fang, and Xinming Zhang. 2023. Learning to count isomorphisms with graph neural networks. In *AAAI*.

[56] Xingtong Yu, Zemin Liu, Yuan Fang, and Xinming Zhang. 2024. HGPROMPT: Bridging Homogeneous and Heterogeneous Graphs for Few-shot Prompt Learning. In *AAAI*.

[57] Xingtong Yu, Chang Zhou, Yuan Fang, and Xinming Zhang. 2024. MultiGPrompt for Multi-Task Pre-Training and Prompting on Graphs. In *WWW*.

[58] Seongjun Yun, Minbyul Jeong, Raehyun Kim, Jaewoo Kang, and Hyunwoo J Kim. 2019. Graph transformer networks. *NeurIPS* 32 (2019).

[59] Yuhang Zang, Wei Li, Kaiyang Zhou, Chen Huang, and Chen Change Loy. 2022. Unified vision and language prompt learning. *arXiv preprint arXiv:2210.07225* (2022).

[60] 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.

[61] 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. *AI open* (2020), 57–81.

[62] Kaiyang Zhou, Jingkang Yang, Chen Change Loy, and Ziwei Liu. 2022. Conditional prompt learning for vision-language models. In *CVPR*. 16816–16825.

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

[64] Jiong Zhu, Ryan A Rossi, Anup Rao, Tung Mai, Nedim Lipka, Nesreen K Ahmed, and Danai Koutra. 2021. Graph neural networks with heterophily. In *AAAI*. 11168–11176.

[65] Jiong Zhu, Yujun Yan, Lingxiao Zhao, Mark Heimann, Leman Akoglu, and Danai Koutra. 2020. Beyond homophily in graph neural networks: Current limitations and effective designs. *NeurIPS* (2020), 7793–7804.

[66] Yanqiao Zhu, Yichen Xu, Feng Yu, Qiang Liu, Shu Wu, and Liang Wang. 2020. Deep graph contrastive representation learning. *arXiv preprint arXiv:2006.04131* (2020).

Appendices

A Algorithm

We detail the main steps for conditional prompt generation and tuning in Algorithm 1. In brief, we iterate through each downstream task to learn the corresponding prompt vectors. In lines 3–5, we compute the embedding for each node using the pre-trained graph encoder, with the pre-trained weights Θ_0 frozen throughout the adaptation process. In lines 6–22, we optimize the condition-net. Specifically, we perform similarity-weighted readout (lines 9–11), generate prompts (lines 12–13), modify nodes' embeddings using these prompts (lines 14–15), calculate the embedding of the corresponding graph (line 16), and update the embeddings for the prototypical nodes/graphs based on the few-shot labeled data provided in the task (lines 17–19).

B Homophily and Non-Homophily Methods

We provide further details about the set of positive samples $\mathcal{A}$ and negative samples $\mathcal{B}$ for various homophily and non-homophily methods in Table 5.

C Further Experimental Details

Datasets. We summarize the statistics of the ten datasets used in our experiments in Table 6.

Details of baselines. We use the authors' code for all baselines, if available. To ensure a fair comparison, each model is tuned while referencing the settings recommended in their respective publications. We use early stopping strategy in training and set the patience value to 50 steps. The number of training epochs is set to 2,000.

- For GCN [18], we employ a 3-layer architecture on Wisconsin, Squirrel, Chameleon, Cornell datasets and a 2-layer architecture on Cora, Citeseer, ENZYMES, PROTEINS, COX2, BZR datasets. The hidden dimension is set to 256.
- For GAT [41], we employ a 2-layer architecture and set the hidden dimension to 256. Additionally, we apply 8 attention heads in the first GAT layer.
- For H2GCN [65], we employ a 2-layer architecture and set the hidden dimension to 256.
- For FAGCN [2], we employ a 2-layer architecture. The hyper-parameter settings are: eps = 0.3, dropout = 0.5, hidden = 256. We use RELU as the activation function.

Algorithm 1 CONDITIONAL PROMPT LEARNING FOR PRONOG

```
Input: Pre-trained graph encoder with parameters $\Theta_0$, a set of downstream
    tasks $\mathcal{T} = \{t_1, \dots, t_n\}$.
Output: Optimized parameters $\{\phi_{t_1}, \dots, \phi_{t_n}\}$ of $n$ condition-nets
 1: for $i \leftarrow 1$ to $n$ do
 2:     /* Encoding graphs via pre-trained graph encoder */
 3:     for each graph $G = (V, E, \mathbf{X})$ in task $t_i$ do
 4:         $\mathbf{H} \leftarrow \text{GRAPHENCODER}(G; \Theta_0)$
 5:         $\mathbf{h}_v \leftarrow \mathbf{H}[v]$, where $v$ is a node in $G$
 6:     $\phi_i \leftarrow$ initialization
 7:     while not converged do
 8:         for each node $v \in V$ in task $t_i$ do
 9:             /* Subgraph sampling and readout by Eq. (7) */
10:             Sample $v$'s $k$-hop subgraph $S_v$
11:             $\mathbf{s}_v \leftarrow \text{AVERAGE}(\{\mathbf{h}_u \cdot \mathtt{sim}(\mathbf{h}_u, \mathbf{h}_v) : u \in V(S_v)\})$
12:             /* Generate pattern-based prompts by Eq. (8) */
13:             $\mathbf{p}_{t_i,v} \leftarrow \text{CONDNET}(\mathbf{s}_v; \phi_{t_i})$
14:             /* Prompt modification by Eq. (9) */
15:             $\tilde{\mathbf{h}}_{t_i,v} \leftarrow \mathbf{p}_{t_i,v} \odot \mathbf{h}_v$
16:         $\mathbf{h}_{t_i,G} = \text{AVERAGE}(\tilde{\mathbf{h}}_{t_i,v} : v \in \mathcal{V})$
17:         /* Update prototypical subgraphs */
18:         for each class $c$ in task $t_i$ do
19:             $\tilde{\mathbf{h}}_{t_i,c} \leftarrow \text{AVERAGE}(\tilde{\mathbf{h}}_{t_i,x}$: instance $x$ belongs to class $c$)
20:         /* Optimizing the parameters in condition-net */
21:         Calculate $\mathcal{L}_{\text{down}}(\phi_i)$ by Eq. (10)
22:         Update $\phi_i$ by backpropagating $\mathcal{L}_{\text{down}}(\phi_{t_i})$
23: return $\{\phi_{t_1}, \dots, \phi_{t_n}\}$
```

- For DGI [41], we utilize a 1-layer GCN as the base model and set the hidden dimension to 256. Additionally, we employ PRELU as the activation function.
- For GraphCL [52], a 1-layer GCN is also employed as its base model, with the hidden dimension set to 256. Specifically, we select edge dropping as the augmentation, with a default augmentation ratio of 0.2.
- For DSSL [47], we search the hidden dimension in {64, 256, 2048}. We report the best performance on PROTEINS and ENZYMES with a hidden size of 64, Cora and Citeseer with 2048, and the rest datasets with 256.
- For GraphACL [48], we search the hidden dimension in {64, 256, 1024, 2048}. We report the best performance on PROTEINS and ENZYMES with a hidden size of 64, Cora and Citeseer with 2048, and the rest datasets with 256.
- For GPPT [37], we utilize a 2-layer GraphSAGE as its base model, setting the hidden dimensions to 256. For the GraphSAGE backbone, we employ a mean aggregator.
- For GraphPrompt [23], we employ a 3-layer architecture on Wisconsin, Squirrel, Chameleon, and Cornell, and a 2-layer architecture on the rest. Hidden dimensions are set to 256. We use link prediction as the pre-training task.
- For GraphPrompt+ [54], we employ a 2-layer GCN on Cora, Citeseer, ENZYMES, PROTEINS, COX2, and BZ, and a 3-layer GCN on the rest. Hidden dimensions are set to 256. We use link prediction as the pre-training task.

Details of PRONOG. For our proposed PRONOG, we utilize a 2-layer FAGCN architecture as the backbone for pre-training on the

Table 5: Positive and negative samples for homophily and non-homophily contrastive methods.

Pre-training task	Positive instances $\mathcal{A}_u$	Negative instances $\mathcal{B}_u$	Homophily task
Link prediction [23, 54, 56]	a node connected to node u	nodes disconnected to node u	Yes
DGI [42]	nodes in graph G	nodes in corrupted graph G'	No
GraphCL [52]	an augmented graph from graph G	augmented graphs from $G' \neq G$	No
GraphACL [48]	nodes with similar ego-subgraph to node u	nodes with dissimilar ego-subgraph to node u	No

Table 6: Summary of datasets.

	Graphs	Homophily ratio	Graph classes	Avg. nodes	Avg. edges	Node features	Node classes
Wisconsin	1	0.21	-	251	199	1,703	5
Squirrel	1	0.22	-	5,201	217,073	2,089	5
Chameleon	1	0.23	-	2,277	36,101	2,325	5
Cornell	1	0.30	-	183	295	1,703	5
PROTEINS	1,113	0.66	2	39.06	72.82	1	3
ENZYMES	600	0.67	6	32.63	62.14	18	3
Citeseer	1	0.74	-	3,327	4,732	3,703	6
Cora	1	0.81	-	2,708	5,429	1,433	7
BZR	405	-	2	35.75	38.36	3	-
COX2	467	-	2	41.22	43.45	3	-

Homophily ratios are calculated by Eq. (1). Note that *BZR* and *COX2* do not have any node label, and thus no homophily ratios can be calculated.

Table 7: Comparison of the number of tunable parameters during the downstream adaptation phase.

Methods	Wisconsin	Chameleon	Citeseer	Cora
GCN	501,504	660,736	947,968	366,848
FAGCN	440,654	601,130	956,654	370,994
GraphCL	1,280	1,280	1,536	1,792
GraphACL	1,280	1,280	12,288	14,336
GraphPrompt	256	256	256	256
GraphPrompt+	512	512	512	512
ProNoG	32,768	32,768	32,768	32,768

non-homophilic graphs, namely, Wisconsin, Squirrel, Chameleon, and Cornell, with edge-dropping implemented on the subgraph level and hidden dimensions set to 256. For the remaining more homophilic graphs, we employ a 1-layer GCN for pre-training, with the hidden dimensions set to 256, except for PROTEINS and ENZYMES, which use hidden dimensions of 64. We adopt a non-homophily pre-training task GraphCL [52] for all datasets except for PROTEINS and ENZYMES. Specifically, GraphCL does not work well for ProNoG on the two datasets, and instead we use link prediction [23] and DSSL [47], respectively. Note that the non-homophily task GraphCL still works well on most datasets including all of the non-homophilic graphs. For the condition-net, we set the hidden dimension to 64 for all datasets. All experiments are conducted with a random seed of 39.

D Parameter Efficiency

We evaluate the parameter efficiency of ProNoG compared to other notable methods. Specifically, we evaluate the number of parameters that need to be updated or tuned during the downstream adaptation phase, and present the results in Table 7. For GCN and FAGCN, since these models are trained end-to-end, all model weights must be updated, leading to the least parameter efficiency. In contrast, for GraphCL and GraphACL, only the downstream classifier is fine-tuned, while the pre-trained model weights are frozen, significantly reducing the number of tunable parameters in the downstream phase. Prompt-based methods GraphPrompt and GraphPrompt+ are the most parameter-efficient, as prompts are lightweight and typically contain fewer parameters than the downstream classifier. Although our conditional prompt design requires updating more tunable parameters than GraphPrompt and GraphPrompt+ during downstream adaptation, the increase is minor compared to updating the entire model weights, and thus does not pose a major issue.

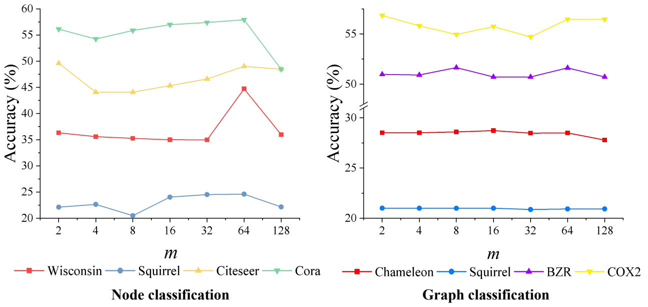


Figure 5: Impact of hidden dimension m in the condition-net.

E Hyperparameter Analysis

In our experiments, we use a 2-layer MLP with a bottleneck structure as the condition-net. We evaluate the impact of the hidden dimension m of the condition-net, and report the corresponding performance in Fig. 5. We observe that for both node and graph classification, $m = 64$ generally yields optimal performance, which we have adopted in all other experiments. Specifically, smaller values of m may lack sufficient model capacity, while larger m may introduce too many learnable parameters, leading to overfitting in few-shot settings.