CacheGNN: Enhancing Graph Neural Networks with Global Information Caching

Anonymous authors
Paper under double-blind review

Abstract

Graph neural networks (GNNs) have achieved impressive results on various graph learning tasks. Most GNNs merely leverage information from a limited range of local neighbors, which is difficult to effectively capture global information in the graph. However, utilising global information enables GNNs to capture long-range dependencies and learn more informative node representations. To this end, we propose CacheGNN, an approach that leverages information from global similar nodes to enhance GNNs. Our CacheGNN uses a cache to store node representations and utilises those cached embeddings to efficiently find global similar nodes. To quickly and efficiently making predictions at test time, our CacheGNN retrieves global similar nodes from a set of representative nodes, which is selected from a sparse node selection distribution with Dirichlet prior. We conduct node classification experiments on seven real-world datasets under inductive and transductive settings. Experimental results verify the effectiveness of our CacheGNN.

1 Introduction

Graph Neural Networks (GNNs) (Kipf & Welling, 2017; Hamilton et al., 2017; Veličković et al., 2018) have been receiving growing popularity due to their effectiveness in a wide spectrum of graph learning tasks (Xu et al., 2018a; Liu et al., 2020). The essential idea of GNNs is to learn informative node representations through a message passing scheme, which recursively aggregates information from local neighbors. Due to the useful inductive bias and powerful neural structure, GNNs have achieved impressive results on many real-world datasets (Tang et al., 2008; Hamilton et al., 2017).

However, existing GNNs suffer from the following two defects. Firstly, most GNNs learn node embeddings by leveraging information from a limited range of local neighbors, which limits their ability to explicilty and effectively leverage the *global information*, i.e., features, structural patterns and label information in the entire graph (Bodnar et al., 2021; Bouritsas et al., 2022; Wu et al., 2021). This drawback hinders GNNs from capturing the long-range dependencies among nodes (Wu et al., 2021) and learning more informative node representations (Tang et al., 2015; Zhuang & Ma, 2018), especially for nodes with sparse local connections (Gasteiger et al., 2018). Secondly, there are a few works that try to leverage the global structural information in the graph to improve the performance of GNNs. For instance, DGCN (Zhuang & Ma, 2018) utilises positive pointwise mutual information matrix to preserve the global consistency of graph structure, and PPNP Gasteiger et al. (2018) combines GNNs with personalized PageRank to capture global structural information. However, since these methods require acknowledging the structural information of the whole graph in advance, they are difficult to be applied to the inductive graph learning setting, where predictions should be quickly generated for unseen nodes in an efficient way Hamilton et al. (2017).

To address the aforementioned defects, we propose CacheGNN, an approach to enhance existing GNNs such that they can capture and utilise the global information in an efficient way. To utilise the global information, inspired by the graph attention network Veličković et al. (2018) that focuses on the similar nodes in local neighbors for learning informative representations, we generalize the scope of similar nodes to the whole graph and enhance GNNs by utilising the information of *global similar nodes* which is defined as nodes in the entire graph that have similar features and geometric structures. Therefore, compared with vanilla GNNs, CacheGNN can additionally leverage the information, i.e., features, structural patterns and label information, of global similar nodes to make more accurate predictions. Moreover, CacheGNN uses the similarities between node representations output by GNNs to find the global similar nodes, without introducing additional modules for finding global similar nodes, which can be applied to a variety of GNNs. However, finding global similar nodes in such a way requires calculating representations of all nodes in the graph using GNNs, which is computationally expensive both in space and time Fey et al. (2021). To alleviate the heavy burden of computing all node representations, CacheGNN uses a cache to store the representations of all nodes and leverages the cached node embeddings to find the global similar nodes.

Since CacheGNN only requires finding global similar nodes using cache to enhance GNNs, it can be easily applied to both inductive and transductive graph learning settings. However, one downside of CacheGNN is that we need to maintain a large embedding pool for large-scale graphs, leading to high memory cost and low speed efficiency at test time. Therefore, in order to quickly and efficiently making predictions, CacheGNN retrieves global similar nodes from a set of representative nodes, i.e., a subset of training nodes that are more likely to be selected as global similar nodes, and only stores embeddings of representative nodes in the cache for prediction. Those representative nodes are selected based on a distribution over training nodes, which itself is sampled from a Dirichlet prior. The use of Dirichlet prior with appropriate hyperparameters is able to encourage a sparse node selection distribution (He et al., 2020), allowing us to select as few representative nodes as possible, which could reduce the cache size and improve efficiency of our model.

Our contributions can be summarized as follows: (1) We leverage the information, i.e., features, structural patterns and label information, of global similar nodes to enhance existing GNNs. (2) We propose a cache-based approach to efficiently retrieve the global similar nodes from the entire graph. (3) To further improve the prediction efficiency of our approach on large-scale graphs, we leverage Dirichlet prior to learn a sparse distribution over training nodes, based on which we select a set of representative nodes and their cached embeddings for prediction. (4) Experimental results on seven real-world datasets show that GNNs with our approach, i.e., CacheGNN, can outperform vanilla GNNs on node classification task under both inductive and transductive settings.

2 Related Work

Graph Neural Networks. Recently, GNNs are widely used to handle graph-structured data. They have achieved remarkable success a wide spectrum of real-world applications, ranging from recommendation systems (Yang et al., 2020; Wang et al., 2021; Cai et al., 2022), knowledge graph reasoning (Zhang et al., 2019; Chen et al., 2022; Guo et al., 2022), medical property prediction (Wishart et al., 2018; Szklarczyk et al., 2019; Gasteiger et al., 2021), natural language processing (Yao et al., 2019; Hao et al., 2021). The current de facto design of GNNs follows the message passing framework (Kipf & Welling, 2017; Gilmer et al., 2017; Hamilton et al., 2017; Veličković et al., 2018; Yang et al., 2022), where they learn node representations by aggregating information from local neighbors. Due to the over-smoothing problem Li et al. (2018); Oono & Suzuki (2019); Zhang et al. (2021), most GNNs are shallow for practical purpose. Some studies try to enlarge the respective filed of GNNs by developing deep GNNs (Liu et al., 2020; Zeng et al., 2021). For instance, the JKnet (Xu et al., 2018b) and GCNII (Chen et al., 2020) combine residual connections with aggregation scheme to increase the number of layers in GNNs. However, both shallow and deep GNNs only have access to local information, difficult to leverage the global information in the graph, while our CacheGNN can effectively utilise the global information by retrieving global similar nodes.

There are some studies (Zhuang & Ma, 2018; Gasteiger et al., 2018; Abu-El-Haija et al., 2019; Wu et al., 2021) that leverage the global information of the graph as well. For example, DGCN (Zhuang & Ma, 2018) uses a positive pointwise mutual information matrix to preserve the global consistency of graph structure, and PPNP Gasteiger et al. (2018) combines GNNs with personalized PageRank to develop an improved propagation scheme. However, they requires knowing the whole graph structure in advance and cannot be applied to the inductive learning setting, while CacheGNN do not have such requirement and can be applied to both inductive and transductive learning settings.

Cached-Based Graph Learning. There are some works that leverage cache in GNNs as well. For example, VR-GCN (Chen et al., 2018) utilises cache to store historic node embeddings to reduce the variance of representations obtained by sampling neighboring nodes. GAS (Fey et al., 2021) proposes to incorporate historic embeddings to account for all available neighborhood information. GraphFM (Yu et al., 2022) proposes to use a momentum step to incorporate historic embeddings to avoid the neighborhood explosion problem (Hamilton et al., 2017). However, these methods require storing embeddings of all nodes in the graph, which suffers from high memory cost and low speed efficiency for large-scale graphs. In contrast, our CacheGNN only stores the most informative node embeddings in the cache, which are sampled from a Dirichlet prior, and therefore reduce the memory cost and improve the efficiency without compromising performance.

3 Problem Formulation

In this paper, we instantiate graph learning tasks with node classification problem, which is a fundamental problem in graph machine learning community (Hamilton et al., 2017; Qu et al., 2021). We formulate this problem under the inductive setting (Hamilton et al., 2017; Gao et al., 2018; Zeng et al., 2020; Chiang et al., 2019). However, our method can be also applied to the easier transductive setting. The inductive setting assumes that we are given a set of labeled nodes, and the goal is to predict the labels of unseen nodes, i.e., nodes that do not appear during training.

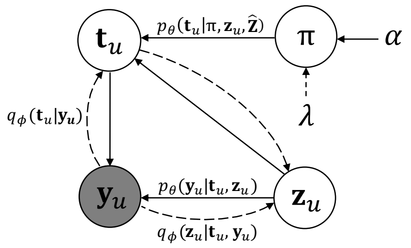
Formally, we denote the set of labeled nodes as a single training graph $\mathcal{G} = \{\mathcal{V}, \mathcal{E}, \mathbf{X}, \mathbf{Y}\}$, where $\mathcal{V}$ and $\mathcal{E}$ denote the set of nodes and edges, respectively. $\mathbf{X} \in \mathbb{R}^{N\times F}$ represents the feature matrix, and $\mathbf{Y} \in \{0,1\}^{N\times C}$ denotes the labels of training nodes. Give a test graph $\tilde{\mathcal{G}} = \{\tilde{\mathcal{V}}, \tilde{\mathcal{E}}, \tilde{\mathbf{X}}\}$, we aim to predict the labels of nodes $\tilde{\mathbf{Y}}$ in the test graph. Following (Qu et al., 2021), we employ a probabilistic framework to formalize the inductive node classification problem, which requires solving the following two problems: (1) **Training**: Learn model parameters θ by approximating the marginal distribution $p_\theta(\mathbf{Y} \mid \mathbf{X}, \mathcal{E})$ on training graph. (2) **Prediction**: Infer label distributions of unlabeled nodes in the test graph $\tilde{\mathcal{G}}$, i.e., $p_\theta(\tilde{\mathbf{Y}} \mid \tilde{\mathbf{X}}, \tilde{\mathcal{E}})$, based on the learned model parameters.

4 Methodology

In this section, we will first introduce the technical details of the proposed method, and then introduce the training and prediction process of our method.

4.1 Model Structure

Following previous works (Kipf & Welling, 2016; Hasanzadeh et al., 2019; Qu et al., 2021), we adopt the Bayesian formulation of GNNs, where we regard node representations as stochastic latent variables. Normally in such case, the label of node u, i.e., $\mathbf{y}_u$, depends only on its latent representation $\mathbf{z}_u$, which is limited in leveraging global information. However, leveraging the global information in full-graph is beneficial to capture the long-range dependencies among nodes (Wu et al., 2021), and learn more informative node representations (Tang et al., 2015; Zhuang & Ma, 2018), especially for nodes with sparse local connections (Gasteiger et al., 2018). To this end, we take inspiration from graph attention networks and propose to enhance GNNs with the information of *global similar nodes*, i.e., nodes in the whole graph that have similar node features and local geometric structure. Particularly, we assume that the label of node u depends not only on its representation, but also on the representations of its global similar nodes.



$$\begin{tikzcd}
\mathbf{t}_u \arrow[d] \arrow[rr, bend left, dashed] & & \pi \arrow[ll, "p_\theta(\mathbf{t}_u|\pi, \mathbf{z}_u, \hat{\mathbf{Z}})"'] & \alpha \arrow[l] \\
 & & \lambda \arrow[u, dashed] & \\
\mathbf{y}_u \arrow[u, bend left, dashed, "q_\phi(\mathbf{t}_u|\mathbf{y}_u)"] \arrow[rr, bend right, dashed, "q_\phi(\mathbf{z}_u|\mathbf{t}_u, \mathbf{y}_u)"'] & & \mathbf{z}_u \arrow[ll, "p_\theta(\mathbf{y}_u|\mathbf{t}_u, \mathbf{z}_u)"'] \arrow[uull] &
\end{tikzcd}$$

Figure 1: Probabilistic graphical model of our CacheGNN. Generative dependencies are shown in solid arrows, while inference dependencies are shown in dashed arrows.

Specifically, we denote the set of global similar nodes of u as a vector $\mathbf{t}_u \in \{0,1\}^N$, with $\mathbf{t}_{uv} = 1$ indicating that node v is a global similar node of u. Note that we only consider leveraging information from top-K similar nodes. Therefore, there are only K elements in $\mathbf{t}_u$ being 1 while others being 0. Similar to node representations, which are treated as latent variables, we also regard the similar node indicator $\mathbf{t}_u$ as stochastic latent variable. Moreover, those global similar nodes can be conveniently obtained based on the similarities between node representations output by GNNs. We use such method to find global similar nodes because the representations learned by GNNs have aggregated the local features and structural information of nodes, and therefore it is convenient and effective to leverage those representations to find nodes that are similar in features and structure.

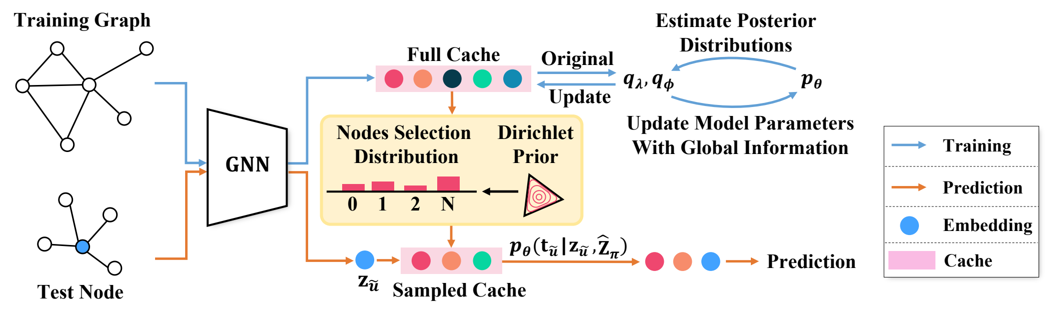

Training Graph
Test Node
GNN
Full Cache
Original
Update
Estimate Posterior Distributions
q_λ, q_ϕ
p_θ
Update Model Parameters With Global Information
Nodes Selection Distribution
Dirichlet Prior
0 1 2 N
$p_\theta(\mathbf{t}_{\tilde{u}}|\mathbf{z}_{\tilde{u}},\hat{\mathbf{Z}}_\pi)$
$\mathbf{z}_{\tilde{u}}$
Sampled Cache
Prediction
Training
Prediction
Embedding
Cache

Figure 2: Overview of the proposed CacheGNN model. In the training process, CacheGNN uses a variational EM algorithm to estimate the posterior distributions, i.e., q_λ and q_ϕ, and update the model parameters p_θ, as well as learning a sparse node selection distribution over training nodes. In the prediction process, based on the node selection distribution, CacheGNN only stores the most informative node embeddings in the cache, i.e., $\hat{\mathbf{Z}}_{\boldsymbol{\pi}}$, for prediction, and infers node labels based on the embeddings of nodes, i.e., $\mathbf{z}_{\tilde{u}}$, and the global similar nodes, i.e., $\mathbf{t}_{\tilde{u}}$.

Another benefit is that we do not introduce additional module for finding global similar nodes and it can be naturally applied to a variety of GNNs.

Unfortunately, finding global similar nodes using node representations requires calculating the representations of all nodes in the graph using GNNs, which is computationally expensive both in space and time for large-scale graphs Fey et al. (2021). In order to alleviate the intensive computation burden, we propose to cache previously calculated node embeddings and leverage the cached node embeddings to find global similar nodes. Specifically, the prior distribution of $\mathbf{t}_u$ is defined as $p_\theta(\mathbf{t}_u \mid \mathbf{z}_u, \hat{\mathbf{Z}})$, where $\hat{\mathbf{Z}}$ is the cached embeddings of all nodes. In such case, only the representation of node u, i.e., $\mathbf{z}_u$, is required to obtain the prior distribution, which is more efficient for training and prediction. Moreover, in order to accurately approximate node embeddings $\mathbf{Z}$ with the cached embedding table $\hat{\mathbf{Z}}$, we constantly update the cached embedding table at each training iteration.

One downside of the cache method is that we need to maintain a large embedding table for large-scale graphs, leading to significant high memory cost and low speed efficiency during the prediction stage. To further address this problem, we hypothesize that there exists some representative nodes in the training graph, and it is sufficient for our model to achieve satisfactory performance by only aggregating similar nodes from this representative node list. Experimental results in Section 6.3 verify our hypothesis that selecting similar nodes from a subset of representative nodes can achieve comparable results as selecting similar nodes from all training nodes. Since the representative nodes in the training graph are unknown in advance, we introduce a latent variable $\boldsymbol{\pi} \in [0,1]^N, s.t. \sum_{n=1}^{N} \boldsymbol{\pi}_n = 1$, with $\boldsymbol{\pi}_n$ representing the probability that the n-th node in the training set is a representative node. Since we aim to find a small set of representative nodes, we would like to model $p(\boldsymbol{\pi})$ as a sparse distribution, where the probability mass concentrates on only a few representative nodes. To achieve this purpose, we introduce a prior over $\boldsymbol{\pi}$, i.e., $p_\alpha(\boldsymbol{\pi})$, with parameters α, which is able to encourage a sparse probability distribution over $\boldsymbol{\pi}$.

Therefore, the marginal likelihood function of our model is defined as:

$$p_\theta(\mathbf{Y} \mid \mathbf{X}, \mathcal{E}, \hat{\mathbf{Z}}) = \int_{\boldsymbol{\pi}} p_\alpha(\boldsymbol{\pi}) \left[\int_{\mathbf{Z}} \left[\sum_{\mathbf{T}} p_\theta(\mathbf{Y}, \mathbf{T}, \mathbf{Z} \mid \boldsymbol{\pi}, \mathbf{X}, \mathcal{E}, \hat{\mathbf{Z}}) \right] \mathrm{d}\mathbf{Z} \right] \mathrm{d}\boldsymbol{\pi} \,, \tag{1}$$

where $\mathbf{T} = [\mathbf{t}_u]_{u\in\mathcal{V}}^\top$, and $\mathbf{Z} = [\mathbf{z}_u]_{u\in\mathcal{V}}^\top$ represent the global similar node sets and the representations of all nodes, respectively. The joint distribution of latent variables is defined as:

$$p_\theta(\mathbf{Y}, \mathbf{T}, \mathbf{Z} \mid \boldsymbol{\pi}, \mathbf{X}, \mathcal{E}, \hat{\mathbf{Z}}) = \prod_{u\in\mathcal{V}} p_\theta(\mathbf{z}_u \mid \mathbf{X}, \mathcal{E}) \, p_\theta(\mathbf{t}_u \mid \boldsymbol{\pi}, \mathbf{z}_u, \hat{\mathbf{Z}}) \, p_\theta(\mathbf{y}_u \mid \mathbf{t}_u, \mathbf{z}_u) \,. \tag{2}$$

The graphical model of the proposed method is depicted in Figure 1. Next we will introduce the instantiation of our probabilistic framework.

Prior distribution over $\boldsymbol{\pi}$. We employ Dirichlet distribution as the prior distribution over $\boldsymbol{\pi}$, given by $p_\alpha(\boldsymbol{\pi}) \propto \prod_{i=1}^{N} \boldsymbol{\pi}_i^{\alpha_i - 1}$, where α_i is the concentration parameter of the distribution. Since we do not have prior knowledge about representative nodes, we set all the concentration parameters as α, i.e., $\alpha_i = \alpha, \forall i = 1, \ldots, N$. The concentration parameter is a positive value and smaller α prefers a sparser distribution over $\boldsymbol{\pi}$ (He et al., 2020). Therefore, we use Dirichlet distribution as the prior distribution over $\boldsymbol{\pi}$ to encourage spare distribution of $\boldsymbol{\pi}$, and set $\alpha \leq 1$ in our experiments.

Prior distribution over node representations $\mathbf{Z}$. We model prior distribution over node representation as Gaussian distribution, and instantiate this prior distribution with GNN, which is defined as: $p_\theta(\mathbf{Z} \mid \mathbf{X}, \mathcal{E}) = \prod_{u \in \mathcal{V}} \mathcal{N}(\mathbf{z}_u \mid \mathrm{GNN}_\theta(\mathbf{X}, \mathcal{E}), \sigma_1^2 \mathbf{I})$, where σ_1^2 is the variance of prior distributions and GNN_θ is an L-layer GNN with parameter θ.

Prior distribution over $\mathbf{t}_u$. To obtain global similar nodes of node u, we define a prior distribution over the global similar node indicator as: $p_\theta(\mathbf{t}_u \mid \boldsymbol{\pi}, \mathbf{z}_u, \hat{\mathbf{Z}}) = \mathrm{Mul}(\mathbf{t}_u \mid K, f_\theta(\boldsymbol{\pi}, \mathbf{z}_u, \hat{\mathbf{Z}}))$, where $\mathrm{Mul}(\cdot)$ represents multinomial distribution, K is the predefined number of similar nodes, and f_θ is a function that outputs the parameters of the multinomial distribution. Let $\boldsymbol{\beta}_u$ represent the output of function f_θ, which is calculated as: $\boldsymbol{\beta}_u = \mathrm{Softmax}(\boldsymbol{\pi} \odot \boldsymbol{\pi}'_u)\,, \boldsymbol{\pi}'_{uv} = \frac{\varphi(\mathbf{z}_u, \hat{\mathbf{z}}_v)}{\sum_{v' \in \hat{\mathcal{V}}} \varphi(\mathbf{z}_u, \hat{\mathbf{z}}_{v'})}\,, \forall v \in \hat{\mathcal{V}}$, where $\varphi(\cdot, \cdot)$ is the inner product, $\boldsymbol{\pi}'_u = [\boldsymbol{\pi}'_{uv}]_{v \in \hat{\mathcal{V}}}$ is a vector containing the normalized similarities between node u and other nodes in the cache, i.e., $\hat{\mathcal{V}}$, $\odot$ is element-wise multiplication operation.

Prediction of labels. Finally, we need to instantiate the prediction probability $p_\theta(\mathbf{y}_u \mid \mathbf{z}_u, \mathbf{t}_u)$. In this paper, we define this distribution as the fusion of predicted label distribution of node u and the label distributions of its similar nodes $\mathbf{t}_u$. Specifically, we first predict the label of node u based on its representation: $p_{\mathrm{LC}}(\mathbf{y}_u \mid \mathbf{z}_u) = \mathrm{Softmax}(\mathbf{W}_c \mathbf{z}_u + \mathbf{b}_c)\,$, where $\mathbf{W}_c$ and $\mathbf{b}_c$ are the parameters of the linear classifier. Then, we predict the label of node u using the labels of its similar nodes: $p_{SIM}(\mathbf{y}_u \mid \mathbf{t}_u) \propto \sum_{v \in \hat{\mathcal{V}}} \mathbf{t}_{uv} \cdot \exp(\varphi(\mathbf{z}_u, \hat{\mathbf{z}}_v)) \cdot \mathbf{y}_v\,$, where $\mathbf{t}_{uv}$ is the v-th element of $\mathbf{t}_u$, $\mathbf{y}_v$ denotes the one-hot labels of node v. Finally, the predicted label distribution is defined as:

$$p_\theta(\mathbf{y}_u \mid \mathbf{z}_u, \mathbf{t}_u) = \eta\, p_{\mathrm{LC}}(\mathbf{y}_u \mid \mathbf{z}_u) + (1 - \eta)\, p_{SIM}(\mathbf{y}_u \mid \mathbf{t}_u)\,, \tag{3}$$

where $\eta \in [0, 1]$ is a trade-off hyperparameter. When $\eta = 1$, our model only uses local representations of nodes for prediction, which degrades to vanilla GNNs. In contrast, when $0 < \eta < 1$, our model predicts the labels of nodes by leveraging information from both local neighbors and global similar nodes, which is benefical for improving the performance.

4.2 Training

Since the marginal likelihood function of our model is intractable, we develop a variational Expectation-Maximization (EM) algorithm (Beal, 2003; Qu et al., 2019) to learn the model parameters θ. Following the principle of variational inference, we approximate the posterior distributions of latent variables with the following variational distributions [1]:

$$q_\phi(\mathbf{T}, \mathbf{Z}, \boldsymbol{\pi} \mid \mathbf{Y}) = \prod_{u \in \mathcal{V}} q_\phi(\mathbf{t}_u \mid \mathbf{Y}) q_\phi(\mathbf{z}_u \mid \mathbf{t}_u, \mathbf{Y}) q_\lambda(\boldsymbol{\pi})\,, \tag{4}$$

where ϕ and λ are variational parameters. We next introduce the details of variational distributions.

Variational distribution over $\mathbf{t}_u$. We assume variational distribution of $\mathbf{t}_u$ is a multinomial distribution, defined as: $q_\phi(\mathbf{t}_u \mid \mathbf{Y}) = \mathrm{Mul}(\mathbf{t}_u \mid K, g_\phi(\hat{\mathbf{Z}}, \mathbf{Y}))$, where g_ϕ is a function that outputs the parameters of the multinomial distribution. Let $\boldsymbol{\gamma}_u = g_\phi(\hat{\mathbf{Z}}, \mathbf{Y})$ denote the output of g_ϕ, which is calculated as: $\boldsymbol{\gamma}_u = \mathrm{Softmax}(s_\phi([\hat{\mathbf{z}}_u, \mathbf{y}_u], [\hat{\mathbf{Z}}, \mathbf{Y}]))$, where $[\cdot, \cdot]$ is the concatenation operation, $s_\phi(\cdot, \cdot)$ is a function that measures the similarities between node u and other nodes in the cache by leveraging both node representations and label information.

Variational distribution over $\mathbf{z}_u$. We assume the variational distribution over node representation is Gaussian distribution, defined as: $q_\phi(\mathbf{z}_u \mid \mathbf{t}_u, \mathbf{Y}) = \mathcal{N}(\mathbf{z}_u \mid \mathrm{GNN}_\theta(\mathbf{X}, \mathcal{E}) + \delta k_\phi(\mathbf{t}_u, \mathbf{Y}, \hat{\mathbf{Z}}), \sigma_2^2 \mathbf{I})\,$, where GNN_θ is the same GNN as in the prior distribution, δ is a hyperparameter and σ_2^2 is the variance of posterior distribution. k_ϕ is a defined as: $k_\phi(\mathbf{t}_u, \mathbf{Y}, \hat{\mathbf{Z}}) = \mathrm{Neural}(\sum_{v \in \hat{\mathcal{V}}} b_{uv} \cdot \hat{\mathbf{z}}_v)\,, b_{uv} = \frac{\mathbb{I}(\mathbf{y}_u, \mathbf{y}_v) \cdot \mathbf{t}_{uv}}{\sum_{v' \in \hat{\mathcal{V}}} \mathbb{I}(\mathbf{y}_u, \mathbf{y}_{v'}) \cdot \mathbf{t}_{uv'}}\,$, where $\mathrm{Neural}(\cdot)$ is a feed-forward neural network, $\mathbb{I}(\cdot, \cdot)$ is a scalar function

[1]We omit the dependence of variational distributions on node features $\mathbf{X}$ and edges $\mathcal{E}$ for brevity.

that measure the similarities between two labels, which is defined as:

$$\mathbb{I}(\mathbf{y}_u, \mathbf{y}_v) = \begin{cases} 1, & \text{if } \arg\max_{\mathbf{y}_u} = \arg\max_{\mathbf{y}_v}, \\ 0, & \text{if Otherwise }, \end{cases} \tag{5}$$

where $\arg\max_{\mathbf{y}_u}$ is the index of the largest element in $\mathbf{y}_u$.

Variational distribution over $\boldsymbol{\pi}$. We assume the variational distribution over $\boldsymbol{\pi}$ is a Dirichlet distribution: $q_\lambda(\boldsymbol{\pi}) \propto \prod_{i=1}^{N} \boldsymbol{\pi}_i^{\lambda_i - 1}$, where the variational parameter λ_i is defined as: $\lambda_i = \alpha + \sum_{u \in \mathcal{V}} K \boldsymbol{\gamma}_{ui}, \forall i = \{1, \ldots, N\}$, where $\boldsymbol{\gamma}_{ui}$ is the i-th element of $\boldsymbol{\gamma}_u$. However, it's computationally expensive to calculate λ_i at each iteration as it requires summing over all nodes. To enable mini-batch training, inspired by the the technique used in (He et al., 2020), we update λ_i as: $\lambda_i^{(t)} = (1 - \rho_t)\lambda_i^{(t-1)} + \rho_t\big(\alpha + \frac{N}{|\overline{\mathcal{V}}|} \sum_{u \in \overline{\mathcal{V}}} K \boldsymbol{\gamma}_{ui}\big), \rho_t = (t + \nu)^{-\xi}$, where $\overline{\mathcal{V}}$ is a small batch of training nodes, $|\overline{\mathcal{V}}|$ is the batch size, t is the iteration step, $\xi \in (0.5, 1]$ is the forgetting rate, and ν is a hyperparameter to down-weight early iterations.

With above variational distributions, we can derive the Evidence Lower Bound (ELBO) of the log-likelihood function of our model as:

$$\begin{aligned} \mathcal{L}_{\text{ELBO}} = &- D_{\text{KL}}[q_\lambda(\boldsymbol{\pi}) \,||\, p_\alpha(\boldsymbol{\pi})] + \sum_{u \in \mathcal{V}} \mathbb{E}_{q_\phi(\mathbf{t}_u \mid \mathbf{Y}) q_\phi(\mathbf{z}_u \mid \mathbf{t}_u, \mathbf{Y})} \big[\log p_\theta(\mathbf{y}_u \mid \mathbf{t}_u, \mathbf{z}_u)\big] \\ &- \mathbb{E}_{q_\phi(\mathbf{t}_u \mid \mathbf{Y})} \big[D_{\text{KL}}\big[q_\phi(\mathbf{z}_u \mid \mathbf{t}_u, \mathbf{Y}) \,||\, p_\theta(\mathbf{z}_u \mid \mathbf{X}, \mathcal{E})\big]\big] \\ &- \mathbb{E}_{q_\lambda(\boldsymbol{\pi}) q_\phi(\mathbf{z}_u \mid \mathbf{t}_u, \mathbf{Y})} \big[D_{\text{KL}}\big[q_\theta(\mathbf{t}_u \mid \mathbf{Y}) \,||\, p_\theta(\mathbf{t}_u \mid \boldsymbol{\pi}, \mathbf{z}_u, \hat{\mathbf{Z}})\big]\big]. \end{aligned} \tag{6}$$

We use variation EM algorithm to learn the parameters of our model, which alternatively learns the variational parameters ϕ and the model parameters θ through an E-step and M-step.

E-step. In E-step, the goal is to estimate the posterior distributions of latent variables. In such case, we fix the model parameters θ and update the variational distributions to approximate the true posteriors, which is achieved by minimizing the KL-divergence between the variational distribution and true posterior of latent variables, i.e., $D_{\text{KL}}\big[\log q_\phi(\mathbf{T}, \mathbf{Z}, \boldsymbol{\pi} \mid \mathbf{Y}) \,||\, p_\theta(\mathbf{T}, \mathbf{Z}, \boldsymbol{\pi} \mid \mathbf{Y}, \mathbf{X}, \mathcal{E})\big]$. According to the principle of variational inference, minimizing this KL divergence is equivalent to maximizing the ELBO $\mathcal{L}_{\text{ELBO}}$. Therefore, in E-step, we fix the model parameters θ and update variational parameters by maximizing $\mathcal{L}_{\text{ELBO}}$. After updating variational parameters, we update cache embeddings with variational distributions of node representations.

M-step. In M-step, the goal is to update the model parameters θ. In such case, we fix variational parameters ϕ and update the model parameters by maximizing the expectation of log-likelihood function with respective to variational distributions: $\mathbb{E}_{q_\phi(\mathbf{T}, \mathbf{Z}, \boldsymbol{\pi} \mid \mathbf{Y})}[\log p_\theta(\mathbf{Y}, \mathbf{T}, \mathbf{Z}, \boldsymbol{\pi} \mid \mathbf{X}, \mathcal{E})]$, where we sample from the variational distributions to approximate the expectation.

4.3 Prediction

We next introduce how to predict the labels of test nodes. Note that we introduce a latent variable $\boldsymbol{\pi}$ to select a subset of representative nodes to improve the prediction efficiency. After training, we expect to obtain a sparse distribution over $\boldsymbol{\pi}$, i.e., $q_\lambda(\boldsymbol{\pi})$, based on which we can select a subset of representative nodes and their cached embeddings. Specifically, we calculate the expected value of $q_\lambda(\boldsymbol{\pi})$, which is give by: $\mathbb{E}_{q_\lambda(\boldsymbol{\pi})}[\boldsymbol{\pi}_n] = \lambda_n / \sum_{j=1}^{N} \lambda_j, \forall n = 1, \ldots, N$, and then select the top-M nodes that occupy 90% of the probability mass as representative nodes. We represent the selected representative node set as $\mathcal{V}_{\boldsymbol{\pi}}$ and their corresponding cached embeddings as $\hat{\mathbf{Z}}_{\boldsymbol{\pi}}$. We then leverage the cached node embeddings $\hat{\mathbf{Z}}_{\boldsymbol{\pi}}$ and p_θ to predict the labels of test nodes. Specifically, the predicted label distribution of a test node $\tilde{u}$ is calculated as:

$$p(\mathbf{y}_{\tilde{u}} \mid \tilde{\mathbf{X}}, \tilde{\mathcal{E}}) = \int_{\mathbf{z}_{\tilde{u}}} p_\theta(\mathbf{z}_{\tilde{u}} \mid \tilde{\mathbf{X}}, \tilde{\mathcal{E}}) \Bigg[\sum_{\mathbf{t}_{\tilde{u}}} p_\theta(\mathbf{t}_{\tilde{u}} \mid \mathbf{z}_{\tilde{u}}, \hat{\mathbf{Z}}_{\boldsymbol{\pi}}) p_\theta(\mathbf{y}_{\tilde{u}} \mid \mathbf{t}_{\tilde{u}}, \mathbf{z}_{\tilde{u}})\Bigg] \mathrm{d}\mathbf{z}_{\tilde{u}} \,, \tag{7}$$

where $p_\theta(\mathbf{t}_{\tilde{u}} \mid \mathbf{z}_{\tilde{u}}, \hat{\mathbf{Z}}_{\boldsymbol{\pi}}) = \mathrm{Mul}(\mathbf{t}_{\tilde{u}} \mid K, f_\theta(\frac{1}{|\mathcal{V}_{\boldsymbol{\pi}}|}\mathbf{I}, \mathbf{z}_{\tilde{u}}, \hat{\mathbf{Z}}_{\boldsymbol{\pi}}))$ is the distribution of global similar nodes of test node $\tilde{u}$. We use Monte Carlo method to approximate the expectation.

Table 1: Overall performance of CacheGNN and baselines in node classification tasks.

	Inductive				Transductive		
Model	**Computer**	**Photo**	**PPI**	**DBLP**	**Cora**	**Citeseer**	**Pubmed**
GCN	80.74 ± 0.52	88.19 ± 0.97	53.90 ± 0.21	**80.24 ± 0.63**	86.88 ± 1.15	75.19 ± 0.45	84.82 ± 0.02
+ CacheGNN	**83.08 ± 0.83**	**89.69 ± 1.38**	**95.49 ± 0.95**	77.65 ± 0.94	**88.08 ± 0.13**	**76.09 ± 0.64**	**85.61 ± 0.02**
SGC	82.04 ± 0.80	89.35 ± 0.18	49.41 ± 0.76	74.17 ± 0.25	86.51 ± 0.26	72.33 ± 0.85	82.98 ± 0.47
+ CacheGNN	**83.52 ± 0.42**	**90.20 ± 0.46**	**98.11 ± 0.06**	**77.78 ± 0.4**	**87.71 ± 0.39**	**73.53 ± 0.64**	**84.16 ± 0.56**
GraphSAGE	88.63 ± 0.37	94.15 ± 0.64	59.61 ± 0.23	74.68 ± 0.46	88.66 ± 0.59	75.64 ± 0.43	88.61 ± 0.36
+ CacheGNN	**89.22 ± 0.22**	**94.84 ± 0.33**	**97.25 ± 0.58**	**75.29 ± 0.61**	**88.85 ± 0.75**	**76.47 ± 0.53**	**88.68 ± 0.02**
GCNII	88.35 ± 1.10	93.34 ± 0.44	94.25 ± 0.14	84.41 ± 0.74	86.75 ± 0.65	71.20 ± 0.74	84.64 ± 0.05
+ CacheGNN	**88.96 ± 0.53**	**94.20 ± 0.21**	**98.46 ± 0.08**	**86.44 ± 0.22**	**86.78 ± 0.39**	**73.16 ± 0.53**	**85.29 ± 0.14**
GAT	87.24 ± 0.46	92.22 ± 0.14	89.54 ± 0.59	70.87 ± 2.64	85.09 ± 1.76	71.88 ± 0.85	86.32 ± 0.34
+ CacheGNN	**88.88 ± 0.45**	**93.33 ± 0.20**	**98.20 ± 0.41**	**76.56 ± 1.93**	**86.81 ± 2.04**	**72.93 ± 0.43**	**86.66 ± 0.11**
GATv2Conv	87.15 ± 0.60	92.84 ± 0.54	96.83 ± 1.26	70.46 ± 1.20	84.41 ± 1.89	**72.56 ± 0.74**	86.37 ± 0.20
+ CacheGNN	**88.89 ± 1.07**	**93.33 ± 0.30**	**98.73 ± 0.12**	**78.90 ± 1.94**	**85.95 ± 0.52**	72.26 ± 0.53	**87.12 ± 0.61**

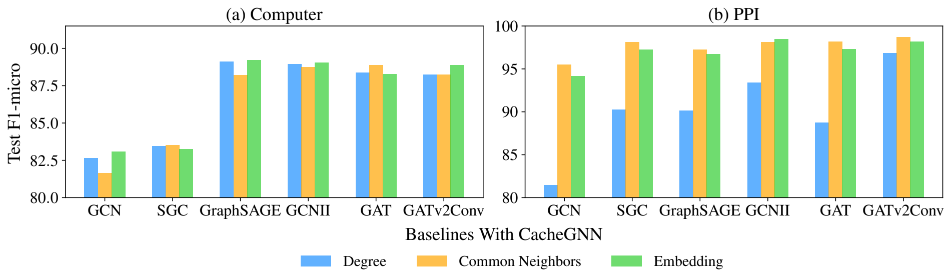


Figure 3: Performance of CacheGNN with different base models under different similarity functions on Computer and PPI benchmark datasets.

5 Discussions

Our CacheGNN differs from most existing GNNs at the following aspects: (1) Unlike previous works (Kipf & Welling, 2017; Hamilton et al., 2017; Veličković et al., 2018) that leverage a limited range of local neighbors, our CacheGNN aims at leveraging the comprehensive information, i.e., features, structural patterns and label information, of all nodes in the graph. (2) Unlike previous works (Zhuang & Ma, 2018; Gasteiger et al., 2018) that leverage global structural information but are difficult to be applied to inductive learning setting, our CacheGNN captures global information by finding global similar nodes using cache and can be applied to both inductive and transductive settings. (3) Unlike previous works (Chen et al., 2018; Fey et al., 2021; Yu et al., 2022) that store embeddings of all nodes in the cache which consumes large memory when dealing with large-scale graphs, our CacheGNN only stores embeddings of a small set of representative nodes, which is selected from a sparse node selection distribution. (4) We are the first attempt to encourage a sparse node selection distribution via the use of Dirichlet prior with appropriate hyperparameters, which in turn enables our CacheGNN to select as few representative nodes as possible, and therefore reduce the cache size and improve the prediction efficiency.

6 Experiments

We mainly study the following four research questions: (***RQ1***) Can CacheGNN outperform baselines in node classification task? (***RQ2***) How does CacheGNN perform when using different similarity functions to find global similar nodes? (***RQ3***) How does the number of global similar nodes K and the trade-off hyperparameter η affect the performance of CacheGNN? (***RQ4***) What are the distinctions, i.e., model performance and cache size, between using full cache which stores the embeddings of all nodes, and sampled cache which stores the embeddings of representative nodes?

6.1 Experimental Settings

Datasets. To evaluate the performance of CacheGNN in inductive node classification, we compare our method with baselines on four benchmark datasets, including Protein-Protein Interaction (PPI)

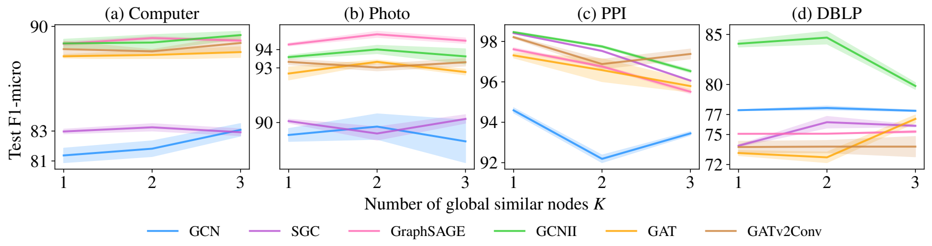


Figure 4: Performance of CacheGNN with different base models under different number of similar nodes on four benchmark datasets.

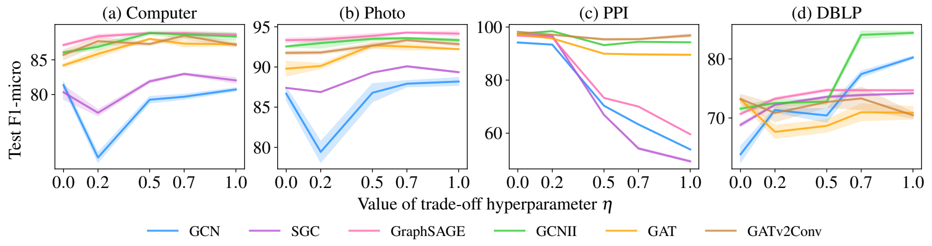


Figure 5: Performance of CacheGNN with different base models under different values of η.

networks Hamilton et al. (2017), DBLP dataset from the citation network Tang et al. (2008) and two Amazon product co-purchase networks Shchur et al. (2018), namely Photo and Computer. To evaluate the performance of CacheGNN in transductive node classification, we conduct experiment on three benchmark citation networks Yang et al. (2016), including Cora, Citeseer and Pubmed.

Baselines. We make comparisons between CacheGNN and existing state-of-the-art GNNs, including GCN (Kipf & Welling, 2017), GraphSAGE (Hamilton et al., 2017), GAT (Veličković et al., 2018), SGC (Wu et al., 2019), GCNII (Chen et al., 2020) and GATv2Conv (Brody et al., 2021).

Note that our CacheGNN aims to enhance existing GNNs by global information caching. Therefore, we report the performance of our CacheGNN with different GNNs as base models. We use micro-F1 score (in %) with the standard errors over 5 repetitions as evaluation metric. Detailed experimental settings are provided in section C of the Appendix. Resource code of our CacheGNN is publicly available from `https://github.com/anonynz12/anonymous3`

6.2 Results

To answer (***RQ1***), we compare CacheGNN with different GNN baselines in node classification task under both inductive and transductive settings. Experimental results are reported in Table 1, which suggest that CacheGNN can improve the performance of base GNNs in most cases. For example, in inductive node classification, CacheGNN improves the performance of GCN, SGC and GraphSAGE on PPI dataset by 41.6%, 48.7% and 37.8%, respectively. Likewise, in transductive node classification, it can be observed that GNNs equipped with CacheGNN outperform corresponding baselines in most cases. Theses results demonstrate that it is effective to utilise the information of global similar nodes to enhance existing GNNs and improve their performance on node classification task under both inductive and transductive settings.

Next, we explore (***RQ2***) by examining the performance of CacheGNN with different similarity functions. Specifically, we compare three different similarity functions, namely degree based function, common neighborhood based function and embedding based function. For degree based similarity function, we calculate the similarity of two nodes based on their degree. As for common neighborhood based similarity function, we compute node similarity by measuring the number of common neighborhoods. For embedding based similarity function, we measure node similarity by calculating cosine similarity of their node embeddings learned from GNNs. Experimental results are reported in Figure 3, which shows that different similarity functions lead to different model performance on Computer and PPI datasets. For instance, on Computer dataset, while GCN equipped with CacheGNN achieves the best performance under embedding similarity metric, SGC equipped with CacheGNN using common neighbors similarity metric achieves the best performance over different similarity functions. This can be explained by the fact that similarity functions influence the process

Table 2: Overall performance of CacheGNN using full cache and sampled cache on Computer, Photo, PPI and DBLP benchmark datasets.

	Computer		**Photo**		**PPI**		**DBLP**	
Model	Micro-F1	Cache Size	Micro-F1	Cache Size	Micro-F1	Cache Size	Micro-F1	Cache Size
GCN + CacheGNN (Full Cache)	83.11 ± 0.84	100%	89.78 ± 1.39	100%	96.26 ± 0.21	100%	77.77 ± 0.06	100%
GCN + CacheGNN (Sampled Cache)	83.08 ± 0.83	50.34%	89.69 ± 1.38	52.55%	94.17 ± 0.52	59.53%	77.65 ± 0.94	56.62%
SGC + CacheGNN (Full Cache)	83.38 ± 0.97	100%	90.48 ± 0.58	100%	98.41 ± 0.04	100%	76.27 ± 2.79	100%
SGC + CacheGNN (Sampled Cache)	83.52 ± 0.42	43.70%	90.20 ± 0.46	45.56%	97.26 ± 0.26	57.84%	76.23 ± 2.86	49.45%
GraphSAGE + CacheGNN (Full Cache)	89.26 ± 0.27	100%	94.89 ± 0.43	100%	97.61 ± 0.17	100%	75.33 ± 0.45	100%
GraphSAGE + CacheGNN (Sampled Cache)	89.22 ± 0.22	48.96%	94.84 ± 0.33	47.86%	96.75 ± 0.19	59.74%	75.29 ± 0.61	40.67%
GCNII + CacheGNN (Full Cache)	89.12 ± 0.17	100%	94.10 ± 0.16	100%	98.66 ± 0.04	100%	84.71 ± 3.00	100%
GCNII + CacheGNN (Sampled Cache)	88.96 ± 0.53	37.84%	94.20 ± 0.21	42.90%	98.46 ± 0.08	66.78%	84.68 ± 3.15	49.61%
GAT + CacheGNN (Full Cache)	88.36 ± 0.55	100%	93.34 ± 0.22	100%	98.48 ± 0.19	100%	75.03 ± 1.48	100%
GAT + CacheGNN (Sampled Cache)	88.88 ± 0.45	47.64%	93.33 ± 0.20	47.76%	97.31 ± 0.25	61.09%	76.56 ± 1.93	41.74%
GATv2Conv + CacheGNN (Full Cache)	88.91 ± 1.04	100%	93.42 ± 0.36	100%	99.00 ± 0.05	100%	77.47 ± 1.86	100%
GATv2Conv + CacheGNN (Sampled Cache)	88.89 ± 1.07	44.08%	93.33 ± 0.30	58.54%	98.21 ± 0.10	61.08%	78.90 ± 1.94	44.61%

of finding global similar nodes, and therefore CacheGNN obtains non-identical model performance under different similarity functions on the same dataset.

6.3 Analysis

Subsequently, we turn to (***RQ3***) by investigating the performance of CacheGNN under different number of global similar nodes K and trade-off hyperparameter η. To understand the effect of the number of global similar nodes, we conduct node classification experiments under different values of K. As indicated in Figure 4, CacheGNN can improve the performance of base GNNs by using a few global similar nodes. However, the performance of CacheGNN may degrade with the increase of K in some cases. This is because some irrelevant information may be introduced when utilising many global similar nodes. Therefore, the results suggest that a very small number of global similar nodes, such as 1 or 2, are sufficient to increase the model performance in most cases.

Recall that in section 4.1, we introduce a hyperparameter η to obtain the predicted label distribution. When $\eta = 1$, our model only uses local representations of nodes for prediction, which degrades to vanilla GNNs. In contrast, when $0 < \eta < 1$, our model predicts the labels of nodes by leveraging information from both local neighbors and global similar nodes. To further understand the effect of the trade-off hyperparameter η, we investigate the sensitivity of CacheGNN to the value of η. The experimental results are presented in Figure 5. We find that compared with $\eta = 1$, CacheGNN can achieve improved performance when $0 < \eta < 1$ in most cases. For instance, by tuning η smaller on PPI dataset, the performance of CacheGNN can be further improved by leveraging more information from global similar nodes. This result quantifies the contribution of utilizing information from global similar nodes, which leads to the improvement of model performance.

Finally, we answer (***RQ4***) by studying the performance of CacheGNN under different cache size. Recall that in section 4.3, CacheGNN leverages Dirichlet prior to learn a sparse distribution over training nodes, based on which we select a set of representative nodes and their cached embeddings for prediction. To evaluate the effectiveness of such method, we compare the performance of CacheGNN under full cache and sampled cache. Experimental results are provided in Table 2, where "Cache size" is the proportion of sampled nodes to the total number of training nodes. The results suggest that our CacheGNN is able to reduce the cache size while achieving comparable performance when utilising only a small fraction of training nodes. This verifies our hypothesis that selecting global similar nodes from a small of representative nodes can achieve comparable results as selecting global similar nodes from all training nodes.

7 Conclusion

In this paper, we propose CacheGNN to leverage information from global similar nodes to enhance GNNs. Considering the computation cost, our CacheGNN uses a cache to store node representations and leverages those cached embeddings to efficiently find global similar nodes. To further address the low efficiency problem at test time, our CacheGNN retrieves global similar nodes from a small set of representative nodes which are selected based on a sparse node selection distribution. Extensive experiments on seven real-world benchmark datasets demonstrate the superior performance of our CacheGNN. In the future we will extend CacheGNN to other graph learning applications such as link prediction and graph classification.

REFERENCES

Sami Abu-El-Haija, Bryan Perozzi, Amol Kapoor, Nazanin Alipourfard, Kristina Lerman, Hrayr Harutyunyan, Greg Ver Steeg, and Aram Galstyan. Mixhop: Higher-order graph convolutional architectures via sparsified neighborhood mixing. In *International Conference on Machine Learning*, pp. 21–29, 2019.

Matthew James Beal. *Variational algorithms for approximate Bayesian inference*. University of London, University College London (United Kingdom), 2003.

Cristian Bodnar, Fabrizio Frasca, Yuguang Wang, Nina Otter, Guido F Montufar, Pietro Lio, and Michael Bronstein. Weisfeiler and lehman go topological: Message passing simplicial networks. In *International Conference on Machine Learning*, pp. 1026–1037, 2021.

Giorgos Bouritsas, Fabrizio Frasca, Stefanos P Zafeiriou, and Michael Bronstein. Improving graph neural network expressivity via subgraph isomorphism counting. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2022.

Shaked Brody, Uri Alon, and Eran Yahav. How attentive are graph attention networks? In *International Conference on Learning Representations*, 2021.

Jie Cai, Xin Wang, Chaoyu Guan, Yateng Tang, Jin Xu, Bin Zhong, and Wenwu Zhu. Multimodal continual graph learning with neural architecture search. In *Proceedings of the ACM Web Conference 2022*, pp. 1292–1300, 2022.

Guanzheng Chen, Jinyuan Fang, Zaiqiao Meng, Qiang Zhang, and Shangsong Liang. Multi-relational graph representation learning with bayesian gaussian process network. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 36, pp. 5530–5538, 2022.

Jianfei Chen, Jun Zhu, and Le Song. Stochastic training of graph convolutional networks with variance reduction. In *International Conference on Machine Learning*, pp. 942–950, 2018.

Ming Chen, Zhewei Wei, Zengfeng Huang, Bolin Ding, and Yaliang Li. Simple and deep graph convolutional networks. In *International Conference on Machine Learning*, pp. 1725–1735, 2020.

Wei-Lin Chiang, Xuanqing Liu, Si Si, Yang Li, Samy Bengio, and Cho-Jui Hsieh. Cluster-gcn: An efficient algorithm for training deep and large graph convolutional networks. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pp. 257–266, 2019.

Djork-Arné Clevert, Thomas Unterthiner, and Sepp Hochreiter. Fast and accurate deep network learning by exponential linear units (elus). In *International Conference on Learning Representations*, 2016.

Matthias Fey and Jan Eric Lenssen. Fast graph representation learning with pytorch geometric. In *ICLR 2019 (RLGM Workshop)*, 2019.

Matthias Fey, Jan E Lenssen, Frank Weichert, and Jure Leskovec. Gnnautoscale: Scalable and expressive graph neural networks via historical embeddings. In *International Conference on Machine Learning*, pp. 3294–3304, 2021.

Hongyang Gao, Zhengyang Wang, and Shuiwang Ji. Large-scale learnable graph convolutional networks. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pp. 1416–1424, 2018.

Johannes Gasteiger, Aleksandar Bojchevski, and Stephan Günnemann. Predict then propagate: Graph neural networks meet personalized pagerank. In *International Conference on Learning Representations*, 2018.

Johannes Gasteiger, Florian Becker, and Stephan Günnemann. Gemnet: Universal directional graph neural networks for molecules. *Advances in Neural Information Processing Systems*, 34:6790–6802, 2021.

Justin Gilmer, Samuel S Schoenholz, Patrick F Riley, Oriol Vinyals, and George E Dahl. Neural message passing for quantum chemistry. In *International Conference on Machine Learning*, pp. 1263–1272, 2017.

Lingbing Guo, Qiang Zhang, Zequn Sun, Mingyang Chen, Wei Hu, and Huajun Chen. Understanding and improving knowledge graph embedding for entity alignment. In *International Conference on Machine Learning*, pp. 8145–8156, 2022.

Will Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. *Advances in Neural Information Processing Systems*, 30, 2017.

Yu Hao, Xin Cao, Yufan Sheng, Yixiang Fang, and Wei Wang. Ks-gnn: Keywords search over incomplete graphs via graphs neural network. *Advances in Neural Information Processing Systems*, 34:1700–1712, 2021.

Arman Hasanzadeh, Ehsan Hajiramezanali, Krishna Narayanan, Nick Duffield, Mingyuan Zhou, and Xiaoning Qian. Semi-implicit graph variational auto-encoders. *Advances in Neural Information Processing Systems*, 32, 2019.

Junxian He, Taylor Berg-Kirkpatrick, and Graham Neubig. Learning sparse prototypes for text generation. *Advances in Neural Information Processing Systems*, 33:14724–14735, 2020.

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *International Conference on Learning Representations*, 2015.

Thomas N Kipf and Max Welling. Variational graph auto-encoders. *NIPS Workshop on Bayesian Deep Learning*, 2016.

Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. In *International Conference on Learning Representations*, 2017.

Qimai Li, Zhichao Han, and Xiao-Ming Wu. Deeper insights into graph convolutional networks for semi-supervised learning. In *Thirty-Second AAAI conference on Artificial Intelligence*, 2018.

Meng Liu, Hongyang Gao, and Shuiwang Ji. Towards deeper graph neural networks. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pp. 338–348, 2020.

Vinod Nair and Geoffrey E Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th International Conference on International Conference on Machine Learning*, pp. 807–814, 2010.

Kenta Oono and Taiji Suzuki. Graph neural networks exponentially lose expressive power for node classification. In *International Conference on Learning Representations*, 2019.

Meng Qu, Yoshua Bengio, and Jian Tang. Gmnn: Graph markov neural networks. In *International Conference on Machine Learning*, pp. 5241–5250, 2019.

Meng Qu, Huiyu Cai, and Jian Tang. Neural structured prediction for inductive node classification. In *International Conference on Learning Representations*, 2021.

Oleksandr Shchur, Maximilian Mumme, Aleksandar Bojchevski, and Stephan Günnemann. Pitfalls of graph neural network evaluation. *Relational Representation Learning Workshop, NeurIPS 2018*, 2018.

Damian Szklarczyk, Annika L Gable, David Lyon, Alexander Junge, Stefan Wyder, Jaime Huerta-Cepas, Milan Simonovic, Nadezhda T Doncheva, John H Morris, Peer Bork, et al. String v11: protein–protein association networks with increased coverage, supporting functional discovery in genome-wide experimental datasets. *Nucleic Acids Research*, 47(D1):D607–D613, 2019.

Jian Tang, Meng Qu, Mingzhe Wang, Ming Zhang, Jun Yan, and Qiaozhu Mei. Line: Large-scale information network embedding. In *Proceedings of the 24th International Conference on World Wide Web*, pp. 1067–1077, 2015.

Jie Tang, Jing Zhang, Limin Yao, Juanzi Li, Li Zhang, and Zhong Su. Arnetminer: extraction and mining of academic social networks. In *Proceedings of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 990–998, 2008.

Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph attention networks. In *International Conference on Learning Representations*, 2018.

S Wang, L Hu, Y Wang, X He, QZ Sheng, MA Orgun, L Cao, F Ricci, and PS Yu. Graph learning based recommender systems: A review. In *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence*, 2021.

David S Wishart, Yannick D Feunang, An C Guo, Elvis J Lo, Ana Marcu, Jason R Grant, Tanvir Sajed, Daniel Johnson, Carin Li, Zinat Sayeeda, et al. Drugbank 5.0: a major update to the drugbank database for 2018. *Nucleic Acids Research*, 46(D1):D1074–D1082, 2018.

Felix Wu, Amauri Souza, Tianyi Zhang, Christopher Fifty, Tao Yu, and Kilian Weinberger. Simplifying graph convolutional networks. In *International Conference on Machine Learning*, pp. 6861–6871, 2019.

Zhanghao Wu, Paras Jain, Matthew Wright, Azalia Mirhoseini, Joseph E Gonzalez, and Ion Stoica. Representing long-range context for graph neural networks with global attention. *Advances in Neural Information Processing Systems*, 34:13266–13279, 2021.

Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? In *International Conference on Learning Representations*, 2018a.

Keyulu Xu, Chengtao Li, Yonglong Tian, Tomohiro Sonobe, Ken-ichi Kawarabayashi, and Stefanie Jegelka. Representation learning on graphs with jumping knowledge networks. In *International Conference on Machine Learning*, pp. 5453–5462, 2018b.

Mingqi Yang, Renjian Wang, Yanming Shen, Heng Qi, and Baocai Yin. Breaking the expression bottleneck of graph neural networks. *IEEE Transactions on Knowledge and Data Engineering*, 2022.

Zhilin Yang, William Cohen, and Ruslan Salakhudinov. Revisiting semi-supervised learning with graph embeddings. In *International Conference on Machine Learning*, pp. 40–48, 2016.

Zhiyong Yang, Qianqian Xu, Xiaochun Cao, and Qingming Huang. Task-feature collaborative learning with application to personalized attribute prediction. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 43(11):4094–4110, 2020.

Liang Yao, Chengsheng Mao, and Yuan Luo. Graph convolutional networks for text classification. In *Proceedings of the AAAI conference on Artificial Intelligence*, volume 33, pp. 7370–7377, 2019.

Haiyang Yu, Limei Wang, Bokun Wang, Meng Liu, Tianbao Yang, and Shuiwang Ji. Graphfm: Improving large-scale gnn training via feature momentum. In *International Conference on Machine Learning*, pp. 25684–25701, 2022.

Hanqing Zeng, Hongkuan Zhou, Ajitesh Srivastava, Rajgopal Kannan, and Viktor Prasanna. GraphSAINT: Graph sampling based inductive learning method. In *International Conference on Learning Representations*, 2020.

Hanqing Zeng, Muhan Zhang, Yinglong Xia, Ajitesh Srivastava, Andrey Malevich, Rajgopal Kannan, Viktor Prasanna, Long Jin, and Ren Chen. Deep graph neural networks with shallow subgraph samplers. In *Advances in Neural Information Processing Systems*, 2021.

Shuai Zhang, Yi Tay, Lina Yao, and Qi Liu. Quaternion knowledge graph embeddings. *Advances in Neural Information Processing Systems*, 32, 2019.

Wentao Zhang, Zeang Sheng, Yuezihan Jiang, Yikuan Xia, Jun Gao, Zhi Yang, and Bin Cui. Evaluating deep graph neural networks. *arXiv preprint arXiv:2108.00955*, 2021.

Chenyi Zhuang and Qiang Ma. Dual graph convolutional networks for graph-based semi-supervised classification. In *Proceedings of the 2018 World Wide Web Conference*, pp. 499–508, 2018.

Marinka Zitnik and Jure Leskovec. Predicting multicellular function through multi-layer tissue networks. *Bioinformatics*, 33(14):i190–i198, 2017.

A NOTATION TABLE

Table 3 summarizes the main notations appeared across the paper.

Table 3: Main notations used across the paper.

Symbol	Description
$\mathcal{G}$	a graph with the set of labeled nodes
$\mathcal{V}$	a set of nodes
$\mathcal{E}$	a set of edges between nodes
L	number of GNN layers
$N = \|\mathcal{V}\|$	number of nodes
C	number of classes
K	number of the most similar nodes
F	feature dimension
η	trade-off hyperparameter
α	concentration hyperparameter
$\boldsymbol{\pi}$	latent representative node distribution
θ	model parameters
ϕ	variational parameters
λ	variational parameters
$\mathbf{z}_u$	representation of node u
$\mathbf{Z}$	representations of all nodes
$\mathbf{t}_u$	the set of global similar nodes of u
$\mathbf{T}$	the set of all global similar nodes
$\hat{\mathbf{Z}}$	cached embeddings of all nodes
$\hat{\mathbf{Z}}_{\boldsymbol{\pi}}$	cached embeddings of top M representative nodes
$\mathbf{X} \in \mathbb{R}^{N \times F}$	feature matrix of nodes
$\mathbf{Y} \in \{0,1\}^{N \times C}$	one-hot encoding label matrix of nodes

B PSEUDO CODES OF CACHEGNN

In this section, we first introduce message passing framework. Specifically, for an L-layer GNN, the representation learning function of the l-th layer is represented as:

$$\mathbf{b}_i^{(l)} = \text{AGGREGATION}(\{\mathbf{h}_j^{(l-1)} : j \in Ne(i)\}), \tag{8}$$

$$\mathbf{h}_i^{(l)} = \text{COMBINE}(\mathbf{h}_i^{(l-1)}, \mathbf{b}_i^{(l)}), \tag{9}$$

where $\mathbf{h}_i^{(l)}$ is the representation of node v_i at the l-th layer with $\mathbf{h}_i^{(0)} = \mathbf{x}_i$, $Ne(i)$ is a set of nodes adjacent to v_i, and $\text{AGGREGATION}(\cdot)$ and $\text{COMBINE}(\cdot)$ are the component functions of GNN layers. Different GNNs adopt different formulations for these two component functions. For instance, GraphSAGE (Hamilton et al., 2017) uses mean function for $\text{AGGREGATION}(\cdot)$ and a one-layer feed-forward neural network for $\text{COMBINE}(\cdot)$.

After obtaining the node representations, GNNs apply a linear softmax layer to the node representations to calculate the label distributions of nodes, which are usually modeled as factorized categorical distributions, i.e., $p_\theta(\mathbf{Y} \mid \mathbf{X}, \mathcal{E}) = \prod_{u \in \mathcal{V}} p_\theta(\mathbf{y}_u \mid \mathbf{X}, \mathcal{E})$. The model parameters of GNNs are learned by maximizing the log-likelihood of label distributions through stochastic gradient descent.

We further provide an overview of the optimization process of CacheGNN model for node classification in in Algorithm 1.

Algorithm 1 The proposed CacheGNN approach for node classification task.

Input: A training graph with labeled nodes $\mathcal{G} = \{\mathcal{V}, \mathcal{E}, \mathbf{X}, \mathbf{Y}\}$ and a test graph $\tilde{\mathcal{G}} = \{\tilde{\mathcal{V}}, \tilde{\mathcal{E}}, \tilde{\mathbf{X}}\}$.
Output: Node labels $\tilde{\mathbf{Y}}$ for the unlabeled nodes in $\tilde{\mathcal{G}}$.

```
1: Pre-train $p_\theta$ according to Equation 8 and 9.
2: while no converge do
3:     ⊡ E-step                                                  ▷ Estimate the posterior distributions
4:     Calculate $q_\phi$ based on $q_\phi(\mathbf{t}_u \mid \mathbf{Y})$ and $q_\phi(\mathbf{z}_u \mid \mathbf{t}_u, \mathbf{Y})$
5:     Calculate $q_\lambda(\boldsymbol{\pi})$ based on $\prod_{i=1}^{N} \boldsymbol{\pi}_i^{\lambda_i - 1}$.
6:     Update $q_\phi$ and $q_\lambda(\boldsymbol{\pi})$ based on Equation 6.
7:     ⊡ M-step                                                  ▷ Maximizing the expectation
8:     Calculate $p_\theta$ based on Equation 2.
9:     Update $p_\theta$ based on $\mathbb{E}_{q_\phi(\mathbf{T},\mathbf{Z},\boldsymbol{\pi}|\mathbf{Y})}[\log p_\theta(\mathbf{Y}, \mathbf{T}, \mathbf{Z}, \boldsymbol{\pi} \mid \mathbf{X}, \mathcal{E})]$.
10: end while
11: Select the top $M$ nodes that occupy $90\%$ of the probability mass and their corresponding cached
    embeddings as $\hat{\mathbf{Z}}_{\boldsymbol{\pi}}$.
12: Classify each unlabeled node in graph with $p_\theta$ and cache $\hat{\mathbf{Z}}_{\boldsymbol{\pi}}$ based on Equation 7.
```

C Experimental details

C.1 Datasets

We conduct experiments on the following seven real-world network datasets [2], which consists of three standard citation network benchmark datasets, two Amazon Co-purchase networks, DBLP dataset and propein-protein interaction (PPI) dataset, statistical information of which is provided in Table 4.

Table 4: Statistics of the datasets used in the experiments.

Dataset	Task	#Nodes	#Edges	#Features	#Labels
Cora	Multi-class	2,708	5,429	1,433	7
Citeseer	Multi-class	3,327	4,732	3,703	6
Pubmed	Multi-class	19,717	44,338	500	3
Photo	Multi-class	7,650	238,162	745	8
Computers	Multi-class	13,752	491,722	767	10
DBLP	Multi-class	47,443	214,792	100	3
PPI	Multi-label	56,944	818,716	50	121

- **Cora, Citeseer, Pubmed** (Yang et al., 2016): The three datasets are citation networks where the nodes represent the documents and the edges represent the citation links between publications. Each node feature corresponds to the bag-of-words representation of the document and belongs to one of the academic topics. We randomly select 70% of nodes as training nodes, 10% as validation nodes and the remaining nodes are treated as test nodes.
- **Photo, Computers** (Shchur et al., 2018): The two datasets are the Amazon product co-purchase networks where the nodes represent the products and the edges between two nodes indicate that two products are frequently bought together. The features of each node are the bag-of-word product reviews and the labels are given by the product category. We randomly select 20% of nodes as training nodes, 40% of nodes as validation nodes and the remaining nodes are treated as test nodes.
- **DBLP** (Tang et al., 2008): The DBLP dataset is constructed from the citation network where the nodes represent the documents and the edges represent the citation links between publication. Following the previous work (Qu et al., 2021), We compute the mean GloVe embedding [3] of words in the title and abstract as features. We split the DBLP dataset

[2]The datasets are available from `https://pytorch-geometric.readthedocs.io/en/latest/modules/datasets.html`

[3]`http://nlp.stanford.edu/data/glove.6B.zip`

into three disjoint graphs for training/validation/test. The training graph contains papers published before 1999 (with 1999 included). The validation graph contains papers between 2000 and 2009 (with 2000 and 2009 included). The test graph contains papers published after 2010 (with 2010 included).

- **PPI** (Hamilton et al., 2017): The PPI dataset is constructed from human tissues (Zitnik & Leskovec, 2017). Each node has 50 features that are composed of positional gene sets, motif gene sets and immunological signatures. There are 121 labels for each node set from gene ontology and a node can possess several labels simultaneously. Following the previous work (Hamilton et al., 2017), the dataset contains 20 graphs for training, 2 for validation and 2 for testing.

C.2 Hyperparameter choices

Throughout the experiments, we use the Adam SGD optimizer (Kingma & Ba, 2015) with a learning rate of 0.001 and early stopping with a patience of 100 epochs. We run all experiments on a single Nvidia 2080 Ti GPU. We set the Dirichlet hyperparameter α to 0.1 to encourage encourage a sparse node selection distribution. We use the GNN module implementations of PyTorch Geometric (Fey & Lenssen, 2019), and follow the GNN models provided in the examples of the repository.

GCN (Kipf & Welling, 2017). We set the number of hidden neurons to 16, and the number of layers to 2. RELU (Nair & Hinton, 2010) is used as the activation function. We do not dropout between GNN layers.

GraphSAGE (Hamilton et al., 2017). We set the number of hidden neurons to 64, and the number of layers to 2. ELU (Clevert et al., 2016) is used as the activation function. We do not dropout between GNN layers.

GAT (Veličković et al., 2018). We set the number of hidden neurons to 64 per attention head and the number of layers to 3 for the PPI dataset. For the rest datasets (DBLP, Photo, Amazon, Cora, Citeseer and Pubmed), we set the number of hidden neurons to 128 per attention head and the number of layers to 3. The number of heads for each layer is set to 4, 4 and 6. ELU (Clevert et al., 2016) is used as the activation function. We do not dropout between GNN layers.

SGC (Wu et al., 2019). We set the number of hidden neurons to 256, the number of layers to 2 and the number of hops to 2. We do not dropout between GNN layers.

GCNII (Chen et al., 2020). We set the number of hidden neurons to 512. We set the number of layers to 9. ReLU Nair & Hinton (2010) is used as the activation function. We set the strength α of the initial residual connection to 0.5, and the hyperparameter θ to compute the strength of the identity mapping to 1.

GATv2Conv (Brody et al., 2021). We set the number of hidden neurons to 64 per attention head and the number of layers to 3 for the PPI dataset. For the rest datasets (DBLP, Photo, Amazon, Cora, Citeseer and Pubmed), we set the number of hidden neurons to 128 per attention head and the number of layers to 3. The number of heads for each layer is set to 4, 4 and 6. ELU (Clevert et al., 2016) is used as the activation function. We do not dropout between GNN layers.

D Computational Complexity

In order to verify the fact that our CacheGNN can reduce the computation cost, we analysis the computational complexity of CacheGNN equipped with GCN under the situation of using cache and not using cache. If not using cache, the computational complexity of our method equipped with GCN is $\mathcal{O}(N(LN^2F + LNF^2))$, where L, N, F represent the number of layers, the number of nodes in the graph and the dimension of embeddings respectively. If using cache, the computation complexity of our method equipped with GCN is $\mathcal{O}(LN^2F + LNF^2)$, which shows that leveraging cache is much more efficient and could reduce the computation cost.