

GCoT: Chain-of-Thought Prompt Learning for Graphs

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

Chang Zhou
University of Science and Technology
of China
China
zhouchang21sy@mail.ustc.edu.cn

Zhongwei Kuai
University of Science and Technology
of China
China
asagiri@mail.ustc.edu.cn

Xinming Zhang
University of Science and Technology
of China
China
xinming@ustc.edu.cn

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

Abstract

Chain-of-thought (CoT) prompting has achieved remarkable success in natural language processing (NLP). However, its vast potential remains largely unexplored for graphs. This raises an interesting question: How can we design CoT prompting for graphs to guide graph models to learn step by step? On one hand, unlike natural languages, graphs are non-linear and characterized by complex topological structures. On the other hand, many graphs lack textual data, making it difficult to formulate language-based CoT prompting. In this work, we propose the first CoT prompt learning framework for text-free graphs, GCoT. Specifically, we decompose the adaptation process for each downstream task into a series of inference steps, with each step consisting of prompt-based inference, “thought” generation, and thought-conditioned prompt learning. While the steps mimic CoT prompting in NLP, the exact mechanism differs significantly. Specifically, at each step, an input graph, along with a prompt, is first fed into a pre-trained graph encoder for prompt-based inference. We then aggregate the hidden layers of the encoder to construct a “thought”, which captures the working state of each node in the current step. Conditioned on this thought, we learn a prompt specific to each node based on the current state. These prompts are fed into the next inference step, repeating the cycle. To evaluate and analyze the effectiveness of GCoT, we conduct comprehensive experiments on eight public datasets, which demonstrate the advantage of our approach.

CCS Concepts

• Information systems → Data mining.

Keywords

Graph learning, chain-of-thought, prompt learning.

ACM Reference Format:

Xingtong Yu, Chang Zhou, Zhongwei Kuai, Xinming Zhang, and Yuan Fang. 2025. GCoT: Chain-of-Thought Prompt Learning for Graphs. In *Proceedings*

of the 31st ACM SIGKDD Conference on Knowledge Discovery and Data Mining V.2 (KDD '25), August 3–7, 2025, Toronto, ON, Canada. ACM, New York, NY, USA, 11 pages. <https://doi.org/10.1145/3711896.3736974>

KDD Availability Link:

The source code of this paper has been made publicly available at <https://doi.org/10.5281/zenodo.15501903>.

1 Introduction

Recently, Chain-of-Thought (CoT) prompting has demonstrated significant advancement in the field of natural language processing (NLP) [3, 45, 47], which mimics the logical process a human may employ to solve a task. Instead of directly providing an answer, CoT prompting decomposes a problem into several steps, guiding the pre-trained language model to follow these steps that lead to the final answer. For example, given the math question in Fig. 1(a), a language model utilizes CoT prompting (e.g., initial quantity, eaten quantity, and subtraction) to reach the final answer of 4.

However, the vast potential of CoT prompt learning remains unexplored on pre-trained graph models. Graphs can capture the interactions between entities in a wide range of domains, exhibiting a non-linear topological structure, such as molecular graphs [22, 24, 46], citation networks [19, 25, 50], and social networks [14, 66]. Conventional approaches typically retrain graph neural networks (GNNs) [21, 42] or graph transformers [54, 65] for each specific task in an end-to-end supervised manner, relying on abundant labeled data. More recent pre-training methods [43, 55] learn task-invariant, general properties from unlabeled graphs through self-supervised pretext tasks and are then fine-tuned via task-specific labels to adapt to various downstream applications [13, 34]. To further narrow the gap between pre-training and downstream tasks, prompt learning [6, 30, 56] has emerged as a low-resource alternative. They unify the objectives of pre-training and downstream tasks using the same template and employ a lightweight prompt to modify either the input features or hidden embeddings of the pre-trained graph encoder, while keeping the pre-trained weights frozen, as shown in Fig. 1(b). A contemporary work [18] leverages graphs to guide reasoning; however, it is built on the CoT mechanism from NLP and relies on textual data to construct thoughts, precluding its application to general *text-free* graphs. For text-free graphs, existing graph learning methods—including supervised, pre-training, and prompting approaches—produce a “final



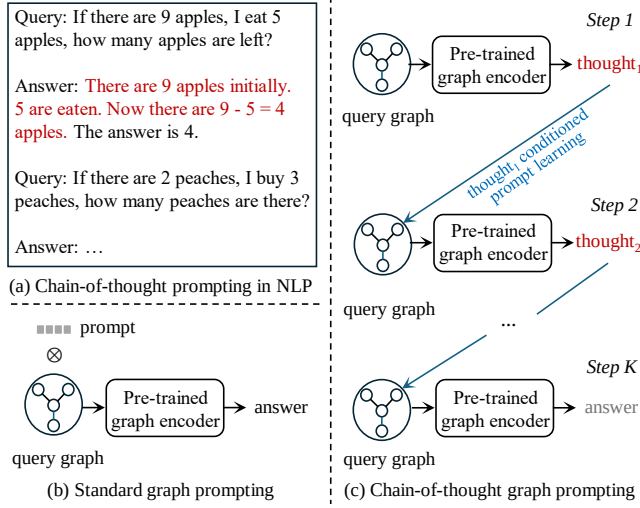


Figure 1: Illustration of chain-of-thought in NLP, standard graph prompts and graph chain-of-thought prompting.

answer” in a single inference step, which may limit the refinement of the final prediction. In this work, we explore the following question: *Would introducing additional inference steps in a CoT style enhance the ability of pre-trained graph models to refine their predictions?*

Due to the significant differences between language and graph data, replicating CoT prompting from NLP for graphs is challenging. In NLP, a CoT prompt can be handcrafted before the learning phase and typically consists of a structured text in the form (input, chain of thought, output) [47]. This prompt serves as an example to guide the model in generating intermediate thoughts that lead to the final answer. In contrast, our work explores a different prompting format for text-free graphs, which mimics the CoT approach in NLP but is not a direct application. Instead of designing prompts prior to the learning phase, we generate them step by step based on intermediate “thoughts”, improving the model’s inference ability on downstream tasks by incorporating additional inference steps while freezing the pre-trained weights. To realize this vision, we must address two questions.

First, *what should be the inference steps and thoughts for a graph task?* In NLP, a thought is defined as a short instruction that reflects a single reasoning step, with each intermediate textual answer serving as a thought [3]. However, in general text-free graphs, we cannot directly leverage text as prompts or thoughts. In this work, we aim to improve the inference capability of a pre-trained graph model by incorporating additional steps to refine the answer. We design an *inference step* with three substages: prompt-based inference, thought construction, and prompt learning. For prompt-based inference, we feed the input graph for the downstream task, along with some prompts, into a pre-trained graph encoder. Then, we construct a “thought” by fusing embeddings from each layer of the pre-trained encoder to capture the current working state with varying levels of topological knowledge [21]. Lastly, the thought is used to learn a set of prompts that guide the next step.

Second, *how can we leverage a “thought” to learn prompts and guide the next-step inference?* In NLP, CoT prompting is typically

implemented by appending specific phrases such as “let’s think step by step” or by providing few-shot CoT examples [8, 47]. Then, following a given prompt template, the language model generates new thoughts based on the query and prior thoughts, which in turn facilitate the next reasoning step. In our work, the absence of textual data prevents us from explicitly guiding the next step. Moreover, since each node in a graph exhibits unique characteristics, inference may benefit from node-specific prompts. Thus, we propose a *thought-conditioned prompt learning* method to guide the next inference step. Specifically, inspired by conditional prompt learning [68], we generate a unique prompt for each node via a conditional network (condition-net), which is conditioned on the node-specific element of the previously constructed thought. The generated prompts then feed into the graph encoder to initiate and guide the next step, repeating the process.

In summary, the contributions of this work are fourfold. (1) We propose GCoT, a Graph CoT prompting approach to guide pre-trained graph models to perform step-by-step inference. To the best of our knowledge, this is the first exploration of CoT-style prompting on text-free graphs. (2) We design an inference step with three substages: prompt-based inference, thought construction, and thought-conditioned prompt learning. In particular, a thought is constructed by fusing embeddings from each layer of the graph encoder to capture fine-grained topological knowledge in each step. (3) We employ a condition-net to generate node-specific prompts based on the previous thought, enabling the model to perform inference for the downstream task through a step-by-step, node-specific adaptation. (4) We conduct extensive experiments on eight benchmark datasets, demonstrating the superior performance of GCoT compared to a suite of state-of-the-art methods.

2 Related Work

In this section, we briefly review related work on CoT prompting, graph learning, and graph prompt learning.

Chain-of-Thought prompting. Chain-of-Thought (CoT) prompting has emerged as a groundbreaking technique in NLP, empowering language models to address complex reasoning tasks by producing intermediate reasoning steps [9, 45, 47]. By breaking down problems into a sequence of logical steps, CoT emulates human-like thought processes, leading to significant improvements in model performance on tasks requiring structured reasoning [8, 53]. Despite its remarkable success in NLP applications, the potential of CoT prompting for graphs remains largely unexplored. A contemporary work, GraphCoT [18], leverages the inherent relational information in text-attributed graphs [48, 52] to guide the reasoning process. However, GraphCoT primarily focuses on question-answering tasks for natural language and cannot be extended to general text-free graphs that lack textual descriptions.

Graph representation learning. GNNs [16, 17, 21, 42, 60] are the dominant technique for graph representations learning. They generally update node embeddings iteratively by aggregating information from their local neighborhoods based on a message-passing mechanism [11, 26, 51]. Despite their success, GNNs often demand substantial amounts of task-specific labeled data and necessitate retraining for each new task, limiting their flexibility and scalability. Recently, researchers have extensively investigated pre-training

techniques for graphs [12, 13, 15, 20, 28, 31]. These approaches involve pre-training a graph encoder using self-supervised objectives, and then adapt the pre-trained knowledge to downstream tasks. However, a significant gap exists between the objectives of pre-training and those of downstream tasks, resulting in suboptimal performance.

Graph prompt learning. First proposed in NLP, prompt learning has emerged as a powerful framework for bridging the gap between pre-training and downstream tasks [2, 23, 27]. Recently, this paradigm has been extended to the graph domain as a compelling alternative to fine-tuning approaches [30, 40, 57, 59, 63]. These methods typically utilize a universal template to align pre-training and downstream tasks, followed by task-specific prompts that facilitate seamless adaptation to downstream tasks while keeping the pre-trained model frozen. However, current graph prompt learning methods directly produce a final answer in a single step, resulting in insufficient refinement to the answer [7, 58, 64].

3 Preliminaries

In this section, we present the background and preliminaries relevant to our work.

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

Graph encoder. Towards graph representation learning, one of the most widely used families of graph encoders is graph neural networks (GNNs), which generally rely on message passing to capture structural knowledge [49, 67]. Each node updates its representation by aggregating information from its neighbors, and stacking multiple GNN layers enables iterative message propagation across the graph. Formally, let $\mathbf{H}^l$ denote the embedding matrix at the l -th layer, where each row $\mathbf{h}_i^l$ represents the embedding of node v_i . This matrix is iteratively computed using the embeddings from the preceding layer:

$$\mathbf{H}^l = \text{MP}(\mathbf{H}^{l-1}, G; \theta^l), \quad (1)$$

where $\text{MP}(\cdot)$ is the message passing function, and θ^l represents the learnable parameters of the graph encoder at layer l . The initial embedding matrix, $\mathbf{H}^0$, is the input feature matrix, *i.e.*, $\mathbf{H}^0 = \mathbf{X}$. The output after a total of L layers is then $\mathbf{H}^L$; for brevity we may simply write $\mathbf{H}$. We abstract the multi-layer encoding process as

$$\{\mathbf{H}^1, \mathbf{H}^2, \dots, \mathbf{H}^L\} = \text{GRAPHENCODER}(\mathbf{X}, G; \Theta), \quad (2)$$

where $\{\mathbf{H}^1, \mathbf{H}^2, \dots, \mathbf{H}^L\}$ denotes the embedding matrix of each layer of the graph encoder, respectively. $\Theta = (\theta^1, \dots, \theta^L)$ is the collection of weights across the layers.

Pre-training. As prior studies [59, 62] have suggested, mainstream contrastive pre-training tasks on graphs [30, 43, 55] can be unified under the task template of similarity calculation. Formally, the unified pre-training objective is defined as follows:

$$\mathcal{L}(\Theta) = - \sum_{o \in \mathcal{T}_{\text{pre}}} \ln \frac{\sum_{a \in \text{Pos}_o} \exp(\text{sim}(\mathbf{h}_a, \mathbf{h}_o)/\tau)}{\sum_{b \in \text{Neg}_o} \exp(\text{sim}(\mathbf{h}_b, \mathbf{h}_o)/\tau)}, \quad (3)$$

where Pos_o and Neg_o denote the sets of positive and negative samples for a target instance o , respectively. $\mathbf{h}_o$ represents the embedding of the target instance, while $\mathbf{h}_a$ and $\mathbf{h}_b$ correspond to the embeddings of positive and negative samples. The hyperparameter τ controls the temperature scaling of the similarity function $\text{sim}(\cdot, \cdot)$. In our framework, we follow previous work [30, 59] by employing similarity calculation as the task template and using link prediction as the pre-training task.

Problem definition. In this work, we explore a CoT prompt learning framework for text-free graphs. We focus on two widely used tasks in graph learning: node classification and graph classification, in few-shot scenarios. For node classification, given a graph $G = (V, E)$ with a set of node classes Y , each node $v_i \in V$ is associated with a label $y_i \in Y$. In contrast, graph classification considers a collection of graphs $\mathcal{G}$, where each graph $G_i \in \mathcal{G}$ is assigned a class label $Y_i \in Y$. In the few-shot setting, only m labeled examples per class are available (*e.g.*, $m \leq 10$), which is referred to as m -shot classification [30, 59].

4 Chain-of-Thought Graph Prompt Learning

In this section, we present our approach, GCoT, starting with an overview of its framework. Then we detail its core components and conclude with a complexity analysis of the algorithm.

4.1 Overall Framework

We illustrate the overall framework of GCoT in Fig. 2, which consists of two phases: pre-training and CoT prompting.

First, we pre-train a graph encoder as shown in Fig. 2(a). Details of the pre-training phase are provided in Sect. 3.

Second, given a pre-trained graph encoder, to guide the model with additional inference steps before finalizing its predictions, we propose CoT prompting, as illustrated in Fig. 2(c). Specifically, we design an inference step with three substages: prompt-based inference, thought construction, and thought-conditioned prompt learning. Specifically, in each inference step, we first feed the prompt and query graph into the pre-trained graph encoder. Then, we construct a “thought” by fusing embeddings of all hidden layers of the pre-trained graph encoder. Lastly, as detailed in Fig. 2(b), conditioned on the constructed thought, we employ a condition-net to generate a series of node-specific prompts. These conditional prompts capture individualized patterns for nodes in a fine-grained and parameter-efficient manner. We repeat the inference step one or more times before making the final prediction for the downstream task. It is worth noting that, to better align the downstream task with the pre-training task, we still incorporate a standard prompt similar to those used in prior graph prompting methods [6, 30].

4.2 Chain-of-Thought Prompting

Next, we introduce the details of each inference step and the standard prompt learning employed in our framework.

4.2.1 Inference step. Each inference step consists of three substages, as described in the following.

Prompt-based inference. In the k -th inference step, we first feed the query graph $G = (V, E)$, along with its prompt-modified feature

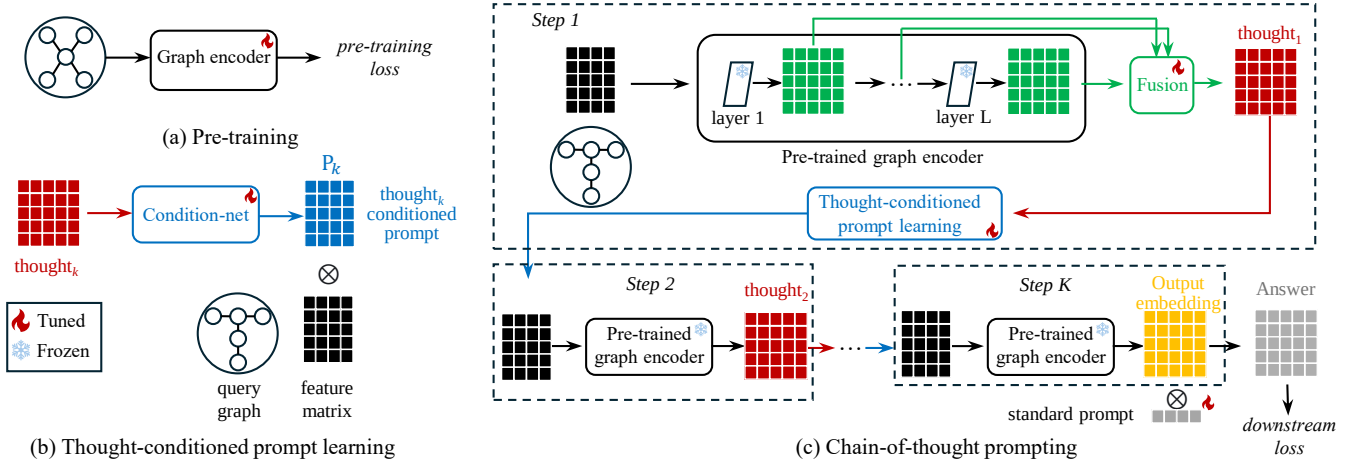


Figure 2: Overall framework of GCoT.

matrix, into the pre-trained encoder:

$$\{\mathbf{H}_k^1, \mathbf{H}_k^2, \dots, \mathbf{H}_k^L\} = \text{GRAPHENCODER}(\mathbf{X}_k, G; \Theta_0), \quad (4)$$

where Θ_0 denotes the frozen pre-trained weights in the graph encoder, and $\mathbf{X}_k$ is the node feature matrix modified by the prompts generated in the previous (i.e., $k-1$ -th) step. The generation of the prompts and the modification of the feature matrix are elaborated later in *thought-conditioned prompt learning*. Note that for the first step ($k=1$), the original feature matrix is used without modification, i.e., $\mathbf{X}_1 = \mathbf{X}$.

Thought construction. To leverage the hierarchical knowledge across multiple layers of the graph encoder, we construct a “thought” by fusing the hidden embeddings from each layer of the pre-trained graph encoder as follows:

$$\mathbf{T}_k = \text{FUSE}(\mathbf{H}_k^1, \mathbf{H}_k^2, \dots, \mathbf{H}_k^L), \quad (5)$$

where each $\mathbf{H}_k^l$ denotes the hidden embedding from the l -th layer during the k -th inference step. The $\text{FUSE}(\cdot)$ function can be implemented in various ways. For example, if all hidden layers of the graph encoder share the same dimensionality, we can adopt a simple weighted summation¹, as follows.

$$\mathbf{T}_k = w^1 \cdot \mathbf{H}_k^1 + w^2 \cdot \mathbf{H}_k^2 + \dots + w^L \cdot \mathbf{H}_k^L, \quad (6)$$

where $W = \{w^l \in \mathbb{R} : 1 \leq l \leq L\}$ are learnable parameters. The resulting thought, $\mathbf{T}_k$, captures the current working state of the graph encoder, storing multi-layer structural knowledge. In particular, the i -th row of $\mathbf{T}_k$ reflects the thought pertaining specifically to node v_i .

Thought-conditioned prompt learning. The thought constructed in Eq. (6) is then used for prompt generation, in order to further guide the next inference step. Moreover, since different nodes may exhibit distinct characteristics with respect to the downstream task, it is beneficial to adapt the pre-trained model to node-specific patterns. Thus, rather than employing a single prompt for all nodes, we propose to leverage a conditional network (condition-net) [61, 62,

68] to generate node-specific prompts and facilitate node-specific adaptation. Specifically, conditioned on the current thought $\mathbf{T}_k$, the condition-net generates a series of prompts, collectively represented as $\mathbf{P}_k \in \mathbb{R}^{|V| \times d}$, as follows.

$$\mathbf{P}_k = \text{CONDNET}(\mathbf{T}_k; \phi), \quad (7)$$

where CONDNET is the condition-net parameterized by ϕ . The condition-net can be viewed as a hypernetwork [10]—a lightweight auxiliary network such as a multi-layer perceptron (MLP)—that generates a distinct prompt for each node from the thought, yet avoids parameterizing a separate prompt vector for each node. In particular, the i -th row of $\mathbf{P}_k$ represents a unique node-specific prompt vector $\mathbf{p}_{k,i}$ for node $v_i \in V$. Subsequently, the prompt vectors are used to modify the node features of the query graph for *prompt-based inference* in the next (i.e., $k+1$ -th) step, as follows.

$$\mathbf{X}_{k+1} = \mathbf{P}_k \odot \mathbf{X}, \quad (8)$$

where $\odot$ denotes element-wise multiplication, and $\mathbf{X}_{k+1}$ represents the input to the pre-trained graph encoder in the $k+1$ -th step.

4.2.2 Standard prompt learning. To align the objectives of pre-training and downstream tasks, we also leverage a standard prompt following prior work, which typically modifies node features [6, 40, 59] or embeddings [30, 59]. In particular, we adopt GPF+ [6] to generate the standard prompts. However, we emphasize that our framework is compatible with any standard graph prompting technique. As demonstrated in Sect. 5.6, our CoT-style prompting can be combined with several prevailing graph prompting approaches to further enhance their performance.

Specifically, we train N bias prompts $\{\mathbf{p}_{\text{bias}}^1, \dots, \mathbf{p}_{\text{bias}}^N\}$ and leverage attention-based aggregation to generate node-specific prompts. While GPF+ [6] applies these prompts to graph features $\mathbf{X}$, we use them to modify the output embeddings $\mathbf{H}_K$ after K inference steps. Concretely, the standard prompt for node v_i is computed as

$$\mathbf{p}_{\text{std},i} = \sum_{j=1}^N \alpha_{i,j} \mathbf{p}_{\text{bias}}^j, \quad \text{where } \alpha_{i,j} = \frac{\exp(\mathbf{a}^j \mathbf{h}_{K,i})}{\sum_{n=1}^N \exp(\mathbf{a}^n \mathbf{h}_{K,i})}. \quad (9)$$

¹This is the case in our experiments. If the dimensionalities differ, linear layers can be potentially used to project them to a common dimension.

Here, $\mathbf{h}_{K,i}$ denotes the i -th row of the output embedding matrix $\mathbf{H}_K$, *i.e.*, the embedding vector of node v_i after K inference steps. $\{\mathbf{a}^1, \mathbf{a}^2, \dots, \mathbf{a}^N\}$ are N learnable linear projection vectors. The standard prompts for all nodes are stacked to form the matrix $\mathbf{P}_{\text{std}}$, which is then used to modify the output embeddings $\mathbf{H}_K$:

$$\tilde{\mathbf{H}} = \mathbf{P}_{\text{std}} \odot \mathbf{H}_K. \quad (10)$$

We call $\tilde{\mathbf{H}}$ the *answer matrix*, since it is used by downstream tasks to produce final predictions (or answers) in Sect. 4.3.

Note that for clarity and to maintain a unified representation of various standard prompt mechanisms, we use $\mathbf{P}_{\text{std}}$ to denote the trainable parameters associated with these mechanisms throughout the remainder of this paper.

4.2.3 Summary. In GCoT, we adopt the following mechanisms, which collectively ensure the effectiveness of GCoT. (1) We follow standard graph prompt learning methods [6, 30] to align pre-training and downstream tasks, ensuring that GCoT can efficiently adapt to different downstream tasks even in few-shot settings. (2) The CoT-style prompting performs multiple inference steps, allowing iterative refinement of its prediction for a given downstream task. (3) The thoughts in GCoT fuse hierarchical topological knowledge from graphs, enabling the capture of fine-grained structural information. (4) Conditioned on the thought, GCoT generates a series of node-specific prompts that reflect individualized node characteristics in a parameter-efficient manner.

4.3 Prompt Tuning

Consider a downstream task with a labeled training set

$$\mathcal{D} = \{(x_1, y_1), (x_2, y_2), \dots\},$$

where each x_i represents either a node or a graph and $y_i \in Y$ denotes its corresponding class label. Subsequently, the loss function of the task is defined as $\mathcal{L}_{\text{down}}(\mathcal{D}; W, \phi, \mathbf{P}_{\text{std}}) =$

$$- \sum_{(x_i, y_i) \in \mathcal{D}} \ln \frac{\exp(\text{sim}(\tilde{\mathbf{h}}_{x_i}, \tilde{\mathbf{h}}_{y_i}) / \tau)}{\sum_{c \in Y} \exp(\text{sim}(\tilde{\mathbf{h}}_{x_i}, \tilde{\mathbf{h}}_c) / \tau)}, \quad (11)$$

where $\tilde{\mathbf{h}}_{x_i}$ represents the final embedding of a node v or a graph G . Specifically, for node classification, $\tilde{\mathbf{h}}_v$ corresponds to a row in the answer matrix $\tilde{\mathbf{H}}$; for graph classification, we apply a readout operation and compute the graph embedding as $\tilde{\mathbf{h}}_G = \sum_{v \in V} \tilde{\mathbf{h}}_v$. Lastly, $\tilde{\mathbf{h}}_c$ denotes the prototype embedding for class c , which is obtained by averaging the embeddings of all labeled nodes or graphs belonging to that class.

During prompt tuning, only the weights of thought construction (W) and the condition-net (ϕ), as well as the standard prompts ($\mathbf{P}_{\text{std}}$), are updated, while the pre-trained weights of the graph encoder remain frozen. This parameter-efficient design makes our approach well-suited for few-shot learning, where the training set $\mathcal{D}$ contains only a few labeled examples.

4.4 Algorithm and Complexity Analysis

Algorithm. We outline the main steps for GCoT in Algorithm 1. In lines 3–13, we iterate through the first $K - 1$ inference steps for a downstream task while keeping the pre-trained weights Θ_0 frozen. Specifically, in lines 8–9, we generate the thought for the

Algorithm 1 CHAIN-OF-THOUGHT GRAPH PROMPT LEARNING

Input: Pre-trained graph encoder with parameters Θ_0 , labeled data $\mathcal{D}$ for a given downstream task

Output: Optimized parameters $W, \phi, \mathbf{P}_{\text{std}}$.

```

1:  $W, \phi, \mathbf{P}_{\text{std}} \leftarrow$  initialization
2: while not converged do
3:    $\mathbf{X}_1 \leftarrow \mathbf{X}$ 
4:   /* First  $K - 1$  inference steps */
5:   while inference step  $1 \leq k < K$  do
6:     /* Prompt-based inference */
7:      $\{\mathbf{H}_k^1, \mathbf{H}_k^2, \dots, \mathbf{H}_k^L\} \leftarrow \text{GRAPHENCODER}(G, \mathbf{X}_k; \Theta_0)$ 
8:     /* Thought construction */
9:      $\mathbf{T}_k \leftarrow \text{Fuse}(\mathbf{H}_k^1, \mathbf{H}_k^2, \dots, \mathbf{H}_k^L)$ 
10:    /* Thought-conditioned prompt generation by Eq. (7) */
11:     $\mathbf{P}_k \leftarrow \text{CondNet}(\mathbf{T}_k; \phi)$ 
12:    /* Feature modification by Eq. (8) */
13:     $\mathbf{X}_{k+1} \leftarrow \mathbf{P}_k \odot \mathbf{X}$ 
14:    /* Last (i.e.,  $K$ -th) inference step */
15:     $\{\mathbf{H}_K^1, \mathbf{H}_K^2, \dots, \mathbf{H}_K^L\} \leftarrow \text{GRAPHENCODER}(G, \mathbf{X}_K; \Theta_0)$ 
16:     $\mathbf{H}_K \leftarrow \mathbf{H}_K^L$ 
17:    /* Standard prompt modification by Eq. (10) */
18:     $\tilde{\mathbf{H}} \leftarrow \mathbf{P}_{\text{std}} \odot \mathbf{H}_K$ 
19:    /* Update prototypical instance */
20:    for each class  $c$  do
21:       $\tilde{\mathbf{h}}_c \leftarrow \text{MEAN}(\{\mathbf{h}_x: x \text{ is any instance in class } c \text{ from } \mathcal{D}\})$ 
22:    /* Optimizing the parameters */
23:    Calculate  $\mathcal{L}_{\text{down}}(\mathcal{D}; W, \phi, \mathbf{P}_{\text{std}})$  by Eq. (11)
24:    Update  $W, \phi, \mathbf{P}_{\text{std}}$  by backpropagating  $\mathcal{L}_{\text{down}}(\mathcal{D}; W, \phi, \mathbf{P}_{\text{std}})$ 
25: return  $\{W, \phi, \mathbf{P}_{\text{std}}\}$ 

```

k -th inference step by fusing the embeddings from each layer of the pre-trained graph encoder. In lines 10–11, we leverage this thought to generate node-specific prompts that guide the next inference step. In lines 14–18, we perform the last inference step, employing a standard prompt to modify the output embedding and obtaining the answer matrix. Finally, in lines 19–24, we calculate and backpropagate the downstream loss to update the parameters $W, \phi, \mathbf{P}_{\text{std}}$.

Complexity analysis. For a downstream graph G , we perform K inference steps. In each step, we first conduct prompt-based inference, the complexity of which is determined by the architecture of the graph encoder. In a standard GNN, each node aggregates information from up to D neighboring nodes per layer. Consequently, computing node embeddings over L layers results in a complexity of $O(D^L|V|)$, where $|V|$ denotes the number of nodes. This is followed by thought generation, which fuses embeddings from all L layers, introducing an additional complexity of $O(L|V|)$. The generated thought is subsequently used for thought-conditioned prompt learning, with a complexity of $O(|V|)$. Therefore, the computational complexity for the K inference steps is $O(K(D^L + L)|V|)$. Additionally, we employ a standard prompt to modify node features or embeddings. The complexity of this process depends on the specific prompting mechanism. Here, we assign it a complexity of $O(|V|)$, which is common among graph prompting methods [6, 30]. Thus, the overall complexity of GCoT is $O(K(D^L + L)|V|)$, or more simply $O(KD^L|V|)$, since it is typical that $L \ll D^L$.

Table 1: Summary of datasets.

Datasets	Graphs	Graph classes	Avg. nodes	Avg. edges	Node features	Node classes	Task* (N/G)
Cora	1	-	2,708	5,429	1,433	7	N
Citeseer	1	-	3,327	4,732	3,703	6	N
Pubmed	1	-	19,717	88,648	500	3	N
Photo	1	-	7,650	238,162	745	8	N
MUTAG	188	2	17.9	18.9	7	-	G
COX2	467	2	41.2	43.5	3	-	G
BZR	405	2	35.8	38.4	3	-	G
PROTEINS	1,113	2	39.1	72.8	4	3	G

* This column indicates the type of downstream task conducted for each dataset: “N” denotes node classification, while “G” denotes graph classification.

5 Experiments

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

5.1 Experimental Setup

Datasets. We conduct experiments on eight widely used benchmark datasets, spanning citation networks, e-commerce, protein structures, and molecular graphs. *Cora* [32], *Citeseer* [37], and *Pubmed* [37] are citation networks, each consisting of a single graph. In these datasets, nodes represent academic papers and edges denote citation relationships between them. *Photo* [38] is an e-commerce co-purchase network derived from Amazon’s photography-related product categories, where nodes correspond to products and edges indicate frequently co-purchased items. *PROTEINS* [1] is a dataset of protein structures. In each graph, nodes correspond to secondary structures, and edges capture spatial or sequential relationships within the amino acid sequence. *MUTAG* [35], *BZR* [35], and *COX2* [35] are molecular graph datasets, representing nitroaromatic compounds, ligands associated with benzodiazepine receptors, and molecular structures related to cyclooxygenase-2 inhibitors, respectively. These datasets are summarized in Table 1, with further descriptions in Appendix A.

Baselines. We compare GCoT with state-of-the-art methods across three categories. (1) *Supervised GNNs*: GCN [21] and GAT [42] are trained directly on downstream labels in a supervised manner, without any pre-training. (2) *Graph pre-training models*: DGI/InfoGraph² [39, 42] and GraphCL [55] adopt a “pre-train, fine-tune” strategy. They first perform self-supervised pre-training using unlabeled graphs and are later fine-tuned for downstream tasks, where a classifier is trained with few-shot labels while keeping the pre-trained encoder frozen. (3) *Graph prompt learning models*: ProG [41], GPF [6], GPF+ [6], and GraphPrompt [30] employ self-supervised pre-training followed by prompt tuning. Unlike the fine-tuning methods, these methods leverage a unified task template, and train task-specific prompts for downstream adaptation. Further descriptions of these baselines are presented in Appendix B.

²DGI is originally designed for node-level tasks, while InfoGraph extends it to graph-level classification. In our experiments, we apply DGI to node classification and InfoGraph to graph classification.

Downstream tasks and evaluation. We perform experiments on two downstream tasks: node classification and graph classification. Both types of task follow an m -shot classification setup, where for each class, we randomly select m instances (nodes or graphs) as labeled examples. The remaining instances are treated as the test set. In our main results, we set $m = 1$ for both node and graph classification tasks. Additionally, to examine the robustness of our method, we vary m within the range $[1, 10]$, allowing us to analyze the performance under different few-shot scenarios. We construct 100 independent m -shot tasks for each type of classification through repeated sampling. Each task is evaluated using five different random seeds, resulting in a total of 500 experimental runs per task type. We adopt accuracy as the performance metric and report both the mean and standard deviation across these runs, in line with previous studies [29, 30, 44].

5.2 Implementation Details

We outline key settings for the baselines and GCoT.

Baseline settings. We utilize the official codes for all open-source baselines. Each model was tuned based on the settings recommended in their respective work to achieve optimal performance.

For the baseline GCN [21], we employ a 2-layer architecture, and set the hidden dimensions to 64. For GAT [42], we employ a 2-layer architecture and set the hidden dimension to 64. Additionally, we apply 8 attention heads in the first GAT layer. For DGI [42], we utilize a 1-layer GCN as the backbone and set the hidden dimensions to 256. Additionally, we employ PReLU as the activation function. For InfoGraph [39], a 1-layer GCN is used as the backbone, with its hidden dimensions set to 256. For GraphCL [55], a 1-layer GCN is also employed as its backbone, with the hidden dimensions set to 256. Specifically, we select edge dropping as the augmentations, with a default augmentation ratio of 0.2. For GraphPrompt [30], a 3-layer GCN is used as the backbone for all datasets, with the hidden dimensions set to 256. For GPF and GPF+ [6], we employ a 3-layer GCN as the backbone for all datasets. The hidden dimensions are set to 256. For ProG [41], we employ a 2-layer GCN as the backbone for all datasets. The hidden dimensions are set to 100.

GCoT settings. For our proposed GCoT, we utilize a 3-layer GCN as the backbone for all datasets, with the hidden dimensions set to 256. We set the number of inference step as 2 for node classification task, and 3 for graph classification task. The condition-net is implemented as a two-layer MLP with a bottleneck architecture [61, 62]. Its input dimension is 256, while the hidden dimension is set to 32 for node classification and 8 for graph classification.

Environment. All experiments were conducted on Ubuntu 22.04.2, using a machine equipped with AMD EPYC 7742 64-core processors and NVIDIA GeForce RTX 3090 (24 GB) GPUs.

5.3 Performance Evaluation

We first evaluate one-shot classification tasks. Then, we examine the model performance as the number of shots increases. Note that for the other experiments in Sect. 5.4 and thereafter, we adopt the one-shot setting.

Table 2: Accuracy (%) evaluation of node and graph classification.

Methods	Node classification				Graph classification			
	Cora	Citeseer	Pubmed	Photo	MUTAG	COX2	BZR	PROTEINS
GCN	32.50 $\pm$ 14.21	26.36 $\pm$ 9.03	52.18 $\pm$ 8.70	60.18 $\pm$ 12.04	43.44 $\pm$ 15.14	50.95 $\pm$ 23.48	47.25 $\pm$ 16.59	40.28 $\pm$ 0.03
GAT	31.00 $\pm$ 16.22	27.71 $\pm$ 8.74	50.02 $\pm$ 8.88	51.79 $\pm$ 12.85	37.33 $\pm$ 10.81	50.58 $\pm$ 26.16	46.55 $\pm$ 16.57	40.39 $\pm$ 0.04
DGI/INFOGRAPH	54.11 $\pm$ 9.60	45.00 $\pm$ 9.19	47.46 $\pm$ 12.19	58.89 $\pm$ 10.97	53.17 $\pm$ 17.29	53.82 $\pm$ 14.19	49.33 $\pm$ 15.11	52.51 $\pm$ 10.29
GRAPHCL	51.96 $\pm$ 9.43	43.12 $\pm$ 9.61	46.80 $\pm$ 9.04	57.78 $\pm$ 11.31	54.92 $\pm$ 17.09	53.81 $\pm$ 14.21	49.73 $\pm$ 14.66	53.81 $\pm$ 8.97
ProG	50.59 $\pm$ 14.64	43.17 $\pm$ 8.49	63.07 $\pm$ 11.96	66.50 $\pm$ 9.46	51.99 $\pm$ 4.50	53.45 $\pm$ 15.01	53.52 $\pm$ 11.97	52.73 $\pm$ 6.57
GPF	57.60 $\pm$ 13.88	43.11 $\pm$ 8.80	55.63 $\pm$ 10.96	65.29 $\pm$ 10.07	56.55 $\pm$ 13.95	54.16 $\pm$ 14.07	48.65 $\pm$ 13.96	53.05 $\pm$ 7.62
GPF+	57.42 $\pm$ 13.87	43.28 $\pm$ 8.82	57.16 $\pm$ 10.99	65.07 $\pm$ 10.01	<u>56.81</u> $\pm$ 12.93	<u>55.24</u> $\pm$ 13.29	50.83 $\pm$ 19.74	<u>54.58</u> $\pm$ 8.70
GRAPHPROMPT	54.25 $\pm$ 9.38	<u>45.34</u> $\pm$ 10.53	<u>63.11</u> $\pm$ 10.01	<u>66.62</u> $\pm$ 9.90	55.44 $\pm$ 12.56	54.34 $\pm$ 14.77	<u>54.59</u> $\pm$ 10.52	53.80 $\pm$ 7.93
GCoT	59.67 $\pm$ 15.51	46.21 $\pm$ 8.78	64.43 $\pm$ 9.96	67.16 $\pm$ 10.46	58.75 $\pm$ 15.42	56.26 $\pm$ 15.52	58.03 $\pm$ 23.44	56.24 $\pm$ 8.60

Best results are **bolded** and runner-up results are underlined.

One-shot performance. We present the results for one-shot node and graph classification in Table 2. We make several major observations, as follows.

(1) GCoT consistently outperforms the baseline methods across both node and graph classification, demonstrating its overall advantage and robustness.

(2) Supervised methods tend to underperform compared to others, as they do not leverage any pre-trained model. On the other hand, graph pre-training models achieve improved performance through self-supervised pre-training on unlabeled graphs.

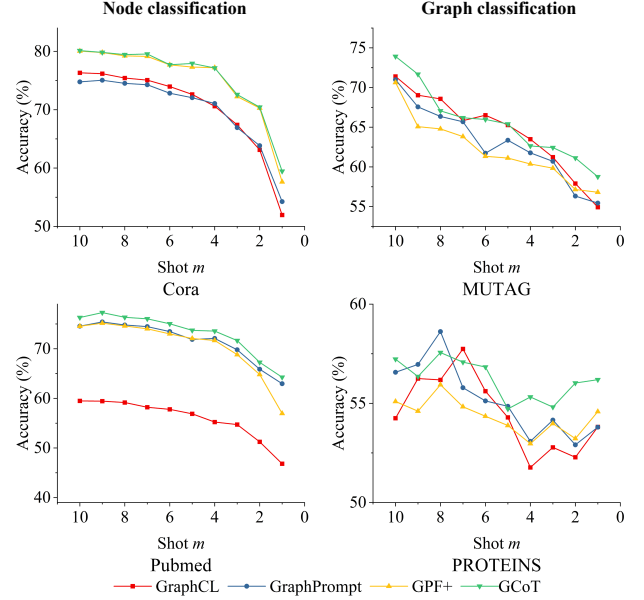
(3) Standard graph prompt learning approaches—ProG, GPF, GPF+, and GraphPrompt—often outperform fine-tuning of the pre-trained models, due to their alignment between pre-training and downstream objectives, as well as their parameter efficiency. However, they still fall short compared to GCoT, due to their single-step inference process that lacks iterative refinement of the final answer. This highlights the effectiveness of our CoT-style prompting, which enables step-by-step inference.

Few-shot performance. To investigate the effect of labeled data size on performance, we vary the number of shots in both node and graph classification tasks. The results are presented in Fig. 3, comparing GCoT against several competitive baselines. We make the following observations.

(1) GCoT generally outperforms the baselines on both node and graph classification tasks, particularly in low-shot settings (e.g., $m \leq 5$), where labeled data are limited.

(2) As the number of shots increases (e.g., $m > 5$), all methods generally exhibit improved performance, as expected. Nevertheless, GCoT remains highly competitive, often achieving the best or near-best results.

(3) On the *PROTEINS* dataset, the performance of all methods tends to fluctuate more. A possible reason is that this dataset exhibits greater variability in graph sizes compared to other datasets: The standard deviation of graph sizes in *PROTEINS* is 45.78, whereas other datasets fall within the range of 4.04 to 15.29. This may contribute to the unstable performance. Despite this, GCoT demonstrates greater robustness than the competing methods.

**Figure 3: Impact of labeled data size (number of shots) on node and graph classification.**

5.4 Ablation Study and Visualization

To thoroughly evaluate the contribution of CoT-style prompting to graph models, we perform an ablation study and visualization.

Ablation study. To investigate the effect of step-by-step inference and the impact of thoughts constructed from different layers, we compare GCoT with four variants. The first variant is GCoT\CoT, which produces the final answer in a single inference step without CoT-style prompting. The other three variants are GCoT-L1, GCoT-L2, and GCoT-L3, which utilize only the hidden embeddings from the first, second, and third layers of the pre-trained graph encoder as the thought, respectively. (Recall that we employ a 3-layer GCN as the graph encoder.)

As shown in Table 3, GCoT consistently outperforms all variants. Specifically, the advantage over GCoT\CoT underscores the

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

Methods	Node classification		Graph classification	
	Cora	Pubmed	MUTAG	PROTEINS
GCoT\CoT	56.65±13.97	62.80±10.08	56.49±16.61	53.40±6.66
GCoT-L1	57.18±14.34	63.31±10.05	56.54±14.12	54.71±8.57
GCoT-L2	57.00±14.48	63.20±10.08	57.68±13.84	54.77±8.81
GCoT-L3	57.01±14.66	63.33±10.05	57.85±16.10	56.22±8.45
GCoT	59.67±15.51	64.43± 9.96	58.75±15.42	56.24±8.60

importance of step-by-step inference, which enables iterative refinement of the predictions. Moreover, the advantage over single-layer-based thoughts demonstrates the effectiveness of fusing hierarchical knowledge from different layers of the pre-trained graph encoder in constructing the thoughts.

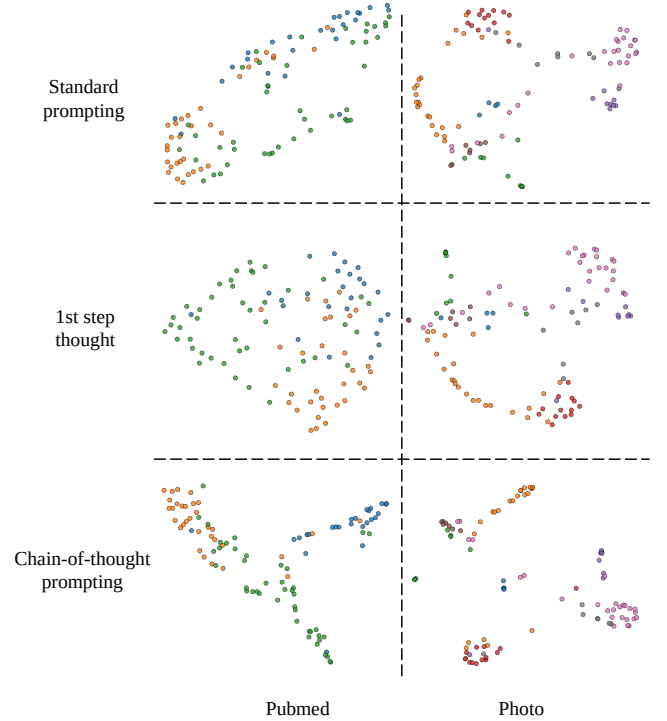
Visualization. To further demonstrate the impact of our CoT design, we visualize the output embeddings produced by GCoT\CoT (the first variant in the ablation study) and the answer embeddings by GCoT, as well as the embeddings of thoughts constructed in the first inference step. As shown in Fig. 4, each point represents the embeddings associated with a node, and different colors represent different classes of the nodes. With CoT-style prompting, node embeddings from different classes exhibit a clearer separation compared to those produced by GCoT\CoT, underscoring the effectiveness of GCoT in enhancing class distinction. Moreover, the thoughts generated in the first inference step already show some clustering—though not as well as the final node embeddings produced by GCoT—since they only reflect an intermediate state before the final step (two steps in total are used).

5.5 Heterophily Sensitivity

To examine the robustness of GCoT on heterophilic graphs [62], we conduct one-shot node classification on *Wisconsin* [33] and *Squirrel* [36]. Specifically, *Wisconsin* is a network of 251 nodes representing webpages, with 199 edges indicating hyperlink connections among them. Node features are constructed using a bag-of-words approach based on webpage content. *Squirrel* [36] consists of 5,201 Wikipedia pages discussing predefined topics. Nodes represent individual webpages, while edges capture 217,073 hyperlink connections between them. Node features are constructed using key informative nouns extracted from the text content of the Wikipedia pages. As shown in Table 4, GCoT outperforms other competitive baselines on heterophilic node classification. These results indicate that our CoT-style prompting can generalize across both homophilic and heterophilic graphs.

5.6 Flexibility of Graph Prompting Methods

To evaluate the flexibility and robustness of GCoT, we evaluate its performance using various standard graph prompting methods as the standard prompt. We integrate ProG [41], GPF [6], GPF+ [6], and GRAPH PROMPT [30] into our framework. Specifically, ProG, GPF, and GPF+ are employed to modify the input features at the first inference step, whereas GRAPH PROMPT is utilized to modify the output embeddings at the final step. The results for both node and graph classification on four datasets are presented in Table 5. Across

**Figure 4: Visualization of node and thought embeddings. Different colors represent different node classes.****Table 4: Accuracy (%) evaluation of node classification on heterophilic graphs.**

Methods	Wisconsin	Chameleon
DGI	28.04 ± 6.47	19.33 ± 4.57
GraphCL	29.85 ± 8.46	<u>27.16 ± 4.31</u>
ProG	28.99 ± 11.14	25.90 ± 3.95
GPF	<u>32.18 ± 11.36</u>	25.98 ± 3.71
GPF+	32.02 ± 11.17	26.12 ± 3.75
GraphPrompt	31.48 ± 5.18	25.36 ± 3.99
GCoT	32.80 ± 11.59	27.96 ± 3.89

all cases, GCoT outperforms its standard prompting counterparts, highlighting the robustness and flexibility of our CoT design.

5.7 Hyperparameter Analysis

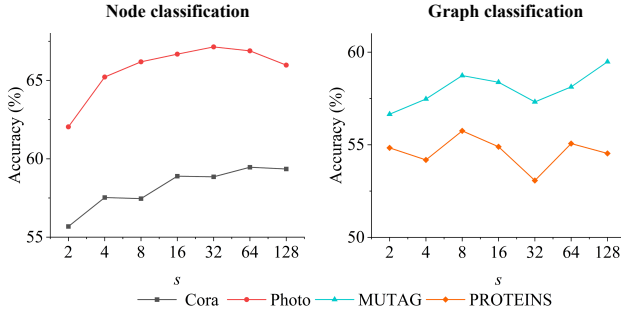
Next, we study the sensitivity of two key hyperparameters, namely, the hidden dimension of the condition-net, and the number of inference steps performed.

Hidden dimension of condition-net. In our experiments, we implement the condition-net as a two-layer MLP with a bottleneck design. To examine the effect of the MLP architecture, we vary its hidden dimension s and present the results in Fig. 5. The results reveal that $s = 32$ generally yields strong results for node classification, and $s = 8$ for graph classification. A smaller s may restrict the model’s representational capacity, limiting its effectiveness,

Table 5: Accuracy (%) evaluation of GCoT with different standard prompting methods.

Prompting	Node classification		Graph classification	
	Cora	Pubmed	MUTAG	PROTEINS
ProG	50.59±14.64	63.07±11.96	51.99±14.50	52.73±6.57
with GCoT	52.62±15.27	64.01±11.27	58.42±15.39	55.82±8.79
GPF	57.60±13.88	55.63±10.96	56.55±13.95	53.05±7.62
with GCoT	58.98±15.28	62.63±10.01	57.66±13.11	56.98±8.30
GPF+	57.42±13.87	57.16±10.99	56.81±12.93	54.58±8.70
with GCoT	59.35±15.37	62.18±10.49	58.54±13.29	55.45±9.20
GRAPHPROMPT	54.25± 9.38	63.11±10.01	55.44±12.56	53.80±7.93
with GCoT	58.40±14.74	63.79± 9.40	58.28±15.47	55.32±8.43

In each group, the first row represents a standard prompting method, and the second row represents the standard prompting coupled with our CoT design.

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

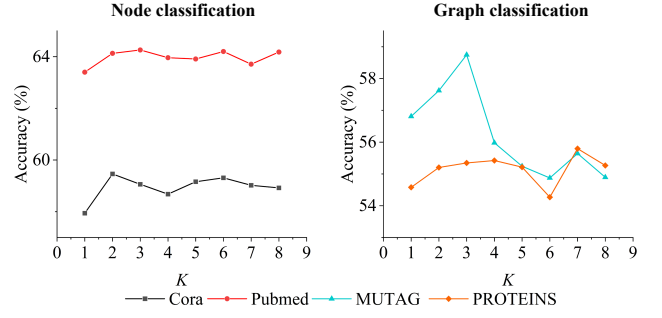
whereas a larger s introduces additional trainable parameters and increases the risk of overfitting in few-shot scenarios.

Number of inference steps. We further investigate the influence of the number of inference steps, K , with the results summarized in Fig. 6. For node classification, we observe that $K = 2$ generally yields the best performance, whereas $K = 3$ is optimal for graph classification. This discrepancy may stem from the fact that graph classification relies more on global structural information, which may be better captured with more inference steps. Notably, for the PROTEINS dataset, $K = 7$ achieves the highest performance, likely due to its intrinsic complexity and variability, as discussed in Sect. 5.3.

5.8 Computational Efficiency Analysis

To evaluate the efficiency of GCoT, we perform one-shot node classification on a large-scale dataset called CLUSTER [5], which comprises 1,172,320 nodes and 43,038,630 edges. The accuracy evaluation is shown in Table 6, whereas inference time comparisons with strong baselines are reported in Table 7.

While GCoT maintains higher accuracy on the large-scale dataset, we observe additional inference time due to its multi-step inference on the test set. Such a limitation of CoT has also been observed in NLP. In our case, inference time increases roughly linearly with the number of steps, which is acceptable considering the significant accuracy gains.

**Figure 6: Impact of number of inference steps K .****Table 6: Accuracy (%) evaluation of node classification on the large-scale CLUSTER dataset.**

Methods	DGI	GRAPHCL	GRAPHPROMPT	GPF+	GCoT
Accuracy	21.27	23.82	22.18	24.27	27.54

Table 7: Inference time (milliseconds) on the test set of a node classification task.

Methods	Cora	Pubmed	MUTAG	CLUSTER
GRAPHPROMPT	3.90	5.62	5.85	955.25
GPF+	5.25	5.92	4.79	713.08
GCoT	7.95	13.95	11.44	2340.00

6 Conclusions

In this paper, we proposed GCoT, the first CoT prompting framework for text-free graphs. Specifically, we introduced the notion of step-by-step inference, where each inference step consists of three substages: prompt-based inference, thought construction, and thought-conditioned prompt learning. Concretely, we first feed the prompt-modified query graph into a pre-trained graph encoder, and then construct a thought by fusing the hidden embeddings from all layers of the pre-trained encoder to capture hierarchical structural knowledge. Subsequently, conditioned on the thought, we generate a series of node-specific prompts to guide the next inference step. After multiple such inference steps, GCoT makes a prediction on the final “answer.” Lastly, we conducted extensive experiments on eight public datasets, demonstrating that GCoT significantly outperforms a range of 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] Karsten M Borgwardt, Cheng Soon Ong, Stefan Schöner, SVN Vishwanathan, Alex J Smola, and Hans-Peter Kriegel. 2005. Protein function prediction via graph kernels. *Bioinformatics* 21, suppl_1 (2005), i47–i56.
- [2] 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.
- [3] Zheng Chu, Jingchang Chen, Qianglong Chen, Weijiang Yu, Tao He, Haotian Wang, Weihua Peng, Ming Liu, Bing Qin, and Ting Liu. 2023. A survey of chain of thought reasoning: Advances, frontiers and future. *arXiv preprint arXiv:2309.15402* (2023).
- [4] Asim Kumar Debnath, Rosa L Lopez de Compadre, Gargi Debnath, Alan J Shusterman, and Corwin Hansch. 1991. Structure-activity relationship of mutagenic aromatic and heteroaromatic nitro compounds. correlation with molecular orbital energies and hydrophobicity. *Journal of medicinal chemistry* 34, 2 (1991), 786–797.
- [5] Vijay Prakash Dwivedi, Chaitanya K Joshi, Anh Tuan Luu, Thomas Laurent, Yoshua Bengio, and Xavier Bresson. 2023. Benchmarking graph neural networks. *Journal of Machine Learning Research* 24, 43 (2023), 1–48.
- [6] Taoran Fang, Yunchao Zhang, Yang Yang, Chunping Wang, and Lei Chen. 2024. Universal prompt tuning for graph neural networks. *NeurIPS* (2024).
- [7] Yuan Fang, Yuxia Wu, Xingtong Yu, and Shirui Pan. 2025. Few-Shot Learning on Graphs: From Meta-Learning to LLM-empowered Pre-Training and Beyond. In *Companion Proceedings of WWW*, 9–12.
- [8] Guhao Feng, Bohang Zhang, Yuntian Gu, Haotian Ye, Di He, and Liwei Wang. 2024. Towards revealing the mystery behind chain of thought: a theoretical perspective. *NeurIPS* 36 (2024).
- [9] Luyu Gao, Aman Madaan, Shuyan Zhou, Uri Alon, Pengfei Liu, Yiming Yang, Jamie Callan, and Graham Neubig. 2023. Pal: Program-aided language models. In *ICML*. 10764–10799.
- [10] David Ha, Andrew M Dai, and Quoc V Le. 2022. HyperNetworks. In *ICLR*.
- [11] Will Hamilton, Zitao Ying, and Jure Leskovec. 2017. Inductive representation learning on large graphs. *NeurIPS* (2017), 1025–1035.
- [12] 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*.
- [13] 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.
- [14] Shuo Ji, Xiaodong Lu, Mingzhe Liu, Leilei Sun, Chuanren Liu, Bowen Du, and Hui Xiong. 2023. Community-based dynamic graph learning for popularity prediction. In *SIGKDD*. 930–940.
- [15] Xinke Jiang, Zidi Qin, Jiarong Xu, and Xiang Ao. 2023. Incomplete graph learning via attribute-structure decoupled variational auto-encoder. In *WSDM*. 304–312.
- [16] Xinke Jiang, Rihong Qiu, Yongxin Xu, Wentao Zhang, Yichen Zhu, Ruizhe Zhang, Yuchen Fang, Xu Chu, Junfeng Zhao, and Yasha Wang. 2024. RAGraph: A General Retrieval-Augmented Graph Learning Framework. In *NeurIPS*.
- [17] Xinke Jiang, Dingyi Zhuang, Xianghui Zhang, Hao Chen, Jiayuan Luo, and Xiaowei Gao. 2023. Uncertainty quantification via spatial-temporal tweedie model for zero-inflated and long-tail travel demand prediction. In *CIKM*.
- [18] Bowen Jin, Chulin Xie, Jiawei Zhang, Kashob Kumar Roy, Yu Zhang, Zheng Li, Ruirui Li, Xianfeng Tang, Suhang Wang, Yu Meng, et al. 2024. Graph chain-of-thought: Augmenting large language models by reasoning on graphs. *arXiv preprint arXiv:2404.07103* (2024).
- [19] Anshul Kanakia, Zhihong Shen, Darrin Eide, and Kuansan Wang. 2019. A scalable hybrid research paper recommender system for microsoft academic. In *WWW*. 2893–2899.
- [20] Thomas N Kipf and Max Welling. 2016. Variational graph auto-encoders. In *Bayesian Deep Learning Workshop*.
- [21] Thomas N Kipf and Max Welling. 2017. Semi-supervised classification with graph convolutional networks. In *ICLR*.
- [22] Namkyeong Lee, Kanghoon Yoon, Gyoung S Na, Sein Kim, and Chanyoung Park. 2023. Shift-robust molecular relational learning with causal substructure. In *SIGKDD*. 1200–1212.
- [23] Brian Lester, Rami Al-Rfou, and Noah Constant. 2021. The Power of Scale for Parameter-Efficient Prompt Tuning. In *EMNLP*. 3045–3059.
- [24] Yibo Li, Yuan Fang, Mengmei Zhang, and Chuan Shi. 2024. FineMolTex: Towards Fine-grained Molecular Graph-Text Pre-training. *arXiv preprint arXiv:2409.14106* (2024).
- [25] Yibo Li, Xiao Wang, Hongrui Liu, and Chuan Shi. 2024. A Generalized Neural Diffusion Framework on Graphs. In *AAAI*. 8707–8715.
- [26] Yibo Li, Xiao Wang, Yujie Xing, Shaohua Fan, Ruijia Wang, Yaoqi Liu, and Chuan Shi. 2024. Graph Fairness Learning under Distribution Shifts. In *WWW*. 676–684.
- [27] 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).
- [28] Yang Liu, Deyu Bo, Wenxuan Cao, Yuan Fang, Yawen Li, and Chuan Shi. 2025. Graph Positional Autoencoders as Self-supervised Learners. *arXiv:2505.23345*
- [29] 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.
- [30] 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.
- [31] Yuanfu Lu, Xunqiang Jiang, Yuan Fang, and Chuan Shi. 2021. Learning to pre-train graph neural networks. In *AAAI*. 4276–4284.
- [32] Andrew Kachites McCallum, Kamal Nigam, Jason Rennie, and Kristie Seymore. 2000. Automating the construction of internet portals with machine learning. *Information Retrieval* (2000).
- [33] 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).
- [34] 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.
- [35] Ryan A. Rossi and Nesreen K. Ahmed. 2015. The Network Data Repository with Interactive Graph Analytics and Visualization. In *AAAI*. 4292–4293.
- [36] Benedek Rozemberczki, Carl Allen, and Rik Sarkar. 2021. Multi-scale attributed node embedding. *Journal of Complex Networks* (2021), cnab014.
- [37] Prithviraj Sen, Galileo Namata, Mustafa Bilgic, Lise Getoor, Brian Galligher, and Tina Eliassi-Rad. 2008. Collective classification in network data. *AI magazine* (2008).
- [38] Oleksandr Shchur, Maximilian Mumme, Aleksandar Bojchevski, and Stephan Günnemann. 2018. Pitfalls of graph neural network evaluation. *arXiv preprint arXiv:1811.05868* (2018).
- [39] Fan-Yun Sun, Jordan Hoffmann, Vikas Verma, and Jian Tang. 2020. Infograph: Un-supervised and semi-supervised graph-level representation learning via mutual information maximization. In *ICLR*.
- [40] 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.
- [41] Xiangguo Sun, Hong Cheng, Jia Li, Bo Liu, and Jihong Guan. 2023. All in one: Multi-task prompting for graph neural networks. In *SIGKDD*. 2120–2131.
- [42] Petar Velićković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. 2018. Graph attention networks. In *ICLR*.
- [43] Petar Velićković, William Fedus, William L Hamilton, Pietro Liò, Yoshua Bengio, and R Devon Hjelm. 2019. Deep Graph Infomax. In *ICLR*.
- [44] 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.
- [45] Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc Le, Ed Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. 2023. Self-consistency improves chain of thought reasoning in language models. *ICLR* (2023).
- [46] Xu Wang, Huan Zhao, Wei-wei Tu, and Quanming Yao. 2023. Automated 3D pre-training for molecular property prediction. In *SIGKDD*. 2419–2430.
- [47] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. 2022. Chain-of-thought prompting elicits reasoning in large language models. *NeurIPS* 35 (2022), 24824–24837.
- [48] Zhihao Wen and Yuan Fang. 2023. Augmenting Low-Resource Text Classification with Graph-Grounded Pre-training and Prompting. In *SIGIR*. 506–516.
- [49] 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.
- [50] Chenyan Xiong, Russell Power, and Jamie Callan. 2017. Explicit semantic ranking for academic search via knowledge graph embedding. In *WWW*. 1271–1279.
- [51] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. 2019. How powerful are graph neural networks?. In *ICLR*.
- [52] Hao Yan, Chaozhao Li, Ruosong Long, Chao Yan, Jianan Zhao, Wenwen Zhuang, Jun Yin, Peiyan Zhang, Weihao Han, Hao Sun, et al. 2023. A comprehensive study on text-attributed graphs: Benchmarking and rethinking. *NeurIPS* 36 (2023), 17238–17264.
- [53] Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Tom Griffiths, Yuan Cao, and Karthik Narasimhan. 2024. Tree of thoughts: Deliberate problem solving with large language models. *NeurIPS* 36 (2024).
- [54] 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.
- [55] Yuning You, Tianlong Chen, Yongduo Sui, Ting Chen, Zhangyang Wang, and Yang Shen. 2020. Graph contrastive learning with augmentations. *NeurIPS* 33 (2020), 5812–5823.
- [56] 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).

- [57] Xingtong Yu, Yuan Fang, Zemin Liu, and Xinming Zhang. 2024. Hgprompt: Bridging homogeneous and heterogeneous graphs for few-shot prompt learning. In *AAAI*, Vol. 38. 16578–16586.
- [58] Xingtong Yu, Zechuan Gong, Chang Zhou, Yuan Fang, and Hui Zhang. 2025. SAMGPT: Text-free graph foundation model for multi-domain pre-training and cross-domain adaptation. In *Proceedings of the ACM on Web Conference 2025*. 1142–1153.
- [59] 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 TKDE* (2024).
- [60] Xingtong Yu, Zemin Liu, Yuan Fang, and Xinming Zhang. 2023. Learning to count isomorphisms with graph neural networks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 37. 4845–4853.
- [61] Xingtong Yu, Zhenghao Liu, Xinming Zhang, and Yuan Fang. 2025. Node-Time Conditional Prompt Learning In Dynamic Graphs. In *ICLR*.
- [62] Xingtong Yu, Jie Zhang, Yuan Fang, and Renhe Jiang. 2025. Non-homophilic graph pre-training and prompt learning. In *SIGKDD*.
- [63] Xingtong Yu, Chang Zhou, Yuan Fang, and Xinming Zhang. 2024. Multigprompt for multi-task pre-training and prompting on graphs. In *WWW*. 515–526.
- [64] Xingtong Yu, Chang Zhou, Yuan Fang, and Xinming Zhang. 2024. Text-Free Multi-domain Graph Pre-training: Toward Graph Foundation Models. *arXiv preprint arXiv:2405.13934* (2024).
- [65] Seongjun Yun, Minbyul Jeong, Raehyun Kim, Jaewoo Kang, and Hyunwoo J Kim. 2019. Graph transformer networks. *NeurIPS* 32 (2019).
- [66] Shiqi Zhang, Yiqian Huang, Jiachen Sun, Wenqing Lin, Xiaokui Xiao, and Bo Tang. 2023. Capacity constrained influence maximization in social networks. In *SIGKDD*. 3376–3385.
- [67] 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.
- [68] Kaiyang Zhou, Jingkang Yang, Chen Change Loy, and Ziwei Liu. 2022. Conditional prompt learning for vision-language models. In *CVPR*. 16816–16825.

Appendices

A Further Descriptions of Datasets

We provide more comprehensive descriptions of the benchmark datasets^{3,4,5} used in our experiments.

- *Cora* [32] is a citation network composed of 2,708 research papers in the field of computing, each classified into one of seven categories. The network consists of 5,429 citation links between papers. Each paper is represented by a binary word vector, where each entry indicates the presence or absence of a word from a predefined vocabulary of 1,433 unique terms.
- *Citeseer* [37] includes 3,312 computer science publications, categorized into six distinct classes, separate from those in *Cora*. The citation network contains 4,732 edges. Each document is encoded as a binary word vector that captures the presence or absence of words from a dictionary comprising 3,703 unique terms.
- *PubMed* [37] is a citation network of 19,717 biomedical articles related to diabetes, divided into three categories. The network includes 44,338 citation edges. Unlike *Cora* and *Citeseer*, each document is represented by a TF/IDF-weighted word vector derived from a dictionary of 500 unique terms.
- *Photo* [38] consists of 7,487 photography-related products, each assigned to one of eight categories. The co-purchase network contains 119,043 edges, where connections indicate products frequently bought together. Each product is described by a feature vector extracted from metadata and customer reviews, with category labels corresponding to product types.
- *MUTAG* [4] is a dataset of nitroaromatic compounds aimed at predicting their mutagenic effects on *Salmonella typhimurium*.

Each compound is modeled as a graph, where nodes represent atoms with categorical labels (encoded as one-hot vectors) based on atom types, and edges depict the chemical bonds connecting them. The dataset comprises 188 molecular graphs with 7 unique node types.

- *BZR* [35] consists of 405 molecular graphs representing ligands that interact with benzodiazepine receptors. Each molecule is treated as an independent graph and is classified into one of two categories.
- *COX2* [35] includes 467 molecular structures of cyclooxygenase-2 inhibitors. In this dataset, nodes correspond to atoms, while edges define chemical bonds—which may be single, double, triple, or aromatic. The molecules are divided into two distinct classes.
- *PROTEINS* [1] is a dataset of protein structure graphs that encode both biochemical and structural properties. In this dataset, nodes represent secondary structural elements, while edges capture connectivity based on spatial proximity or amino acid sequence adjacency. Each node falls into one of three categories, and graphs are classified into two broader groups.

B Further Descriptions of Baselines

We provide additional details about the baseline methods used in our experiments.

(1) Supervised GNNs.

- **GCN** [21]: A graph neural network that aggregates node information using mean-pooling, thereby enabling nodes to capture structural information from their neighbors.
- **GAT** [42]: Unlike GCN, GAT incorporates attention mechanisms to assign different weights to neighboring nodes, refining the aggregation process based on their relative importance.

(2) Graph Pre-training Models.

- **DGI** [42]: A self-supervised pre-training method that maximizes mutual information between local node embeddings and the graph’s global representation, thereby enhancing structural awareness.
- **InfoGraph** [39]: An extension of DGI designed for graph-level tasks, aligning node and graph representations by optimizing their similarity.
- **GraphCL** [55]: A contrastive learning framework that leverages diverse graph augmentations to extract structural patterns, aiming to improve representation consistency across transformations.

(3) Standard Graph Prompting Models.

- **ProG** [41]: Reformulates node- and edge-level tasks as graph-level problems by employing prompt graphs with task-specific structures to guide adaptation.
- **GPF** [6]: A universal prompt-tuning strategy for pre-trained graph models that transforms input graph features to mimic various prompting effects.
- **GPF+** [6]: An enhanced version of GPF that integrates an attention mechanism to dynamically refine prompt representations.
- **GraphPrompt** [30]: Bridges pre-training and downstream tasks using subgraph similarity-based prompting, where a learnable prompt is optimized to incorporate task-relevant information for both node and graph classification.

³<https://github.com/shchur/gnn-benchmark/raw/master/data/npz/>

⁴<https://huggingface.co/datasets/graphs-datasets/MUTAG>

⁵<https://chrsmrrs.github.io/datasets/docs/datasets/>