Asymmetric Learning for Graph Neural Network based Link Prediction

KAI-LANG YAO and WU-JUN LI, National Key Laboratory for Novel Software Technology, Department of Computer Science and Technology, Nanjing University, China

Link prediction is a fundamental problem in many graph-based applications, such as protein-protein interaction prediction. Recently, graph neural network (GNN) has been widely used for link prediction. However, existing GNN-based link prediction (GNN-LP) methods suffer from scalability problem during training for large-scale graphs, which has received little attention from researchers. In this paper, we first analyze the computation complexity of existing GNN-LP methods, revealing that one reason for the scalability problem stems from their symmetric learning strategy in applying the same class of GNN models to learn representation for both head nodes and tail nodes. We then propose a novel method, called asymmetric learning (AML), for GNN-LP. More specifically, AML applies a GNN model to learn head node representation while applying a multi-layer perceptron (MLP) model to learn tail node representation. To the best of our knowledge, AML is the first GNN-LP method to adopt an asymmetric learning strategy for node representation learning. Furthermore, we design a novel model architecture and apply a row-wise mini-batch sampling strategy to ensure promising model accuracy and training efficiency for AML. Experiments on three real large-scale datasets show that AML is 1.7×~7.3× faster in training than baselines with a symmetric learning strategy while having almost no accuracy loss.

CCS Concepts: • **Computing methodologies** → **Machine learning approaches**;

Additional Key Words and Phrases: Graph neural networks, link prediction, large-scale graphs

ACM Reference Format:
Kai-Lang Yao and Wu-Jun Li. 2024. Asymmetric Learning for Graph Neural Network based Link Prediction. *ACM Trans. Knowl. Discov. Data.* 18, 5, Article 106 (February 2024), 18 pages. https://doi.org/10.1145/3640347

1 INTRODUCTION

Link prediction [Liben-Nowell and Kleinberg 2007], a fundamental problem in many graph-based applications, aims to predict the existence of a link that has not been observed. Link prediction problem widely exists in real applications, like drug response prediction [Stanfield et al. 2017], protein-protein interaction prediction [Qi et al. 2006], friendship prediction in social networks [Adamic and Adar 2003], knowledge graph completion [Nickel et al. 2016; Rossi et al. 2021; Shomer et al. 2023] and product recommendation in recommender systems [Koren et al. 2009].

This work is supported by NSFC Project (No. 12326615, No. 62192783) and Fundamental Research Funds for the Central Universities (No. 020214380108).
Authors' address: K.-L. Yao and W.-J. Li (Corresponding author), National Key Laboratory for Novel Software Technology, Department of Computer Science and Technology, Nanjing University, 163 Xianlin Avenue, Qixia District, Nanjing 210023, China; e-mails: yaokl@smail.nju.edu.cn, liwujun@nju.edu.cn.

ACM 1556-4681/2024/02-ART106
https://doi.org/10.1145/3640347

Its increased importance in real applications also promotes a great interest in research for link prediction algorithms in the machine learning community.

Link prediction algorithms have been studied for a long time [Lü and Zhou 2011; Martinez et al. 2017; Samad et al. 2020], while learning-based algorithms have been one dominant class in the past decades. Learning-based algorithms aim to learn a deterministic model [Koren et al. 2009; Menon and Elkan 2011; Zhang and Chen 2017] or a probabilistic model [Goldenberg et al. 2009; Guimerà and Sales-Pardo 2009; Salakhutdinov and Mnih 2007] which can fit the observed data. In most learning-based algorithms, models learn or generate a representation for each node [Zhang et al. 2021b], which is used to generate a score or probability of link existence. Traditional learning-based algorithms typically do not adopt graph neural network (GNN) for node representation learning. Although these non-GNN based learning methods have achieved much progress in many applications, they are less expressive than GNN in node representation learning.

Recently, graph neural network based link prediction (GNN-LP) methods have been proposed and have become one of the most popular algorithms due to their superior accuracy performance. The key to the success of GNN-LP methods is that they learn node representation from graph structure and node features in a unified way with GNN, which is a major difference between them and traditional non-GNN based learning methods. Existing GNN-LP methods mainly include local methods [Li et al. 2020; You et al. 2021; Zhang and Chen 2018; Zhang et al. 2021a] and global methods [Hasanzadeh et al. 2019; Kipf and Welling 2016; Pan et al. 2018; Yun et al. 2021]. Local methods apply GNN to subgraphs that capture local structural information. More specifically, they first extract an enclosed k-hop subgraph for each link and then apply various labeling tricks [Zhang et al. 2021a] to capture the relative positions of nodes in the subgraph. After that, they learn node representation by applying a GNN model to the labeled subgraphs, and then they extract subgraph representation with a readout function [Gilmer et al. 2017] for prediction. Global methods learn node representation by directly applying a GNN model to the global graph and then make predictions based on head node and tail node representation. Even though differences exist between local methods and global methods, existing GNN-LP methods share a common characteristic that they adopt a symmetric learning strategy for node representation learning. In particular, they apply the same class of GNN models to learn representation for both head nodes and tail nodes. An illustration of existing representative GNN-LP methods is presented in Figure 1. Although existing GNN-LP methods have made much progress in learning expressive models, they suffer from scalability problem during training for large-scale graphs, which has received little attention from researchers. By taking a graph dataset ogbl-citation2 (comprising millions of nodes) as an example, existing GNN-LP methods take at least four days for training on an NVIDIA RTX A6000 GPU. The graphs in many applications may be an order of magnitude larger than ogbl-citation2. Such low training efficiency will impose undesirable limitations on existing GNN-LP methods. For example, optimizing models and tuning hyper-parameters within reasonable time overhead becomes challenging, leading to suboptimal model accuracy. Moreover, real applications typically accumulate data over time, necessitating continual model retraining to learn new knowledge from fresh data. Hence, it is important to improve the computation efficiency during training.

In this paper, we propose a novel method, called asymmetric learning (AML), for GNN-LP. The contributions of this paper are listed as follows:

- We analyze the computation complexity of existing GNN-LP methods, revealing that one reason for the scalability problem stems from their symmetric learning strategy in applying the same class of GNN models to learn representation for both head nodes and tail nodes.
- AML is the first GNN-LP method to adopt an asymmetric learning strategy for node representation learning. Furthermore, we design a novel model architecture and apply a

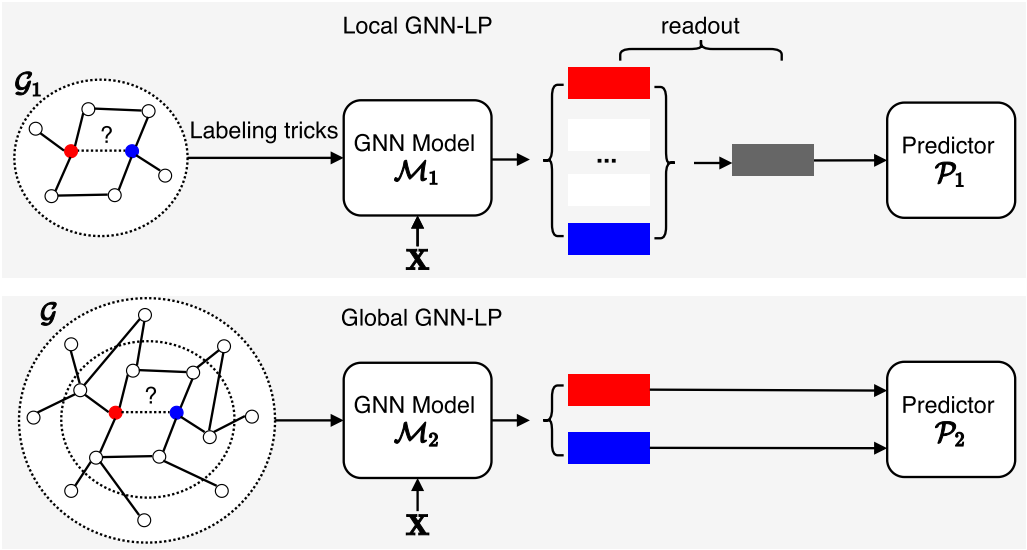


Fig. 1. An illustration of existing representative GNN-LP methods. A circle denotes a node. $\mathcal{G}$ denotes the input graph. $\mathcal{G}_1$ denotes a subgraph extracted from $\mathcal{G}$. The dashed line between the red circle and the blue circle denotes the target link we aim to predict. $\mathbf{X}$ denotes the node feature matrix. The red rectangle and the blue rectangle denote the representation of the red circle and the blue circle, respectively. The gray rectangle denotes the subgraph representation of $\mathcal{G}_1$ generated by applying a readout function on representation of the nodes within the subgraph.

row-wise mini-batch sampling strategy to ensure promising model accuracy and training efficiency for AML.

- Experiments on three real large-scale datasets show that AML is 1.7×~7.3× faster in training than baselines with a symmetric learning strategy, while having almost no accuracy loss.

2 PRELIMINARY

In this section, we introduce notations and some related works for link prediction.

2.1 Notations

We use a boldface uppercase letter, such as $\mathbf{B}$, to denote a matrix. We use a boldface lowercase letter, such as $\mathbf{b}$, to denote a vector. $\mathbf{B}_{i*}$ and $\mathbf{B}_{*j}$ denote the ith row and the jth column of $\mathbf{B}$, respectively. $\mathbf{X} \in \mathbb{R}^{N \times u}$ denotes the node feature matrix, where u is the feature dimension and N is the number of nodes. $\mathbf{A} \in \{0,1\}^{N \times N}$ denotes the adjacency matrix of a graph $\mathcal{G}$. $A_{ij} = 1$ iff there is an edge from node i to node j, otherwise $A_{ij} = 0$. L denotes the number of layers for GNN models. $\mathcal{E}$ denotes the set of links for training. For a link (i, j), we call node i a *head node* and call node j a *tail node*.

2.2 Related Works

2.2.1 Graph Neural Network. GNN [Bruna et al. 2014; Gori et al. 2005; Hamilton et al. 2017; Jin et al. 2022; Kipf and Welling 2017; Scarselli et al. 2009; Velickovic et al. 2018] is a class of models for learning over graph data. In GNN, nodes can iteratively encode their first-order and high-order neighbor information in the graph through message passing between neighbor nodes [Gilmer et al. 2017]. Due to the iteratively dependent nature, the computation complexity for a node

exponentially increases with iterations. Although some works [Gao et al. 2018; Yao and Li 2021; Zeng et al. 2020; Zou et al. 2019] propose solutions for the above problem of exponential complexity, the computation complexity of GNN is still much higher than that of a multi-layer perceptron (MLP).

2.2.2 Graph Neural Network based Link Prediction. Benefited from the powerful ability of GNN in modeling graph data, GNN-LP methods are more expressive than traditional non-GNN based learning methods in node representation learning. GNN-LP methods include two major classes, local methods and global methods. For local methods, different methods vary in the labeling tricks they apply, which mainly include double radius node labeling (DRNL) [Zhang and Chen 2018], distance encoding (DE) [Li et al. 2020], partial zero-one labeling trick [You et al. 2021] and zero-one labeling trick [Zhang et al. 2021a]. As shown in Zhang et al. [2021a], local methods with DRNL and DE perform better than other methods. However, one bottleneck for DRNL and DE is that they need to compute the shortest path distance (SPD) between target nodes and other nodes within subgraphs, and computing SPD is time-consuming during the training process [Zhang et al. 2021a]. Although we can compute SPD in the preprocessing step, costly storage overhead for subgraphs occurs instead. For global methods [Hasanzadeh et al. 2019; Kipf and Welling 2016; Pan et al. 2018; Yun et al. 2021], they mainly apply different GNN models on the global graphs to generate node representation. A recently proposed local method, called ELPH [Chamberlain et al. 2023], bridges previous local methods and global methods with additional structural features, which involve counts of generalized common neighbors (CN) between head nodes and tail nodes. ELPH reveals that it is unnecessary to propagate conditional information between head nodes and tail nodes with GNN. As a result, ELPH decouples conditional information (e.g., SPD) from node representations by introducing supplementary structural features as inputs to the link prediction module. Almost all existing GNN-LP methods adopt a symmetric learning strategy, which applyies the same class of GNN models to learn representation for both head nodes and tail nodes.

2.2.3 Non-GNN based Methods. Besides GNN-LP methods, WLNM [Zhang and Chen 2017] and SUREL [Yin et al. 2022] also show competitive performance in accuracy. The main difference between them and GNN-LP methods is that they do not apply GNN to learn from graphs or subgraphs. For example, WLNM applies an MLP model to learn subgraph representation from the adjacency matrices of extracted subgraphs. SUREL proposes an alternative sampler for subgraph extraction and applies a sequential model, like recurrent neural networks (RNNs), to learn subgraph representation.

2.2.4 Scalable Training of Graph Neural Network. When applied to large-scale graphs, GNN has a computation complexity that increases with the number of model layers. Many works have proposed solutions to reduce the computation complexity of GNN models. These solutions mainly include model simplification methods and sampling-based methods. Model simplification methods [Chen et al. 2020; Klicpera et al. 2019; Wu et al. 2019; Zhang et al. 2021c] avoid the problem of exponential computation complexity by fusing graph structure information into node features during the preprocessing step. However, these methods suffer from weak expressive power despite being computation-efficient. Instead, sampling-based methods have gained much interest recently due to their promising model accuracy. Sampling-based methods reduce the average number of sampled neighbors via various sampling strategies. These methods mainly include node-wise sampling methods [Chen et al. 2018b; Cong et al. 2020; Hamilton et al. 2017; Yao and Li 2021], layer-wise sampling methods [Chen et al. 2018a; Zou et al. 2019] and subgraph sampling methods [Chiang et al. 2019; Fey et al. 2021; Shi et al. 2023; Yu et al. 2022; Zeng et al. 2020, 2021]. Even though

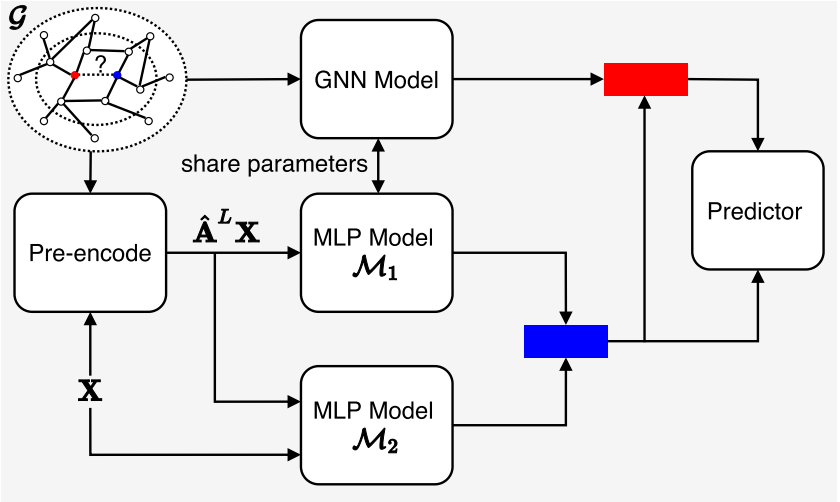


Fig. 2. An illustration of AML. The MLP model $\mathcal{M}_1$ is for transfering knowledge from head nodes to tail nodes by sharing parameters with the GNN model. The MLP model $\mathcal{M}_2$ is for learning over the residual between $\hat{\mathbf{A}}^L\mathbf{X}$ and $\mathbf{X}$. AML obtains the tail node representation, marked with the blue rectangle, by summing up the outcomes of $\mathcal{M}_1$ and $\mathcal{M}_2$. AML obtains the head node representation, marked with the red rectangle, by summing up the tail node representation and the outcomes of the GNN model. The predictor is for generating predictions according to the input vector representation.

sampling-based methods can reduce the growth rate by a large margin, they still have a computation complexity much larger than that of an MLP model.

In general, GNN-LP methods can achieve higher accuracy than non-GNN based methods [Zhang et al. 2021a], but suffer from scalability problem during training when applied to large-scale graphs. Even though global GNN-LP methods are more efficient than local GNN-LP methods, they still have high computation complexity in training due to the adopted symmetric learning strategy. Unlike existing works in scalable training of GNN models, this paper does not aim to reduce the computation complexity of GNN models. Instead, this paper aims to reduce the overall computation complexity of GNN-LP in a different way. In particular, we achieve this by decreasing the frequency of applying a GNN model to generate node representations during training. As a result, our proposed method in this paper can be combined with any existing scalable GNN training methods to further reduce the computation complexity of GNN-LP.

3 ASYMMETRIC LEARNING FOR GNN-LP

Like most deep learning methods, GNN-LP methods are typically trained in a mini-batch manner. Suppose the number of links in the training set is $|\mathcal{E}|$. Existing GNN-LP methods with a symmetric strategy need to perform $2|\mathcal{E}|$ times of GNN computation within each epoch. In particular, $|\mathcal{E}|$ times of GNN computation are for head nodes and another $|\mathcal{E}|$ times of GNN computation are for tail nodes. It is easy to verify that $|\mathcal{E}|$ times of GNN computation are inevitable for both head node and tail node representation learning with a symmetric strategy. Since GNN is of exponential computation complexity and $|\mathcal{E}|$ is considerably large, existing GNN-LP methods incur a huge computation burden for large-scale graphs.

To solve the scalability problem caused by symmetric learning, we propose AML which is illustrated in Figure 2. Unlike existing GNN-LP methods with a symmetric strategy, AML designs a novel model architecture to learn node representations. More specifically, AML applies a GNN

model to learn head node representations while applying an MLP model to learn tail node representations. Meanwhile, AML pre-encodes graph structure to avoid information loss for MLP. The following content of this section will present the details of AML.

3.1 Node Representation Learning with AML

We use $\mathbf{U} \in \mathbb{R}^{N\times r}$ to denote the representation of all head nodes and use $\mathbf{V} \in \mathbb{R}^{N\times r}$ to denote the representation of all tail nodes, where each row of $\mathbf{U}$ and $\mathbf{V}$ corresponds to a node. We apply a GNN model to learn representation for head nodes while applying an MLP model to learn representation for tail nodes.[1]

We take SAGE [Hamilton et al. 2017], one of the most representative GNN models, as an example to describe the details. Note that other GNN models can also be used in AML. Let $\hat{\mathbf{A}}$ denote the normalization of $\mathbf{A}$, which can be row-normalization, column-normalization, or symmetric normalization. Formulas for one layer in SAGE are as follows:

$$\mathbf{U}_{i*}^{(\ell)} = f\left(\hat{\mathbf{A}}_{i*}\mathbf{U}^{(\ell-1)}\mathbf{W}_1^{(\ell)} + \mathbf{U}_{i*}^{(\ell-1)}\mathbf{W}_2^{(\ell)}\right), \tag{1}$$

where ℓ is layer number, $f(\cdot)$ is an activation function, $\mathbf{W}_1^{(\ell)} \in \mathbb{R}^{r\times r,}$ and $\mathbf{W}_2^{(\ell)} \in \mathbb{R}^{r\times r}$ are learnable parameters at layer ℓ. $\mathbf{U}^{(\ell)} \in \mathbb{R}^{N\times r}$ is the node representation at layer ℓ and $\mathbf{U}^{(0)} = \mathbf{X}$. In (1), node i encodes neighbor information via $\hat{\mathbf{A}}_{i*}\mathbf{U}^{(\ell-1)}\mathbf{W}_1^{(\ell)}$ and updates its own message together with $\mathbf{U}_{i*}^{(\ell-1)}\mathbf{W}_2^{(\ell)}$.

Different from existing GNN-LP methods which apply the same class of GNN models to learn representation for both head nodes and tail nodes, AML applies an MLP model to learn representation for tail nodes. However, naively applying MLP for tail nodes will deteriorate the accuracy. Hence, as in Chen et al. [2020], Klicpera et al. [2019], and Wu et al. [2019], we first pre-encode graph structure information into node features in the preprocessing step. The pre-encoding step is as follows:

$$\tilde{\mathbf{V}}^{(0)} = \hat{\mathbf{A}}^L\mathbf{X}. \tag{2}$$

To further improve the representation learning for tail nodes, we propose to transfer knowledge from head nodes to tail nodes by sharing parameters. The formula is as follows:

$$\tilde{\mathbf{V}}^{(\ell)} = f\left(\tilde{\mathbf{V}}^{(\ell-1)}\mathbf{W}_1^{(\ell)} + \tilde{\mathbf{V}}^{(\ell-1)}\mathbf{W}_2^{(\ell)}\right), \tag{3}$$

where we perform knowledge transfer between head nodes and tail nodes by sharing $\mathbf{W}_1^{(\ell)}$ and $\mathbf{W}_2^{(\ell)}$. Since sharing parameters somewhat restricts the expressiveness of $\tilde{\mathbf{V}}^{(L)}$, we propose to apply an additional MLP model to learn over the residual $\Delta^{(0)}$. Adding the residual $\Delta^{(L)}$ to $\tilde{\mathbf{V}}^{(L)}$, we obtain the representation for tail nodes which is shown as follows:

$$\Delta^{(0)} = \mathbf{X} - \hat{\mathbf{A}}^L\mathbf{X}, \tag{4}$$

$$\Delta^{(\ell)} = f\left(\Delta^{(\ell-1)}\mathbf{W}^{(\ell)}\right), \tag{5}$$

$$\mathbf{V}^{(L)} = \tilde{\mathbf{V}}^{(L)} + \Delta^{(L)}, \tag{6}$$

where $\mathbf{W}^{(\ell)} \in \mathbb{R}^{r\times r}$ is a learnable parameter at layer ℓ.

[1] We can also apply a GNN model to learn representation for tail nodes while applying an MLP model to learn representation for head nodes. For convenience, we only present the technical details of one case. The technical details for the reversed case are the same as the presented one.

Note that if BatchNorm [Ioffe and Szegedy 2015] is applied to $\mathbf{U}^{(\ell)}$ and $\tilde{\mathbf{V}}^{(\ell)}$, we keep individual parameters of BatchNorm for $\mathbf{U}^{(\ell)}$ and $\tilde{\mathbf{V}}^{(\ell)}$. Since $\mathbf{U}^{(\ell)}$ and $\tilde{\mathbf{V}}^{(\ell)}$ have different scales and lie in different representation spaces, it is reasonable to keep individual parameters of BatchNorm for them. Here, pre-encoding step only needs to be performed once and the resulting $\hat{\mathbf{A}}^L\mathbf{X}$ in (2) can be saved for the entire training process.

3.2 Learning Objective of AML

According to the definitions in (1) and (6), modeling links with $\mathbf{U}^{(L)}$ and $\mathbf{V}^{(L)}$ can capture directed relations while being a little difficult to capture undirected relations. Our solutions are twofold. Firstly, we formulate each undirected link by two directed ones with opposite directions. Secondly, motivated by the work in Li et al. [2011], in AML we propose to model both *homophily* and *stochastic equivalence* [Hoff 2007]. As a result, the formulas for the prediction of a pair (i, j) are as follows:

$$\mathbf{U} = \mathbf{U}^{(L)} + \mathbf{V}^{(L)}, \quad \mathbf{V} = \mathbf{V}^{(L)}, \tag{7}$$

$$S_{ij} = f_{\mathbf{\Theta}}\left(\mathbf{U}_{i*} \odot \mathbf{V}_{j*}\right), \tag{8}$$

where $\mathbf{U}$ and $\mathbf{V}$ are representation for head nodes and tail nodes, respectively. $\mathbf{V}^{(L)}$ in $\mathbf{U}$ is included to model the *homophily* feature in graph data. S_{ij} is the prediction for the pair (i, j). $\odot$ denotes element-wise multiplication. $f_{\mathbf{\Theta}}(\cdot)$ is an MLP model with parameter $\mathbf{\Theta}$. As in ELPH [Chamberlain et al. 2023], AML can also encode conditional information between head nodes and tail nodes by introducing additional structural features as inputs to the link prediction module $f_{\mathbf{\Theta}}(\cdot)$. AML can achieve this by introducing counts of generalized CN between head nodes and tail nodes as additional inputs to $f_{\mathbf{\Theta}}(\cdot)$ [Chamberlain et al. 2023].

Given node representation $\mathbf{U}$ and $\mathbf{V}$, the learning objective of link prediction is as follows:

$$\min_{\mathcal{W}} \frac{1}{|\mathcal{E}|} \sum_{(i,j)\in\mathcal{E}} f_{loss}(S_{ij}, Y_{ij}) + \frac{\lambda}{2} \sum_{\mathbf{W}\in\mathcal{W}} \|\mathbf{W}\|_F^2, \tag{9}$$

where $\mathcal{E}$ denotes the training set. $\mathcal{W}$ is the set of all learnable parameters. Y_{ij} is the ground-truth label for the pair (i, j). $f_{loss}(\cdot, \cdot)$ is a loss function, such as cross-entropy loss. λ is a coefficient for the regularization term of $\mathcal{W}$. $\|\cdot\|_F$ denotes the Frobenius norm of a matrix.

3.3 Row-wise Sampling for Generating Mini-Batch

Like most deep learning methods, existing GNN-LP methods are typically trained in a mini-batch manner. But most GNN-LP methods adopt an edge-wise sampling strategy to generate mini-batch for training. In particular, they first randomly sample a mini-batch of edges $\mathcal{E}_\Omega$ from $\mathcal{E}$ at each iteration and then optimize the objective function based on $\mathcal{E}_\Omega$. For example, if we adopt the edge-wise sampling strategy for the objective function of AML in (9), the corresponding learning objective at each iteration will be as follows:

$$\min_{\mathcal{W}} \frac{1}{|\mathcal{E}_\Omega|} \sum_{(i,j)\in\mathcal{E}_\Omega} f_{loss}(S_{ij}, Y_{ij}) + \frac{\lambda}{2} \sum_{\mathbf{W}\in\mathcal{W}} \|\mathbf{W}\|_F^2. \tag{10}$$

Suppose $\mathcal{E}_\Omega = \{(i_1, j_1), \ldots, (i_B, j_B)\}$ with B as the mini-batch size. We, respectively, use C and M to denote the computation complexity for generating a node representation by GNN and MLP. Since $\mathcal{E}_\Omega$ is edge-wise randomly sampled from $\mathcal{E}$ and N is of a large value for large-scale graphs, $(i_1, \ldots, i_B)$ will have a relatively small number of repeated nodes. As a result, the edge-wise sampling strategy for AML has a computation complexity of $O(|\mathcal{E}| \cdot (C + M))$ for each epoch, which

ALGORITHM 1: Learning Algorithm for AML

Require: N (number of nodes in the input graph), L (number of model layers), $\mathcal{E}$ (the training set), B (mini-batch size), T (maximum number of epochs).
Ensure: $\mathcal{W}$ (model parameters).

```
Pre-encoding graph structure by (2);
B̃ = B/(|E|/N);
for t = 1 : T do
    for q = 1 : (N/B̃) do
        Sample V_Ω of B̃ head nodes from {1, ..., N};
        Generate E_Ω with V_Ω by (11);
        Compute U_{i*}^{(L)} for each node i in V_Ω by (1);
        Compute V_{j*}^{(L)} for each node j in E_Ω by (3)–(6);
        Compute U_{i*} and V_{j*} for each (i, j) in E_Ω by (7);
        Compute S_ij for each (i, j) in E_Ω by (8);
        Update model parameters W by optimizing (10); /*Note that the objective function of row-wise
        sampling is the same as that of edge-wise sampling in (10).*/
    end for
end for
```

has the same order of magnitude in computation complexity as existing GNN-LP methods and hence is undesirable.

To solve this high computation complexity problem of edge-wise sampling strategy adopted by existing GNN-LP methods, in AML we apply a row-wise sampling strategy for generating mini-batch. More specifically, we first sample a number of row indices $\mathcal{V}_\Omega$ (head nodes) from $\{1, 2, \ldots, N\}$ for each mini-batch iteration. Then we construct the mini-batch $\mathcal{E}_\Omega$ as follows:

$$\mathcal{E}_\Omega = \{(i, j) | i \in \mathcal{V}_\Omega \wedge (i, j) \in \mathcal{E}\}. \tag{11}$$

By applying this row-wise sampling strategy, AML has a computation complexity of $O(|\mathcal{V}_\Omega| \cdot C + (|\mathcal{V}_\Omega|/N)|\mathcal{E}| \cdot M)$ for each mini-batch iteration. Since $\mathcal{V}_\Omega$ iterates over $\{1, \ldots, N\}$ for $N/|\mathcal{V}_\Omega|$ times to go through the whole training set, this row-wise sampling strategy for AML has a computation complexity of $O(N \cdot C + |\mathcal{E}| \cdot M)$ for each epoch. Therefore, the row-wise sampling strategy enables AML to decouple the factor of $|\mathcal{E}|$ from the computation complexity of GNN, leading to a complexity reduction by orders of magnitude compared to the edge-wise sampling strategy.

Algorithm 1 summarizes the whole learning algorithm for AML.

3.4 Complexity Analysis

The computation complexity for different methods are summarized in Table 1. Here, we assume N is large such that we cannot compute all node representations at once by feeding the entire input graph into a GNN model without exceeding the graphics memory capacity. In such a case, we need to sample neighboring nodes layer by layer for a batch of target nodes when applying a GNN model to compute node representations. As a result, a GNN model has an exponential growth in computation complexity when applied to large-scale graphs. Therefore, we often have $s^L < N$ and $s^k < N$ for large-scale graphs. Typically, $s^L r^2$ has the same order of magnitude as $Ls^k r^2$. According to (1), GNN has a computation complexity of $C = O(s^L \cdot r^2)$ to generate a node representation while the corresponding computation complexity of MLP is $M = O(L \cdot r^2)$ according to (3)–(6). It is easy to verify that $M \ll C$. Although many works [Hamilton et al. 2017; Yao and Li 2021; Zeng et al. 2020; Zou et al. 2019] have proposed solutions to reduce C, C is still much larger than M. Here,

Table 1. Complexity Analysis

Method	Computation complexity	Memory complexity
Local GNN-LP	$O\left(Ls^k r^2 \cdot \lvert\mathcal{E}\rvert\right)$	$O\left(BLs^k r + Lr^2\right)$
Local GNN-LP (w/RWS)	$O\left(Ls^k r^2 \cdot \lvert\mathcal{E}\rvert\right)$	$O\left(BLs^k r + Lr^2\right)$
Global GNN-LP	$O\left(s^L r^2 \cdot \lvert\mathcal{E}\rvert\right)$	$O\left(2Bs^L r + Lr^2\right)$
Global GNN-LP (w/RWS)	$O\left(s^L r^2 \cdot (\lvert\mathcal{E}\rvert + N)\right)$	$O\left(Bs^L r + Lr^2\right)$
AML (w/o RWS)	$O\left(\left(s^L r^2 + Lr^2\right) \cdot \lvert\mathcal{E}\rvert\right)$	$O\left(Bs^L r + Lr^2\right)$
AML	$O\left(s^L r^2 \cdot N + Lr^2 \cdot \lvert\mathcal{E}\rvert\right)$	$O\left(\frac{1}{\lvert\mathcal{E}\rvert/N} \cdot Bs^L r + BLr + Lr^2\right)$

Computation complexity is analyzed within an epoch, while memory complexity is analyzed within a mini-batch training step. B is the mini-batch size. L is the number of model layers. $s = \|\mathbf{A}\|_0/N$ is the average number of neighbors for a node in $\mathcal{G}$. r is the dimension of node representation. $|\mathcal{E}|$ is the number of links in the training set $\mathcal{E}$. N is the number of nodes in graph $\mathcal{G}$. k in local GNN-LP is the number of hops for the enclosed subgraphs. "RWS" denotes row-wise sampling. "w/ RWS" in local GNN-LP and global GNN-LP means generating mini-batch with RWS. "w/o RWS" in AML means generating mini-batch without RWS but with an edge-wise sampling strategy.

we suppose all methods are trained in a mini-batch manner which has been adopted by almost all deep learning models including GNN.

From Table 1, we can get the following results. Firstly, AML has a computation complexity of $O(N \cdot C + |\mathcal{E}| \cdot M)$, which is much lower than $O(|\mathcal{E}| \cdot C)$ of existing GNN-LP methods. Secondly, even with our proposed row-wise sampling strategy, existing GNN-LP methods still have a computation complexity of $O((|\mathcal{E}| + N) \cdot C)$, without change in the order of magnitude. The reason is that they still need to perform $O(|\mathcal{E}|)$ times of GNN computation for tail nodes within each epoch. Thirdly, AML achieves $O(|\mathcal{E}|/N)$ lower memory complexity than existing global GNN-LP methods, where N is usually an order of magnitude smaller than $|\mathcal{E}|$. During inference, where we only need to generate a representation for one head node, AML has a memory complexity comparable to existing global GNN-LP methods. However, AML is superior to other methods during mini-batch training, which is the focus of this paper. Existing global GNN-LP methods must generate representations for $O(B)$ head nodes using a GNN model. By contrast, AML only needs to generate representations for $B/(|\mathcal{E}|/N)$ head nodes using a GNN model (Line 2 and Line 5 in Algorithm 1).

It is worth noting that AML performs pre-encoding graph structure only once in the preprocessing step, without repeating during training. It is because $\hat{\mathbf{A}}^L\mathbf{X}$ can be saved as $\tilde{\mathbf{V}}^{(0)}$ in (2), and AML can use $\tilde{\mathbf{V}}^{(0)}$ as a direct result during training without the need for recalculation. It is evident that the computation overhead during training is at least T times larger than the computation overhead for calculating $\hat{\mathbf{A}}^L\mathbf{X}$ in the preprocessing step, where T is the maximum number of epochs in Algorithm 1. Therefore, pre-encoding graph structure introduces negligible computation overhead for AML.

3.5 Comparison to Related Works

Some existing studies [Kong et al. 2022; Zhang and Yao 2022; Zhu et al. 2021] have also applied a row-wise sampling strategy for generating mini-batch. However, an asymmetric learning strategy to learn node representations in AML is not applied in existing studies. The asymmetric learning strategy introduces a new learning paradigm for GNN-LP and constitutes the primary contribution of this paper. It is worth noting that a naive asymmetric learning strategy alone cannot guarantee a model with promising accuracy and training efficiency, as will be shown in Table 5. Therefore, we design a novel model architecture to ensure high model accuracy, including knowledge transfer from head nodes to tail nodes, encoding residual information, modeling homophily property and pre-encoding graph structure. We will demonstrate in experiments

Table 2. Statistics of Datasets

Datasets	ogbl-collab	ogbl-ppa	ogbl-citation2
#Nodes	235,868	576,289	2,927,963
#Edges	1,285,465	30,326,273	30,561,187
#Features/Node	128	128	128
#Training links	1,179,052	21,231,931	30,387,995
#Validation links	160,084	9,062,562	86,682,596
#Test links	146,329	6,031,780	86,682,596
Metric	Hits@50	Hits@100	MRR

that each model component is indispensable for ensuring high model accuracy. Moreover, we must apply a row-wise mini-batch sampling strategy to achieve promising training efficiency, as shown in Table 1. Hence, making the asymmetric learning strategy really work is challenging and non-trivial, constituting another important contribution of this paper. Since methods proposed by Kong et al. [2022], Zhang and Yao [2022], and Zhu et al. [2021] have a computation complexity of at least $O(C \cdot |\mathcal{E}|)$, AML is at least $O(|\mathcal{E}|/N)$ times more computation-efficient than them.

4 EXPERIMENTS

In this section, we evaluate AML and baselines on three real datasets. All methods are implemented with Pytorch [Paszke et al. 2019] and Pytorch-Geometric Library [Fey and Lenssen 2019]. All experiments are run on an NVIDIA RTX A6000 GPU server with 48 GB of graphics memory.

4.1 Settings

4.1.1 Datasets. Datasets for evaluation include ogbl-collab,[2] ogbl-ppa and ogbl-citation2.[3] The first two are medium-scale datasets with hundreds of thousands of nodes. The last one is a large-scale dataset with millions of nodes. For ogbl-ppa, since the provided node features are uninformative, we apply matrix factorization [Menon and Elkan 2011] to generate new features for nodes. The first two datasets are for undirected link prediction, while the last one is for directed link prediction. The statistics of datasets are summarized in Table 2. Since most GNN-LP methods adopt the evaluation metrics provided by Hu et al. [2020], we also follow these evaluation settings.

4.1.2 Baselines. AML is actually a global GNN-LP method. We first compare AML with existing global GNN-LP baselines by adopting the same GNN for both AML and baselines. Since almost all existing global GNN-LP methods are developed based on the graph autoencoder framework proposed in Kipf and Welling [2016], we mainly adopt the GNNs under the graph autoencoder framework. In particular, we adopt SAGE [Hamilton et al. 2017] and GAT [Velickovic et al. 2018] as the GNNs for both AML and baselines, because SAGE and GAT are respectively representative non-attention based and attention based GNN models under the graph autoencoder framework.

Then, we compare AML with non-GNN baselines and local GNN-LP baselines. Non-GNN baselines include common neighbors (CN) [Newman 2001], Adamic-Adar (AA) [Adamic and Adar 2003], Node2vec [Grover and Leskovec 2016] and matrix factorization (MF) [Menon and Elkan 2011]. Local GNN-LP baselines inlcude DE-GNN [Li et al. 2020] and SEAL [Zhang and Chen 2018; Zhang et al. 2021a]. For local GNN-LP methods, we extract enclosed subgraphs in an online way to simulate large-scale settings by following the original work [Zhang et al. 2021a].

[2]In ogbl-collab, there are data leakage issues in the provided graph adjacency matrix $\mathbf{A}$. We remove those positive links in the validation and testing set from $\mathbf{A}$.
[3]https://ogb.stanford.edu/docs/linkprop/

Table 3. Comparison with Global GNN-LP Baselines

(a) SAGE as the GNN model.

Methods	ogbl-collab				ogbl-ppa			
	Hits@50 (%) ↑	Gap	Time (s) ↓	Mem (MB) ↓	Hits@100 (%) ↑	Gap	Time (s) ↓	Mem (MB) ↓
SAGE	54.57 ± 0.82	0	7.9×10^3	591	**50.13 ± 0.55**	0	1.0×10^5	1489
AML	**57.26 ± 1.25**	+2.69	$\mathbf{4.4 \times 10^3}$	**301**	49.73 ± 0.89	−0.40	$\mathbf{3.0 \times 10^4}$	**781**

Methods	ogbl-citation2			
	MRR (%) ↑	Gap	Time (s) ↓	Mem (MB) ↓
SAGE	86.39 ± 0.15	0	7.3×10^4	3009
AML	**86.55 ± 0.06**	+0.16	$\mathbf{1.0 \times 10^4}$	**428**

(b) GAT as the GNN model.

Methods	ogbl-collab				ogbl-ppa			
	Hits@50 (%) ↑	Gap	Time (s) ↓	Mem (MB) ↓	Hits@100 (%) ↑	Gap	Time (s) ↓	Mem (MB) ↓
GAT	56.43 ± 0.86	0	7.8×10^3	591	49.71 ± 0.48	0	1.0×10^5	1489
AML	**57.60 ± 0.71**	+1.17	$\mathbf{4.5 \times 10^3}$	**301**	**50.23 ± 0.78**	+0.52	$\mathbf{3.2 \times 10^4}$	**781**

Methods	ogbl-citation2			
	MRR (%) ↑	Gap	Time (s) ↓	Mem (MB) ↓
GAT	86.50 ± 0.20	0	7.4×10^4	3009
AML	**86.70 ± 0.05**	+0.20	$\mathbf{1.0 \times 10^4}$	**428**

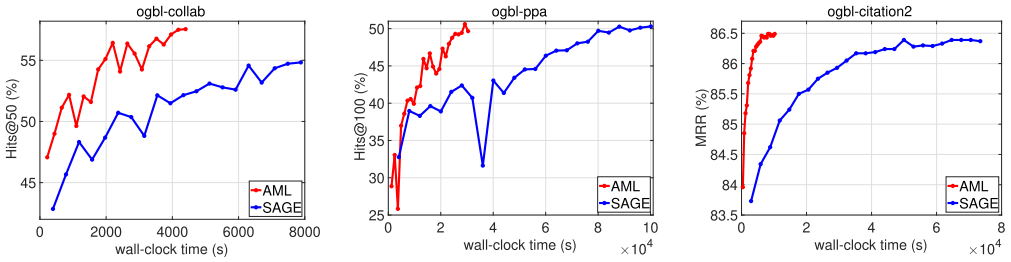
"Time" denotes the whole time to complete training. "Gap" denotes the accuracy of AML minus that of baselines. "Mem" denotes the memory footprint within a mini-batch training step, where we only consider the memory footprint during forward computation.

4.1.3 Hyper-parameter Settings. Hyper-parameters include L (layer number), r (hidden dimension), λ (coefficient for the regularization of parameters), T (maximum number of epochs), η (learning rate) and B (mini-batch size). On ogbl-collab, $L = 3$, $r = 256$, $\lambda = 0$, $T = 400$, $\eta = 0.001$, and $B = 65{,}536$. On ogbl-ppa, $L = 3$, $r = 256$, $\lambda = 0$, $T = 50$, $\eta = 0.01$, and $B = 65{,}536$. On ogbl-citation2, $L = 3$, $r = 256$, $\lambda = 0$, $T = 50$, $\eta = 0.005$, and $B = 65{,}536$. We use Adam [Kingma and Ba 2015] as the optimizer. We use GraphNorm [Cai et al. 2021] to accelerate the training. We adopt BNS [Yao and Li 2021] as the neighbor sampling strategy for large-scale training. In BNS, there are three hyper-parameters, including $\tilde{s}_0$, $\tilde{s}_1$ and δ. $\tilde{s}_0$ denotes the number of sampled neighbors for nodes at output layer. $\tilde{s}_1$ denotes the number of sampled neighbors for nodes at other layers. δ denotes the ratio of blocked neighbors. On ogbl-collab, $\tilde{s}_0 = 7$, $\tilde{s}_1 = 2$, $\delta = 1/2$. On ogbl-ppa, $\tilde{s}_0$ equals the number of all neighbors, $\tilde{s}_1 = 4$, $\delta = 5/6$. On ogbl-citation2, $\tilde{s}_0 = 7$, $\tilde{s}_1 = 2$, $\delta = 1/2$. It is worth noting that all GNN-LP methods can adopt other scalable GNN training methods as an alternative to BNS. We run each setting for five times and report the mean with standard deviation.

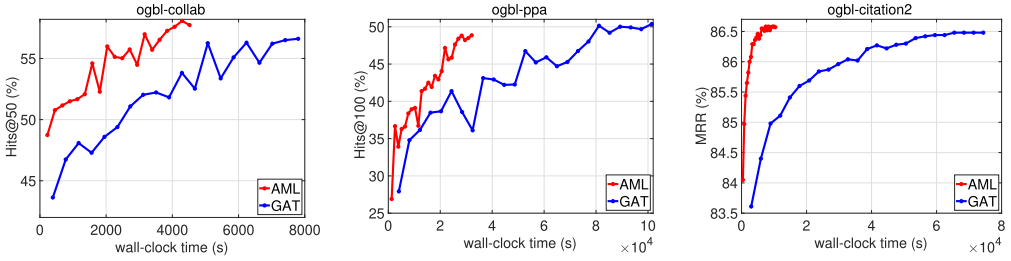
4.2 Comparison with Baselines

4.2.1 Comparison with Global GNN-LP Baselines. Comparison with global GNN-LP baselines is shown in Table 3 and Figure 3. According to the results, we can draw the following conclusions. Firstly, AML has almost no accuracy loss in all cases compared to baselines.[4] For example, AML's accuracy is within the standard deviation of baselines' accuracy on ogbl-ppa and ogbl-citation2. Instead, AML achieves an accuracy gain of 1.17%∼2.69% on ogbl-collab compared to baselines.

[4]When AML's accuracy is within the standard deviation of baselines' accuracy, we say that AML achieves almost no accuracy loss compared to baselines.



(a) SAGE as the GNN model.



(b) GAT as the GNN model.

Fig. 3. Test accuracy-time curves of AML and global GNN-LP baselines.

Secondly, AML is about 1.7×~7.3× faster in training than baselines when achieving almost no accuracy loss. For example, AML is 1.7× faster than baselines on ogbl-collab, 3.1× faster on ogbl-ppa, and 7.3× faster on ogbl-citation2. In particular, baselines need about 5.8 days and 4.2 days to achieve the mean of results while AML only needs 1.7 days and 0.6 days to achieve the mean of results, on ogbl-ppa and ogbl-citation2, respectively. Thirdly, the speedup of AML relative to baselines increases with the size of graphs. For example, the number of nodes in ogbl-collab, ogbl-ppa, and ogbl-citation2 increases in an ascending order, and the speedup of AML relative to baselines increases in a consistent order on these three graph datasets. Finally, AML has a better accuracy-time trade-off than baselines, which can be concluded from Figure 3. For example, we can find that AML is faster than baselines when achieving the same accuracy.

4.2.2 Comparison with Non-GNN and Local GNN-LP Baselines. Comparison with non-GNN and local GNN-LP baselines is shown in Table 4. According to the results, we can draw the following conclusions. Firstly, AML is comparable with state-of-the-art local GNN-LP methods in accuracy. For example, AML has almost no accuracy loss compared to the best baseline DE-GNN on obgl-collab. On ogbl-ppa, AML achieves an accuracy gain of 3.43% compared to the best baseline SEAL. On ogbl-citation2, AML achieves an accuracy loss less than 1% compared to the best baseline SEAL. Secondly, AML is about 13×~110× faster than local GNN-LP baselines. For example, AML is about 13× faster than DE-GNN and SEAL on ogbl-collab, about 66× faster on ogbl-ppa, and about 110× faster on ogbl-citation2. In particular, DE-GNN and SEAL need 3.5 days to achieve the mean accuracy on ogbl-collab which is a relatively small-scale dataset. By contrast, AML only needs 1.2 hours to achieve the mean accuracy. Finally, GNN-LP methods can achieve better accuracy than non-GNN methods. For example, DE-GNN improves by 13% in accuracy over MF on ogbl-ppa and improves by 17% over Node2vec on ogbl-citation2. AML improves by 17% in accuracy over MF on ogbl-ppa and improves by 16% over Node2vec on ogbl-citation2.

Table 4. Comparison with Non-GNN and Local GNN-LP Baselines

Category	Methods	ogbl-collab		ogbl-ppa		ogbl-citation2	
		Hits@50(%)↑	Time(s)↓	Hits@100(%)↑	Time(s)↓	MRR(%)↑	Time(s)↓
Non-GNN	CN	49.96 ± 0.00*	-	27.60 ± 0.00	-	51.47 ± 0.00	-
	AA	56.49 ± 0.00*	-	32.45 ± 0.00	-	51.89 ± 0.00	-
	Node2vec	49.29 ± 0.64*	-	22.26 ± 0.88	-	61.41 ± 0.11	-
	MF	37.93 ± 0.76*	-	32.29 ± 0.94	-	51.86 ± 4.43	-
Local GNN-LP	DE-GNN	**57.87 ± 0.79***	6.1×10^4	45.70 ± 3.46	2.0×10^6	78.85 ± 0.17	1.1×10^6
	SEAL	57.55 ± 0.72*	6.5×10^4	48.80 ± 3.16	2.0×10^6	**87.67 ± 0.32**	1.1×10^6
Global GNN-LP	AML (SAGE)	57.26 ± 1.25	$\mathbf{4.4 \times 10^3}$	49.73 ± 0.89	$\mathbf{3.0 \times 10^4}$	86.55 ± 0.06	$\mathbf{1.0 \times 10^4}$
	AML (GAT)	57.60 ± 0.71	4.5×10^3	**50.23 ± 0.78**	3.2×10^4	86.70 ± 0.05	$\mathbf{1.0 \times 10^4}$

"(SAGE)" in AML means using SAGE as the GNN model, and "(GAT)" in AML means using GAT as the GNN model. Accuracy of non-GNN and local GNN-LP baselines on ogbl-ppa and ogbl-citation2 is from Zhang et al. [2021a]. "*" denotes the results achieved by rerunning the authors' code on the clean ogbl-collab.

Table 5. Ablation Study

(a) SAGE as the base GNN model.

Methods	ogbl-collab		ogbl-ppa		ogbl-citation2	
	Hits@50 (%) ↑	Gap	Hits@100 (%) ↑	Gap	MRR (%) ↑	Gap
AML	**57.26 ± 1.25**	0	**49.73 ± 0.89**	0	**86.55 ± 0.06**	0
AML - w/o KT	54.50 ± 0.90	−2.76	49.16 ± 1.07	−0.57	86.24 ± 0.05	−0.31
AML - w/o $\Delta^{(L)}$	54.61 ± 0.62	−2.65	49.00 ± 0.26	−0.73	86.26 ± 0.02	−0.29
AML - w/o Homophily	55.70 ± 1.75	−1.56	47.74 ± 0.66	−1.99	85.75 ± 0.10	−0.80
AML - w/o Pre-encoding	43.90 ± 0.78	−13.36	1.97 ± 0.27	−47.76	60.82 ± 0.14	−25.73

(b) GAT as the base GNN model.

Methods	ogbl-collab		ogbl-ppa		ogbl-citation2	
	Hits@50 (%) ↑	Gap	Hits@100 (%) ↑	Gap	MRR (%) ↑	Gap
AML	**57.60 ± 0.71**	0	**50.23 ± 0.78**	0	**86.70 ± 0.05**	0
AML - w/o KT	54.60 ± 0.60	−3.00	47.38 ± 0.28	−2.85	86.28 ± 0.05	−0.42
AML - w/o $\Delta^{(L)}$	54.56 ± 1.43	−3.04	47.04 ± 1.03	−3.19	86.30 ± 0.01	−0.40
AML - w/o Homophily	55.73 ± 0.34	−1.87	47.86 ± 1.28	−2.37	85.83 ± 0.04	−0.87
AML - w/o Pre-encoding	43.95 ± 1.01	−13.65	2.95 ± 0.35	−47.28	60.89 ± 0.22	−25.81

"KT" represents knowledge transfer. "$\Delta^{(L)}$" indicates the residual variable defined in (6). "Homophily" means modeling the homophily property within graphs. "Pre-encoding" means pre-encoding graph structure. "Gap" denotes the accuracy of variants of AML minus that of AML.

4.3 Ablation Study

In this subsection, we study the effectiveness of different components in AML, including knowledge transfer, residual term $\Delta^{(L)}$, pre-encoding graph structure and modeling the homophily. Results are shown in Table 5. We can find the following phenomenons. Firstly, knowledge transfer between head nodes and tail nodes effectively improves the accuracy of AML. For example, knowedge transfer can improve the accuracy of AML by 3.00% on ogbl-collab, by 2.85% on ogbl-ppa and by 0.42% on ogbl-citation2. Secondly, $\Delta^{(L)}$ is beneficial for AML. For example, including $\Delta^{(L)}$ in AML can improve accuracy by 3.04% on ogbl-collab, by 3.19% on ogbl-ppa and by 0.40% on ogbl-citation2. Thirdly, modeling homophily is helpful for AML. For example, modeling homophily in AML can improve accuracy by 1.87% on ogbl-collab, by 2.37% on ogbl-ppa and by 0.87% on ogbl-citation2. Finally, pre-encoding graph structure plays a crucial role in AML. For example, AML has

Table 6. Experiments to Verify the Necessity of GNN in AML

Methods	ogbl-collab		ogbl-ppa		ogbl-citation2	
	Hits@50 (%) ↑	Gap	Hits@100 (%) ↑	Gap	MRR (%) ↑	Gap
SMLP	47.25 ± 0.89	0	47.42 ± 1.37	0	69.82 ± 0.05	0
AML (SAGE)	57.26 ± 1.25	+10.01	49.73 ± 0.89	+2.31	86.55 ± 0.06	+16.73
AML (GAT)	**57.60 ± 0.71**	+10.35	**50.23 ± 0.78**	+2.81	**86.70 ± 0.05**	+16.88

SMLP applies MLP with pre-encoding to learn representation for both head nodes and tail nodes. "Gap" denotes the accuracy of AML minus that of SMLP.

Table 7. Reversed Asymmetric Learning

Methods	ogbl-collab	ogbl-ppa	ogbl-citation2
	Hits@50 (%) ↑	Hits@100 (%) ↑	MRR (%) ↑
AML-R (SAGE)	57.15 ± 0.32	49.73 ± 0.45	85.70 ± 0.10
AML (SAGE)	57.26 ± 1.25	49.73 ± 0.89	86.55 ± 0.06
AML-R (GAT)	57.08 ± 1.19	**50.30 ± 0.61**	85.91 ± 0.04
AML (GAT)	**57.60 ± 0.71**	50.23 ± 0.78	**86.70 ± 0.05**

AML-R denotes the reversed case of AML, which learns representation for head nodes with an MLP model while learning representation for tail nodes with a GNN model.

Table 8. Ability to Encode Conditional Information

(a) SAGE as the base GNN model.

Methods	ogbl-collab		ogbl-ppa		ogbl-citation2	
	Hits@50 (%) ↑	Gap	Hits@100 (%) ↑	Gap	MRR (%) ↑	Gap
ELPH	62.77 ± 0.52	0	51.05 ± 0.04	0	**88.00 ± 0.11**	0
AML-ELPH	**63.84 ± 0.57**	+1.07	**53.87 ± 0.13**	+2.82	87.78 ± 0.08	−0.22

(b) GAT as the base GNN model.

Methods	ogbl-collab		ogbl-ppa		ogbl-citation2	
	Hits@50 (%) ↑	Gap	Hits@100 (%) ↑	Gap	MRR (%) ↑	Gap
ELPH	62.80 ± 0.66	0	53.60 ± 0.34	0	87.88 ± 0.11	0
AML-ELPH	**63.44 ± 0.47**	+0.64	**53.70 ± 0.71**	+0.10	**88.00 ± 0.06**	+0.12

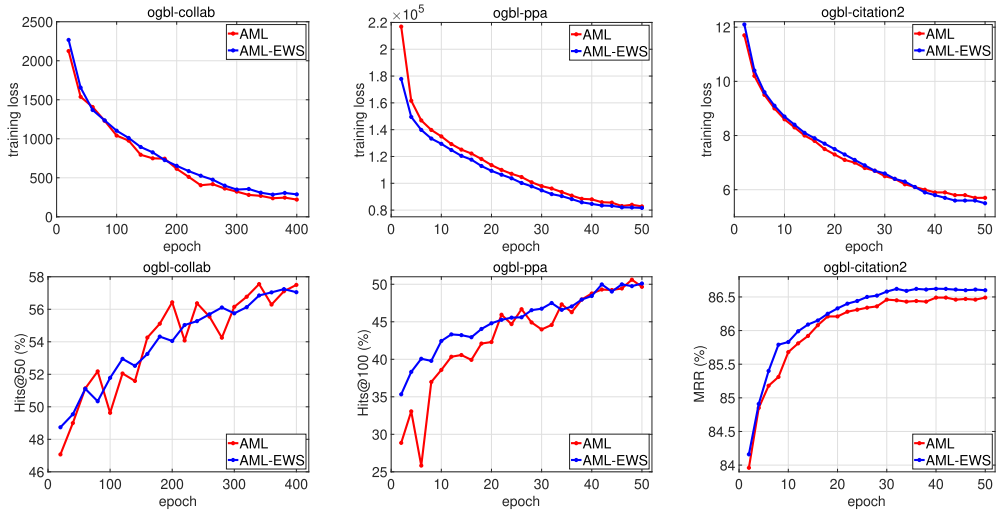
AML-ELPH represents the method that applies AML to ELPH. "Gap" denotes the accuracy of AML-ELPH minus that of baselines. Here we omit training time and memory footprint because those results are close to that in Table 3.

an accuracy loss of about 13% on ogbl-collab, 47% on ogbl-ppa, and 25% on ogbl-citation2 without pre-encoding graph structure.

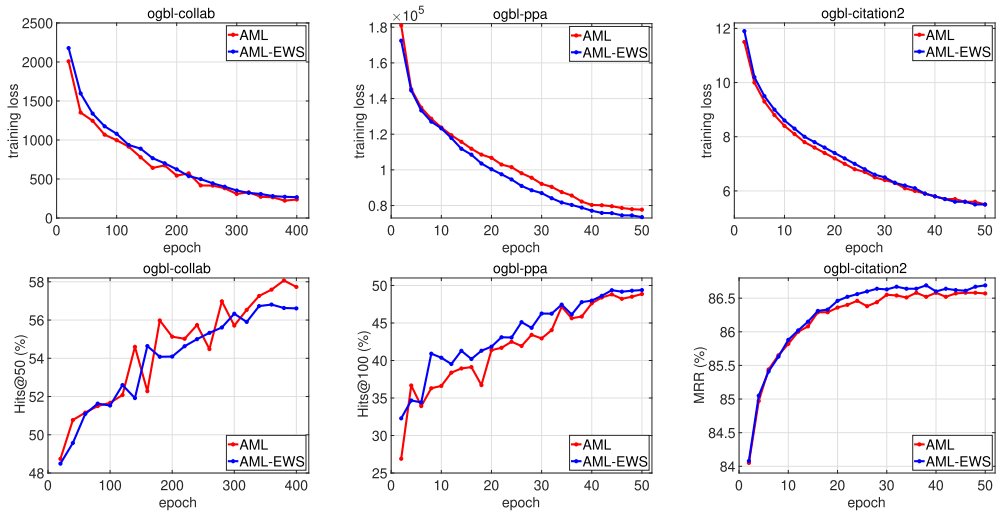
4.4 Necessity of GNN in AML

Here, we perform experiment to verify the necessity of GNN in AML. We design a method called symmetric MLP (SMLP), which applies MLP with pre-encoding to learn representation for both head nodes and tail nodes. More specifically, SMLP adopts the techniques for learning tail node representation in AML, that is, (2) and (3), to learn representation for both head nodes and tail nodes.

Results are shown in Table 6. We can find that SMLP is much worse than AML on all datasets. This shows that training a GNN model for node representation learning is necessary to achieve high accuracy.



(a) SAGE as the GNN model.



(b) GAT as the GNN model.

Fig. 4. Effect of sampling strategy. We present accuracy-epoch curves and loss-epoch curves of both AML and AML-EWS. AML-EWS represents the method that replaces the row-wise sampling strategy in AML with an edge-wise sampling strategy.

4.5 Reversed Asymmetric Learning

In above experiments, AML learns representation for head nodes with a GNN model while learning representation for tail nodes with an MLP model. Here, we verify whether the reversed case can also behave well. We denote the reversed case as AML-R, which learns representation for head nodes with an MLP model while learning representation for tail nodes with a GNN model. Results are shown in Table 7. Results show that AML and AML-R have similar accuracy.

4.6 Ability to Encode Conditional Information

This subsection verifies that AML can also encode conditional information between head nodes and tail nodes when applied to ELPH [Chamberlain et al. 2023]. To this end, we apply AML to ELPH by introducing counts of generalized common neighbors between head nodes and tail nodes as additional inputs to $f_\Theta(\cdot)$ in (8) and denote such a method as AML-ELPH. Experimental results are summarized in Table 8. We can find that AML-ELPH achieves no accuracy loss in five out of six cases compared to ELPH.

4.7 Effect of Sampling Strategy

A row-wise mini-batch sampling strategy may incur biased estimation for gradients. This subsection analyzes the impact on model accuracy and training convergence caused by such a biased estimation. To this end, we replace the row-wise sampling strategy in AML with an edge-wise sampling strategy and denote such a method as AML-EWS. We present accuracy-epoch curves and loss-epoch curves in Figure 4. By comparing AML with AML-EWS, we can find that the biased estimation caused by a row-wise mini-batch sampling strategy has minimal impact on accuracy performance and training convergence with respect to epoch. It is worth noting that a row-wise mini-batch sampling strategy is beneficial for AML to reduce training time.

5 CONCLUSIONS

Graph neural network based link prediction (GNN-LP) methods have achieved better accuracy than non-GNN based link prediction methods but suffer from scalability problem for large-scale graphs. Our computation complexity analysis reveals that one reason for the scalability problem of existing GNN-LP methods stems from their symmetric learning strategy for node representation learning. Motivated by this finding, we propose a novel method, called asymmetric learning (AML), for GNN-LP. Furthermore, we design a novel model architecture and apply a row-wise mini-batch sampling strategy to ensure promising model accuracy and training efficiency for AML. To the best of our knowledge, AML is the first GNN-LP method to adopt an asymmetric learning strategy for node representation learning. Extensive experiments show that AML is significantly faster than baselines with a symmetric learning strategy while having almost no accuracy loss.

REFERENCES

Lada A. Adamic and Eytan Adar. 2003. Friends and neighbors on the web. *Social Networks* 25, 3 (2003), 211–230.

Joan Bruna, Wojciech Zaremba, Arthur Szlam, and Yann LeCun. 2014. Spectral networks and locally connected networks on graphs. In *Proceedings of International Conference on Learning Representations*.

Tianle Cai, Shengjie Luo, Keyulu Xu, Di He, Tie-Yan Liu, and Liwei Wang. 2021. GraphNorm: A principled approach to accelerating graph neural network training. In *Proceedings of International Conference on Machine Learning*.

Benjamin Paul Chamberlain, Sergey Shirobokov, Emanuele Rossi, Fabrizio Frasca, Thomas Markovich, Nils Hammerla, Michael M. Bronstein, and Max Hansmire. 2023. Graph neural networks for link prediction with subgraph sketching. In *Proceedings of International Conference on Learning Representations*.

Jie Chen, Tengfei Ma, and Cao Xiao. 2018a. FastGCN: Fast learning with graph convolutional networks via importance sampling. In *Proceedings of International Conference on Learning Representations*.

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

Ming Chen, Zhewei Wei, Bolin Ding, Yaliang Li, Ye Yuan, Xiaoyong Du, and Ji-Rong Wen. 2020. Scalable graph neural networks via bidirectional propagation. In *Proceedings of Advances in Neural Information Processing Systems*.

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

Weilin Cong, Rana Forsati, Mahmut T. Kandemir, and Mehrdad Mahdavi. 2020. Minimal variance sampling with provable guarantees for fast training of graph neural networks. In *Proceedings of ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*.

Matthias Fey and Jan E. Lenssen. 2019. Fast graph representation learning with PyTorch Geometric. In *Proceedings of International Conference on Learning Representations Workshop*.

Matthias Fey, Jan Eric Lenssen, Frank Weichert, and Jure Leskovec. 2021. GNNAutoScale: Scalable and expressive graph neural networks via historical embedddings. In *Proceedings of International Conference on Machine Learning*.

Hongyang Gao, Zhengyang Wang, and Shuiwang Ji. 2018. Large-scale learnable graph convolutional networks. In *Proceedings of ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*.

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

Anna Goldenberg, Alice X. Zheng, Stephen E. Fienberg, and Edoardo M. Airoldi. 2009. A survey of statistical network models. *Foundations and Trends in Machine Learning* 2, 2 (2009), 129–233.

Marco Gori, Gabriele Monfardini, and Franco Scarselli. 2005. A new model for learning in graph domains. In *Proceedings of International Joint Conference on Neural Networks*.

Aditya Grover and Jure Leskovec. 2016. node2vec: Scalable feature learning for networks. In *Proceedings of ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*.

Roger Guimerà and Marta Sales-Pardo. 2009. Missing and spurious interactions and the reconstruction of complex networks. *Proceedings of National Academy of Sciences* 106, 52 (2009), 22073–22078.

William L. Hamilton, Zhitao Ying, and Jure Leskovec. 2017. Inductive representation learning on large graphs. In *Proceedings of International Conference on Neural Information Processing Systems*.

Arman Hasanzadeh, Ehsan Hajiramezanali, Krishna R. Narayanan, Nick Duffield, Mingyuan Zhou, and Xiaoning Qian. 2019. Semi-implicit graph variational auto-encoders. In *Proceedings of International Conference on Neural Information Processing Systems*.

Peter D. Hoff. 2007. Modeling homophily and stochastic equivalence in symmetric relational data. In *Proceedings of International Conference on Neural Information Processing Systems*.

Weihua Hu, Matthias Fey, Marinka Zitnik, Yuxiao Dong, Hongyu Ren, Bowen Liu, Michele Catasta, and Jure Leskovec. 2020. Open graph benchmark: Datasets for machine learning on graphs. In *Proceedings of International Conference on Neural Information Processing Systems*.

Sergey Ioffe and Christian Szegedy. 2015. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *Proceedings of International Conference on Machine Learning*.

Wei Jin, Xiaorui Liu, Yao Ma, Charu C. Aggarwal, and Jiliang Tang. 2022. Feature overcorrelation in deep graph neural networks: A new perspective. In *Proceedings of ACM SIGKDD Conference on Knowledge Discovery and Data Mining*.

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

Thomas N. Kipf and Max Welling. 2016. Variational graph auto-encoders. In *Proceedings of Advances in Neural Information Processing Systems Workshop on Bayesian Deep Learning*.

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

Johannes Klicpera, Stefan Weißenberger, and Stephan Günnemann. 2019. Diffusion improves graph learning. In *Proceedings of Advances in Neural Information Processing Systems*.

Lecheng Kong, Yixin Chen, and Muhan Zhang. 2022. Geodesic graph neural network for efficient graph representation learning. In *Proceedings of Advances in Neural Information Processing Systems*.

Yehuda Koren, Robert M. Bell, and Chris Volinsky. 2009. Matrix factorization techniques for recommender systems. *Computer* 42, 8 (2009), 30–37.

Pan Li, Yanbang Wang, Hongwei Wang, and Jure Leskovec. 2020. Distance encoding: Design provably more powerful neural networks for graph representation learning. In *Proceedings of International Conference on Neural Information Processing Systems*.

Wu-Jun Li, Dit-Yan Yeung, and Zhihua Zhang. 2011. Generalized latent factor models for social network analysis. In *Proceedings of International Joint Conference on Artificial Intelligence*.

David Liben-Nowell and Jon M. Kleinberg. 2007. The link-prediction problem for social networks. *Journal of the Association for Information Science and Technology* 58, 7 (2007), 1019–1031.

Linyuan Lü and Tao Zhou. 2011. Link prediction in complex networks: A survey. *Physica A: Statistical Mechanics and its Applications* 390, 6 (2011), 1150–1170.

Victor Martinez, Fernando Berzal, and Juan Carlos Cubero Talavera. 2017. A survey of link prediction in complex networks. *Computing Surveys* 49, 4 (2017), 69:1–69:33.

Aditya Krishna Menon and Charles Elkan. 2011. Link prediction via matrix factorization. In *Proceedings of European Conference on Machine Learning and Knowledge Discovery in Databases*.

Mark E. J. Newman. 2001. Clustering and preferential attachment in growing networks. *Physical Review E* 64, 2 (2001), 025102.

Maximilian Nickel, Kevin Murphy, Volker Tresp, and Evgeniy Gabrilovich. 2016. A review of relational machine learning for knowledge graphs. *IEEE* 104, 1 (2016), 11–33.

Shirui Pan, Ruiqi Hu, Guodong Long, Jing Jiang, Lina Yao, and Chengqi Zhang. 2018. Adversarially regularized graph autoencoder for graph embedding. In *Proceedings of International Joint Conference on Artificial Intelligence*.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Köpf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. 2019. PyTorch: An imperative style, high-performance deep learning library. In *Proceedings of International Conference on Neural Information Processing Systems*.

Yanjun Qi, Ziv Bar-Joseph, and Judith Klein-Seetharaman. 2006. Evaluation of different biological data and computational classification methods for use in protein interaction prediction. *Proteins: Structure, Function, and Bioinformatics* 63, 3 (2006), 490–500.

Andrea Rossi, Denilson Barbosa, Donatella Firmani, Antonio Matinata, and Paolo Merialdo. 2021. Knowledge graph embedding for link prediction: A comparative analysis. *ACM Transactions on Knowledge Discovery from Data* 15, 2 (2021), 14:1–14:49.

Ruslan Salakhutdinov and Andriy Mnih. 2007. Probabilistic matrix factorization. In *Proceedings of International Conference on Neural Information Processing Systems*.

Abdul Samad, Mamoona Qadir, Ishrat Nawaz, Muhammad Arshad Islam, and Muhammad Aleem. 2020. A comprehensive survey of link prediction techniques for social network. *EAI Endorsed Transactions on Industrial Networks Intelligent Systems* 7, 23 (2020), e3.

Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. 2009. The graph neural network model. *IEEE Transactions on Neural Networks* 20, 1 (2009), 61–80.

Zhihao Shi, Xize Liang, and Jie Wang. 2023. LMC: Fast training of GNNs via subgraph sampling with provable convergence. In *Proceedings of International Conference on Learning Representations*.

Harry Shomer, Wei Jin, Wentao Wang, and Jiliang Tang. 2023. Toward degree bias in embedding-based knowledge graph completion. In *Proceedings of International World Wide Web Conference*.

Zachary Stanfield, Mustafa Coskun, and Mehmet Koyutürk. 2017. Drug response prediction as a link prediction problem. In *Proceedings of ACM International Conference on Bioinformatics, Computational Biology, and Health Informatics*.

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

Felix Wu, Amauri H. Souza Jr., Tianyi Zhang, Christopher Fifty, Tao Yu, and Kilian Q. Weinberger. 2019. Simplifying graph convolutional networks. In *Proceedings of International Conference on Machine Learning*.

Kai-Lang Yao and Wu-Jun Li. 2021. Blocking-based neighbor sampling for large-scale graph neural networks. In *Proceedings of International Joint Conference on Artificial Intelligence*.

Haoteng Yin, Muhan Zhang, Yanbang Wang, Jianguo Wang, and Pan Li. 2022. Algorithm and system co-design for efficient subgraph-based graph representation learning. In *Proceedings of International Conference on Very Large Databases*.

Jiaxuan You, Jonathan Michael Gomes Selman, Rex Ying, and Jure Leskovec. 2021. Identity-aware graph neural networks. In *Proceedings of AAAI Conference on Artificial Intelligence*.

Haiyang Yu, Limei Wang, Bokun Wang, Meng Liu, Tianbao Yang, and Shuiwang Ji. 2022. GraphFM: Improving large-scale GNN training via feature momentum. In *Proceedings of International Conference on Machine Learning*.

Seongjun Yun, Seoyoon Kim, Junhyun Lee, Jaewoo Kang, and Hyunwoo J. Kim. 2021. Neo-GNNs: Neighborhood overlap-aware graph neural networks for link prediction. In *Proceedings of Advances in Neural Information Processing Systems*.

Hanqing Zeng, Muhan Zhang, Yinglong Xia, Ajitesh Srivastava, Andrey Malevich, Rajgopal Kannan, Viktor Prasanna, Long Jin, and Ren Chen. 2021. Decoupling the depth and scope of graph neural networks. In *Proceedings of Advances in Neural Information Processing Systems*.

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

Muhan Zhang and Yixin Chen. 2017. Weisfeiler-lehman neural machine for link prediction. In *Proceedings of ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*.

Muhan Zhang and Yixin Chen. 2018. Link prediction based on graph neural networks. In *Proceedings of International Conference on Neural Information Processing Systems*.

Muhan Zhang, Pan Li, Yinglong Xia, Kai Wang, and Long Jin. 2021a. Labeling trick: A theory of using graph neural networks for multi-node representation learning. In *Proceedings of Advances in Neural Information Processing Systems*.

Wentao Zhang, Ziqi Yin, Zeang Sheng, Yang Li, Wen Ouyang, Xiaosen Li, Yangyu Tao, Zhi Yang, and Bin Cui. 2022. Graph attention multi-layer perceptron. In *Proceedings of (ACM) (SIGKDD) Conference on Knowledge Discovery and Data Mining*.

Yao Zhang, Yun Xiong, Xiangnan Kong, Zhuang Niu, and Yangyong Zhu. 2021b. IGE+: A framework for learning node embeddings in interaction graphs. *IEEE Transactions on Knowledge and Data Engineering* 33, 3 (2021), 1032–1044.

Yongqi Zhang and Quanming Yao. 2022. Knowledge graph reasoning with relational digraph. In *Proceedings of International World Wide Web Conference*.

Zhaocheng Zhu, Zuobai Zhang, Louis-Pascal Xhonneux, and Jian Tang. 2021. Neural bellman-ford networks: A general graph neural network framework for link prediction. In *Proceedings of Advances in Neural Information Processing Systems*.

Difan Zou, Ziniu Hu, Yewen Wang, Song Jiang, Yizhou Sun, and Quanquan Gu. 2019. Layer-dependent importance sampling for training deep and large graph convolutional networks. In *Proceedings of International Conference on Neural Information Processing Systems*.

Received 28 February 2023; revised 21 September 2023; accepted 15 December 2023