NeighSqueeze: Compact Neighborhood Grouping for Efficient Billion-Scale Heterogeneous Graph Learning

Xinyue Feng
Rutgers University
xinyue.feng@rutgers.edu

Shuxin Zhong
HKUST(GZ)
shuxinzhong@hkust-gz.edu.cn

Jinquan Hang
Rutgers University
jinquan.hang@rutgers.edu

Yuequn Zhang
Rutgers University
yz1127@cs.rutgers.edu

Guang Yang
Rutgers University
gy121@cs.rutgers.edu

Haotian Wang
JD Logistics
wanghaotian18@jd.com

Desheng Zhang
Rutgers University
desheng@cs.rutgers.edu

Guang Wang
Florida State University
guang@cs.fsu.edu

ABSTRACT

The rapid growth of online shopping has intensified competition among logistics companies, highlighting the importance of customer expansion, i.e., identifying customers willing to establish long-term contracts. Although existing approaches frame customer expansion as a node classification task using heterogeneous graph learning to capture complex interactions between a customer and other items, it is computationally infeasible to utilize all neighboring interactions on large-scale logistics graphs. Current sub-sampling methods reduce computational load by sampling a small part of neighborhood for training. However, they introduce substantial information loss, particularly affecting high-degree nodes and decreasing predictive accuracy. To address this, we introduce **NeighSqueeze**, a novel approach that groups structurally and semantically similar nodes, substantially reducing the neighbors count and facilitating full-neighbor learning. NeighSqueeze consists of three modules designed to efficiently and effectively enable node grouping on billion-scale heterogeneous graphs: (1) Structure-tightness-based neighbor filtering reduces the high redundancy and complexity in similarity computations. (2) Hybrid similarity graph construction addresses the difficulty of measuring node similarity at scale; and (3) A two-level grouping strategy resolves the label dominance issue within groups. We evaluate NeighSqueeze on JD Logistics, one of the largest logistics companies in China. Compared with sub-sampling methods, our NeighSqueeze exhibits lower runtime and memory usage with full-neighbor training on the compressed graph, while simultaneously improving average precision over 28.9% in offline evaluation and increase new customer exploration rate by 18.6% in online A/B testing.

1 INTRODUCTION

The surge in online shopping has significantly increased logistics demands, driving growth for logistics providers like Amazon [1], FedEx [8], JD Logistics [25], and SF Express [7]. In this competitive landscape, logistics companies are increasingly focused on **customer expansion** [23, 28, 32, 38], i.e., seeking **high-intent customers** that are willing to sign long-term contracts. Establishing such partnerships not only ensures stable revenue streams but also reduces customer churn risks posed by business competitors, making it a key strategic priority for the industry.

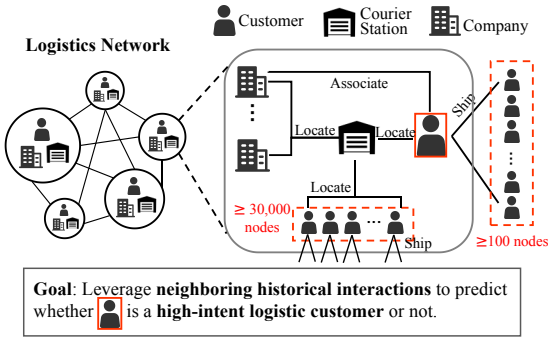


Figure 1: Rich interactions form the Logistics network including more than 1 billion nodes and 10 billion edges.

Recent methods [33, 38] formulate customer expansion as a node classification problem in heterogeneous graph learning [31], allowing them to capture rich interactions between customers and other entities to improve predictive capabilities. Yet, in large-scale graphs, it is computationally infeasible to utilize all neighboring interactions. For example, in logistics networks (Figure 1), customers may engage in hundreds of shipments monthly, while courier stations process over 60,000 packages, making full-neighbor aggregation prohibitively expensive. To address this, mainstream approaches adopt sub-sampling [5, 15, 19, 36], which train the model on selected k neighbors from a total of N neighbors where $k \ll N$ to reduce computation. However, such strategy struggles to learn the true neighborhood distribution when N is large with capturing only a limited subset of information. **As shown in Figure 2, on high-degree nodes, the predictive accuracy of sampling methods drops significantly compared to the method that utilizes all neighbors for training, highlighting severe information loss.**

Thus, this raises a key question: how can we efficiently process a large number of neighboring nodes in the training while minimizing information loss? Interestingly, our empirical observation reveals that some nodes' neighbors often exhibit **strong clustering properties**, i.e., many neighbors share similar semantics or structural roles (Figure 3). Such clustering is reasonable in our scenarios: for example, a station may be connected to tens of thousands of surrounding customers, many of whom are inactive,

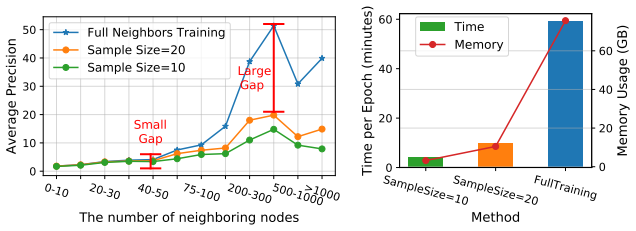


Figure 2: The left figure compares the prediction accuracy of models trained with different sample sizes, evaluated across nodes grouped by their number of neighbors (degree). *Sample Size = n* indicates that for each node, up to *n* neighbors of *each node type* are randomly sampled at each hop during training. The right figure shows that larger sample sizes significantly increase time and memory usage. As full-neighbor training is infeasible on billion-scale graphs, all results here are reported on a subgraph.

with minimal transaction history and have not signed a contract with our company. As these nodes tend to learn similar embeddings during graph learning, it is reasonable to merge them into a single group for joint representation learning. Motivated by this, we can aggregate similar nodes into representative groups, which can significantly reduce the number of neighbors while preserving the neighborhood information.

However, implementing grouping in billion-scale heterogeneous graphs introduces three main challenges:

- **High Complexity in Determining Group Range (C1).** Computing similarity between all node pairs in the graph [4, 17] incurs an impractical $O(n^2)$ complexity, making it impractical for our graph with over billions of nodes.
- **Embedding Dimensionality Explosion in Measuring Similarity at Scale (C2).** In graphs, similarity between nodes is typically measured by jointly considering node attributes and structural information. While node feature embeddings can directly represent similarity in attribute space, structural similarity is harder to encode. [4] uses the rows of the adjacency matrix as structural embeddings, but it results in extremely high-dimensional vectors on billion-scale graphs, making clustering computationally infeasible. An alternative approach represents structural embeddings by aggregating neighboring features (e.g., via mean pooling) [6, 10], but this is ineffective in our setting where many nodes lack attributes.
- **Label Overwhelming in Assigning Group Label (C3).** Nodes in a group may have different labels. Common strategies to label the group include majority voting [10, 17] or using the positive-to-negative ratio in binary tasks [18, 30]. However, in our case, positive samples are very sparse, making these methods prone to overwhelming the signal with negatives and unlabeled instances.

To address these challenges, we propose **NeighSqueeze**, a neighborhood grouping method for efficient billion-scale heterogeneous graphs learning. Specifically, it compresses the graph efficiently and effectively through three key steps: **(1) Structure-tightness-based neighbor filtering.** We posit that similar nodes are necessarily closely interconnected. Hence, given a node A, instead of calculating its similarity with every other node in the graph (C1), we restrict similarity computations to only those nodes within a specified range around A. **(2) Hybrid similarity graph construction.** Rather than

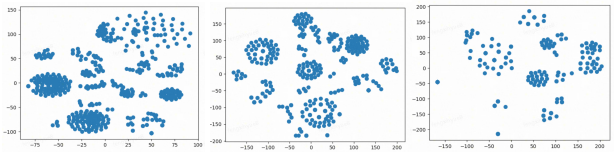


Figure 3: To demonstrate the strong clustering properties in the neighboring nodes, we selected three customer nodes as examples. For each node, we visualize the distribution of its neighboring nodes' embeddings (obtained after the model is fully trained) using the t-SNE method.

encoding attributes and structure into a flattening embedding space, which is inefficient and prone to loss of structural details (C2), we take the opposite approach: preserving structural relations as the backbone and integrating attribute similarity into the structure, constructing a similarity graph. We then apply the graph partitioning algorithm on this graph for node grouping. This method preserves complete structural information, integrates attribute signals seamlessly, and avoids the overhead and information loss of previous encoding methods. **(3) Two-level grouping strategy.** To prevent positive supervision signals from being overwhelmed (C3), we introduce a two-level grouping strategy. Initially, nodes are grouped based on their attribute and structural similarity. Subsequently, we further partition the nodes within each group based on their labels: nodes labeled "1", nodes labeled "0", and unlabeled nodes are separated into distinct subgroups.

Our main contributions are summarized as follows:

- We identify a critical issue: Existing graph sampling methods perform poorly on high-degree nodes, as they capture only a limited subset of the extensive neighborhood information, resulting in significant information loss. To address this, we propose grouping semantically and structurally similar neighbors as an efficient alternative, effectively reducing computational cost while preserving the integrity of neighborhood information.
- We propose NeighSqueeze, a scalable neighbor grouping framework that can be applied on billion-scale heterogeneous graphs. It contains three key steps: (1) Structure-tightness-based neighbor filtering. (2) Hybrid similarity graph construction. (3) Two-level grouping strategy to prevent information confusion.
- We implement, evaluate, and deploy NeighSqueeze in JD Logistics, one of the largest logistics companies in China. Compared with sub-sampling methods, our NeighSqueeze exhibits lower runtime and memory usage with full-neighbor training on the compressed graph, while simultaneously improving average precision by 28.9% in offline evaluation and increasing new customer exploration rate by 18.6% in online A/B testing. This demonstrates that our method effectively addresses the information loss in sub-sampling approaches while maintaining high computational efficiency.

2 PRELIMINARIES AND NOTATIONS

Definition 1 (Heterogeneous Graph [26]). A heterogeneous graph is defined as $\mathcal{G} = \{\mathcal{V}, \mathcal{E}, \mathcal{A}, \mathcal{R}\}$, where $\mathcal{V}$ and $\mathcal{E}$ denote the sets of nodes and edges, respectively. Each node $v \in \mathcal{V}$ has a type $\boldsymbol{A}(v)$, forming the node type set $\mathcal{A} = \{\boldsymbol{A}(v) : \forall v \in \mathcal{V}\}$. Each edge $e \in \mathcal{E}$ has a type $\boldsymbol{R}(e)$, forming the edge type set $\mathcal{R} = \{\boldsymbol{R}(e) : \forall e \in$

$\mathcal{E}\}$. For a heterogeneous graph, $|\mathcal{A}| + |\mathcal{R}| > 2$. If $|\mathcal{A}| = |\mathcal{R}| = 1$, it will degenerate to a homogeneous graph.

Definition 2 (Heterogeneous Graph Neural Networks (HGNNs)). Existing heterogeneous graph neural networks [14, 15, 26, 27] typically involve three operations in each layer: (1) **Message**, which extracts useful information from neighboring nodes by linear transformations; (2) **Attention**, which estimates the importance of neighboring nodes to the central node based on node features and edge types. (3) **Aggregate**, which weights neighboring nodes based on their importance and aggregates them. Let h_v^{l-1} denote the embedding of node v output from the l^{th} layer, we input the central node h_v^{l-1} and its neighboring nodes $\{h_u^{l-1}, e_{v,u} \in E\}$ into the l^{th} layer, then these three operations can be formulated as:

$$h_v^l \leftarrow \underset{\forall u, e_{v,u} \in E}{Agg} \left(Att(h_v^{l-1}, h_u^{l-1}, \boldsymbol{R}(e_{v,u})) \cdot Mes(h_u^{l-1}, \boldsymbol{A}(u)) \right).$$

Detailed implementations of these three operations differ depending on the methods used. After going through L layers, the representation of the target node will be input into several fully connected layers to get the final classification result.

Definition 3 (Metapath [31]). A metapath m is based on a network schema $\mathcal{S}$, and is denoted as

$$m = A_1 \xrightarrow{R_1} A_2 \xrightarrow{R_2} \cdots \xrightarrow{R_l} A_{l+1}$$

where $A_1, A_2, \ldots, A_{l+1} \in \mathcal{A}$ are node types and $R_1, R_2, \ldots, R_l \in \mathcal{R}$ are edge types.

3 METHODOLOGY

3.1 Model Overview

In this paper, we aim to propose a highly efficient node grouping method for billion-scale heterogeneous graphs, through which we can reduce the neighborhood size of high-degree nodes, and enable efficient training on all neighbors. To achieve this goal, we propose NeighSqueeze, which consists of the following three modules:

- **Structure-tightness-based neighbor filtering:** it restricts the range of nodes considered for grouping, significantly reducing similarity computations in subsequent stages.
- **Hybrid similarity graph construction:** building a similarity graph fusing both the structure and attribute similarities.
- **Two-level grouping strategy:** it first groups nodes by applying the graph partitioning algorithm on the similarity graph, then refines each group based on node labels.

After node grouping, we perform learning on the compressed graph using a heterogeneous graph neural network to obtain the final predictions. As there are multiple types of nodes in a heterogeneous graph, it should be noted that our grouping method is type-specific: **each node is only grouped with others of the same type to preserve semantic consistency.** In the following sections, we use the *Customer* node as an example to illustrate our method.

3.2 Structure-tightness-based neighbor filtering

Since computing similarity between all node pairs in the graph incurs an impractical $O(n^2)$ complexity, we introduce the Structure-tightness-based neighbor filtering module to restrict the range of nodes considered for grouping.

The key intuition of this module is that similar nodes are necessarily closely interconnected. Hence, we limit similarity computations to a filtered set of candidates that are most likely to be relevant. Specifically, we define the filtering scope using a predefined metapath set $\mathcal{M}$, where each metapath $m \in \mathcal{M}$ encoding a strong semantic relationship between nodes. For instance, given a customer node v, metapath *Customer-(locate)-Station-(locate)-Customer* identifies customers served by the same delivery station, implying location similarity. The metapath *Customer-(ship)-Customer* captures direct transaction relationships.

Given a target node v with node type $\boldsymbol{A}(v)$, the filtered nodes sides can be represented by

$$\mathcal{N}(v) = \{u | u \in \boldsymbol{A}(v); v \rightsquigarrow u \text{ via } m, m \in \mathcal{M}\} \tag{1}$$

where $v \rightsquigarrow u$ denotes that there exists a metapath m from v to u.

This design has the potential to substantially reduce the similarity computations in subsequent stages, improving efficiency without sacrificing much semantic relevance.

3.3 Hybrid similarity graph construction

After defining the group range, the next step is to compute node similarity for grouping. In graphs, node similarity is typically derived from both attributes and structural context. While node-feature embeddings can directly represent similarity in attribute space, structural similarity is harder to encode. Common approaches, such as aggregating neighboring features [6, 10] or using adjacency rows [4], either fail when nodes lack attributes or become computationally infeasible on billion-scale graphs due to high dimensionality. To address this, rather than encoding attributes and structure into a flattening embedding space, which is inefficient and prone to loss of structural detail, we take the opposite approach: preserving structural relations as the backbone and integrating attribute similarity into the structure, forming a **Hybrid Similarity Graph**. Then, we can apply graph partitioning algorithms for node grouping. This design preserves complete structural similarities, integrates attribute similarities seamlessly, and avoids the overhead and information loss of previous encoding methods.

Specifically, **a Hybrid Similarity Graph is a weighted graph constructed over nodes of the same type, where each edge weight represents a fusion of structure-based and attribute-based similarities**. Such a graph is built in three steps:

Step 1: Structure-based Similarity Graph Construction. Given a specific node type (e.g., Customer), to model the structural similarity between nodes, we extract multiple **metapath-based adjacency matrices** from different structural views. For instance, we define Adj_1 as the adjacency matrix derived from the *Customer-(ship)-Customer* metapath, define Adj_2 as the adjacency matrix derived from the *Customer-(associate)-Company-(associate)-Customer* metapath, and so on. $Adj_k[i, j]$ is the number of metapath m_k existing between nodes v_i and v_j. These adjacency matrices, each reflecting a different perspective of structural similarity between two nodes. We then compute a weighted sum of these matrices to construct the Structure-based Similarity Graph.

$$\text{Adj}^{StrucSim}(i, j) = \frac{1}{K} \sum_{k=1}^{K} \text{Adj}_k(i, j) \tag{2}$$

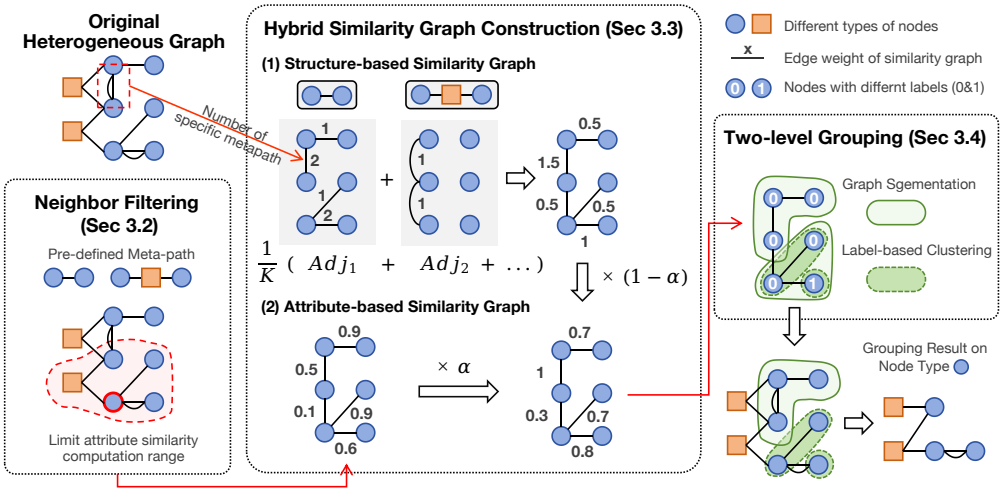


Figure 4: The overview of our NeighSqueeze model

Step 2: Attribute-based Similarity Graph Construction. Next, we compute pairwise attribute similarity between nodes using their feature embeddings. It should be noted that to improve efficiency and prevent noise from distant pairs, we limit the similarity computations to node pairs, **avoiding a fully connected similarity graph (as mentioned in Section 3.2)**.

$$\mathrm{Adj}^{AttSim}(i,j) = \exp\left(-\frac{\|\mathbf{x}_i - \mathbf{x}_j\|^2}{\sigma^2}\right) \tag{3}$$

This step results in a sparse Attribute-based Similarity Graph, whose edge weights reflect semantic similarity between node attributes.

Step 3: Hybrid Similarity Graph Construction. Finally, we fuse these two graphs into a unified Hybrid Similarity Graph, which captures both structural and semantic relationships between nodes:

$$\mathrm{Adj}^{Hybrid}(i,j) = (1-\alpha)\cdot \mathrm{Adj}^{StrucSim}(i,j) + \alpha \cdot \mathrm{Adj}^{AttSim}(i,j) \tag{4}$$

where $\alpha \in [0,1]$ controls the importance trade-off between structural and semantic similarity.

3.4 Two-level grouping strategy

3.4.1 The first-level grouping: graph partitioning. Based on the constructed Hybrid similarity graph, the next step is to utilize graph partitioning algorithms to group similar nodes.

Why Partitioning the Hybrid Similarity Graph Captures Both Structural and Semantic Similarity between nodes? Graph partitioning algorithms aim to divide the graph into groups of nodes that are strongly connected by high-weight edges. When applied to a Hybrid Similarity Graph, where edge weights combine both structural and semantic similarity, the algorithm naturally tends to put nodes together that are close not only in the graph structure but also in their attributes. As a result, the final groups reflect both types of similarity without needing separate processing.

We adopt the Leiden algorithm [29] for graph partitioning due to its efficiency, scalability, and ability to leverage edge weights. Leiden is a greedy optimization method that partitions a graph by maximizing the total connection strength within each group. It works iteratively in two phases: first, it moves each node to the group of its neighbors that increases the total internal edge weight the most; then, it compresses each group into a super node and repeats the process on the reduced graph. This procedure continues until no further improvement can be made. The resolution parameter γ in the Leiden algorithm allows us to control the granularity of partitioning, where lower values produce coarser partitions with larger clusters, leading to a more compact compressed graph.

3.4.2 The second-level grouping: label-based clustering. **However, after grouping, we find that nodes within a group may possess different labels.** Common solutions involve assigning the group's label based on the majority class [10, 17] or, in binary classification scenarios, representing the group's label by the ratio of positive to negative samples [18, 30]. But in our scenario, positive samples are extremely sparse compared to abundant negative samples and numerous unlabeled instances. As a result, applying these conventional methods would likely cause the positive signals to be overwhelmed by other sample types.

To prevent the issue, we introduce the second-level grouping. We further partition nodes within each group according to their labels: nodes labeled "1" and "0" are grouped separately, while unlabeled nodes are assigned to subgroups based on their attribute similarity to labeled samples, computed using the Euclidean distance as defined in Equation 3.

3.4.3 Grouping of Node and Edge Features. We first apply the proposed grouping modules for each node type, and then aggregate node and edge features to construct the final compressed heterogeneous graph. Specifically, the feature of a grouped node is computed as the average of its member node features. Let group G consist of nodes $\{v_1, v_2, \ldots, v_k\}$, each with feature $\mathbf{x}_i$. The aggregated node feature $\mathbf{x}_G$ is computed as:

$$\mathbf{x}_G = \frac{1}{k}\sum_{i=1}^{k} \mathbf{x}_i \tag{5}$$

For edges, each original edge e_{ij} is reassigned to connect the groups containing nodes v_i and v_j, respectively. If multiple edges

map to the same grouped edge, their edge features are summed. For example, suppose $v_1, v_2 \in G_A$ and $v_3, v_4 \in G_B$, and original edges e_{13}, e_{14}, e_{23} exist with edge features $\mathbf{r}_{13}, \mathbf{r}_{14}, \mathbf{r}_{23}$. The resulting edge feature $\mathbf{r}_{AB}$ between groups G_A and G_B is:

$$\mathbf{r}_{AB} = \sum_{(i,j)\in\{(1,3),(1,4),(2,3)\}} \mathbf{r}_{ij} \tag{6}$$

3.5 Training and Inference

Training is conducted directly on this compressed graph, where the number of neighbors per node is substantially reduced. This enables efficient full-neighbor aggregation without the need for subsampling. During inference, we bypass the grouping step and apply the learned heterogeneous GNN model directly on the original graph to preserve prediction fidelity, which is significantly less resource-intensive.

4 EXPERIMENTS

4.1 Datasets

We utilize the offline dataset collected from JD Logistics to evaluate our model. To build the logistics graph, we first collected over 4 million labeled customers based on sales feedback over 4 months, among which 2% (around 93 thousand) are high-intent customers (i.e., positive sample). Then, using these labeled customers as target nodes, along with their 3-hop neighbors connected through the relations described in Figure 1, we constructed a heterogeneous graph comprising over **1 billion nodes** and **10 billion edges**. To ensure temporal relevance and control graph size, we applied a one-year cutoff when defining key relations. Specifically, for the *Customer-(ship)-Customer* relation, we only include node pairs with shipping interactions within the past year. Similarly, for the Station-(locate)-Customer relation, we retain only customers who sent or received packages at the station during the same period. Empirical analysis confirms that interactions beyond this window contribute little to predicting current customer behavior while significantly inflating graph size, making a one-year window an effective trade-off between informativeness and scalability. For label nodes, we adopt a time-based split: days 0–92 for training, 93–107 for validation, and 108–122 for testing.

4.2 Experimental Settings

Backbone Model. We build our method and baselines upon two representative HGNNs, **SimpleHGN** [26] and **HGT** [15], to validate that our method consistently outperforms baseline sampling approaches regardless of the underlying model architecture. For both methods, we implement 3 layers and the hidden dimension is set to 64 across all layers.

Baseline Methods. We compare our method with the two types of mainstream sampling models: (1) **Random Sampling** [36], which is used by most graph learning methods, selects neighbors uniformly at random. (2) **Importance Sampling** [5, 15] uses node degrees to retain important neighbors. *Sample Size = n* indicates that for each node, up to n neighbors of **each node type** are randomly sampled at each hop. Although recent sampling methods [19, 37] attempt to adaptively learn sampling probabilities for potentially better performance, they incur significant computational overhead. This makes them impractical for billion-scale graphs and thus were excluded from our comparison.

Our Method. (1) **Metapath:** Neighbor filtering and Structure-based Similarity Graph Construction modules require predefined metapaths to guide filtering and structural similarity calculation. We use the following: `c-c`, `c-s-c`, and `c-comp-c` for Customer node; `s-c-s` and `s-comp-s` for Station node; `comp-c-comp` and `comp-s-comp` for Company node. Here `c`, `comp`, `s` denotes customer, company, and station, respectively.

(2) **Graph Fusion Ratio:** α controls the fusion ratio between the attribute-based similarity graph and the structure-based similarity graph. A higher α emphasizes attribute similarity. We set $\alpha_{\text{customer}} = 0.5$ (both factors important), $\alpha_{\text{station}} = 1$ (as station has no attributes), and $\alpha_{\text{company}} = 0.8$ (structure-dominant).

(3) **Compression Ratio:** We control the node compression ratio by adjusting the resolution parameter γ in the graph partitioning algorithm. Lower γ values lead to more aggressive grouping and higher compression. Since customer nodes are the most numerous and highly redundant, we apply stronger compression to them and lighter compression to other node types. Compression ratios in Table 4.3 are roughly computed as the total number of all types of nodes after compression divided by that before compression.

Training and Evaluation. We adopt batch-wise training, where each batch randomly samples 1024 target nodes along with all their neighbors. For graph sampling methods, additional sampling is performed on the neighbors at each hop. An epoch is completed once all target nodes have been processed. We use the Adam optimizer [21] for training and perform a grid search for the learning rate to optimize model performance. Models are trained on the training set, halted when the model's performance on the validation set no longer improves after 15 epochs, and then evaluated on the test set. We use AUC (area under the ROC curve) and AP (Average Precision) as the evaluation metrics. Our implementation is tested on CentOS 7 with 4 24GB Tesla P40 GPUs. We use Python 3.7, Pytorch 1.10.1, and CUDA 11.1.

4.3 Experimental Results

As shown in Table 1, existing graph sampling methods show the following limitations:

- **Reduced accuracy and slow convergence at low sampling rates.** Limited sample size leads to incomplete neighborhood information, weakening signal propagation, and requiring more training steps to converge.
- **Importance Sampling can not fully address information loss.** Although it selects high-degree neighbors to better preserve structural importance, it still overlooks semantically important neighbors, thus still incurring information loss.
- **Rapid memory usage growth due to neighborhood explosion**. Sampling more neighbors causes exponential growth in sampled nodes with each hop [36], leading to high computational and memory overhead.

In contrast, our NeighSqueeze significantly reduces the number of neighbors through grouping, enabling the model to incorporate all available neighbors during training. Consequently, it: (1) Significantly **mitigates information loss and improves overall**

Table 1: Model performance on test dataset. *Time/Ep* denotes the training time per epoch. *#Eps* denotes the total epochs to convergence. Each model is trained 5 times with different seeds. The best results are marked in bold.

Backbone		SimpleHGN					HGT				
Methods		AUC	AP	Time/Ep	#Eps	Mem	AUC	AP	Time/Ep	#Eps	Mem
Random Sampling (size=)	**5**	56.24 ± 0.41	3.28 ± 0.17	5m 13s	72	4.1G	56.13 ± 0.56	3.18 ± 0.19	4m 2s	60	6.5G
	10	58.43 ± 0.34	3.96 ± 0.12	27m 44s	68	24.5G	59.34 ± 0.43	4.11 ± 0.21	23m 32s	52	32.8G
	20	59.21 ± 0.35	4.21 ± 0.07	52m 58s	61	59.2G	60.45 ± 0.39	4.39 ± 0.12	41m 17s	48	73.2G
Importance Sampling (size=)	**5**	56.86 ± 0.39	3.52 ± 0.11	7m 9s	71	4.9G	57.21 ± 0.46	3.87 ± 0.13	6m 23s	56	7.8G
	10	59.37 ± 0.32	4.29 ± 0.09	32m 21s	64	29.3G	60.09 ± 0.27	4.23 ± 0.07	28m 48s	49	36.9G
	20	60.45 ± 0.23	4.42 ± 0.06	51m 16s	58	63.8G	61.75 ± 0.26	4.85 ± 0.09	46m 5s	46	79.4G
NeighSqueeze (ratio~)	**1.1%**	63.17 ± 0.27	4.88 ± 0.07	**4m 54s**	**42**	**3.9G**	63.21 ± 0.22	5.02 ± 0.10	**4m 13s**	39	**4.4G**
	5.3%	64.21 ± 0.18	5.35 ± 0.03	22m 27s	46	19.6G	64.74 ± 0.21	5.47 ± 0.08	21m 56s	38	24.6G
	11.7%	**64.72 ± 0.25**	**5.56 ± 0.08**	42m 15s	43	47.5G	**65.32 ± 0.15**	**5.75 ± 0.07**	39m 45s	**35**	58.5G

*Note that for all methods, we use all neighbors during validation and testing; neighbor sampling is applied only during training.

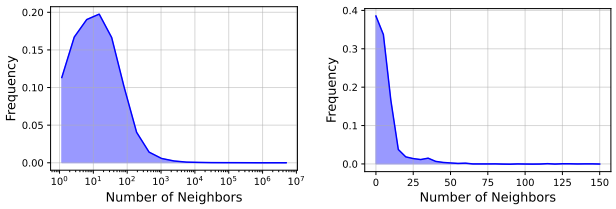
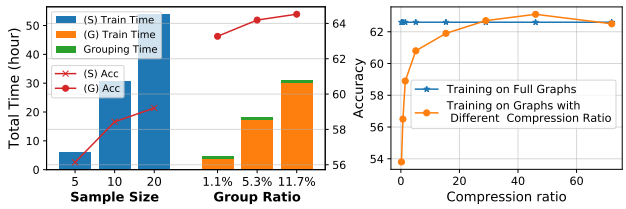


Figure 5: Neighborhood size distribution before (left) and after grouping (right) (compression ratio=5.3%)



(a) Efficiency analysis **(b) Impact of compression ratio**

Figure 6: Efficiency Analysis and Hyper-parameter Analysis. S and G denote important Sampling method and our Grouping method, respectively.

performance; (2) Captures more comprehensive neighborhood information, markedly **accelerating convergence**; (3) Maintains **memory and time usages within acceptable limits.**

4.4 Efficiency Analysis

We evaluate the efficiency of our method from two perspectives: (1) Whether our approach effectively compresses neighborhood sets, enhancing computational efficiency; (2) Whether the grouping process itself introduces excessive computational overhead.

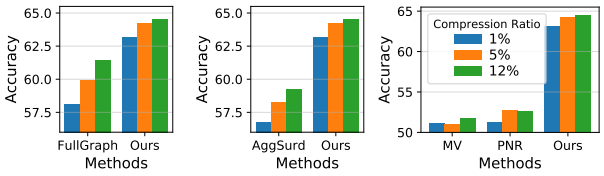
- (1) Figure 5 illustrates the distribution of neighbor counts for customer nodes before and after grouping. Prior to grouping, many customer nodes had more than hundreds of neighbors, making it challenging to efficiently leverage all neighborhood information. After grouping, the neighbor count for most customer nodes is significantly compressed to under 15, enabling efficient training utilizing complete neighbors.
- (2) The computational overhead of our grouping approach primarily arises from the graph partitioning. As demonstrated in Figure 6(a), the total runtime (grouping plus subsequent model training) remains considerably lower than the training time required by the sampling-based method, confirming that our grouping approach introduces minimal additional computational cost and is overall more efficient.

4.5 Hyper-parameter Study

In this section, we analyze how compression ratios affect model performance. On the full graph, training with a compression ratio above 12% leads to an out-of-memory issue. Therefore, we extract a subgraph for model training here, and the result is shown in Figure 6(b): performance degrades only slightly or even improves at the early stage of the compression, possibly because the compression process implicitly performs denoising. When compressing the graph size to 5.1%, we still retain 97.3% of the full-graph training accuracy, indicating effective removal of redundancy. However, when the ratio is lower than 1%, a significant performance drop is observed, suggesting that further compression leads to substantial information loss. In practice, we recommend testing a range of compression levels to identify the most suitable balance.

4.6 Ablation Study

- **Impact of Group Range:** We experiment with two similarity calculation ranges: (a) computing attribute similarities between each node and all other nodes of the same type in the graph, and (b) applying our Structure-tightness-based neighbor filtering method. Our method yields the best results, as (a) introduces excessive noise, making it less effective.
- **Impact of Structure Encoding Methods:** We compare two methods to encode structural information for subsequent similarity calculation, including (a) aggregating neighboring features, where we assign random features to nodes that lack initial attributes; and (b) our Hybrid Similarity Graph. We do not include the method that represents structural embeddings using rows of the one-hop adjacency matrix in our comparison, as this approach would result in embedding dimensions exceeding one



(a) Group Range **(b) Structure Encoding** **(c) Group Label**

Figure 7: Ablation study results.

hundred million in our graph, making subsequent computation infeasible. As shown in Figure 7(b), our method delivers better performance by retaining richer structural context and avoiding information loss caused by flattening structural features.

- **Assign Group Label:** We evaluated three methods: (a) Majority Voting (MV), assigning group labels based on the majority class (0 or 1); (b) Positive-to-Negative Ratio (PNR), assigning labels based on the proportion of positive samples; and (c) our method, which divides nodes with different labels into distinct subgroups. Majority Voting performs poorly, as groups labeled as positive are extremely rare. The Positive-to-Negative Ratio method shows slight improvement, but averaging positive and negative samples' embeddings together introduces substantial feature noise despite capturing positive sample presence in labels. Our method effectively addresses these issues, delivering the best overall performance.

4.7 Online A/B Testing

We deployed our model at JD Logistics, one of China's largest logistics platforms. Its customer expansion system includes three stages: (1) A dynamic pool of unconnected potential customers is maintained. (2) The customer expansion model estimates contract-signing probabilities for all customers in the pool every two weeks. (3) Customers are ranked on these probabilities, and the top K were selected for telephone outreach to identify high-intent ones, i.e., those who show strong interest in signing a contract. Phone call feedback ((whether customers show interest or decline)) is used to update the model biweekly. In the A/B test, the treatment group used our method with compression ratio 5.3% while the control group used JD's rule-based approach and the importance sampling-based HGT (sample size=10). For each method, we select 5,000 customers for telephone outreach, and results are shown in Table 2. Compared to JD's previous method, although the sampling-based HGNN model achieves a slight improvement by leveraging graph learning, its performance is constrained by the information loss issue. In contrast, our NeighSqueeze significantly outperforms both, which demonstrates the effectiveness of preserving neighborhood information through grouping.

5 RELATED WORKS

Customer Expansion aims to identify new customers similar to existing "seed" customers, thus broadening campaign reach [38]. Existing approaches [3, 23, 28] relied only on customer attributes (e.g., demographics, basic behaviors) to predict prospective customers. Recent approaches [9, 33, 38] recast customer expansion as a node classification task on heterogeneous graphs to capture complex structural patterns around a customer, achieving stronger predictive accuracy and coverage than attribute-only models.

Table 2: Results on online A/B testing.

Group	Method	# High-Intent	Rate
Control	JD's Previous	118	2.36%
	Sampling-based	129	2.58% (↑9.3%)
Treatment	NeighSqueeze	153	3.06% (↑29.7%)

Heterogeneous Graph Representation Learning aims at learning node representations using node features and graph structures in the heterogeneous graph [31, 34], supporting many downstream tasks like node classification [12, 35], and link prediction [13]. Compared with homogeneous graph learning, the key challenge in heterogeneous graph representation learning is handling the diversity of node and edge types. RGCN [27] proposes an edge-specific graph convolution network. HetSANN [14] and HGT [15] use the attention mechanism to better integrate diverse edge and node information. Lv et al. [26] establish a comprehensive benchmark for fair comparison of all existing methods and propose the efficient model SimpleHGN inspired by their comparisons.

Graph Sampling aims to reduce the computational cost of training on large graphs by selecting a subset of nodes while preserving essential information [24].Existing methods are broadly categorized into three categories:(1) Random strategies [36] select neighbors at random, while (2) fixed strategies [2, 5, 22] use predefined metrics (node degree or PageRank) to retain important neighbors. Although these methods are high-efficient, they often lead to information loss: random sampling may overlook crucial neighbors, and fixed strategies tend to ignore semantically important ones. (3) Although adaptive strategies [16, 19, 37] improve accuracy by learning to sample informative neighbors via prediction loss minimization, they involve learning to update sampling probabilities for all neighbors at each iteration, resulting in high computational overhead that is particularly prohibitive for billion-scale graphs.

Graph Condensation aims to synthesize a compact yet highly representative graph, allowing GNNs trained on it to achieve performance comparable to those trained on the original large-scale graph [11]. Existing approaches can be categorized into two types: (1) New graph generation [10, 20] constructs a synthetic graph by matching gradients from the original graph to learn node embeddings and edge connections; however, this process is extremely time and memory consuming. (2) In contrast, in-graph condensation [4, 6, 17] operates directly on the original graph by aggregating nodes to form a smaller graph. Yet, as discussed in the Section 1, existing in-graph methods suffer from high computational overhead due to expensive group range selection and similarity calculations, limiting their scalability to billion-scale heterogeneous graphs.

6 CONCLUSION

In this work, we identify and address the critical information loss problem in existing graph sampling methods for high-degree nodes. We propose NeighSqueeze, a scalable neighbor grouping framework featuring structure-tightness filtering, hybrid similarity graph construction, and a two-level grouping strategy. Our deployment at JD Logistics demonstrates significant improvements: a 28.9% increase in average precision offline and an 18.6% increase in customer exploration rate online, while achieving superior runtime and memory efficiency compared to state-of-the-art sub-sampling approaches.

REFERENCES

[1] Amazon. 2025. Amazon. https://www.amazon.com/.

[2] Muhammed Fatih Balin and Ümit ꞓCatalyürek. 2023. Layer-Neighbor Sampling—Defusing Neighborhood Explosion in GNNs. *Advances in Neural Information Processing Systems* 36 (2023), 25819–25836.

[3] Gromit Yeuk-Yin Chan, Tung Mai, Anup B Rao, Ryan A Rossi, Fan Du, Cláudio T Silva, and Juliana Freire. 2021. Interactive audience expansion on large scale online visitor data. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*. 2621–2631.

[4] Hao Chen, Yuanchen Bei, Qijie Shen, Yue Xu, Sheng Zhou, Wenbing Huang, Feiran Huang, Senzhang Wang, and Xiao Huang. 2024. Macro Graph Neural Networks for Online Billion-Scale Recommender Systems. In *Proceedings of the ACM on Web Conference 2024, WWW 2024, Singapore, May 13-17, 2024*, Tat-Seng Chua, Chong-Wah Ngo, Ravi Kumar, Hady W. Lauw, and Roy Ka-Wei Lee (Eds.). ACM, 3598–3608.

[5] Jie Chen, Tengfei Ma, and Cao Xiao. 2018. FastGCN: Fast Learning with Graph Convolutional Networks via Importance Sampling. In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*. OpenReview.net.

[6] Charles Dickens, Edward Huang, Aishwarya Reganti, Jiong Zhu, Karthik Subbian, and Danai Koutra. 2024. Graph coarsening via convolution matching for scalable graph neural network training. In *Companion Proceedings of the ACM Web Conference 2024*. 1502–1510.

[7] SF Express. 2025. SF Express. https://www.sf-express.com/

[8] Fedex. 2025. Fedex. https://www.fedex.com/

[9] Xinyue Feng, Jinquan Hang, Yuequn Zhang, Haotian Wang, Desheng Zhang, and Guang Wang. 2024. InLINE: Inner-Layer Information Exchange for Multi-task Learning on Heterogeneous Graphs. *arXiv preprint arXiv:2410.22089* (2024).

[10] Jian Gao, Jianshe Wu, and Jingyi Ding. 2024. Heterogeneous graph condensation. *IEEE Transactions on Knowledge and Data Engineering* 36, 7 (2024), 3126–3138.

[11] Xinyi Gao, Junliang Yu, Tong Chen, Guanhua Ye, Wentao Zhang, and Hongzhi Yin. 2025. Graph condensation: A survey. *IEEE Transactions on Knowledge and Data Engineering* (2025).

[12] Jinquan Hang, Zhiqing Hong, Xinyue Feng, Guang Wang, Dongjiang Cao, Jiayang Qiao, Haotian Wang, and Desheng Zhang. 2024. Complex-Path: Effective and Efficient Node Ranking with Paths in Billion-Scale Heterogeneous Graphs. *Proceedings of the VLDB Endowment* 17, 12 (2024), 3973–3986.

[13] Jinquan Hang, Zhiqing Hong, Xinyue Feng, Guang Wang, Guang Yang, Feng Li, Xining Song, and Desheng Zhang. 2024. Paths2pair: Meta-path based link prediction in billion-scale commercial heterogeneous graphs. In *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*. 5082–5092.

[14] Huiting Hong, Hantao Guo, Yucheng Lin, Xiaoqing Yang, Zang Li, and Jieping Ye. 2020. An Attention-Based Graph Neural Network for Heterogeneous Structural Learning. In *The Thirty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2020*. AAAI Press, 4132–4139.

[15] Ziniu Hu, Yuxiao Dong, Kuansan Wang, and Yizhou Sun. 2020. Heterogeneous Graph Transformer. In *WWW '20: The Web Conference 2020*. ACM / IW3C2, 2704–2710.

[16] Wenbing Huang, Tong Zhang, Yu Rong, and Junzhou Huang. 2018. Adaptive sampling towards fast graph representation learning. *Advances in neural information processing systems* 31 (2018).

[17] Zengfeng Huang, Shengzhong Zhang, Chong Xi, Tang Liu, and Min Zhou. 2021. Scaling up graph neural networks via graph coarsening. In *Proceedings of the 27th ACM SIGKDD conference on knowledge discovery & data mining*. 675–684.

[18] Ahmet Iscen, Giorgos Tolias, Yannis Avrithis, and Ondrej Chum. 2019. Label propagation for deep semi-supervised learning. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 5070–5079.

[19] Yugang Ji, Mingyang Yin, Hongxia Yang, Jingren Zhou, Vincent W Zheng, Chuan Shi, and Yuan Fang. 2020. Accelerating large-scale heterogeneous interaction graph embedding learning via importance sampling. *ACM Transactions on Knowledge Discovery from Data (TKDD)* 15, 1 (2020), 1–23.

[20] Wei Jin, Lingxiao Zhao, Shichang Zhang, Yozen Liu, Jiliang Tang, and Neil Shah. 2022. Graph Condensation for Graph Neural Networks. In *The Tenth International Conference on Learning Representations, ICLR 2022, Virtual Event, April 25-29, 2022*.

[21] Diederik P. Kingma and Jimmy Ba. 2015. Adam: A Method for Stochastic Optimization. In *3rd International Conference on Learning Representations, ICLR 2015*, Yoshua Bengio and Yann LeCun (Eds.).

[22] Johannes Klicpera, Aleksandar Bojchevski, and Stephan Günnemann. 2019. Predict then Propagate: Graph Neural Networks meet Personalized PageRank. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net. https://openreview.net/forum?id=H1gL-2A9Ym

[23] Haishan Liu, David Pardoe, Kun Liu, Manoj Thakur, Frank Cao, and Chongzhe Li. 2016. Audience expansion for online social network advertising. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 165–174.

[24] Xin Liu, Mingyu Yan, Lei Deng, Guoqi Li, Xiaochun Ye, and Dongrui Fan. 2021. Sampling methods for efficient training of graph convolutional networks: A survey. *IEEE/CAA Journal of Automatica Sinica* 9, 2 (2021), 205–234.

[25] JD Logistics. 2025. JD Logistics. https://www.jingdonglogistics.com/

[26] Qingsong Lv, Ming Ding, Qiang Liu, Yuxiang Chen, Wenzheng Feng, Siming He, Chang Zhou, Jianguo Jiang, Yuxiao Dong, and Jie Tang. 2021. Are we really making much progress?: Revisiting, benchmarking and refining heterogeneous graph neural networks. In *KDD '21: The 27th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*. ACM, 1150–1160.

[27] Michael Sejr Schlichtkrull, Thomas N. Kipf, Peter Bloem, Rianne van den Berg, Ivan Titov, and Max Welling. 2018. Modeling Relational Data with Graph Convolutional Networks. In *The Semantic Web - 15th International Conference, ESWC 2018 (Lecture Notes in Computer Science, Vol. 10843)*. Springer, 593–607.

[28] Jianqiang Shen, Sahin Cem Geyik, and Ali Dasdan. 2015. Effective audience extension in online advertising. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 2099–2108.

[29] Vincent A Traag, Ludo Waltman, and Nees Jan Van Eck. 2019. From Louvain to Leiden: guaranteeing well-connected communities. *Scientific reports* 9, 1 (2019), 1–12.

[30] Hongwei Wang and Jure Leskovec. 2020. Unifying graph convolutional neural networks and label propagation. *arXiv preprint arXiv:2002.06755* (2020).

[31] Xiao Wang, Deyu Bo, Chuan Shi, Shaohua Fan, Yanfang Ye, and Philip S. Yu. 2023. A Survey on Heterogeneous Graph Embedding: Methods, Techniques, Applications and Sources. *IEEE Trans. Big Data* 9, 2 (2023), 415–436.

[32] Dongbo Xi, Zhen Chen, Peng Yan, Yinger Zhang, Yongchun Zhu, Fuzhen Zhuang, and Yu Chen. 2021. Modeling the Sequential Dependence among Audience Multi-step Conversions with Multi-task Learning in Targeted Display Advertising. In *KDD '21: The 27th ACM SIGKDD Conference on Knowledge Discovery and Data Mining, Virtual Event, Singapore, August 14-18, 2021*, Feida Zhu, Beng Chin Ooi, and Chunyan Miao (Eds.). ACM, 3745–3755.

[33] Hua Yan, Yingqiang Ge, Haotian Wang, Desheng Zhang, and Yu Yang. 2023. Logistics Audience Expansion via Temporal Knowledge Graph. In *Proceedings of the 32nd ACM International Conference on Information and Knowledge Management, CIKM 2023, Birmingham, United Kingdom, October 21-25, 2023*, Ingo Frommholz, Frank Hopfgartner, Mark Lee, Michael Oakes, Mounia Lalmas, Min Zhang, and Rodrygo L. T. Santos (Eds.). ACM, 4879–4886.

[34] Carl J. Yang, Yuxin Xiao, Yu Zhang, Yizhou Sun, and Jiawei Han. 2022. Heterogeneous Network Representation Learning: A Unified Framework With Survey and Benchmark. *IEEE Trans. Knowl. Data Eng.* 34, 10 (2022), 4854–4873.

[35] Guang Yang, Yuequn Zhang, Jinquan Hang, Xinyue Feng, Zejun Xie, Desheng Zhang, and Yu Yang. 2023. CARPG: Cross-City Knowledge Transfer for Traffic Accident Prediction via Attentive Region-Level Parameter Generation. In *Proceedings of the 32nd ACM International Conference on Information and Knowledge Management*. 2939–2948.

[36] Hanqing Zeng, Hongkuan Zhou, Ajitesh Srivastava, Rajgopal Kannan, and Viktor K. Prasanna. 2020. GraphSAINT: Graph Sampling Based Inductive Learning Method. In *8th International Conference on Learning Representations, ICLR 2020*. OpenReview.net.

[37] Zaixi Zhang, Qi Liu, Qingyong Hu, and Chee-Kong Lee. 2022. Hierarchical graph transformer with adaptive node sampling. *Advances in Neural Information Processing Systems* 35 (2022), 21171–21183.

[38] Chenyi Zhuang, Ziqi Liu, Zhiqiang Zhang, Yize Tan, Zhengwei Wu, Zhining Liu, Jianping Wei, Jinjie Gu, Guannan Zhang, Jun Zhou, and Yuan Qi. 2020. Hubble: An Industrial System for Audience Expansion in Mobile Marketing. In *KDD '20: The 26th ACM SIGKDD Conference on Knowledge Discovery and Data Mining, Virtual Event, CA, USA, August 23-27, 2020*, Rajesh Gupta, Yan Liu, Jiliang Tang, and B. Aditya Prakash (Eds.). ACM, 2455–2463.

Received 08 February 2024; revised ; accepted