An Efficient Subgraph-Inferring Framework for Large-Scale Heterogeneous Graphs

Wei Zhou, Hong Huang*, Ruize Shi, Kehan Yin, Hai Jin

National Engineering Research Center for Big Data Technology and System
Services Computing Technology and System Lab, Cluster and Grid Computing Lab
School of Computer Science and Technology, Huazhong University of Science and Technology, Wuhan, 430074, China
{weizhou2021, honghuang, rzshi, kehanyin, hjin}@hust.edu.cn

Abstract

Heterogeneous Graph Neural Networks (HGNNs) play a vital role in advancing the field of graph representation learning by addressing the complexities arising from diverse data types and interconnected relationships in real-world scenarios. However, traditional HGNNs face challenges when applied to large-scale graphs due to the necessity of training or inferring on the entire graph. As the size of the heterogeneous graphs increases, the time and memory overhead required by these models escalates rapidly, even reaching unacceptable levels. To address this issue, in this paper, we present a novel framework named SubInfer, which conducts training and inferring on subgraphs instead of the entire graphs, hence efficiently handling large-scale heterogeneous graphs. The proposed framework comprises three main steps: 1) partitioning the heterogeneous graph from multiple perspectives to preserve various semantic information, 2) completing the subgraphs to improve the convergence speed of subgraph training and the performance of subgraph inferring, and 3) training and inferring the HGNN model on distributed clusters to further reduce the time overhead. The framework applies to the vast majority of HGNN models. Experiments on five benchmark datasets demonstrate that SubInfer effectively optimizes the training and inferring phase, delivering comparable performance to traditional HGNN models while significantly reducing time and memory overhead.

Introduction

Representing real-world data with rich structural and semantic information is a crucial task in many fields, and *heterogeneous graphs* (HGs) have proven to be a powerful tool for this purpose (Luo et al. 2023; Zeng et al. 2023; Zhou et al. 2023; Huang et al. 2021). However, learning representations from large-scale HGs remains a challenge in the field of heterogeneous graph representation learning.

To address this challenge, an increasing number of *heterogeneous graph neural network* (HGNN) methods have been proposed for large-scale HGs, which can generally be divided into three categories: decoupled-based, mini-batch-based, and partition-based methods. Decoupled-based methods, such as NARS (Yu et al. 2020) and SeHGNN (Yang et al. 2023), decouple the message propagation and update operations of the HGNN, saving node embeddings in pre-processing to greatly accelerate training. Mini-batch-based methods, such as R-GraphSAGE[1], R-GraphSAINT[2], HGT (Hu et al. 2020) and R-HGNN (Yu et al. 2023), sample a set of neighbors associated with the target node, enabling models to be trained more efficiently in terms of both time and memory consumption. Partition-based methods, such as Cluster-RGCN[2], partition the entire graph into subgraphs and apply the message passing over the induced subgraphs.

Method	Type	Training	Inferring
NARS	Decoupled	15s/38.4G	13s/36.5G
SeHGNN	Decoupled	19s/16.9G	18s/14.7G
R-HGNN	Mini-batch	221s/16.8G	140s/16.1G
Cluster-RGCN	Partition	**15s/7.4G**	42s/29.5G

Table 1: Time and memory overhead of HGNN methods for training and inferring on the ogbn-mag dataset. The table shows the time spent in one epoch.

However, these methods only reduce part of the overhead in training and inferring. As shown in Table 1, decoupled-based methods achieve a relatively balanced time overhead during training and inferring but require more memory to save the results of message propagation, while partition-based methods like Cluster-RGCN have less training time and memory overhead due to their training being confined to subgraphs, but more overhead during inference as they require entire graph inferring. Considering the superior performance of partition-based methods during training and the natural potential for distributed processing offered by subgraphs, we aim to develop a subgraph-inferring framework suitable for large-scale HGs, which can leverage the benefits of subgraphs on the one hand, and significantly reduce the overhead during inferring on the other hand.

Nevertheless, subgraph inferring encounters three major challenges: **1. How to partition large-scale HGs?** There is still no efficient method to partition large-scale HGs. Existing methods (NetClus (Sun, Yu, and Han 2009), Sclump (Li

*Corresponding author

[1]https://ogb.stanford.edu/docs/lsc/leaderboards/#mag240m
[2]https://ogb.stanford.edu/docs/leader_nodeprop/#ogbn-mag

et al. 2019)) need to factorize a large matrix, which makes them too complex to be applied to large-scale HGs, while other methods (Metis (Karypis and Kumar 1998), Louvain (Blondel et al. 2008)) on large homogeneous graphs ignore the types of nodes and edges, causing loss of semantic information. **2. How to make the model converge when subgraph training?** The higher the label distribution difference between subgraphs, the more difficult it is for the model to converge. To train a good model, the distribution differences between subgraphs need to be minimized. Although some methods like Cluster-GCN (Chiang et al. 2019) have considered this problem for homogeneous graphs, they result in a loss of information, which affects the performance of subgraph inferring. **3. How to guarantee the efficiency and performance of subgraph inferring?** Information loss during partitioning is inevitable, if subgraphs are too small, significant information is lost, which affects the performance of subgraph inferring. On the contrary, if the subgraphs are too large, it consumes more time and memory. Therefore, to ensure efficiency and performance, we need to not only consider the size of the subgraph but also complete the missing information.

To solve these problems, we propose a novel distributed HGNN framework based on **Sub**graph training and **Infer**ring(SubInfer). Firstly, the framework partitions the semantic graph constructed by the meta-path, and the remaining nodes are then allocated to each subgraph using a greedy algorithm that aims to minimize edge loss. With different meta-paths, the framework preserves multiple semantic information. Secondly, the framework completes the subgraphs by sampling the top k degree nodes from the original HG as global information. This improves the model's performance and reduces disparities in label distribution among the subgraphs. Finally, the training and inferring of the HGNN model are performed on the subgraphs, which are independent and easily extendable to distributed clusters. The framework can be applied to most HGNN models. Experiments conducted on five datasets showcase that our framework significantly enhances both training and inferring speeds, while also mitigating memory overhead. These improvements are achieved while maintaining acceptable performance levels, as compared to inferring on the entire graph.

Our contribution can be summarized as follows:

- We propose a distributed HGNN framework SubInfer to conduct the training and inferring of HGNNs on the subgraphs with less time and memory. Without complex adaptations, SubInfer is easy to be applied to most HGNN models.
- We are the first to propose a partition method for large-scale HGs using meta-paths, which analyzes and divides the HG from various perspectives to gather rich semantic information. Subgraph completion is further proposed to make HGNN models converge during subgraph training and improve the performance of subgraph inferring.
- Experiments on five datasets show that using our framework SubInfer, baselines require less time and memory while maintaining acceptable performance.

Preliminary

Definition 1: Heterogeneous Graph. A HG is a graph that contains many types of edges or nodes. Specifically defined as follows: given a graph $G = (V, E)$, which is a HG if it satisfies $v_i \in V, e_j \in E, \psi(v_i) \in \Psi, \phi(e_j) \in \Phi, |\Psi| + |\Phi| > 2$, where V and E denote the set of nodes and edges, while ψ and ϕ are type mapping functions by which we can get the types of nodes and edges. Furthermore, Ψ is the set of node types, and Φ is the set of edge types.

Definition 2: Meta-path. The meta-path means a path that expresses specific semantic information with the combination of the node type and edge type. It can be defined as $\psi_1 \xrightarrow{\phi_1} \psi_2 \xrightarrow{\phi_2} \cdots \xrightarrow{\phi_l} \psi_{l+1}$, where ψ_i denotes the type of node i and ϕ_j denote the type of edge j. For easy to express, the meta-path is generally abbreviated as $\psi_1\psi_2\cdots\psi_{l+1}$.

Definition 3: Semantic Graph. The semantic graph is a homogeneous graph constructed by meta-paths. For any meta-path $\psi_1\psi_2\cdots\psi_2\psi_1$, which is a palindrome string, the adjacency matrix of the corresponding semantic graph is $\hat{W}^{\psi_1\psi_1} = W^{\psi_1\psi_2} * \cdots * W^{\psi_2\psi_1}$, where $W^{\psi_i\psi_j}$ represents the adjacency matrix from ψ_i to ψ_j. Furthermore, $\hat{W}^{\psi_1\psi_1}_{uv}$ represents the number of instances following meta-path $\psi_1\psi_2\cdots\psi_2\psi_1$ between ${\psi_1}_u$ and ${\psi_1}_v$.

Our Framework

In this section, we present our framework SubInfer in detail. As shown in Figure 1, the framework can be divided into three parts: 1. Subgraph partition. It is difficult to preserve all kinds of semantic information in a single partition. To preserve one kind of semantic information, SubInfer first divides the semantic graph constructed by the corresponding meta-path, and then allocates the remaining nodes to subgraphs with a greedy algorithm that aims to minimize edge loss. With different meta-paths, multiple semantic information is well preserved. 2. Subgraph completion. Since the partitioned subgraphs lose a lot of information and have different label distributions, SubInfer selects the top k degree nodes from the original HG to complete the subgraphs. 3. Subgraph training and inferring. To further reduce the time overhead, the subgraphs are batched and uniformly distributed to each compute node for subgraph training and inferring. The individual components of SubInfer are explained below.

Subgraph Partition

Previous partition methods (Karypis and Kumar 1998; Blondel et al. 2008) for large-scale homogeneous graphs do not account for the types of nodes and edges, resulting in the loss of semantic information and poor performance of the HGNN model.

In HGs, diverse types of nodes and edges constitute various forms of semantic information, making it challenging to retain all of them within a single partition. For instance, taking into account an academic network comprising nodes that represent authors ($A = \{a_1, a_2 \cdots\}$), papers ($P = \{p_1, p_2, \cdots\}$), and research fields ($F = \{f_1, f_2, \cdots\}$), as well as relationships such as authorship of papers ($A - P$),

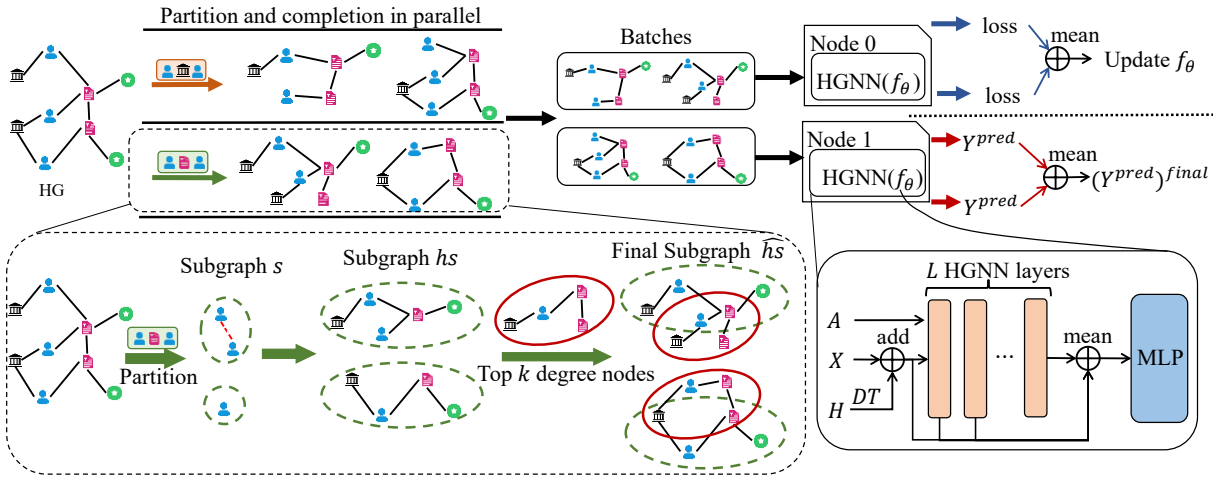


Figure 1: The structure of SubInfer.

citations between papers ($P-P$), and associations between papers and fields ($P-F$), including their inverse relationships. In a partition, node a_1 and its associated semantic information $a_1p_1a_2$ are assigned to subgraph s_1, while nodes p_2, a_3 are assigned to subgraph s_2, resulting in the loss of semantic information $a_1p_1f_1p_2a_3$. Therefore, we design a new partition strategy that retains only one type of semantic information in each partition to reduce information loss.

With the meta-path APA, which represents two authors writing the same paper, the corresponding semantic graph consisting only of author nodes is derived, and its adjacency matrix is

$$\hat{\boldsymbol{W}}^{AA} = \boldsymbol{W}^{AP} * \boldsymbol{W}^{PA}, \tag{1}$$

where $\boldsymbol{W}^{AP}$ represents the adjacency matrix from author to paper, and $\boldsymbol{W}^{PA}$ is its transpose matrix. Next, since the semantic graph is homogeneous, we use Metis (Karypis and Kumar 1998) to partition it, resulting in a set of subgraphs $S^{APA} = \{s_1^{APA}, s_2^{APA}, \cdots, s_\varrho^{APA}\}$ that effectively preserve the semantic information related to the meta-path APA. Here, ϱ is the hyper-parameter representing the number of subgraphs in a single partition.

As only author nodes are included in the subgraphs, we have to allocate the remaining other types of nodes to the subgraphs as well. When assigning the remaining nodes, two constraints are imposed to ensure the training speed and performance of the model: 1) the size of each subgraph should be as similar as possible, and 2) the number of missing edges should be minimized. The entire allocation process can be expressed as an optimization problem

$$\begin{aligned} \min_{\{hs_1^{APA},\cdots,hs_\varrho^{APA}\}} \quad & |E| - \sum_{i=1}^{\varrho} |E_i| \\ s.t. \quad & \forall i \in [1,\varrho], N_i \leq \lfloor N/\varrho \rfloor + 1 \end{aligned}, \tag{2}$$

where $\{hs_1^{APA}, \cdots, hs_\varrho^{APA}\}$ is final subgraphs retaining the semantic information of APA, while S^{APA} denotes its initial state. $|E_i|$ and N_i represent the number of edges and nodes in subgraph hs_i^{APA} respectively. N denotes the number of nodes in the original HG, and $\lfloor\rfloor$ means the floor function.

We employ a greedy algorithm to solve this problem. Specifically, we first compute the degree between paper nodes and the subgraphs as

$$adj = \boldsymbol{W}^{PA} * \boldsymbol{W}^{AS}, \tag{3}$$

where $\boldsymbol{W}^{AS} \in R^{N_A \times \varrho}$ is the relationship matrix between the author nodes and the set of subgraphs S^{APA}, and N_A means the number of author nodes in the original HG. For example, if author node a_i belongs to subgraph s_j^{APA}, the element in the i-th row and j-th column of matrix $\boldsymbol{W}^{AS}$ is set to 1, otherwise it is set to 0.

For nodes except authors, the higher the degree between them and the subgraph, the closer the relationship is. Therefore, the paper node is assigned to the closest subgraph, but if the number of nodes in the subgraph is greater than $\lfloor N/\varrho \rfloor + 1$, it is allocated to the next closest subgraph. Similarly, we assign other types of nodes to the subgraphs (The complete allocation process is shown in Supplementary materials[3] Algorithm 1).

Finally, there are many meta-paths in a HG, so we can get multiple subgraph sets as follows:

$$\mathcal{S} = \{\{hs_1^m, \cdots, hs_\varrho^m\}, m \in M\}, \tag{4}$$

where M is the set of meta-paths, which is set by prior knowledge and experience (Wang et al. 2020b; Shi et al. 2020). In this way, with the meta-path, we can explore and partition the large-scale HG from different perspectives while preserving semantic information.

Subgraph Completion

We find the pain point of subgraph training is that the huge difference in label distribution among subgraphs makes it difficult for the HGNN model to converge. A similar conclusion from experiments has also been drawn in a prior

[3]https://github.com/CGCL-codes/SubInfer

work (Chiang et al. 2019). Therefore, we propose subgraph completion to solve this problem.

For any two subgraphs, assuming that their label distributions are $\mathcal{P}$ and $\mathcal{Q}$. Thus the distance between these two distributions can be expressed in terms of the Kullback-Leibler divergence (Kullback and Leibler 1951) as

$$\begin{aligned} Dis(\mathcal{P},\mathcal{Q}) &= D_{KL}(\mathcal{P}||\mathcal{Q}) + D_{KL}(\mathcal{Q}||\mathcal{P}) \\ &= \sum_i \mathcal{P}(y_i)\log\frac{\mathcal{P}(y_i)}{\mathcal{Q}(y_i)} + \mathcal{Q}(y_i)\log\frac{\mathcal{Q}(y_i)}{\mathcal{P}(y_i)}, \\ &= \sum_i (\mathcal{P}(y_i)-\mathcal{Q}(y_i))\log\frac{\mathcal{P}(y_i)}{\mathcal{Q}(y_i)} \end{aligned} \tag{5}$$

where y_i indicates the i-th label. From Eq. 5, we can see that only when $\forall i, \log\frac{\mathcal{P}(y_i)}{\mathcal{Q}(y_i)} \to 0$, we have $Dis(\mathcal{P},\mathcal{Q}) \to 0$. Suppose we add q_i nodes with label y_i to the two subgraphs, then we have

$$\begin{aligned} \log\frac{\mathcal{P}(y_i)+q_i}{\mathcal{Q}(y_i)+q_i} &= \log\frac{\mathcal{Q}(y_i)+q_i+\mathcal{P}(y_i)-\mathcal{Q}(y_i)}{\mathcal{Q}(y_i)+q_i} \\ &= \log(1+\frac{\mathcal{P}(y_i)-\mathcal{Q}(y_i)}{\mathcal{Q}(y_i)+q_i}) \end{aligned}. \tag{6}$$

Assuming that $N(y=y_i)$ represents the number of nodes with label y_i in the original large-scale HG, if $q_i \to N(y=y_i)$, we will get $\log\frac{\mathcal{P}(y_i)+q_i}{\mathcal{Q}(y_i)+q_i} \to 0$. Therefore, by sampling $k=\sum_{i=1}^{C} q_i$ nodes from the original large-scale HG and adding them to each subgraph, the distribution difference $Dis(\mathcal{P},\mathcal{Q})$ decreases, where C is the total number of labels. In other words, the more identical nodes contained in the two distributions, the more similar they are.

However, randomly adding nodes merely helps the model to converge without completing the missing information during partitioning. The partitioned subgraphs only contain local information, which is the key to the decline in the model's performance. Therefore, it is necessary to use global information to complete the subgraphs. Research has shown that higher degree nodes contain more information and the top k degree nodes hold most of the information of the entire graph (Kong et al. 2021). Hence, we select them as global information I_{global} to complete the subgraph as

$$\hat{hs_i}^m = hs_i^m \cup I_{global}, \tag{7}$$

where m is any meta-path. In fact, an excess of global information can impair performance, while an insufficiency may fail to enhance it, so it is necessary to carefully balance the amount of global information. The most intuitive manifestation of the amount of information is the number of nodes, so we set $k=\lfloor N/\varrho \rfloor \times \sigma$, where σ is a hyper-parameter to control the ratio of global and local information.

Subgraph Training and Inferring

The label distribution between subgraphs with the same meta-path is similar after completing. However, there are multiple meta-paths, so to allow the model to converge better during training, we package these subgraphs into batches as

$$batch_i = \{\hat{hs_i}^m, m \in M\}. \tag{8}$$

In this way, the label distribution between each batch is similar. For any $batch_i$, assuming that the node features are $\boldsymbol{X}_i$ and the number of layers in the HGNN is L, we can get

$$\begin{aligned} \boldsymbol{Z}_i^{(l)} &= HGNN\_layer(\boldsymbol{Z}_i^{(l-1)}, batch_i) \\ \boldsymbol{Z}_i^{(0)} &= \boldsymbol{X}_i + DT(\boldsymbol{H}_i) \end{aligned}, \tag{9}$$

where $\boldsymbol{Z}_i^{(l)}$ represents the output of the l-th layer when HGNN is trained on $batch_i$, and $\boldsymbol{H}_i$ is the label feature. For the nodes in the training set, their label features are represented as one-hot vectors while other nodes are zero vectors. To prevent over-fitting, we mask some label features in the training set, and the masking ratio is controlled by the hyper-parameter α. Furthermore, the dimensional transformation operation (DT) is utilized to match the dimensions of the label feature matrix and node feature matrix, where a learnable parameter matrix is used to perform this mapping.

Followed by recent work (Yu et al. 2020; Yang et al. 2023; Frasca et al. 2020; Duan et al. 2022), we average the outputs of each layer as the final output of the model to get a better performance, which is expressed as

$$\boldsymbol{Z}_i^{final} = mean(\{\boldsymbol{Z}_i^{(l)}, l \in [0, L]\}), \tag{10}$$

then we train the model according to the downstream tasks. For example, in the node classification task, we map the final output Z_i^{final} to the corresponding class by an mlp and calculate the loss with the cross-entropy loss function as

$$\begin{aligned} \boldsymbol{Y}_i^{pred} &= softmax(mlp(\boldsymbol{Z}_i^{final})) \\ loss &= CrossEntropy(\boldsymbol{Y}_i^{pred}, \boldsymbol{Y}_i) \end{aligned}, \tag{11}$$

where $\boldsymbol{Y}_i$ represents the ground truth for the training nodes existing in $batch_i$.

For any node v_i in the test set, it is included in multiple subgraphs. For example, it may be included in subgraph $\hat{hs}_{\nu}^{APA}$ corresponding to meta-path APA and subgraph $\hat{hs}_{\mu}^{PFP}$ corresponding to meta-path PFP. Therefore, during inferring, SubInfer obtains the label prediction probability for node v_i on these subgraphs and averages them to produce the final prediction result as

$$(\boldsymbol{Y}^{pred})^{final} = \frac{1}{|M|}\sum_{m}^{M}(\boldsymbol{Y}^{pred})^m, \tag{12}$$

where $(\boldsymbol{Y}^{pred})^m$ represents the label prediction probability for all test nodes on the set of subgraphs corresponding to meta-path m.

Distributed Implementation

Since subgraphs do not interfere with each other, we can easily speed up training and inferring by distribution. Firstly, to ensure load balancing between distributed nodes, we divide the batch equally into T parts and distribute them to different nodes, where T is the number of distributed nodes. Secondly, we initialize the HGNN model with random parameters on the master node and copy it to each distributed node. Thirdly, in the training process, to ensure that the

model parameters are consistent in all nodes, for each training batch we first aggregate the losses of all nodes to the master node and then distribute the average loss to each node to update the model parameters. This also ensures that distributed training has the same effect as single GPU training. Finally, in the inferring stage, we collect the output results of all nodes to the master node for processing and verify the performance of the model on the validation set and test set.

Analysis of Complexity

Time. For the sake of convenience in the calculation, we assume that N/ϱ is an integer and the time complexity of the HGNN model is $O(N^2)$. In the pre-processing, the main time overhead is using Metis to partition the semantic graph, and the time complexity of partitioning the HG in parallel can be approximated as $O(NlogN)$. During the training, when the model is trained on a single subgraph, the time complexity is $O((1+\sigma)N^2/\varrho^2)$, so the time complexity on all subgraphs is $O((1+\sigma)N^2|M|/\varrho)$, where $|M|$ represents the number of meta-paths. Moreover, we further accelerate the training process by distribution. When the number of distributed nodes is T, the final time complexity can be approximated as $O((1+\sigma)N^2|M|/(\varrho T))$. For example, on the Ogbn-mag dataset, we set $\sigma = 0.1, |M| = 2, \varrho = 100, T = 2$, and the optimized time complexity is $O(1.1N^2/100)$, which is about 1/100 of the original.

Memory. Assuming that the space complexity of the HGNN model is $O(N)$. The size of the subgraph obtained by partitioning and completing does not exceed $(1+\sigma)N/\varrho$, so the batch size is about $(1+\sigma)N|M|/\varrho$. When training and inferring models on the batch, the space complexity is $O((1+\sigma)N|M|/\varrho)$. Similarly, on the Ogbn-mag dataset, the space complexity of training and inferring on the subgraph is $O(1.1N/50)$, about 1/50 of the original.

Experiment

In this section, employing the state-of-the-art HGNN backbones, we demonstrate the performance and efficiency of SubInfer on five datasets. Moreover, we conduct ablation experiments to determine the contribution of each component and study the effect of hyper-parameters and meta-paths on performance. Please see the Supplementary materials for details of experiment results on the MAG240M (Wang et al. 2020a) and the study of meta-paths.

Experimental Setup

Datasets. To validate the performance and efficiency of our framework, we train both our framework (SubInfer) with the baseline on five benchmark datasets: Ogbn-mag (Wang et al. 2020a), DBLP (Yang et al. 2022), PubMed (Yang et al. 2022), Yelp (Yang et al. 2022) and Freebase(Lv et al. 2021). The details of these datasets are provided in Table 2.

Baselines. We compare SubInfer with several state-of-the-art baselines, including basic methods (R-GCN, R-GAT), mini-batch-based methods (R-GraphSAGE, R-GraphSAINT, R-GSN, and R-HGNN), partition-based methods (Cluster-RGCN), and decoupled-based methods (NARS, SeHGNN). Details of the experimental setup are shown in Supplementary materials.

Dataset	#Node/Edge type	#Node	#Edge
Ogbn-mag	4 / 7	1,939,743	42,222,014
DBLP	4 / 9	1,989,077	275,940,913
Freebase	8 / 64	180,098	2,115,376
PubMed	4 / 16	63,109	244,986
Yelp	4 / 7	8,2465	30,542,675

Table 2: Dataset statistics.

Node Classification

In the node classification task, we run the state-of-the-art method and SubInfer on three datasets, Ogbn-mag, DBLP, and Freebase. We show their time and memory overhead during the training and inferring in Table 3 and demonstrate the prediction accuracy on the test set.

As can be seen from Table 3, although R-GraphSAGE, R-GraphSAINT, and Cluster-RGCN all optimize time and memory overhead during training, they still require inferring over the entire graph, resulting in a significant amount of overhead. In contrast, SubInfer remarkably reduces time and memory overhead by subgraph inferring, while maintaining acceptable accuracy and even outperforming baselines on some datasets. For decoupled-based methods, as the reduced memory to store message propagation results when training and inferring on subgraphs, our framework uses less memory, especially on the DBLP dataset, where SubInfer-NARS requires only 8-9% of the original memory.

However, there are some anomalies in experimental results. For instance, R-GraphSAGE and R-GraphSAINT consume less time when training on DBLP and Freebase datasets. First, for these datasets, the size of the training set is only a small fraction of the HG, and mini-batch-based methods only need to learn embeddings of the nodes in the training set, while partition-based methods have to generate embeddings for all nodes in the subgraph. Second, they all use the neighbor sampling strategy, which can further reduce the time overhead. Additionally, SubInfer-SeHGNN performs better than SeHGNN on the DBLP dataset. Our analysis reveals that the high heterogeneity of labels in the DBLP dataset results in substantial noise, while partitioning the subgraph effectively mitigates label heterogeneity, leading to higher-quality pseudo-labels for multi-stage training.

Link Prediction

In the link prediction task, several models are run on three datasets (DBLP, PubMed, and Yelp), with the experimental results presented in Table 4. The partition-based methods result in two nodes of the edge being divided into different subgraphs, making it impossible to perform training and inferring on separate subgraphs. Consequently, the experimental results for these methods are not shown. Fortunately, since the decoupled-based method propagates only in the pre-processing, while training and inferring are independent of the graph structure, SubInfer-SeHGNN and SubInfer-NARS can be modified to meet the requirements of the link prediction task. Instead of training on the subgraph,

		Ogbn-mag			DBLP			Freebase		
		Runtime/Memory		Acc	Runtime/Memory		Acc	Runtime/Memory		Acc
		Training	Inferring		Training	Inferring		Training	Inferring	
	R-GCN	1.0h/40.6G	42s/29.5G	46.08	4.3h/75.6G	296s/62.7G	52.14	34s/8.1G	19s/6.4G	64.47
Mini-batch	R-GraphSAGE*	22s/8.6G	42s/29.5G	47.51	12s/23.6G	296s/62.7G	50.40	4s/2.5G	19s/6.4G	61.42
	R-GraphSAINT*	23s/6.9G	42s/29.5G	47.06	14s/20.8G	296s/62.7G	54.39	4s/1.9G	19s/6.4G	63.25
Partition	Cluster-RGCN*	15s/7.4G	42s/29.5G	42.93	71s/21.9G	296s/62.7G	50.28	10s/2.1G	19s/6.4G	59.92
	SubInfer-RGCN	6s/10.4G	**9s/10.1G**	47.56	23s/24.3G	**26s/22.7G**	52.14	4s/3.8G	**3s/3.5G**	63.40
	R-GAT	1.2h/43.3G	51s/31.9G	49.66	5.0h/82.2G	372s/68.9G	51.30	42s/10.2G	25s/8.3G	65.63
Mini-batch	R-GraphSAGE†	27s/9.8G	51s/31.9G	48.25	16s/25.7G	372s/68.9G	50.81	5s/2.9G	25s/8.3G	62.15
	R-GraphSAINT†	27s/8.3G	51s/31.9G	47.96	19s/23.1G	372s/68.9G	54.10	6s/2.3G	25s/8.3G	63.61
Partition	Cluster-RGCN†	19s/9.0G	51s/31.9G	46.27	80s/24.9G	372s/68.9G	51.84	12s/2.8G	25s/8.3G	61.34
	SubInfer-RGAT	7s/11.5G	**10s/11.3G**	48.13	26s/26.1G	**29s/23.4G**	52.03	5s/4.3G	**5s/3.9G**	63.97
Mini-batch	R-GSN	192s/19.8G	206s/20.3G	50.96	304s/32.5G	331s/34.1G	53.12	10s/6.7G	13s/7.4G	65.69
Partition	SubInfer-RGSN	**14s/16.2G**	**12s/15.7G**	49.38	**30s/30.8G**	**27s/28.9G**	53.09	**6s/5.9G**	**5s/5.2G**	64.08
Mini-batch	R-HGNN	221s/16.8G	140s/16.1G	52.41	338s/29.9G	213s/28.6G	54.22	12s/6.1G	9s/5.6G	66.02
Partition	SubInfer-RHGNN	**16s/15.8G**	**15s/15.2G**	51.61	**35s/27.4G**	**30s/25.9G**	53.73	**7s/5.4G**	**6s/5.0G**	64.92
Decoupled	NARS	15s/38.4G	13s/36.5G	52.96	19s/107G	18s/101G	54.70	2s/9.1G	1s/8.3G	65.80
Partition	SubInfer-NARS	**11s/7.1G**	**9s/6.4G**	50.99	**14s/9.4G**	**11s/8.6G**	53.21	**1s/3.4G**	**1s/2.9G**	64.98
Decoupled	SeHGNN	19s/16.9G	18s/14.7G	56.45	23s/48.8G	21s/45.9G	58.12	3s/5.6G	3s/5.0G	66.59
Partition	SubInfer-SeHGNN	**15s/5.2G**	**12s/5.0G**	54.03	**18s/7.3G**	**16s/6.9G**	62.39	**2s/3.0G**	**1s/2.7G**	66.45

Table 3: Performance of node classification. $*$ means that using the R-GCN aggregator, while † means the R-GAT aggregator.

		DBLP		PubMed		Yelp	
		Runtime/Memory	AUC/Pre*	Runtime/Memory	AUC/Pre*	Runtime/Memory	AUC/Pre*
		Propagation		Propagation		Propagation	
Decoupled	NARS	106s/89.8G	58.55/56.57	6s/3.6G	67.09/63.90	7s/4.3G	73.40/69.53
Decoupled	SubInfer-NARS	**12s/5.8G**	56.78/54.72	**1s/1.0G**	65.34/61.89	**1s/1.6G**	70.92/67.23
Decoupled	SeHGNN	35s/36.7G	56.85/55.21	4s/1.4G	61.91/57.24	4s/1.9G	75.02/70.38
Decoupled	SubInfer-SeHGNN	**10s/2.2G**	54.77/53.60	**1s/0.6G**	60.99/56.61	**1s/1.0G**	71.31/67.98

Table 4: Performance of link prediction. $*$ is short for Precision.

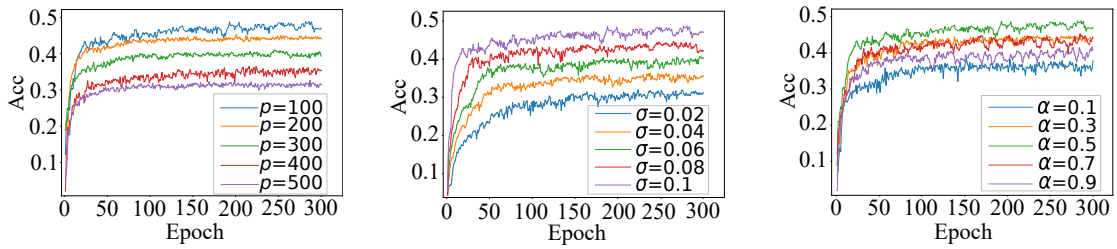
the message propagation is completed on subgraphs to obtain the representation of all nodes before training. The training and inferring processes in them are identical to SeHGNN and NARS, so the time and memory spent by them during training and inferring are also similar to NARS and SeHGNN. To highlight the superiority of SubInfer, we present the time and memory spent during the message propagation.

Experimental results indicate that without the label information and multi-stage training, the performance of SeHGNN is lower than NARS on some datasets. Additionally, the performance gap between SubInfer and baselines is relatively small, suggesting that the node features obtained by propagating on the subgraph are similar to those obtained by propagating on the entire graph. Finally, message propagation on the subgraph consumes significantly less time and memory compared to the entire graph due to its smaller size and independent nature, which enables parallel acceleration.

Ablation Study

In the ablation experiments, the necessity and effectiveness of each component of SubInfer are demonstrated on the Ogbn-mag, and experimental results are presented in Table 5. The model I, involves dividing the entire graph into 100 subgraphs using Metis and then training and inferring directly on the subgraphs without any further processing. The model II, adds global information to complete the subgraphs, without other changes compared to model I. Model III randomly divides the entire graph into ϱ parts, and model IV partitions the large-scale HG with meta-paths. Other components are the same as model II.

A detailed analysis of the final experimental results reveals that training and inferring directly on the subgraphs leads to poor results. The large differences in label distribution between subgraphs, make it difficult for the HGNN



(a) ϱ on Ogbn-mag (b) σ on Ogbn-mag (c) α on Ogbn-mag

Figure 2: Hyper-parameter analysis.

Model	Description	Train Acc	Test Acc
I	Metis	69.61	35.07
II	Metis + Completion*	75.59	47.08
III	Random†+ Completion*	71.49	39.40
IV	SPMP‡+ Completion*	**77.57**	**48.13**

Table 5: Ablation experiments. $*$ indicates completing the subgraphs with global information, † means random partition, and ‡ denotes a subgraph partition with meta-paths.

model to converge. Additionally, inferring from subgraphs results in the loss of much information. Compared to model I, model II performs better on both training and test sets. This suggests that global information can reduce label distribution differences between subgraphs and provide more valid information. A comparison between models II and III indicates that information closely related to nodes is also important, and partitioning with Metis effectively preserves it. Finally, by partitioning large-scale HGs with meta-paths, various semantic information is captured from different views, which helps to improve the model's performance.

Hyper-parameter Study

In this section, the effect of hyper-parameters is analyzed. The same conclusions are reached for all datasets, so only the results for Ogbn-mag is presented in Figure 2. Moreover, the model used is SubInfer-RGAT.

The number of subgraphs in a single partition ϱ. As shown in Figure 2(a), the model's performance deteriorates as the number of partitions increases due to the greater loss of information. Furthermore, the convergence speed of the model under different settings, indicates that completing subgraphs with global information can effectively address the difficulty in subgraph training.

The ratio of the global information σ. The results is presented in Figure 2(b). As indicated by Eq. 6 and $k = \lfloor N/\varrho \rfloor \times \sigma$, a larger value of parameter σ results in smaller differences in label distribution between subgraphs, allowing the model to converge faster. Furthermore, increased information retention within subgraphs and enhanced performance are achieved with a higher value of σ.

Masking ratio of label features α. The results is presented in Figure 2(c). As the masking rate α increases, the model's performance first increases and then decreases. Since low α leads to over-fitting on the training set, resulting in a decrease in generalization ability. Conversely, a particularly high α leads to under-fitting, resulting in poor performance in subgraph inferring.

Related Work

In recent years, large-scale graph representation learning has attracted more and more attention, and many related works have been proposed, which fall into three broad categories. **1. Mini-batch-based methods**, pack the required nodes and edges into a batch, which can significantly reduce the time and memory overhead for training, such as GraphSAGE (Hamilton, Ying, and Leskovec 2017), GraphSAINT (Zeng et al. 2019), HGT (Hu et al. 2020) and R-HGNN (Yu et al. 2023). Moreover, the R-GraphSAGE and R-GraphSAINT we used are extended from GraphSAGE and GraphSAINT by relation-wise aggregation, which is proposed in R-GCN (Schlichtkrull et al. 2018). **2. Partition-based methods**, such as Cluster-GCN (Chiang et al. 2019) and Ada-GNN (Luo et al. 2022), obtain a set of subgraphs by partitioning and then train the model on the subgraphs. Similarly, Cluster-RGCN is an extension of Cluster-GCN. **3. Decoupled-based methods**, by decoupling the message propagation and update operations of GNN, have greatly accelerated the speed of model training and inferring, such as SGC (Wu et al. 2019), LightGCN (He et al. 2020), PPRGO (Bojchevski et al. 2020), SIGN (Frasca et al. 2020), xGCN (Song et al. 2023), and SAGN (Sun, Gu, and Hu 2021). In addition, there are some decoupled-based methods designed for HGs, like NARS (Yu et al. 2020) and SeHGNN (Yang et al. 2023). However, these methods require performing inferring or message propagation on the entire graph, resulting in significant overhead.

Conclusion

We propose a novel distributed HGNN framework based on **Sub**graph Training and **Infer**ring (SubInfer). The framework includes three operations: subgraph partition, subgraph completion, and subgraph training and inferring, which can quickly and effectively learn the representation of large-scale HGs. Experiments on five datasets show that using our framework, the baselines require less time and memory for both training and inferring while maintaining acceptable performance compared to the entire graph inferring.

Acknowledgments

This work is supported by the National Natural Science Foundation of China (No. 62172174, No.61932004).

References

Blondel, V. D.; Guillaume, J.-L.; Lambiotte, R.; and Lefebvre, E. 2008. Fast unfolding of communities in large networks. *Journal of statistical mechanics: theory and experiment*, 2008(10): P10008.

Bojchevski, A.; Klicpera, J.; Perozzi, B.; Kapoor, A.; Blais, M.; Rózemberczki, B.; Lukasik, M.; and Günnemann, S. 2020. Scaling graph neural networks with approximate pagerank. In *Proceedings of the 26th ACM SIGKDD international conference on knowledge discovery and data mining*, 2464–2473.

Chiang, W.-L.; Liu, X.; Si, S.; Li, Y.; Bengio, S.; and Hsieh, C.-J. 2019. 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 and data mining*, 257–266.

Duan, K.; Liu, Z.; Wang, P.; Zheng, W.; Zhou, K.; Chen, T.; Hu, X.; and Wang, Z. 2022. A comprehensive study on large-scale graph training: Benchmarking and rethinking. In *Proceddings of the Thirty-fifth Conference on Neural Information Processing Systems*, 5376–5389.

Frasca, F.; Rossi, E.; Eynard, D.; Chamberlain, B.; Bronstein, M.; and Monti, F. 2020. SIGN: Scalable Inception Graph Neural Networks. In *Proceedings of the ICML 2020 Workshop on Graph Representation Learning and Beyond*, 1–17.

Hamilton, W. L.; Ying, Z.; and Leskovec, J. 2017. Inductive Representation Learning on Large Graphs. In *Proceeding of the Thirtieth Conference on Neural Information Processing Systems*, 1024–1034.

He, X.; Deng, K.; Wang, X.; Li, Y.; Zhang, Y.; and Wang, M. 2020. Lightgcn: Simplifying and powering graph convolution network for recommendation. In *Proceedings of the 43rd International ACM SIGIR conference on research and development in Information Retrieval*, 639–648.

Hu, Z.; Dong, Y.; Wang, K.; and Sun, Y. 2020. Heterogeneous graph transformer. In *Proceedings of the Web Conference 2020*, 2704–2710.

Huang, H.; Shi, R.; Zhou, W.; Wang, X.; Jin, H.; and Fu, X. 2021. Temporal Heterogeneous Information Network Embedding. In *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence*, 1470–1476.

Karypis, G.; and Kumar, V. 1998. A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM Journal on scientific Computing*, 20(1): 359–392.

Kong, J.; Li, W.; Liao, B.; Qiu, J.; Cai, Y.; Zhu, J.; Zhang, S.; et al. 2021. Learning Large-scale Network Embedding from Representative Subgraph. *arXiv preprint arXiv:2112.01442*.

Kullback, S.; and Leibler, R. A. 1951. On information and sufficiency. *The annals of mathematical statistics*, 22(1): 79–86.

Li, X.; Kao, B.; Ren, Z.; and Yin, D. 2019. Spectral clustering in heterogeneous information networks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 4221–4228.

Luo, J.; He, M.; Pan, W.; and Ming, Z. 2023. BGNN: Behavior-aware graph neural network for heterogeneous session-based recommendation. *Frontiers of Computer Science*, 17(5): 175336.

Luo, Z.; Lian, J.; Huang, H.; Jin, H.; and Xie, X. 2022. AdaGNN: Adapting to Local Patterns for Improving Graph Neural Networks. In *Proceedings of the 15th ACM International Conference on Web Search and Data Mining*, 638–647.

Lv, Q.; Ding, M.; Liu, Q.; Chen, Y.; Feng, W.; He, S.; Zhou, C.; Jiang, J.; Dong, Y.; and Tang, J. 2021. Are we really making much progress? Revisiting, benchmarking and refining heterogeneous graph neural networks. In *Proceedings of the 27th ACM SIGKDD conference on knowledge discovery and data mining*, 1150–1160.

Schlichtkrull, M.; Kipf, T. N.; Bloem, P.; Berg, R. v. d.; Titov, I.; and Welling, M. 2018. Modeling relational data with graph convolutional networks. In *Proceedings of the 15th International Conference on Extended Semantic Web Conference*, 593–607.

Shi, R.; Liang, T.; Peng, H.; Jiang, L.; and Dai, Q. 2020. HEAM: Heterogeneous Network Embedding with Automatic Meta-path Construction. In *Proceedings of the International Conference on Knowledge Science, Engineering and Management*, 304–315.

Song, X.; Lian, J.; Huang, H.; Luo, Z.; Zhou, W.; Lin, X.; Wu, M.; Li, C.; Xie, X.; and Jin, H. 2023. xGCN: An Extreme Graph Convolutional Network for Large-scale Social Link Prediction. In *Proceedings of the ACM Web Conference 2023*, 349–359.

Sun, C.; Gu, H.; and Hu, J. 2021. Scalable and adaptive graph neural networks with self-label-enhanced training. *arXiv preprint arXiv:2104.09376*.

Sun, Y.; Yu, Y.; and Han, J. 2009. Ranking-based clustering of heterogeneous information networks with star network schema. In *Proceedings of the 15th ACM SIGKDD international conference on knowledge discovery and data mining*, 797–806.

Wang, K.; Shen, Z.; Huang, C.; Wu, C.-H.; Dong, Y.; and Kanakia, A. 2020a. Microsoft academic graph: When experts are not enough. *Quantitative Science Studies*, 1(1): 396–413.

Wang, X.; Lu, Y.; Shi, C.; Wang, R.; Cui, P.; and Mou, S. 2020b. Dynamic heterogeneous information network embedding with meta-path based proximity. *IEEE Transactions on Knowledge and Data Engineering*, 34(3): 1117–1132.

Wu, F.; Souza, A.; Zhang, T.; Fifty, C.; Yu, T.; and Weinberger, K. 2019. Simplifying graph convolutional networks. In *Proceedings of the International conference on machine learning*, 6861–6871.

Yang, C.; Xiao, Y.; Zhang, Y.; Sun, Y.; and Han, J. 2022. Heterogeneous Network Representation Learning: A Unified Framework With Survey and Benchmark. *IEEE Trans-*

actions on Knowledge and Data Engineering, 34(10): 4854–4873.

Yang, X.; Yan, M.; Pan, S.; Ye, X.; and Fan, D. 2023. Simple and Efficient Heterogeneous Graph Neural Network. In *Proceedings of the the AAAI Conference on Artificial Intelligence*, 10816–10824.

Yu, L.; Shen, J.; Li, J.; and Lerer, A. 2020. Scalable Graph Neural Networks for Heterogeneous Graphs. *arXiv preprint arXiv:2011.09679*.

Yu, L.; Sun, L.; Du, B.; Liu, C.; Lv, W.; and Xiong, H. 2023. Heterogeneous Graph Representation Learning With Relation Awareness. *IEEE Transactions on Knowledge and Data Engineering*, 35(6): 5935–5947.

Zeng, H.; Zhou, H.; Srivastava, A.; Kannan, R.; and Prasanna, V. 2019. GraphSAINT: Graph Sampling Based Inductive Learning Method. In *Proceedings of the International Conference on Learning Representations*, 1–19.

Zeng, Y.; Li, Z.; Chen, Z.; and Ma, H. 2023. Aspect-level sentiment analysis based on semantic heterogeneous graph convolutional network. *Frontiers of Computer Science*, 17(6): 176340.

Zhou, W.; Huang, H.; Shi, R.; Song, X.; Lin, X.; Wang, X.; and Jin, H. 2023. Temporal Heterogeneous Information Network Embedding via Semantic Evolution. *IEEE Transactions on Knowledge and Data Engineering*, 35(12): 13031–13042.