An Efficient Community-Aware Pre-training Method for Graph Neural Networks

Zhenhua Huang[a,b,c,1], Wenhao Zhou[a,1], Yihang Jiang[a,b], Zhaohong Jia[a], Linyuan Lü[b,c], Yunjie Ma[c,d]

[a]*Anhui University, Hefei, 230039, China*
[b]*University of Science and Technology of China, 230026, Hefei, China*
[c]*Dataspace Institute of Hefei Comprehensive National Science Center, 230029, Hefei, China*
[d]*Hefei University of Technology, 230009, Hefei, China*

Abstract

While graph neural networks (GNNs) have demonstrated widespread success in various domains, their pre-training techniques lag behind those in computer vision and natural language processing, typically exhibiting limited performance gains and high computational costs. This paper introduces Community-Aware Pre-training (CAP), a novel approach that leverages the inherent community structures prevalent in real-world networks to enhance GNN pre-training efficiency and effectiveness. CAP employs a self-supervised contrastive learning framework to learn node representations that are highly discriminative of their respective communities. To further optimize the pre-training process, we introduce a Monte Carlo Tree Search-based community sampler that efficiently extracts representative subgraphs, mitigating noise and enhancing sample quality. CAP is versatile and can be applied to a broad range of node classification tasks due to the commonly existing community structures within networks. Extensive evaluations on diverse node classification benchmarks demonstrate that CAP consistently outperforms state-of-the-art methods, achieving accuracy improvements of up to 4.34% while significantly reducing pre-training time by up to 14.87 times compared to existing techniques. Furthermore, CAP enhances the predictive confidence and visualization distinctiveness of node representations, paving a new path for

Email addresses: zhhuangscut@gmail.com (Zhenhua Huang), wenhaozhou@stu.ahu.edu.cn (Wenhao Zhou), yihangjiangahu@gmail.com (Yihang Jiang), zhjia@mail.ustc.edu.cn (Zhaohong Jia), linyuan.lv@ustc.edu.cn (Linyuan Lü), ma.yunjie@foxmail.com (Yunjie Ma)
[1]This is the first author footnote.

effective and efficient GNN pre-training.

---

1. Introduction

Graph-based pattern recognition has emerged as a powerful tool for analyzing data structured in graph form, to which Prof. Edwin R. Hancock has dedicated significant research efforts. His seminal work on representation learning in the node [1] and graph domain [2] has made outstanding contributions to the field. In this paper, we delve deeper into his research domain, particularly focusing on graph neural networks (GNNs), which find applications in diverse domains. However, unlike the significant advancements in pre-training techniques witnessed in computer vision [3] and natural language processing [4], GNN pre-training remains relatively underexplored, with substantial room for improvement in both performance and efficiency. This challenge arises from the unique characteristics of graph data, particularly their non-Euclidean structure and varying semantics across datasets, which hinder the direct transferability of pre-training strategies commonly employed for text and images [5].

Despite these hurdles, recent years have seen notable progress in GNN pre-training. Early works like Hu et al. [6] proposed integrating node and graph-level information during pre-training to enhance generalization and transferability across different molecular structures. You et al. [7] further explored this direction by introducing graph data augmentation strategies for pre-training. Similarly, Xia et al. [8] proposed novel techniques for molecular graph pre-training, including atom vocabulary expansion and representation refinement tasks. While these methods have shown promise on specific graph prediction tasks, their applicability remains limited to certain dataset types. To address cross-domain pre-training, GCC [9] leverages contrastive learning to discover structural patterns across diverse network structures. GPPT [10] introduces masked edge prediction for GNN pre-training, converting nodes into token pairs for node classification. GSR [11] employs multi-view contrastive learning for link prediction and graph structure refinement, fine-tuning the pre-trained model for downstream tasks. GraphPrompt [12] unifies pre-training and downstream tasks into a common template using task-specific prompt vectors, while All-in-One [13] reformulates

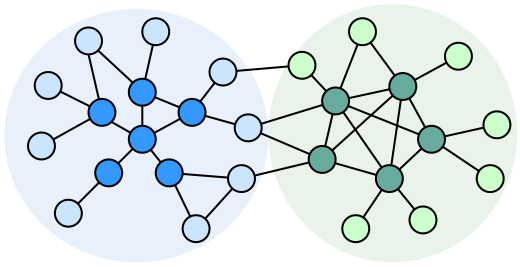


Figure 1: Community structures within networks. Nodes in two communities are depicted in blue and green, and dark blue and green highlight the presence of densely connected substructures ("cores") within each community.

different-level tasks into a unified representation and utilizes a meta-learning technique for prompt graph design. Although these methods reduce the data and training requirements for downstream tasks through fine-tuning, they typically rely on intricate pre-training strategies, leading to high computational costs and limited accuracy improvements in some cases.

Despite being a ubiquitous feature of real-world networks, community structures have often been overlooked in the design of current graph neural networks [14], as illustrated in Fig. 1. Communities, characterized as cohesive groups of densely interconnected nodes within a network [15], hold significant real-world implications across diverse domains [16]. The presence of community structures underscores the existence of rich structural information within graphs, reflecting the inherent complexity and depth of the network topology. Within communities, densely connected substructures known as "cores" often emerge, serving as essential backbones for community formation and functionality [17]. These cores, characterized by their high internal connectivity, act as primary conduits for information exchange among community members. Understanding these core structures is crucial for developing effective graph analysis models. Leveraging the inherent community patterns, particularly these informative core structures, during GNN pre-training holds significant potential for enhancing model performance.

Leveraging the pervasiveness and informativeness of community structures, we propose a **C**ommunity-**A**ware **P**re-training (**CAP**) method of GNNs. CAP aims to learn node representations with enhanced community discriminative power. To achieve this,

we leverage contrastive learning, a self-supervised approach that encourages representations of nodes within the same community to be closer while pushing apart those from different communities. The universality of community structures makes CAP broadly applicable to a wide range of node classification datasets. However, community detection methods often yield noisy structures with varying node sizes, and simply merging community information with GNN [18] produces limited performance gains. Inspired by the concept of cores of communities [17], we introduce a community sampler to identify the representative substructures within communities to reduce the noisy community information in contrastive learning. Unlike conventional pre-training methods that require extensive fine-tuning on large-scale datasets before adaptation to task-specific datasets, the pre-training task of CAP necessitates minimal adjustment, enabling efficient transfer to downstream tasks with a few fine-tuning steps while still achieving competitive performance.

Extensive experiments demonstrate that CAP consistently enhances GNN performance on node classification tasks, achieving state-of-the-art accuracies with improvements of up to 4.34%. Moreover, CAP exhibits significantly lower time consumption than existing pre-training strategies, reducing training time by up to 14.87 times faster than existing techniques. Empirical evidence also reveals that CAP leads to improved predictive confidence, as evidenced by higher and more concentrated confidence distributions than its backbone models. The efficacy of CAP is further investigated through visualizations of node representations and real-world case studies.

This paper makes the following key contributions:

- **Community-Aware Pre-training for GNNs**: We present CAP, an innovative GNN pre-training framework that explicitly utilizes the inherent community structures in graphs. CAP uses a contrastive learning approach to enhance node representations with improved community discriminative power, facilitating effective generalization across various datasets.

- **Efficient Community Sampling**: We introduce a novel community sampler that efficiently extracts representative community information while reducing noise impact. This sampler significantly boosts the effectiveness and efficiency of the

CAP pre-training process, accelerating computations by up to 4.53 times compared to the baseline sampler.

- **Extensive Experimental Validation**: We perform comprehensive experiments on seven real-world datasets, showing that CAP consistently surpasses state-of-the-art GNN models and existing pre-training methods. CAP achieves notable accuracy improvements (up to 4.34%) and substantially reduces training time (up to 14.87 times faster).

2. Related Works

2.1. Graph Neural Networks

GNNs represent a specialized category of deep neural networks tailored for handling data structured in graphs, a domain to which Professor Edwin R. Hancock has dedicated research efforts. The field of GNNs encompasses a rich diversity of architectures, each with unique strengths and characteristics. Notable examples include GCN [19], which leverages neighbor-to-neighbor information propagation; and GAT [20], which introduces a self-attention mechanism to aggregate node features with adjustable weights. GraphSage [21] utilizes localized neighborhood sampling and aggregation to generate novel data embeddings. Further advancements include FusedGAT [22], which optimizes GAT for reduced computational and memory requirements. The ASDGN [23] draws inspiration from ordinary differential equations to provide a stable and non-dissipative framework for designing deep GNNs. SGCN [24] introduces graph-shaped kernels to perform multichannel convolutions on subgraph structures. Wang et al. [25] proposed heterophily-aware graph attention networks to adaptively assign weights to different types, breaking the limitations of graph attention networks to handle heterophilic graphs. Furthermore, a growing body of research focuses on enhancing the interpretability of GNNs [26, 27], aiming to elucidate their decision-making processes. More papers about GNN can be referred to the recent review paper by Xu et al. [28].

2.2. Community Detection

Community detection, a fundamental problem in network analysis and graph theory [14], aims to uncover the underlying community structure within networks. It finds

diverse applications, such as targeted advertising in social networks and identifying research trends in citation networks. Numerous techniques and algorithms have been developed for community detection, which can be broadly categorized based on their underlying principles [29]. Modularity-based methods, such as the Louvain method [30], seek to maximize a modularity function to determine optimal community assignments. Information-theoretic methods, exemplified by Infomap [31], employ random walks to minimize the expected description length of community structures. Label propagation algorithms [32] iteratively assign labels to nodes based on the majority labels of their neighbors. Spectral clustering techniques [33] leverage eigenvectors of similarity matrices to identify communities. More recently, deep learning approaches have been explored, utilizing neural networks [34] and embedding techniques [35] for community detection. To the best of our knowledge, this paper presents the first investigation into leveraging community structures for GNN pre-training.

2.3. Pre-training on Graphs

Pre-training involves training a model on a large corpus of unlabeled data to learn general representations of the input data. Traditional pre-training tasks have demonstrated significant progress in the fields of computer vision and natural language processing. However, due to the unique characteristics of graph-structured data, traditional pre-training methods cannot be directly applied to graphs. GCC [9] employs contrastive learning to discover patterns in diverse network structures, addressing the cross-domain pre-training challenge in graph learning. GPPT [10] integrates features and structure by utilizing masked edge prediction to pre-train graph neural networks. These methods introduce graph-based prompts, transforming individual nodes into token pairs and redefining downstream node classification tasks. GSR [11] employs a multi-view contrastive learning approach for graph structure estimation via link prediction during the pre-training phase, followed by structure refinement through probabilistic edge modification. Fine-tuning involves initializing a GNN with this model for downstream task optimization. GraphPrompt [12] is a novel framework that enhances graph neural networks by unifying tasks into a common template and using task-specific prompt vectors. All-in-One [13] reformulates different-level tasks into

unified ones and designs an effective prompt graph with a meta-learning technique. HGPROMPT [36] unifies not only pre-training and downstream tasks but also homogeneous and heterogeneous graphs via a dual-template design. AAGCN [37] is a dropout-based augmentation framework that incorporates traditional graph augmentation methods into a mathematical model, allowing adaptive learning of mask matrices for node and edge feature augmentation. This framework can be easily integrated into existing graph neural networks.

These models, based on the pre-training and fine-tuning architecture [4], require the design of complex pre-training tasks, leading to extended pre-training times. Additionally, they perform well on a limited subset of datasets and frequently encounter issues with negative transfer.

3. Problem Defnition

GNNs are leveraged for various classical tasks in graph-based learning, such as node classification, graph classification, and link prediction. In this paper, we focus solely on the node classification task. Node classification involves assigning a categorical label to each node in a graph, which can be formally defined as follows:

Let $G = (V, E)$ be a graph where V denotes the set of nodes and E the set of edges. Each node $v \in V$ is associated with a label y_v from a set of labels Y. The goal of node classification is to learn a function f that maps nodes to their respective labels:

$$f : V \to Y, \quad f(v) = \arg\max_{y \in Y} P(y|\mathcal{N}(v); \theta_f) \tag{1}$$

where $\mathcal{N}(v)$ denotes the neighborhood of the node v in the graph G, and θ_f represents the parameters of the function f, learned during the training process. The function f is designed to maximize the conditional probability P of the correct label y given the features of the node v and its neighborhood, parameterized by θ_f.

In this process, the constraint of having labels for training is fundamental. However, pre-training on graphs can be performed in an unsupervised manner, extracting rich feature representations from the graph structure without the need for labels. To this end, it is essential to design self-supervised pre-training tasks that generate embeddings conducive to the downstream task. Such a pre-training task can be represented

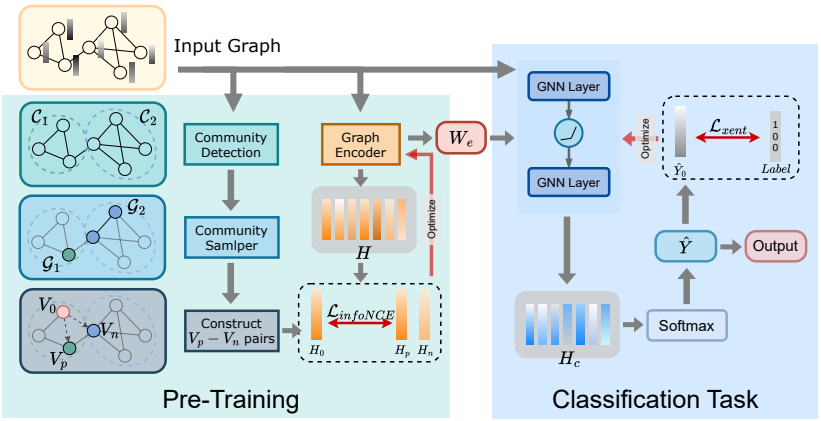


Figure 2: Framework of CAP.

by a function T that operates on the graph and returns a parameter Θ, which can be formalized as:

$$T : (G, \Phi) \rightarrow \Theta, \quad \Theta = \{\theta_f^{(1)}, \theta_f^{(2)}, \ldots, \theta_f^{(n)}\} \tag{2}$$

where Φ represents a set of transformations applied to graph G, and Θ is a set of learned parameters corresponding to each transformation in Φ. The function T maps the graph G under transformations Φ to a parameter space Θ, optimizing a predefined loss function $\mathcal{L}$, which measures the effectiveness of these parameters for the downstream tasks. This parameter Θ guides the optimization of the downstream task, enhancing the performance of the node classification model by providing pre-trained embeddings that encapsulate the intrinsic properties of the graph.

4. Community-Aware Pre-Training Graph Neural Networks

4.1. Community-Aware Pre-Training Task

In community-aware pre-training, the initial step involves using a community sampler to extract crucial subgraphs from communities. Subsequently, positive and negative node representations are generated for each node in the dataset based on these subgraphs and the node representations constructed by the graph encoder. An InfoNCE loss is employed to optimize the parameters of the graph encoder, which are subsequently utilized in downstream prediction tasks. The overall framework of CAP is illustrated in Fig. 2.

4.1.1. Community and Subgraphs Sampling

Community Detection: Community detection is a fundamental task in network analysis aimed at revealing underlying structures within networks. Given an input graph $G(V, A)$, community detection returns community list C. This paper compares two classical non-overlapping community detection approaches: modularity-based [15] and label propagation-based [32]. The modularity-based approach evaluates the quality of a community partition by quantifying modularity, striving to maximize this metric to achieve an optimal partition. The Louvain method [30] is representative of this category. The label propagation-based approach operates by updating the labels of unlabeled nodes based on the label information of their neighboring nodes, iteratively propagating labels throughout the network until convergence. The Label Propagation Algorithm (LPA) [32] is a classic method in this category.

Community Sampler: Communities consist of core nodes, overlapping nodes, and periphery nodes with noisy data. We have devised a community sampler tailored to extract the most representative structural subgraphs within a community and mitigate the impact of noisy information. For each community C, the community sampler initiates the sampling process by first selecting the node with the highest degree v_c as the central node. Subsequently, the sampler extends this subgraph by iteratively adding nodes directly connected to it within the community. During each expansion step, nodes are selected from the community based on their immediate connection to the subgraph, and their impact on the modularity Q of the subgraph is computed as a measure of their importance [15]:

$$Q = \sum_{c=1}^{n_c} \left(\frac{l_c}{M} - \left(\frac{d_c}{2M}\right)^2\right), \tag{3}$$

where n_c is the number of communities, l_c is the number of edges within the community, and d_c is the sum of the degrees of all nodes in the community c, M denotes the total number of edges in the graph.

The node with the largest impact is selected in the subgraph, concluding that particular expansion step. The sampling process within the community continues until the subgraph reaches a predefined scale S, signifying the completion of community sampling.

Algorithm 1 Community Sampler.

Input: Community list C_{list}, subgraph scale S.
Output: Sampled subgraph list $\mathcal{G}$.

```
for C in C_list do
    Initialize empty graph G_i;
    Choose center node v_c with the highest degree;
    G_i ← G_i + v_c;
    while len(G_i) < S do
        Q_c ← Modularity(C);
        Initialize list score
        while v_i ∈ C do
            if v_i not in G_i and have edge(v_i, G_i) then
                G_new ← G_i + v_i;
                score_i ← Q_c − Modularity(C − G_new);
            end if
        end while
        Obtain the best node v_best with max(score);
        G_i ← G_i + v_best;
    end while
end for
return The subgraph list G;
```

To optimize the computational efficiency of the community sampler, we employed a strategy based on the Monte Carlo Tree Search (MCTS). The utilization of MCTS allows for a reduction in time consumption by constraining the number of iterations. Specifically, we build a search tree where the root is associated with the input graph and each other node corresponds to a connected subgraph. Each edge in the search tree indicates that the graph associated with a child node can be obtained by performing node-pruning from the graph associated with its parent node. We define the importance score R of each node in the search tree $\mathcal{N}_i$ as the decrease in modularity of the community when the node is removed from the corresponding subgraph, which can be computed using the following formula:

$$R(\mathcal{N}_i) = \frac{Q(C) - Q(C - G(\mathcal{N}_i))}{Q(C)}, \tag{4}$$

Formally, for node $\mathcal{N}_i$ the action selection criteria of node $\mathcal{N}_j$ are defined as:

$$a^* = \frac{W(\mathcal{N}_j)}{C(\mathcal{N}_j)} + \alpha R(\mathcal{N}_j) \sqrt{\frac{log(C(\mathcal{N}_i))}{C(\mathcal{N}_j)}}, \tag{5}$$

where $W(\mathcal{N}_i)$ represents the total reward for all visits to $\mathcal{N}_i$, $C(\mathcal{N}_i)$ representing the number of times $\mathcal{N}_i$ being selected, α is a hyperparameter to control the trade-off between exploration and exploitation. After each expansion step, $W(\mathcal{N}_i)$ is incremented by $R(\mathcal{N}_i)$, and $C(\mathcal{N}_i)$ is increased by one. The procedures of the community sampler are summarized in Algorithm 1.

4.1.2. Contrast Learning on Community

Graph Encoder: We employ GCN [19] and GAT [38] as backbone graph encoders to obtain node representations. Given an undirected graph $\mathcal{G}$ with the node feature matrix $X \in \mathcal{R}^{N\times C}$ and adjacency matrix $A \in \mathcal{R}^{N\times N}$, the propagation process of an L-layer GCN is represented as:

$$X^{l+1} = \sigma\left(\hat{D}^{-\frac{1}{2}}\hat{A}\hat{D}^{-\frac{1}{2}}X^{l}W^{l}\right), \tag{6}$$

where X^l denotes the representation of layer l, σ is the non-linear activation function. W are learnable weight matrices. For GAT, propagation of the lth layer is represented as:

$$\begin{aligned} X^{l+1} &= \mathrm{ELU}\left(\sum_{i=1}^{N}\sum_{j\in N(i)}\alpha_{ij}W^{l}x_j^{l}\right), \\ \alpha_{ij} &= \frac{\exp\left(\sigma\left(a^{T}[W^{l}x_i^{l}\|W^{l}x_j^{l}]\right)\right)}{\sum k\in N(i)\exp\left(\sigma\left(a^{T}[W^{l}x_i^{l}\|W^{l}x_k^{l}]\right)\right)}, \end{aligned} \tag{7}$$

The attention coefficient α_{ij} determines the contribution of node j to node i when aggregating features from the neighboring nodes. The set $N(i)$ represents the collection of neighbor nodes of node i.

The output of the last layer of Graph Encoder is denoted as $H \in \mathcal{R}^{N\times C_{out}}$, C_{out} represents the dimension of the node representations in the final layer.

Positive and Negative Pairs: We consider each node v_i in the graph as an anchor point and select k positive sample pairs from nodes within the same community (selected via Community Sampler) to form its positive sample set $V_p(v_i)$ and k negative sample pairs from nodes belonging to different communities to constitute its negative sample set $V_n(v_i)$.

In the pre-training loss formulation, we use H_a to represent the anchor representations, which are equivalent to X. Based on the node sets V_p and V_n, we obtain the positive representation H_p and negative representation H_n for the corresponding nodes from the results of graph encoding. where each element in H_p or H_n is $h_i \in \mathcal{R}^{k \times C_{out}}$.

Pre-Training Loss: We use InfoNCE loss to calculate the intrinsic dissimilarities between these samples. The loss function is as follows:

$$\mathcal{L}_{\text{InfoNCE}} = -\log \frac{\sum_{i=0}^{N} \exp(s(h_i, h_i^+)/\tau)}{\sum_{i=0}^{N} \exp(s(h_i, h_i^+) + \sum_{i=0}^{N} \exp(s(h_i, h_i^-)/\tau)} \tag{8}$$

$\mathrm{s}(h_i, h_i^+)$ denotes the similarity between the anchor vector h_i and the positive sample vector h_i^+. The similarity function (sim) could be a dot product or cosine similarity. $\mathrm{s}(h_i, h_i^-)$ denotes the similarity between the anchor vector h_i and the negative sample vector h_i^-. τ is the temperature parameter that controls the scaling of the similarities. It helps in smoothing the distribution of the similarities. The objective of the loss is to minimize the distance between positive samples, ensuring their proximity in the learned embedding space, while simultaneously maximizing the distance between negative samples, promoting their distinctiveness.

4.2. Supervised Node-Level Property Prediction

After community-aware pre-training, we obtain the parameters of the graph encoder. These parameters are employed to initialize the prediction model. The model is then trained using label information, allowing us to make node-level predictions.

The prediction model is consistent with the graph encoder employed in the pre-training task. The result of graph encoder H_c is used to predict the label after a fully connected layer, and the model's prediction $\hat{Y} \in \mathcal{R}^{N \times M}$ is obtained after a $Softmax$, O is the number of output label, W_c is the parameters.

$$\hat{Y} = Softmax(W_c H_c + B_c), \tag{9}$$

The loss function for the prediction is:

$$\mathcal{L}_{xent} = -\sum_{v \in V} \sum_{i=1}^{O} Y \ln \hat{Y} + \lambda \|\theta\|^2, \tag{10}$$

where λ denotes a regularization parameter and θ denotes the parameters of the model. The whole process of CAP is summarized in Algorithm 2.

Algorithm 2 Framework of CAP.

Input: Graph $G(V, X, A)$, subgraph scale S.
Output: Predictions $\hat{Y}$.

```
Obtain C_list by Community Detection(G);
Obtain 𝒢 by Community Sampler(C_list, S);
while epochs < pretrain_epochs do
    H ← Graph Encoder(X, A);
    H_p, H_n ← Split(H);
    Update W_e in Graph Encoder by Eq. 8;
end while
Initialize the classification model GNN by W_e;
while not conerged do
    H_c ← GNN(X, A);
    Ŷ ← Softmax(W_c H_c + B_c);
    Update GNN parameters by Eq. 10;
end while
return Predictions Ŷ;
```

4.3. Time Complexity Analysis

We analyze CAP's time complexity, focusing on its crucial component: the community sampler.

Base Community Sampler: For each of the $|\mathcal{C}_{\text{list}}|$ communities, the base sampler iterates until the subgraph size reaches S. In each iteration, the algorithm: 1) Select Nodes: Given the variable number of candidate nodes expanded, we approximate the analysis of edges around each node using the node-edge density $\beta = |E|/|V|$. The selection process runs $O(S \times \beta)$ times. 2) Calculates Modularity: This step has a $O(|E_c|)$ complexity, where $|E_c|$ denotes the number of edges in the community C. Therefore, the base community sampler's time complexity is $O(\sum_{C \in \mathcal{C}_{\text{list}}} S^2 \times \beta \times |E_c|)$.

MCTS-Based Community Sampler: For each community, the MCTS sampler performs I iterations. Each iteration consists of: 1) Tree Policy: Traversing the MCTS tree to a non-leaf node (potentially expanding a new node) takes $O(S)$ time, as the tree depth is proportional to the subgraph size S. 2) Node Expansion: Selecting and expanding a node takes $O(1)$ time due to the random selection process. 3) Backpropagation and Subgraph Selection: Updating nodes along the tree to the root node takes $O(S)$ time. Traversing the tree to select the optimal subgraph also takes $O(S)$ time. 4) Modularity Calculation: This step has a $O(|E_c|)$ complexity. Thus, the overall time

complexity of the MCTS-based sampler is $O(\sum_{C \in C_{\text{list}}} I \times S \times |E_c|)$.

For the **Graph Encoder**: The complexity of the graph encoder depends on the chosen backbone. For a GCN [19], the complexity is $O(|E| \times C \times F_{\text{hid}} \times M)$, where $|E|$ is the number of edges, C is the input feature dimensionality, F_{hid} is the hidden layer size, and O is the number of categories.

The Total Complexity: Combining the sampler and encoder complexities, the total time complexity for CAP with base sampler is $O(\sum_{C \in C_{\text{list}}} S^2 \times \beta \times |E_c| + |E| \times C \times F_{\text{hid}} \times M)$, and for CAP with MCTS sampler is $O(\sum_{C \in C_{\text{list}}} I \times S \times |E_c| + |E| \times C \times F_{\text{hid}} \times M)$.

4.4. Effectiveness Theoretical Analysis

We use Graph Signal Propagation to theoretically demonstrate the effectiveness of CAP community contrastive learning strategies.

Objective: Establish theoretical connections between contrastive learning objectives and global graph signal smoothness, addressing reviewer concerns about the lack of theoretical grounding for long-range dependency modeling.

4.4.1. Graph Signal Smoothness Definition

Let $\mathbf{X} \in \mathbb{R}^{N \times d}$ denote node features and $\mathbf{L} = \mathbf{D} - \mathbf{A}$ be the graph Laplacian (degree matrix $\mathbf{D}$, adjacency matrix $\mathbf{A}$). The smoothness energy is quantified as:

$$\mathcal{E}(\mathbf{X}) = \text{Tr}(\mathbf{X}^\top \mathbf{L} \mathbf{X}) = \frac{1}{2} \sum_{i,j} \mathbf{A}_{i,j} \|\mathbf{X}_i - \mathbf{X}_j\|^2$$

where lower values indicate smoother signals (greater feature similarity among connected nodes). Contrastive Learning Objective

The InfoNCE loss optimizes representations by:

$$\mathcal{L}_{\text{cont}} = -\log \frac{\exp(s(\mathbf{z}_i, \mathbf{z}_j)/\tau)}{\sum_k \exp(s(\mathbf{z}_i, \mathbf{z}_k)/\tau)}$$

where $(\mathbf{z}_i, \mathbf{z}_j)$ are positive pairs (intra-community nodes) and $\mathbf{z}_k$ are negative samples (inter-community nodes).

4.4.2. Implicit Smoothness Constraints

The contrastive objective induces Local Smoothness and Global Discriminability.

Local Smoothness: Maximizing intra-community similarity through positive pairs.

Global Discriminability: Minimizing inter-community similarity via negative pairs, while preserving information flow through community-aware propagation.

4.4.3. Spectral Graph Interpretation

Through Laplacian eigendecomposition $\mathbf{L} = \mathbf{U}\mathbf{\Lambda}\mathbf{U}^\top$, contrastive learning implicitly performs:

$$\mathcal{L}_{\text{cont}} \approx \alpha \text{Tr}(\mathbf{Z}^\top \mathcal{L}_{\text{local}} \mathbf{Z}) + \beta \text{Tr}(\mathbf{Z}^\top \mathcal{L}_{\text{global}} \mathbf{Z})$$

where: $\mathbf{L}_{\text{local}}$ enforces intra-community smoothness (preserving low-frequency components), $\mathbf{L}_{\text{global}}$ regulates inter-community information flow (attenuating high-frequency noise).

From a spectral perspective, the contrastive objective enforces node representations to align with low-frequency components of the graph signal. By maximizing similarity within community subgraphs (local smoothness) and differentiating across communities (global discriminability), the contrastive loss implicitly approximates a weighted Laplacian regularization term. where $\mathbf{L}_{\text{local}}$ and $\mathcal{L}_{\text{global}}$ denote intra-community and inter-community Laplacians respectively. This regularization promotes global smoothness by suppressing high-frequency noise while preserving long-range dependencies through community-aware message passing.

5. Experiments

5.1. Experimental Setup

5.1.1. Data Description

To evaluate the performance of the CAP on the node classification task, we consider the following real-world datasets:

PolBlogs [39]: A social network with 1,490 vertices and 19,025 edges. Vertices represent political blogs and edges represent links between blogs.

Cora [40]: A citation network of scientific publications in machine learning that consists of 2,708 papers represented by a bag-of-words feature, where edges represent citation links.

CiteSeer [40]: A citation network with 3,327 papers. Each paper is represented by a feature vector based on word occurrences, and edges represent citation links.

CS [41]: A citation network of computer science publications that consists of 18,333 scientific papers. Nodes represent authors and are connected by an edge if they coauthored a paper.

PubMed [40]: A citation network of biomedical literature containing 19,717 papers. Each paper is represented by a word vector, and edges represent citation links.

Physics [41]: A co-authorship network of 34,493 high-energy physics theory papers. Nodes represent authors, and an edge connects authors if they coauthored a paper.

Flickr [42]: A social network of photos with 89,250 photos and their associated tag information. Nodes represent users or accounts, and edges represent relationships between these users.

Reddit [21]: A social network of posts and comments on the website Reddit, comprising 232,965 posts and their associated comments. Nodes represent individual posts or submissions made by users on the Reddit platform, and edges represent replies, comments, or other interactions between posts.

5.1.2. Baselines

For node classification, the following strong baselines have achieved notable performance:

GCN [19] is a widely used graph convolution network that updates the features of a node by averaging its neighboring nodes' features.

GAT [20] aggregates neighbor features using multi-head attention, achieving SOTA performance on various datasets.

GCN and GAT often serve as the backbone for pre-trained models to verify their performance and are also employed in this paper.

GraphSage [21] is the first inductive graph neural network that aggregates node features by sampling neighboring nodes. It predicts graph context and labels using the aggregated information.

CE-GCN [18] has made preliminary explorations on utilizing community information to enhance GNNs, which incorporates the modularity constraint to optimize the GCN training process. A modularity constraint calculates the community information of nodes.

DGI [43] is a general approach that relies on maximizing mutual information between patch representations and corresponding high-level summaries of graphs.

FusedGAT [22] is an optimized version of GAT based on the dgNN [44] that fuses

message passing computation for accelerated execution and lower memory footprint.

ASDGN [23] is an Anti-Symmetric Deep Graph Network, that addresses limitations in traditional DGNs by introducing a stable and non-dissipative design framework based on ordinary differential equations.

SGCN [24] leverages graph-shaped convolutional kernels with multi-channel convolution to achieve superior node classification performance.

For pre-train models, we consider the following advanced open-sourced models suitable for the node classification task:

GPPT [10] proposes a transfer learning paradigm, that adopts the masked edge prediction to pre-train GNNs, using a graph prompting function to modify the standalone node into a token pair, and reformulate the downstream node classification to look the same as edge prediction.

GSR [11] framework uses a pre-train-finetune pipeline to solve the goal of graph structure learning differs from the goal of downstream task and scalability issues in terms of time and space during the process of estimation and optimization of the adjacency matrix.

GraphPrompt [12] is a novel framework that enhances graph neural networks by unifying tasks into a common template and using task-specific prompt vectors, significantly outperforming state-of-the-art models on multiple datasets.

All-in-One [13] is a novel method to reformulate different-level tasks to unified ones and further design an effective prompt graph with a meta-learning technique. However, the process is time-consuming.

5.1.3. Parameter Settings

For the setting of CAP, the community detection method employs the LPA. We utilize an MCTS-based community sampling procedure with parameters set as $I = 100, S = 15$, and 30 pre-training epochs. The datasets are partitioned in a ratio of $6 : 2 : 2$ for train/validate/test datasets. We employ the Adam optimizer with a learning rate of 0.001, a hidden layer size of 128, and set the weight decay to $5e^{-4}$. For the baselines, we use the default parameters in the papers. Our experiments were conducted on a workstation equipped with an Intel Core i9-13900K CPU, 128GB RAM,

Table 1: Node Classification Performance.

Datasets	PloBlogs	Cora	CiteSeer	CS	PubMed	Physics	Flickr	Reddit
$\|V\|$	1490	2,708	3,327	18,333	19,717	34,493	89,250	232,965
$\|E\|$	19025	5,429	4,732	163,788	44,338	495,924	899,756	114,615,892
# classes	2	7	6	15	3	5	7	41
GraphSage	94.29±0.34	88.23±0.32	75.54±0.46	93.11±0.79	85.29±0.27	96.01±1.09	44.05±3.69	91.66±0.25
CEGCN	92.13±0.25	88.63±0.51	75.19±0.41	89.13±0.19	86.31±0.23	94.75±0.68	44.27±1.53	87.54±0.35
DGI	92.53±0.74	84.16±0.41	74.32±0.35	93.21±1.13	84.69±0.33	95.28±0.53	44.94±0.72	87.43±0.26
FusedGAT	94.05±1.79	87.05±0.78	73.69±0.86	92.71±0.32	84.57±0.40	95.84±0.10	47.69±0.63	91.26±0.25
ASDGN	93.81±0.72	86.39±0.24	75.72±0.30	92.90±0.15	86.32±0.06	95.82±0.37	47.32±0.40	91.29±0.28
SGCN	93.71±0.27	88.66±0.33	78.10±0.34	92.89±0.45	86.10±0.14	96.70±0.20	51.19±0.40	93.25±0.14
GPPT	92.37±0.43	86.50±0.37	77.31±0.51	91.44±0.73	85.18±0.69	95.01±0.76	47.01±0.98	88.96±0.32
GSR	92.79±0.36	89.30±0.47	76.31±0.35	91.73±0.46	85.93±0.47	95.73±0.29	48.39±0.41	90.33±0.38
GraphPrompt	93.15±0.40	87.22±0.42	78.45±0.49	92.10±0.68	86.05±0.65	95.30±0.72	48.08±0.95	87.33±0.29
All-in-One	92.88±0.38	88.67±0.45	77.29±0.53	92.37±0.70	85.78±0.62	95.55±0.33	47.76±0.50	85.16±0.37
GCN	93.57±0.33	88.38±0.44	75.35±0.57	90.08±0.11	84.35±0.20	95.13±2.11	45.16±0.10	90.13±0.20
CAP (GCN)	**96.31**±0.29	**90.61**±0.65	**79.21**±0.57	93.29±0.53	85.84±0.22	96.33±0.63	49.06±0.41	93.31±0.32
GAT	93.61±0.57	87.04±0.54	75.90±0.36	91.72±0.44	85.18±0.08	95.83±1.78	46.10±1.58	90.66±0.34
CAP (GAT)	94.94±0.12	90.05±0.61	77.62±0.32	**93.48**±0.21	**86.53**±0.16	**96.42**±0.13	**50.44**±0.53	**93.31**±0.26

and an NVIDIA RTX A6000 GPU with 48GB VRAM.

5.2. Performance Evaluation

5.2.1. Node Classification

We conducted experiments to assess the performance improvement of CAP on node classification on eight real-world datasets, as shown in Table 1. CAP consistently delivered notable improvements compared to advanced models. For instance, it achieved a 2.02% accuracy increase on the PloBlogs dataset, surpassing the second-best model, GraphSage. On the Cora dataset, CAP attained a 1.31% accuracy gain, demonstrating its capacity to handle diverse data scenarios effectively. CAP's performance on the CiteSeer dataset is particularly remarkable, with a significant 3.86% accuracy improvement compared to the GCN backbone and a 1.9% improvement over the second-best model, GPPT. Furthermore, CAP consistently demonstrated competitive or superior results on other datasets, including CS, PubMed, Physics, and Flickr, establishing its robustness across a range of node classification tasks. The CAP variants, CAP (GCN) and CAP (GAT) enhanced the performance of their respective backbone architectures (GCN and GAT) by considerable margins. Specifically, compared to its backbone, CAP (GCN) exhibited enhancements of 3.21%, 1.49%, 1.2%, and 3.9% on the CS, PubMed, Physics, and Flickr datasets, respectively. These advancements highlight the

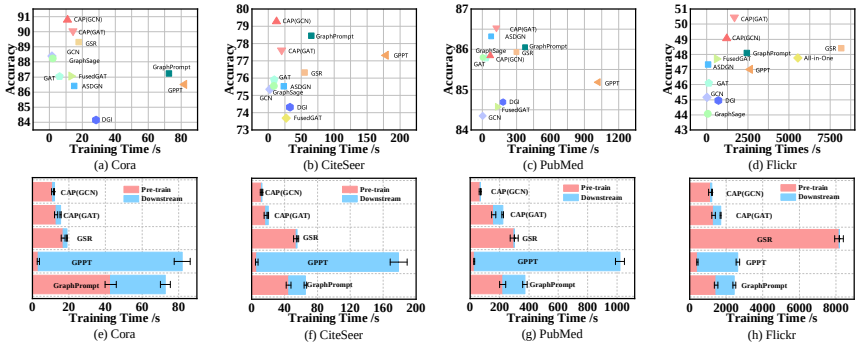


Figure 3: The overall training time and accuracy of different models.

value of integrating community information into graph neural networks, showcasing CAP as a promising model for achieving exceptional node classification results.

5.2.2. Training Time Analysis

Fig. 3 (a)-(d) compares the training time and accuracy of CAP against baseline models on the Cora, CiteSeer, PubMed, and Flickr datasets. Ideally, a model should minimize time consumption while maximizing accuracy, placing it in the upper left quadrant of each plot. Both CAP (GCN) and CAP (GAT) achieve superior results on the Cora and CiteSeer datasets, incurring a limited additional time cost compared to highly efficient models like GCN, GAT, and GraphSage while yielding accuracy improvements of up to 4%. On Flickr, both CAP (GCN) and CAP (GAT) outperform the comparatively strong ASDGN and FusedGAT baselines and significantly exceed the performance of GCN and GAT. Overall, CAP achieves substantial accuracy gains without a proportionally significant increase in training time compared to models trained directly.

Fig. 3 (e)-(h) presents the total training time (including pre-training and downstream task training) for pre-trained models GPPT, GSR, and CAP. In the academic domain, Cora showcases the efficiency of CAP (GCN) with a training time of 11.4 seconds, surpassing both non-pretrained models such as DGI (18.31 seconds) and FusedGAT (13.14 seconds), as well as pretrained models like GPPT (81.95 seconds) and GSR (17.85 seconds). Similarly, in CiteSeer, CAP (GCN) exhibits a notable decrease in training time to 12.52 seconds, in contrast to GPPT's 178.7 seconds (a 14.27×

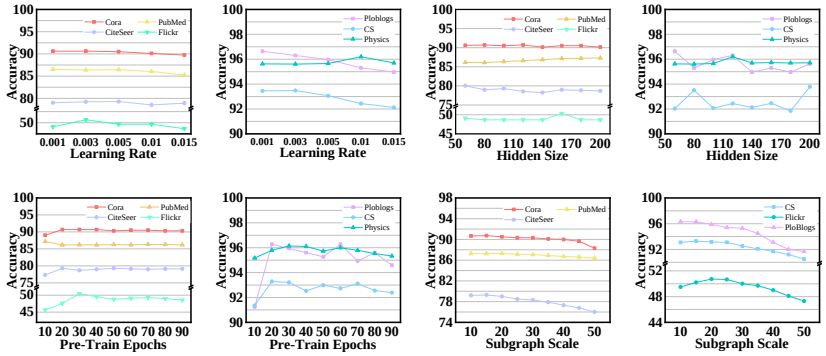


Figure 4: Parameter sensitivity analysis of learning rate, hidden size, pre-train epochs, and subgraph size.

speedup) and GSR's 55.1 seconds (a 4.4× speedup). Moreover, in PubMed, CAP (GCN) achieves a training time of 68.63 seconds, showcasing a significant enhancement over GPPT (1020.75 seconds, a 14.87× speedup) and GSR (301.23 seconds, a 4.38× speedup). In the context of Flickr, CAP (GCN) further underscores its efficiency, demonstrating training times that are 2.17× and 6.75× faster than GPPT and GSR, respectively.

Consistently, CAP (GCN) exhibits faster training than CAP (GAT), reflecting the computational differences in their backbone architectures. The All-in-One model, due to its fixed and extensive pre-training dataset, incurs significantly longer pre-training times compared to other methods, often exceeding them by an order of magnitude on the Cora, CiteSeer, and PubMed datasets.

As shown in Fig. 3 (e)-(h), pre-trained models like GSR, GPPT, and GraphPrompt often sacrifice efficiency for accuracy. For instance, GPPT's training time on Cora, CiteSeer, and PubMed is, on average, three to four times higher than on other datasets. GSR's time consumption notably escalates on Flickr, exceeding 7500 seconds. In contrast, CAP's community-aware pre-training strategy effectively reduces pre-training time, as demonstrated by its consistently lower overall training times. The figure also highlights that, for CAP (GCN), CAP (GAT), GSR, and GraphPrompt, pre-training dominates the overall training time. In contrast, GPPT's downstream task training constitutes the majority of its total training time.

5.2.3. Parameter Sensitivity

We investigate the impact of the learning rate, hidden size, pre-train epochs, and subgraph size on CAP's performance, as shown in Fig. 4.

Learning Rate: We evaluated CAP's performance across five learning rates. On most datasets, CAP's performance declined as the learning rate increased, except for Flickr (improved from 0.001 to 0.003) and Physics (improved from 0.005 to 0.01). Thus, we set the final learning rate to 0.001.

Hidden Layer Size: CAP's performance demonstrates low sensitivity to the hidden layer size on Cora, PubMed, CiteSeer, Flickr, and Physics. In contrast, PloBlogs and CS show significant fluctuations. PloBlogs favors smaller hidden layers, while CS achieves peak performance with 200 hidden layers.

Pre-training Epochs: Most datasets (Cora, CiteSeer, CS, Flickr, Physics, and PloBlogs) exhibit a similar trend: performance initially improves with increasing pre-training epochs, then plateaus or shows minor fluctuations. Peak performance generally reaches 20 epochs (Cora, CiteSeer, CS, PloBlogs) or 30 epochs (Flickr, Physics). PubMed, however, achieves its highest accuracy at 10 epochs, with performance remaining stable thereafter. These findings suggest that a relatively small number of pre-training epochs is sufficient for optimal performance.

Subgraph Scale: We compared CAP's performance across different subgraph scales. Most methods peaked at a scale of 15, except Flickr, which peaked at 20, followed by performance degradation with larger scales. Larger subgraphs introduce noise in contrastive learning by increasing false positives, while overly small subgraphs hinder sufficient community information extraction. Hence, we set the subgraph scale to 15 in experiments.

5.3. Effectiveness Exploration

5.3.1. Confidence Analysis

Confidence analysis is crucial in machine learning as it assesses a model's certainty in its predictions, providing insights beyond predictive performance and guiding decision-making. Higher confidence generally implies greater reliability, while a concentrated confidence distribution indicates consistent certainty, contributing to model stability.

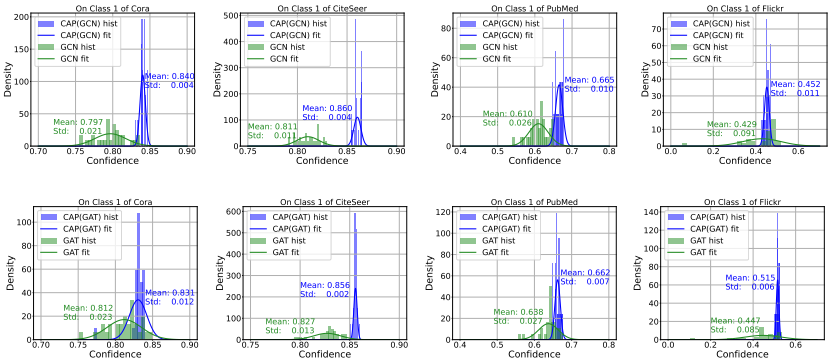


Figure 5: The confidence distribution of the model over 30 training rounds.

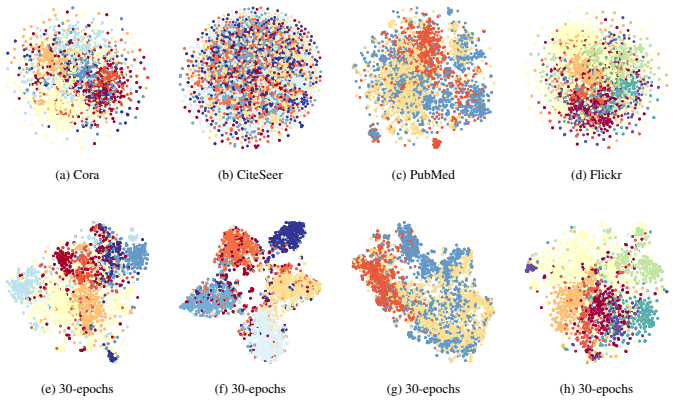
Fig. 5 illustrates the confidence distributions for a specific category after 30 rounds of training. Notably, CAP (GCN) and CAP (GAT) exhibit denser and higher mean confidence distributions compared to their backbone GNNs, GCN and GAT. The sharper curves of CAP (GCN) and CAP (GAT) indicate a more concentrated distribution, reflecting higher confidence and lower variance in predictions. In contrast, the flatter curves for GCN and GAT suggest greater uncertainty. This difference stems from CAP's community-aware pre-training strategy, which leverages community information to learn more discriminative and robust model parameters.

Quantitatively, CAP (GCN) demonstrates significant improvements. On Cora, the mean confidence increases from 0.797 (GCN) to 0.840, while the standard deviation decreases from 0.021 to 0.004. Similar trends are observed on CiteSeer and PubMed, with 3% to 5% improvements in mean confidence and approximately 1% reductions in standard deviation. Flickr exhibits a particularly pronounced decrease in standard deviation, reaching 8%.

These results highlight the effectiveness of CAP in achieving higher and more stable confidence in classification outcomes compared to training with randomly initialized parameters.

5.3.2. Pre-Training Effectiveness

To visually analyze the impact of pre-training on node representations, we employ t-SNE to visualize node representations before and after pre-training on Cora, CiteSeer,



(a) Cora (b) CiteSeer (c) PubMed (d) Flickr

(e) 30-epochs (f) 30-epochs (g) 30-epochs (h) 30-epochs

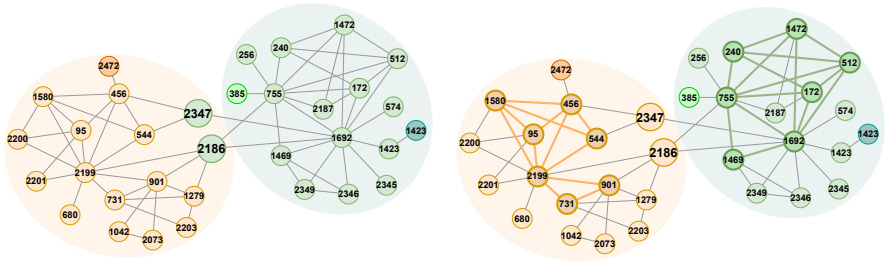
Figure 6: Visualization of node representations before and after 30-epochs pre-training

Table 2: Statistical metrics for embeddings on CiteSeer.

Metric	Silhouette	Calinski-Harabasz
Cora before training	0.06	51.64
Cora after 30-epochs pre-training	**0.19**	**112.85**
CiteSeer before training	-0.06	91.45
After 30-epochs pre-training	**0.13**	**445.13**
PubMed before training	-0.01	116.71
After 30-epochs pre-training	**0.07**	**277.52**
Flickr before training	0.03	61.27
After 30-epochs pre-training	**0.17**	**158.23**

PubMed, and Flickr. Fig. 6 depicts these visualization results. For clarity, we randomly sampled 3000 nodes from the PubMed and Flickr datasets. To qualify the visualization effectiveness of different methods, classical clustering evaluation indicators Silhouette score and Calinski-Harabasz score are calculated, with higher values indicating better outcomes, as shown in Table 2.

Visual inspection reveals a striking clustering effect after just 30 epochs of pre-training, particularly evident in the CiteSeer dataset. Nodes are distinctly grouped into five clusters, corroborated by substantial improvements in both the Silhouette score (from -0.06 to 0.13) and the Calinski-Harabasz index (from 91.45 to 445.13). Similar clustering patterns, accompanied by varying degrees of improvement in the evaluation metrics, are observed in the other datasets. These findings indicate that even a limited number of pre-training epochs can significantly enhance the grouping and discriminative capacity of node representations.



(a) GCN (b) CAP (GCN)

Figure 7: Node classification predictions on the Cora dataset using GCN and CAP (GCN). Orange and green-shaded regions represent two distinct communities. Nodes are color-coded according to their predicted labels. The deepened node and edge representations in CAP (GCN) highlight the densely connected subgraphs identified by the community sampler, enabling accurate prediction of node labels for nodes 2347 and 2186.

Table 3: Performance with Different Community Detection Methods. $|C_{\text{list}}|$ the number of detected communities.

Datasets	PloBlogs	Cora	CiteSeer	CS	PubMed	Physics	Flickr
CAP (Louvain)	95.97±0.36	**90.79**±0.37	**79.28**±0.39	85.43±1.73	85.01±0.91	82.91±1.27	42.33±0.83
$\lvert C_{\text{list}}\rvert$	277	117	468	26	40	25	26
CAP (LPA)	**96.31**±0.29	90.61±0.65	79.21±0.57	**93.29**±0.53	**85.84**±0.22	**96.33**±0.63	**49.06**±0.41
$\lvert C_{\text{list}}\rvert$	278	115	451	1281	1740	1367	61

5.3.3. Case Study

To illustrate the predictive capabilities of CAP, we applied it to the Cora dataset [40] and visualized the prediction results. As shown in Fig. 7, while GCN incorrectly classifies nodes 2347 and 2186 located at the boundary of two communities, CAP accurately predicts their labels. This misclassification by GCN stems from the strong influence of node 1692, despite nodes 2347 and 2186 being structurally part of the yellow community. CAP overcomes this limitation through its community-aware pre-training process. By incorporating community structure into the learning process, CAP enables the model to recognize the communal context of nodes, significantly improving the accuracy of downstream classification tasks.

The pre-training task of CAP is designed to instill in the model an awareness of the community distribution of nodes. This community-centric learning approach enables the model to make predictions not only based on individual node features but also by considering the broader community-aware relationships. Consequently, CAP's

Table 4: Performance with or without Community Sampler (CAP - Sampler).

Datasets	PloBlogs	Cora	CiteSeer	CS	PubMed	Physics	Flickr
CAP (GCN)	**96.31**±0.29	**90.61**±0.65	**79.21**±0.57	**93.29**±0.53	**85.84**±0.22	**96.33**±0.63	**49.06**±0.41
CAP - Sampler	92.63±0.83	86.58±0.97	75.92±0.69	91.75±0.72	84.39±0.86	94.21±0.37	47.10±0.94

Table 5: Performance with Different Strategies of Community Sampler.

Datasets	PloBlogs	Cora	CiteSeer	CS	PubMed	Physics	Flickr
Sampler (base)	**96.39**±0.49	**90.79**±0.37	**79.28**±0.39	92.71±0.73	**84.99**±0.76	95.97±0.27	**49.73**±0.31
Time	16.93s	37.64s	39.68s	913.75s	589.11s	3136.51s	5813.24s
Sampler (MCTS)	96.31±0.29	90.61±0.65	79.21±0.57	**93.29**±0.53	85.84±0.22	**96.33**±0.63	49.06±0.41
Time	9.64s	25.32s	30.13s	297.51s	129.93s	835.41s	1427.31s
Speed Up	1.76×	1.49×	1.32×	3.07×	4.53×	3.75×	4.07×

pre-training facilitates a more accurate and insightful classification. This improvement highlights the potential of incorporating community information in graph neural networks and demonstrates the effectiveness of CAP in leveraging such information for enhanced predictive performance.

5.4. Ablation Study

Fine-grained community partitioning yields better performance. To evaluate the impact of different community partitioning methods on CAP's performance, we conducted a comparative analysis using Louvain and LPA algorithms (Table 3). Specifically, CAP (Louvain) slightly outperforms CAP (LPA) on Cora and CiteSeer. However, significant performance differences emerge across CS, PubMed, Physics, and Flickr datasets, attributable to the disparity in community detection results. For instance, on the CS dataset, Louvain identifies 26 communities compared to LPA's 1281, resulting in a community size ratio of 0.020. Similar discrepancies are observed in PubMed (ratio: 0.023) and Physics (ratio: 0.068). This substantial divergence in community granularity directly influences the performance of the two methods. Notably, CAP (LPA) achieves significant improvements over CAP (Louvain) on CS (7.83%), PubMed (0.83%), Physics (13.42%), and Flickr (6.73%), highlighting the importance of finer-grained community structures for optimal CAP performance.

Community sampling facilitates the provision of more representative samples for contrastive learning. To investigate the impact of community sampling on CAP's

performance, we compared its performance with and without the sampler component in Table 4. *CAP - Sampler* implies that during the contrastive learning process, sampled pairs are selected directly from the communities, rather than the subgraphs obtained from Sampler. This approach broadens the potential selection range for sample pairs. Removing the community sampler from CAP leads to degraded performance across all datasets. We observe performance drops of 3.68%, 4.03%, 3.29%, and 2.12% on PloBlogs, Cora, CiteSeer, and Physics, respectively. This decline is particularly prominent for CAP (GCN), which not only exhibits substantial performance deterioration across all datasets but also falls short of a standard GCN in certain cases. These results underscore the presence of significant noise within communities, which, if not effectively mitigated by the sampler,

Approximate subgraphs of communities achieve comparable performance. We compared the performance and time consumption of two community sampling strategies *Samplers* including the basic strategy and an approximate subgraph sampling method based on MCTS. Table 5 reveals that both methods demonstrate comparable performance with a negligible maximum gap of 1%. The basic strategy exhibited minor advantages on smaller datasets like Ploblogs, Cora, and CiteSeer. MCTS outperforms the base sampler on larger networks such as CS, Physics, and Flickr datasets. Notably, MCTS significantly reduces time consumption compared to the basic strategy and achieves reductions of 40%, 33%, and 25% for PloBlogs, Cora, and CiteSeer, respectively. The speeds are even more accelerated for PubMed, CS, Physics, and Flickr.

Table 6: Different Community Based Pre-Training Strategies.

Datasets	Cora	CiteSeer	PubMed	Flickr
CAP (GCN)	**90.79**±0.37	**79.28**±0.39	**85.84**±0.22	**49.06**±0.41
CAP-L	83.31±4.31	63.71±11.34	72.90±7.17	39.33±5.67
CAP-E	85.31±2.17	72.81±2.93	83.41±1.47	42.14±1.70

Contrastive learning based on communities proves more effective than community labels or embeddings. To assess the effectiveness of our proposed contrastive pre-training strategy, we compared CAP with two alternative methods: CAP-L and CAP-E (Table 6). CAP-L directly utilizes community labels as targets for GCN pre-

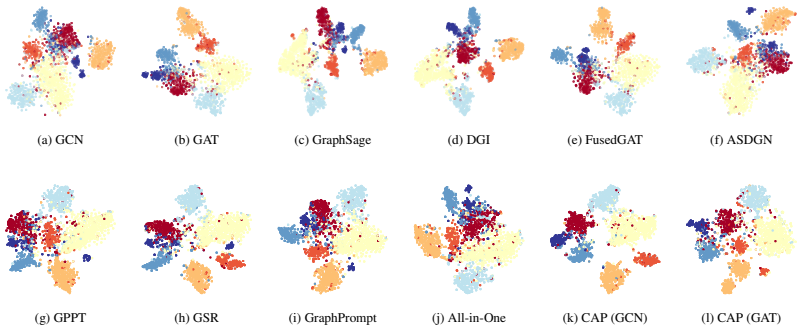


Figure 8: Visualization of node representations on Cora.

training, while CAP-E employs one-hot encoded community representations concatenated with node embeddings. Both CAP-L and CAP-E exhibit performance degradation compared to the baseline GCN. This decline is particularly pronounced in CAP-L, with performance drops of 7.48%, 15.57%, 12.94%, and 9.73% on the four datasets, respectively, suggesting that directly using community labels for pre-training is ineffective. This disparity likely stems from the mismatch between the pre-training task (community classification) and the downstream task (node classification), rendering the learned parameters suboptimal for the latter. While less severe, CAP-E also suffers from performance degradation, potentially due to the high dimensionality of one-hot community representations, which may overshadow node-specific information. In contrast, CAP leverages contrastive learning and community sampling to effectively capture and distill community information while mitigating noise, ultimately enhancing GNN performance on downstream tasks.

5.5. Visualization

To visually assess the quality of learned node representations, we employ t-SNE [45] to visualize node representations on Cora and explore the differences between the CAP and the baseline variants. Fig. 8 depicts these visualization results. The untrained Cora dataset nodes are dispersed and unordered. Training with GCN and GAT leads to the emergence of seven discernible clusters, although nodes belonging to the red and blue classes remain poorly separated. GraphSage, DGI, FusedGAT, ASDGN, and GPPT achieve improved clustering, yet some deep blue nodes still lack clear

separation. GSR, conversely, results in a more dispersed node distribution. Notably, CAP (GCN) demonstrates superior performance, generating well-defined clusters for all seven classes with clear inter-class boundaries and compact intra-class representations.

6. Conclusion

In this study, we introduce CAP, a novel graph pre-training framework that advances upon Professor Edwin R. Hancock's foundational work by addressing key limitations of existing approaches. Unlike current methods that require computationally intensive tasks or large datasets for marginal gains, CAP innovatively combines community-aware self-supervised learning with a noise-reducing community sampler and MCTS to enhance both node representations and computational efficiency. Extensive experiments demonstrate CAP's state-of-the-art performance across diverse datasets with significantly faster training, while ablation studies validate its community-aware pertaining mechanism. While CAP shows promising results, its current implementation faces limitations when handling ultra-large-scale graphs due to constraints in both the community sampling methodology and backbone architecture, suggesting important directions for future improvement.

7. Acknowledgements

This work was supported by the National Natural Science Foundation of China (Grant No. 71971002), the Major Project of Scientific Research in Higher Education Institutions in Anhui Province (Grant No. 2024AH040011), and the Anhui Postdoctoral Scientific Research Program Foundation (Grant No. 2024B828).

References

[1] Rongji Ye, Lixin Cui, Luca Rossi, Yue Wang, Zhuo Xu, Lu Bai, and Edwin R Hancock. C2n-abdp: Cluster-to-node attention-based differentiable pooling. In *International Workshop on Graph-Based Representations in Pattern Recognition*, pages 70–80, 2023.

[2] Dongdong Chen, Yuxing Dai, Lichi Zhang, Zhihong Zhang, and Edwin R Hancock. Position-aware and structure embedding networks for deep graph matching. *Pattern Recognition*, 136:109242, 2023.

[3] Kihyuk Sohn, Huiwen Chang, José Lezama, Luisa Polania, Han Zhang, Yuan Hao, Irfan Essa, and Lu Jiang. Visual prompt tuning for generative transfer learning. In *CVPR*, pages 19840–19851, 2023.

[4] Alec Radford and Karthik Narasimhan. Improving language understanding by generative pre-training. 2018.

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

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

[7] Yuning You, Tianlong Chen, Yongduo Sui, Ting Chen, Zhangyang Wang, and Yang Shen. Graph contrastive learning with augmentations. In *NeurIPS*, page 5812–5823, 2020.

[8] Jun Xia, Chengshuai Zhao, Bozhen Hu, Zhangyang Gao, Cheng Tan, Yue Liu, Siyuan Li, and Stan Z Li. Mole-bert: Rethinking pre-training graph neural networks for molecules. In *The Eleventh International Conference on Learning Representations*, 2022.

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

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

[11] Jianan Zhao, Qianlong Wen, Mingxuan Ju, Chuxu Zhang, and Yanfang Ye. Self-supervised graph structure refinement for graph neural networks. In *WSDM*, page 159–167, 2023.

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

[13] Xiangguo Sun, Hong Cheng, Jia Li, Bo Liu, and Jihong Guan. All in one: Multi-task prompting for graph neural networks. In *SIGKDD*, pages 2120–2131, 2023.

[14] Xing Su, Shan Xue, Fanzhen Liu, Jia Wu, Jian Yang, Chuan Zhou, Wenbin Hu, Cecile Paris, Surya Nepal, Di Jin, et al. A comprehensive survey on community detection with deep learning. *IEEE Transactions on Neural Networks and Learning Systems*, 2022.

[15] Mark E. J. Newman and Michelle Girvan. Finding and evaluating community structure in networks. *Physical Review. E, Statistical, Nonlinear, and Soft Matter Physics*, 69:026113, 2003.

[16] Jaewon Yang, Julian McAuley, and Jure Leskovec. Community detection in networks with node attributes. In *ICDM*, pages 1151–1156, 2013.

[17] Bing-Bing Xiang, Zhong-Kui Bao, Chuang Ma, Xingyi Zhang, Han-Shuang Chen, and Hai-Feng Zhang. A unified method of detecting core-periphery structure and community structure in networks. *Chaos: An Interdisciplinary Journal of Nonlinear Science*, 28(1), 2018.

[18] Yanbei Liu, Qi Wang, Xiao Wang, Fang Zhang, Lei Geng, Jun Wu, and Zhitao Xiao. Community enhanced graph convolutional networks. *Pattern Recognition Letters*, 138:462–468, 2020.

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

[20] Petar Velickovic, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, Yoshua Bengio, et al. Graph attention networks. In *ICLR*, 2018.

[21] Will Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. In *NeurIPS*, page 1025–1035, 2017.

[22] Hengrui Zhang, Zhongming Yu, Guohao Dai, Guyue Huang, Yufei Ding, Yuan Xie, and Yu Wang. Understanding gnn computational graph: A coordinated computation, io, and memory perspective. *Proceedings of Machine Learning and Systems*, 4:467–484, 2022.

[23] Alessio Gravina, Davide Bacciu, and Claudio Gallicchio. Anti-symmetric DGN: a stable architecture for deep graph networks. In *ICLR*, 2023.

[24] Zhenhua Huang, Wenhao Zhou, Kunhao Li, and Zhaohong Jia. Sgcn: A scalable graph convolutional network with graph-shaped kernels and multi-channels. *Knowledge-Based Systems*, 279:110923, 2023.

[25] Junfu Wang, Yuanfang Guo, Liang Yang, and Yunhong Wang. Heterophily-aware graph attention network. *Pattern Recognition*, 156:110738, 2024.

[26] Zhenhua Huang, Wenhao Zhou, Yufeng Li, Xiuyang Wu, Chengpei Xu, Junfeng Fang, Zhaohong Jia, Linyuan Lü, and Feng Xia. Sehg: Bridging interpretability and prediction in self-explainable heterogeneous graph neural networks. In *ACM on Web Conference*, pages 1292–1304, 2025.

[27] Zhenhua Huang, Kunhao Li, Wang Shaojie, Zhaohong Jia, Wentao Zhu, and Sharad Mehrotra. Ses: Bridging the gap between explainability and prediction of graph neural networks. In *ICDE*, 2024.

[28] Lixiang Xu, Jiawang Peng, Xiaoyi Jiang, Enhong Chen, and Bin Luo. Graph neural network based on graph kernel: A survey. *Pattern Recognition*, page 111307, 2024.

[29] Muhammad Aqib Javed, Muhammad Shahzad Younis, Siddique Latif, Junaid Qadir, and Adeel Baig. Community detection in networks: A multidisciplinary review. *Journal of Network and Computer Applications*, 108(C):87–111, 2018.

[30] Vincent D Blondel, Jean-Loup Guillaume, Renaud Lambiotte, and Etienne Lefebvre. Fast unfolding of communities in large networks. *Journal of Statistical Mechanics: Theory and Experiment*, 2008(10):P10008, 2008.

[31] Martin Rosvall and Carl T. Bergstrom. Maps of random walks on complex networks reveal community structure. In *PANS*, volume 105, pages 1118 – 1123, 2007.

[32] Yoshua Bengio, Olivier Delalleau, and Nicolas Le Roux. Label propagation and quadratic criterion. *Semi-Supervised Learning*, pages 193–216, 2006.

[33] Jianbo Shi and J. Malik. Normalized cuts and image segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(8):888–905, 2000.

[34] Xin Xin, Chaokun Wang, Xiang Ying, and Boyang Wang. Deep community detection in topologically incomplete networks. *Physica A: Statistical Mechanics and its Applications*, 469:342–352, 2017.

[35] Fanghua Ye, Chuan Chen, and Zibin Zheng. Deep autoencoder-like nonnegative matrix factorization for community detection. In *CIKM*, page 1393–1402, 2018.

[36] Xingtong Yu, Yuan Fang, Zemin Liu, and Xinming Zhang. Hgprompt: Bridging homogeneous and heterogeneous graphs for few-shot prompt learning. In *AAAI*, volume 38, pages 16578–16586, 2024.

[37] Peng Qin, Yaochun Lu, Weifu Chen, Defang Li, and Guocan Feng. Aagcn: An adaptive data augmentation for graph contrastive learning. *Pattern Recognition*, 163:111471, 2025.

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

[39] Lada A Adamic and Natalie Glance. The political blogosphere and the 2004 us election: divided they blog. In *Workshop on LinkKDD*, pages 36–43, 2005.

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

[41] Oleksandr Shchur, Maximilian Mumme, Aleksandar Bojchevski, and Stephan Günnemann. Pitfalls of graph neural network evaluation. In *Workshop on NeurIPS*, 2018.

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

[43] Petar Veličković, William Fedus, William L Hamilton, Pietro Liò, Yoshua Bengio, and R Devon Hjelm. Deep graph infomax. In *ICLR*, 2019.

[44] Kezhao Huang, Jidong Zhai, Zhen Zheng, Youngmin Yi, and Xipeng Shen. Understanding and bridging the gaps in current gnn performance optimizations. In *PPoPP*, pages 119–132, 2021.

[45] Van Der Maaten Laurens and Geoffrey Hinton. Visualizing data using t-sne. *Journal of Machine Learning Research*, pages 2579–2605, 2008.