Echoless Label-Based Pre-computation for Memory-Efficient Heterogeneous Graph Learning

Jun Hu[1], **Shangheng Chen**[2], **Yufei He**[1], **Yuan Li**[1], **Bryan Hooi**[1], **Bingsheng He**[1]
[1]National University of Singapore
[2]Institute of Automation, CAS
jun.hu@nus.edu.sg, chenshangheng23@mails.ucas.ac.cn, {yufei.he, li.yuan}@u.nus.edu, {dcsbhk, dcsheb}@nus.edu.sg

Abstract

Heterogeneous Graph Neural Networks (HGNNs) are widely used for deep learning on heterogeneous graphs. Typical end-to-end HGNNs require repetitive message passing during training, limiting efficiency for large-scale real-world graphs. Pre-computation-based HGNNs address this by performing message passing only once during preprocessing, collecting neighbor information into regular-shaped tensors, which enables efficient mini-batch training. Label-based pre-computation methods collect neighbors' label information but suffer from training label leakage, where a node's own label information propagates back to itself during multi-hop message passing—the echo effect. Existing mitigation strategies are memory-inefficient on large graphs or suffer from compatibility issues with advanced message passing methods. We propose **Echoless Label-based Pre-computation (Echoless-LP)**, which eliminates training label leakage with Partition-Focused Echoless Propagation (PFEP). PFEP partitions target nodes and performs echoless propagation, where nodes in each partition collect label information only from neighbors in other partitions, avoiding echo while remaining memory-efficient and compatible with any message passing method. We also introduce an Asymmetric Partitioning Scheme (APS) and a PostAdjust mechanism to address information loss from partitioning and distributional shifts across partitions. Experiments on public datasets demonstrate that Echoless-LP achieves superior performance and maintains memory efficiency compared to baselines.

Code — https://github.com/CrawlScript/Echoless-LP

Introduction

Heterogeneous graphs contain multiple types of vertices and edges, unlike their homogeneous counterparts, making them well-suited for modeling complex real-world applications. **Heterogeneous Graph Neural Networks (HGNNs)** emerge as powerful tools for deep learning on heterogeneous graphs, with applications spanning various domains such as social recommendation systems (Zhou and Shen 2023) and research classification (Zhang et al. 2022a; Hu et al. 2020b).

Typical HGNNs, such as RGCN (Schlichtkrull et al. 2018) and HAN (Wang et al. 2019), perform **end-to-end training** on graphs. During each forward pass, neighbors within K hops are aggregated through message passing (Kipf and Welling 2016; Velickovic et al. 2018; Hamilton, Ying, and Leskovec 2017). On heterogeneous graphs, neighbor aggregation is complicated and may involve multiple relation types, diverse node types, and complex meta-path traversals. This **repetitive message passing** occupies over 90% of total training time (Hu, Hooi, and He 2024), causing scalability bottlenecks on large graphs.

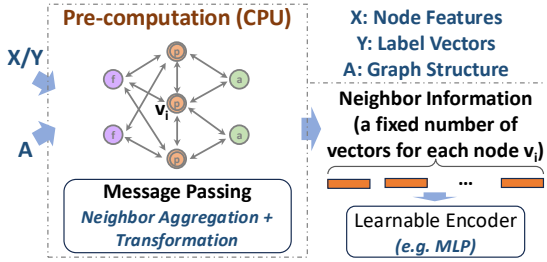
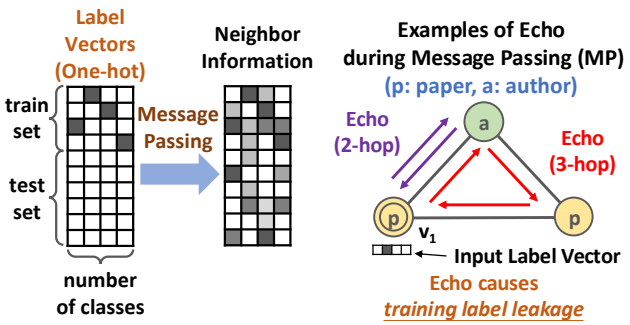


Figure 1: Feature/label-based pre-computation.

Pre-computation for Efficient HGNNs. To address the efficiency limitations caused by repetitive message passing, pre-computation-based HGNNs emerge as a practical solution that only requires one-time message passing in preprocessing, and requires no graph operations during training. As shown in Figure 1, these methods operate in two stages: (1) a pre-computation stage that performs one-time message passing on CPUs to collect neighbor information within K hops, generating a fixed number of vectors for each node to form regular-shaped tensors, and (2) a training stage where the encoder (e.g., MLP) takes the pre-computed tensors as input and performs training without any graph operations, thus enabling efficient mini-batch training. Depending on the input used for message passing, we distinguish between Feature-based Pre-computation (using node features X) and **Label-based Pre-computation** (using label vectors Y). Figure 2a shows an example of $Y \in \{0, 1\}^{N \times C}$, where training nodes use one-hot vectors and other nodes use zero vectors, and N and C are the number of nodes and classes, respectively.

Label-based pre-computation methods suffer from **training label leakage**, where a node's own label information is propagated back to itself during multi-hop message passing, as illustrated in Figure 2b. We define this phenomenon as the



(a) Input/Output of target nodes. (b) Examples of echo.

Figure 2: v_1 is also a multi-hop neighbor of itself, and the 2-hop MP and 3-hop MP propagate v_1's own label back to itself (**echo**), causing **training label leakage** for v_1.

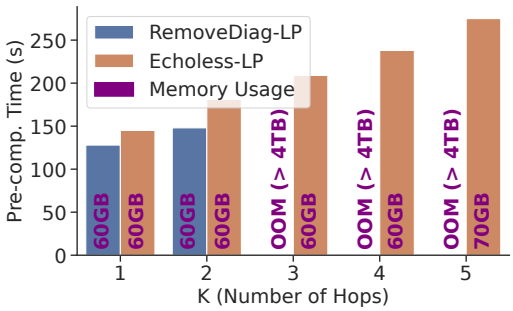


Figure 3: Memory usage (purple text) vs. number of hops K for label-based pre-computation on OAG-Venue (million-scale). Although Echoless-LP incurs a modest increase in pre-computation time (y-axis), it remains memory-efficient for $K > 2$, whereas RemoveDiag-LP (SOTA) runs out of memory (OOM).

echo effect. Echo causes a node's own label information to leak into the collected neighbor information (i.e., the node's representations). During training, the learnable encoder may become reliant on this leaked self-label information, which is only available for training nodes but not for test nodes, leading to poor generalization performance.

Limitations of Existing Mitigation Strategies. Several methods have been proposed to address the echo effect in label-based pre-computation, such as high-hop filtering and propagation adjustment. (1) Last Residual Connection (Zhang et al. 2022b) (referred to as LastResidual-LP in this paper) attempts to reduce label leakage by emphasizing neighbor information from higher-hop neighborhoods, where echo is generally less severe. However, echo can still occur at higher hops, so this approach **only partially alleviates** training label leakage and cannot eliminate it entirely. (2) RemoveDiag-LP (Yang et al. 2023) (a name we adopt for clarity) can avoid echo for linear message passing of the form $A_K A_{K-1} \dots A_1 Y$, with each A_i being a normalized adjacency matrix at hop i. RemoveDiag-LP computes the entire multi-hop propagation as $\tilde{A} = A_K A_{K-1} \dots A_1 \in \mathbb{R}^{N \times N}$ and then removes the diagonal for message passing: $(\tilde{A} - \text{diag}(\tilde{A}))Y$, where the diagonal removal effectively blocks echo. This changes the efficient computation order $A_K(A_{K-1} \dots (A_1 Y) \dots)$, and the diagonal computation may involve the explicit construction of $\tilde{A}$, which may become increasingly dense when $K > 2$, leading to terabyte-scale memory usage for million-node graphs and making the method **memory-inefficient** (see Figure 3). Furthermore, as mentioned before, RemoveDiag-LP is designed upon linear message passing of the form $A_K A_{K-1} \dots A_1 Y$, resulting in **compatibility issues** with more advanced message passing such as RpHGNN, which involve complex operations like feature normalization.

We propose **Echoless Label-based Pre-computation (Echoless-LP)**, a memory-efficient and compatible framework that eliminates training label leakage in label-based pre-computation on large graphs. Echoless partitions target nodes into several partitions, and introduces Partition-Focused Echoless Propagation (PFEP). PFEP performs echoless propagation via partition masks to collect neighbor information for one partition at a time, where each node can only collect information from neighbors in other partitions, avoiding echo. PFEP is **compatible with any message passing method** used in pre-computation, and message passing proceeds as usual without modification, avoiding costly memory overhead and ensuring **memory efficiency**. We also introduce an Asymmetric Partitioning Scheme (APS) and a **PostAdjust mechanism** to address potential information loss from partitioning and distributional shifts across partitions, respectively. In experiments, we show that Echoless-LP integrates seamlessly with various message passing methods for pre-computation, achieves state-of-the-art (SOTA) performance, and efficiently supports high-hop propagation on large graphs (see Figure 3).

Our main contributions are as follows:

- We propose Echoless Label-based Pre-computation (Echoless-LP), which eliminates training label leakage caused by **echo** with a Partition-Focused Echoless Propagation (PFEP) component. PFEP is **compatible with any message passing method**, and message passing proceeds as usual without modification, avoiding costly memory overhead and ensuring **memory efficiency**.
- We introduce an Asymmetric Partitioning Scheme (APS) and a PostAdjust mechanism to address information loss from partitioning and distributional shifts across partitions, respectively.
- Experiments on public datasets demonstrate that Echoless-LP achieves superior performance and maintains memory efficiency.

Related Work

Heterogeneous Graph Neural Networks

HGNNs leverage diverse node and edge types (relations) to capture rich semantics. RGCN (Schlichtkrull et al. 2018) extends GCN (Kipf and Welling 2016) and handles each relation type separately during message passing. HAN (Wang et al. 2019), MAGNN (Fu et al. 2020), and GTN (Yun

et al. 2019) consider meta-path-based message passing, using hierarchical attention, intra/inter-metapath aggregation, and automatic meta-path learning, respectively. RSHN uses coarsened line graphs to capture relation semantics (Zhu et al. 2019). HetSANN (Hong et al. 2020) and HGT (Hu et al. 2020c) use attention-based approaches with type-aware attention layers and type-specific Transformers, respectively. Simple-HGN (Lv et al. 2021) extends GAT (Velickovic et al. 2018) with relation-aware neighbor attention. HINormer (Mao et al. 2023a) proposes to use Transformers with local structure and heterogeneous relation encoders.

Pre-computation-based Heterogeneous Graph Neural Networks

Pre-computation employs one-time CPU message passing to collect neighbor information, avoiding costly graph operations during training. SIGN (Frasca et al. 2020) collects neighbor information at each hop separately for homogeneous graphs and then encodes it via MLPs. NARS (Yu et al. 2020) samples relation-based subgraphs for heterogeneous graphs and applies SIGN on them. SeHGNN (Yang et al. 2023) collects neighbor information via meta-paths. RpHGNN (Hu, Hooi, and He 2024) collects fine-grained multi-hop neighbor information, with complex operations (e.g., random projection, feature normalization) in message passing to ensure efficiency. Given a specific pre-computation method, we can perform feature-based or label-based pre-computation, depending on whether we use node features or label information for message passing.

Label-based pre-computation may suffer from training label leakage caused by echo, and existing research mainly focuses on mitigation strategies. Last Residual Connection (Zhang et al. 2022b) (LastResidual-LP) attempts to reduce label leakage by emphasizing higher-hop neighbors where echo is less severe, but only partially alleviates the problem as echo still occurs at higher hops. RemoveDiag-LP (Yang et al. 2023) removes diagonal elements from multi-hop adjacency matrices to block echo for linear message passing, but this approach is memory-inefficient for large graphs and incompatible with complex operations like feature normalization used in advanced methods.

Different from existing methods, our Echoless-LP avoids training label leakage caused by echo via echoless partition-focused propagation, and is compatible with any message passing method and memory-efficient on large graphs.

Preliminary

Target Node Type (Target Type). It is common for tasks to focus on only one node type. For example, in an academic graph with papers and authors, the task may focus on classifying papers. We refer to this specified type as the target type. Nodes of the target type are defined as **target nodes**.

Linear Message Passing. Linear message passing refers to message passing of the form $A_K A_{K-1} \dots A_1 Y$, where each A_i is a normalized adjacency matrix at hop i. $\tilde{A} = A_K A_{K-1} \dots A_1$ is the entire **multi-hop propagation matrix**, which is not required to be explicitly computed for message passing due to efficiency issues.

Efficiency of Diagonal Computation of Multi-hop Propagation Matrix. The SOTA method RemoveDiag-LP relies on diagonal removal. (1) When the number of hops K is 2, $\tilde{A} = A_2 A_1$, there is a trick to efficiently compute the diagonal without explicitly computing $\tilde{A}$: $\text{diag}(\tilde{A}) = (A_2 \odot A_1^T)\mathbf{1}$, where $\odot$ denotes element-wise multiplication and $\mathbf{1}$ is a vector of ones. Using sparse matrix operations, the time and space complexity is $\mathcal{O}(E+N)$, where N and E are the number of nodes and edges, respectively. (2) However, when $K > 2$, this trick does not apply, and we may have to explicitly compute the potentially dense matrix $\tilde{A} \in \mathbb{R}^{N \times N}$, which can reach terabyte-scale memory usage for million-node graphs, resulting in prohibitive memory overhead.

Method

In this section, we introduce our Echoless-LP framework.

Echoless Label-based Pre-computation

We propose Echoless Label-based Pre-computation (Echoless-LP), which avoids training label leakage caused by echo. The core idea of Echoless-LP is masking a node's input label vector to zero when collecting the node's neighbor information, which completely avoids the node's label information being propagated to itself (echo). For efficiency, we implement Echoless-LP with three key operations:

- **Partition**: Divide target nodes into multiple partitions using a certain partitioning scheme.
- **Partition-Focused Echoless Propagation (PFEP)**: Perform echoless propagation via partition masks to collect neighbor information for one partition at a time, where nodes in each partition collect label information only from neighbors in other partitions, avoiding echo.
- **Partition Output Merge**: Combine the results from all partitions to produce the final node representations.

Formally, given a heterogeneous graph G, let V_t denote the set of target nodes to classify with $|V_t| = N$. We have target node features $X \in \mathbb{R}^{N \times d}$ and label vectors $Y \in \mathbb{R}^{N \times C}$, where d is the feature dimension and C is the number of classes. The three components operate as below:

Partition. We divide the target nodes V_t into M disjoint partitions $\{P_1, P_2, \dots, P_M\}$ where:

$$\bigcup_{i=1}^{M} P_i = V_t, \quad P_i \cap P_j = \emptyset \text{ for } i \neq j \tag{1}$$

The partitioning scheme aims to minimize information loss by considering the graph structure and label distribution. Each partition P_i contains a subset of target nodes that will be processed together in the subsequent propagation step. We discuss our partitioning scheme in the next subsection.

Partition-Focused Echoless Propagation (PFEP). For each partition P_i, we define a partition mask $\mathcal{M}_i$:

$$\mathcal{M}_i \in \{0,1\}^{|V_t|}, \quad \mathcal{M}_i[j] = \begin{cases} 1 & \text{if target node } j \in P_i \\ 0 & \text{otherwise} \end{cases} \tag{2}$$

This mask ensures that only nodes in the current partition have their representations updated. We then apply a message propagation function that produces intermediate representations. Given a message passing operator $\mathcal{F}_{\text{MP}}$ used for

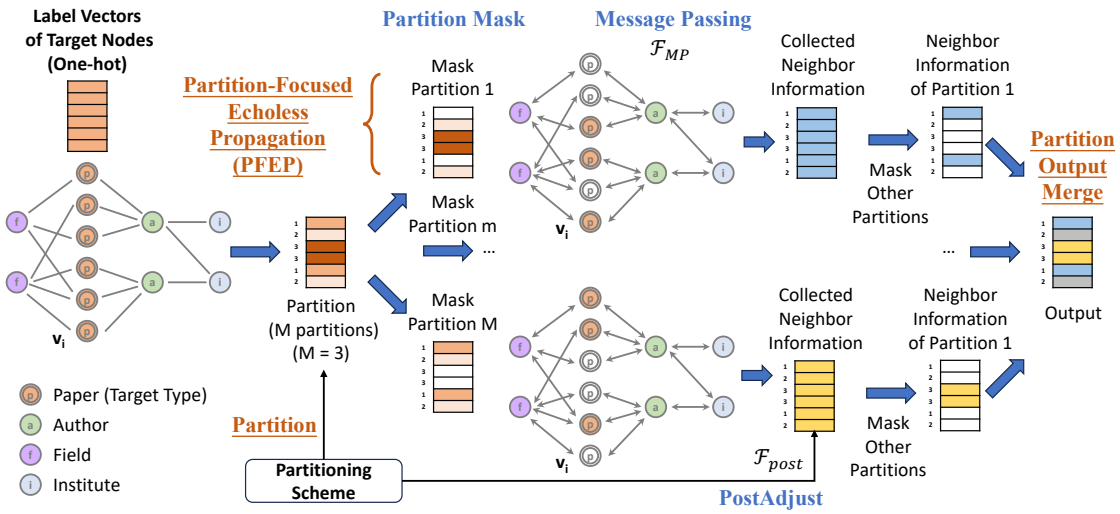
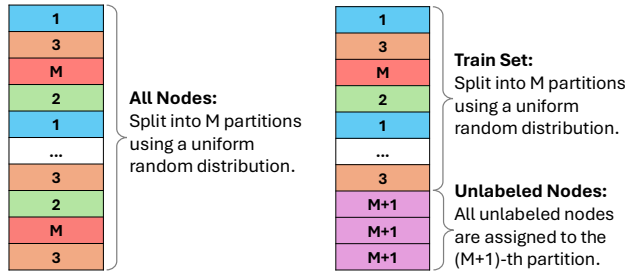


Figure 4: Overall Framework of Echoless Label-based Pre-computation (Echoless-LP).



(a) Uniform Random Partitioning. (b) Asymmetric Partitioning.

Figure 5: Partitioning Schemes.

feature-based pre-computation, which collects neighbor information as $H_i = \mathcal{F}_{\text{MP}}(X, G)$, we adapt it for PFEP by replacing the input node features with masked label vectors:

$$H_i = \mathcal{F}_{\text{MP}}(\text{diag}(\mathbf{1} - \mathcal{M}_i)Y, G) \tag{3}$$

Here, while message passing occurs across the entire graph G, the mask $(\mathbf{1} - \mathcal{M}_i)$ prevents label information of nodes in the current partition P_i from being used when collecting neighbor information, achieving echoless propagation.

To address potential distribution shifts between partitions, we apply a **PostAdjust** mechanism $\mathcal{F}_{\text{post}}$ to each partition's output, which adjusts the output distribution across partitions as $\tilde{H}_i = \mathcal{F}_{\text{post}}(H_i)$. The specific form of $\mathcal{F}_{\text{post}}$ depends on the partitioning scheme. In the next subsection, we introduce the $\mathcal{F}_{\text{post}}$ designed for our specific partitioning scheme.

Partition Output Merge. After PFEP, we obtain adjusted representations $\{\tilde{H}_1, \tilde{H}_2, \ldots, \tilde{H}_M\}$ from all M partitions, which are then merged to produce the final target node representations as $H_{\text{label}} = \sum_{i=1}^{M} \text{diag}(\mathcal{M}_i)\tilde{H}_i$, where $\text{diag}(\mathcal{M}_i)\tilde{H}_i$ selects only the representations of nodes in partition P_i, and the summation combines the partition-specific representations into a unified output.

Asymmetric Partitioning Scheme (APS)

Partitioning schemes affect performance. A naive approach would uniformly and randomly partition all target nodes into M partitions (see Figure 5a), but this may cause information loss. When processing partition P_i, PFEP masks all nodes in that partition. Under uniform random partitioning, for any node $v \in P_i$ collecting neighbor information, each of its neighbors has probability $1/M$ of also being in P_i and thus having their labels masked, causing information loss.

To alleviate the information loss caused by partitioning, we propose APS, ensuring unlabeled nodes retain all neighbor label information without information loss (see Figure 5b). Given labeled target nodes $V_{\text{train}} \subseteq V_t$ and unlabeled target nodes $V_{\text{unlabeled}} \subseteq V_t$, APS creates $M + 1$ partitions:

- **Training nodes:** Uniformly and randomly partitioned into M disjoint partitions $\{P_1, \ldots, P_M\}$ where $\bigcup_{i=1}^{M} P_i = V_{\text{train}}$ and each P_i contains approximately $|V_{\text{train}}|/M$ nodes.
- **Unlabeled nodes (validation/test):** Assigned to a dedicated partition $P_{M+1} = V_{\text{unlabeled}}$.

With APS, when collecting neighbor information for unlabeled nodes, only the nodes in P_{M+1} (unlabeled nodes only) are masked, ensuring no neighbor label information is lost.

PostAdjust in APS. For APS, the PostAdjust mechanism is implemented as renormalization to alleviate distribution shifts. The asymmetry in APS creates distribution shifts where unlabeled nodes may accumulate neighbor information of different magnitude compared to training nodes.

To alleviate this shift, we first augment the input by creating $\overline{Y} = [\mathbf{1}_{\text{train}} \mid Y] \in \mathbb{R}^{N \times (C+1)}$ where $\mathbf{1}_{\text{train}} \in \{0, 1\}^N$ indicates training nodes. The indicator column tracks retained neighbor information after masking. After message passing $H_i = \mathcal{F}_{\text{MP}}(\text{diag}(\mathbf{1} - \mathcal{M}_i)\overline{Y}, G)$, we obtain $[\mathbf{r}_i \mid \overline{H}_i] = H_i$ where $\mathbf{r}_i \in \mathbb{R}^{N \times 1}$ contains the retention ratios. We normalize each row as follows:

$$\tilde{H}_i = \text{diag}\left(\frac{\max(\mathbf{r})}{\mathbf{r}_i}\right)\overline{H}_i \tag{4}$$

where $\mathbf{r} = \sum_{i=1}^{M} \text{diag}(\mathcal{M}_i)\mathbf{r}_i \in \mathbb{R}^{N\times 1}$ is the merged retention vector. The division is element-wise, and the global maximum retention ratio $\max(\mathbf{r}) \in \mathbb{R}$ ensures that all node representations are scaled to the same reference level.

Integration with Feature-based Pre-computation

Our framework operates alongside existing feature-based pre-computation methods (e.g., RpHGNN)—which we term the **backbone**—using their message passing and learnable encoders for both features and labels.

Message Passing. Feature-based pre-computation methods usually apply K^{feat} message passing operations $\{\mathcal{F}_{\text{MP}}^{(1)}, \mathcal{F}_{\text{MP}}^{(2)}, \ldots, \mathcal{F}_{\text{MP}}^{(K^{\text{feat}})}\}$ to collect K^{feat} tensors:

$$H_{\text{feat}}^{(k)} = \mathcal{F}_{\text{MP}}^{(k)}(X, G), \quad k = 1, 2, \ldots, K^{\text{feat}} \tag{5}$$

where $\mathcal{F}_{\text{MP}}^{(k)}$ differs in message passing hyperparameter k. For simplicity, we use the hop index as the hyperparameter k, which is supported by most backbones. For label-based pre-computation, we adopt the same message passing operations (but with a different number K) to collect K tensors:

$$H_{\text{label}}^{(k)} = \text{Echoless-LP}(Y, G, \mathcal{F}_{\text{MP}}^{(k)}), \quad k = 1, 2, \ldots, K \tag{6}$$

Learnable Encoder. The collected tensors for features and labels are combined and processed by the backbone's encoder (e.g., hierarchical MLPs) for final classification:

$$\hat{Y} = \text{Encoder}(\{H_{\text{feat}}^{(1)}, \ldots, H_{\text{feat}}^{(K^{\text{feat}})}, H_{\text{label}}^{(1)}, \ldots, H_{\text{label}}^{(K)}\}) \tag{7}$$

Complexity Analysis

Echoless-LP's complexity depends on the integrated backbone's message passing space/time complexity $\mathcal{O}(\mathcal{F}_{\text{MP}})$.

Space Complexity. Since message passing on each partition can be performed independently, the space complexity is $\mathcal{O}(\mathcal{F}_{\text{MP}}) + \mathcal{O}(NC)$, matching the backbone's complexity, with an additional $\mathcal{O}(NC)$ term that accounts for storing intermediate label vectors during partition masking and merging. Thus, our method remains memory-efficient.

Time Complexity. The overall time complexity of Echoless-LP is $\mathcal{O}(M \cdot \mathcal{O}(\mathcal{F}_{\text{MP}})) + \mathcal{O}(NC)$. The dominant $\mathcal{O}(M \cdot \mathcal{O}(\mathcal{F}_{\text{MP}}))$ term comes from performing message passing on M partitions separately by PFEP, while the $\mathcal{O}(NC)$ term accounts for PostAdjust, partition masking and merging on label vectors. Our experiments show that usually a small M of 2-4 is sufficient, keeping the overhead modest.

Experiments

In this section, we conduct node classification experiments on public datasets to verify the effectiveness of our methods. All experiments are conducted on a Linux machine with dual Intel(R) Xeon(R) E5-2690 v4 CPUs, 128 GB RAM, and a GeForce GTX 1080 Ti (11 GB).

Datasets and Task

We use various public datasets widely adopted by existing work, including three small-scale datasets: DBLP, IMDB, and Freebase from the HGB benchmark (Lv et al. 2021), and three large-scale datasets: OGBN-MAG (Hu et al. 2020a), OAG-Venue, and OAG-L1-Field (Sinha et al. 2015; Tang et al. 2008; Zhang et al. 2019b), which have been used in large-scale HGNN research (Yu et al. 2020; Hu, Hooi, and He 2024). Statistics for all datasets are provided in Table 1.

Dataset	Vertices	Vertex Types	Edges	Edge Types	Target Type	Target Classes
DBLP	26,128	4	239,566	3	`author`	4
IMDB	21,420	4	86,642	3	`movie`	5
Freebase	180,098	8	1,057,688	36	`book`	7
OGBN-MAG	1,939,743	4	21,111,007	4	`paper`	349
OAG-Venue	1,116,162	5	13,985,692	15	`paper`	3505
OAG-L1-Field	1,116,163	5	13,016,549	15	`paper`	275

Table 1: Statistics of datasets.

Each dataset specifies a target node type, and the task is to classify nodes of this type (target nodes). We use the fixed train/valid/test splits officially provided by the datasets.

Baselines

We use two main categories of baselines: (1) HGNN-based approaches, including homogeneous GNNs (GCN (Kipf and Welling 2016), GAT (Velickovic et al. 2018), GraphSAGE (Hamilton, Ying, and Leskovec 2017)), heterogeneous GNNs (RGCN (Schlichtkrull et al. 2018), HAN (Wang et al. 2019), GTN (Yun et al. 2019), RSHN (Zhu et al. 2019), HetGNN (Zhang et al. 2019a), MAGNN (Fu et al. 2020), HetSANN (Hong et al. 2020), HGT (Hu et al. 2020c), Simple-HGN (Lv et al. 2021), HINormer (Mao et al. 2023b)), and pre-computation-based HGNNs (NARS (Yu et al. 2020), SeHGNN (Yang et al. 2023), RpHGNN (Hu, Hooi, and He 2024)); and (2) label-based pre-computation methods that leverage label information as plug-in modules for pre-computation-based HGNNs.

The label-based pre-computation baselines include LastResidual-LP (Zhang et al. 2022b) and RemoveDiag-LP (Yang et al. 2023). All of them are integrated with backbones NARS, SeHGNN, and RpHGNN. One exception is that RemoveDiag-LP only supports linear message passing and is therefore incompatible with RpHGNN, which has complex message passing operations. We use italic font to denote the backbone, e.g., Echoless-LP (*RpHGNN*) denotes Echoless-LP with RpHGNN as the backbone.

Evaluation Metrics and Parameter Settings

Following existing work (Yu et al. 2020; Yang et al. 2023; Hu, Hooi, and He 2024), we use Macro-F1 and Micro-F1 for DBLP, IMDB, and Freebase; accuracy for OGBN-MAG; and NDCG and MRR for OAG-Venue and OAG-L1-Field.

For all baselines except label-based pre-computation methods, we adopt existing results from the literature (Lv et al. 2021; Hu, Hooi, and He 2024). For label-based pre-computation methods, we consider the integration of each method with three backbones, which is not covered by existing research. We carefully implement these integrations using the official code of each label-based pre-computation method and backbone. We perform each experiment using 10 different random seeds and report average performance with standard deviation. Following prior work (Yu et al.

	Small						Large				
	DBLP		IMDB		Freebase		OGBN-MAG	OAG-Venue		OAG-L1-Field	
	Macro-F1	Micro-F1	Macro-F1	Micro-F1	Macro-F1	Micro-F1	Accuracy	NDCG	MRR	NDCG	MRR
GCN	90.84±0.32	91.47±0.34	57.88±1.18	64.82±0.64	27.84±3.13	60.23±0.92	OOM	OOM	OOM	OOM	OOM
GAT	93.83±0.27	93.39±0.30	58.94±1.35	64.86±0.43	40.74±2.58	65.26±0.80	OOM	OOM	OOM	OOM	OOM
GraphSAGE	92.53±0.53	93.06±0.47	58.36±0.70	61.92±0.72	44.17±1.14	61.71±0.53	46.46±0.60	40.32±0.91	23.05±0.80	52.12±0.09	54.09±0.08
RGCN	91.52±0.50	92.07±0.50	58.85±0.26	62.05±0.15	46.78±0.77	58.33±1.57	48.11±0.48	48.93±0.26	31.51±0.20	85.91±0.10	84.92±0.23
HAN	91.67±0.49	92.05±0.62	57.74±0.96	64.63±0.58	21.31±1.68	54.77±1.40	OOM	OOM	OOM	OOM	OOM
GTN	93.52±0.55	93.97±0.54	60.47±0.98	65.14±0.45	OOM	OOM	OOM	OOM	OOM	OOM	OOM
RSHN	93.34±0.58	93.81±0.55	59.85±3.21	64.22±1.03	OOM	OOM	OOM	OOM	OOM	OOM	OOM
HetGNN	91.76±0.43	92.33±0.41	48.25±0.67	51.16±0.65	OOM	OOM	OOM	OOM	OOM	OOM	OOM
MAGNN	93.28±0.51	93.76±0.45	56.49±3.20	64.67±1.67	OOM	OOM	OOM	OOM	OOM	OOM	OOM
HetSANN	78.55±2.42	80.56±1.50	49.47±1.21	57.68±0.44	OOM	OOM	OOM	OOM	OOM	OOM	OOM
HGT	93.01±0.23	93.49±0.25	63.00±1.19	67.20±0.57	29.28±2.52	60.51±1.16	46.78±0.42	47.31±0.32	29.82±0.33	84.13±0.37	82.16±0.89
Simple-HGN	94.01±0.24	94.46±0.22	63.53±1.36	67.36±0.57	47.72±1.48	66.29±0.45	OOM	OOM	OOM	OOM	OOM
HINormer	94.57±0.23	94.94±0.21	64.65±0.53	67.83±0.34	49.94±2.12	66.47±0.47	OOM	OOM	OOM	OOM	OOM
NARS	94.18±0.47	94.61±0.42	63.51±0.68	66.18±0.70	49.98±1.77	63.26±1.26	50.66±0.22	52.28±0.17	34.38±0.21	86.06±0.10	85.15±0.14
SeHGNN	94.86±0.14	95.24±0.13	66.63±0.34	68.21±0.32	50.71±0.44	63.41±0.47	51.45±0.29	46.75±0.27	29.11±0.25	86.01±0.21	84.95±0.20
RpHGNN	95.23±0.31	95.55±0.29	67.53±0.79	69.77±0.66	54.02±0.88	66.55±0.67	52.07±0.17	53.31±0.40	35.46±0.47	87.80±0.06	86.79±0.18
LastResidual-LP (*NARS*)	94.30±0.51	94.70±0.43	63.98±0.33	66.30±0.49	50.18±2.48	63.92±0.84	52.28±0.25	53.03±0.20	35.44±0.17	86.21±0.11	85.30±0.14
LastResidual-LP (*SeHGNN*)	92.51±0.47	93.08±0.37	63.18±0.98	65.43±0.67	44.74±2.35	57.82±3.01	47.59±0.54	42.02±0.13	24.67±0.13	83.98±0.06	82.30±0.10
LastResidual-LP (*RpHGNN*)	95.27±0.32	95.58±0.29	66.67±0.83	70.22±0.54	54.17±0.92	66.63±0.49	45.88±0.25	49.65±0.12	32.17±0.08	87.48±0.11	86.31±0.26
RemoveDiag-LP (*NARS*)	94.48±0.71	94.87±0.61	64.04±0.57	66.31±0.76	50.35±1.40	63.78±0.90	53.99±0.09	53.10±0.35	35.59±0.29	86.85±0.02	86.11±0.10
RemoveDiag-LP (*SeHGNN*)	95.06±0.17	95.42±0.17	66.48±0.57	68.36±0.42	51.87±0.86	65.08±0.66	54.00±0.22	44.21±0.28	27.02±0.22	86.57±0.04	85.13±0.23
RemoveDiag-LP (*RpHGNN*)	–	–	–	–	–	–	–	–	–	–	–
Echoless-LP (*NARS*)	95.18±0.29	95.49±0.28	64.50±0.31	67.11±0.46	50.58±1.61	63.85±1.21	**54.43±0.14**	53.68±0.22	36.16±0.27	86.89±0.07	86.16±0.11
Echoless-LP (*SeHGNN*)	95.31±0.23	95.59±0.22	67.08±0.53	70.12±0.43	52.84±0.51	66.95±0.30	54.17±0.18	47.92±0.33	30.47±0.31	86.81±0.08	85.43±0.16
Echoless-LP (*RpHGNN*)	**95.39±0.26**	**95.70±0.25**	**67.99±0.54**	**70.84±0.39**	**54.28±0.83**	**67.08±0.58**	54.04±0.17	**54.72±0.83**	**37.33±0.80**	**88.11±0.04**	**87.28±0.06**

Table 2: Performance on Small and Large Datasets. Best results are in bold and second-best results are underlined. RemoveDiag-LP (*RpHGNN*) results are marked as "–", since RemoveDiag-LP is incompatible with RpHGNN's complex message passing.

2020; Yang et al. 2023), we use early stopping based on validation performance to save the best model for evaluation.

For parameter settings, we consider them for both backbones and label-based pre-computation methods. For the integrated backbones, we try our best to follow their official parameter settings. The hyperparameters for each backbone remain unchanged when integrated with different label-based pre-computation methods, ensuring fair comparison among them. For label-based pre-computation hyperparameters, the key hyperparameters tuned include the number of hops for label information (K) and the number of partitions (M), with $K \in [1, 8]$ and $M \in [2, 5]$. Besides, for the learnable encoders (e.g., hierarchical MLP), we only consider the input dropout rate d_{in} and hidden dropout rate related to the collected label information (not the feature information), both searched within $[0.0, 0.9]$. All parameters are selected via performance on the validation set.

Performance Analysis

Based on Table 2, we have the following observations:

- Pre-computation-based GNNs, including NARS, SeHGNN, and RpHGNN, show overall superior performance and efficiency over end-to-end GNNs, showing the advantage of pre-computation-based GNNs.
- By introducing labels as extra input data, the label-based pre-computation method LastResidual-LP cannot consistently improve the backbone's performance, outperforming its NARS backbone but underperforming its SeHGNN and RpHGNN backbones. One reason for the performance drop is that LastResidual-LP only alleviates the training label leakage issue but does not address it completely. Additionally, the severity of training label leakage may vary when using different message passing methods (provided by the backbone).

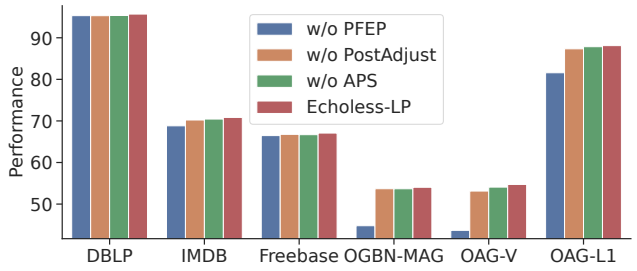
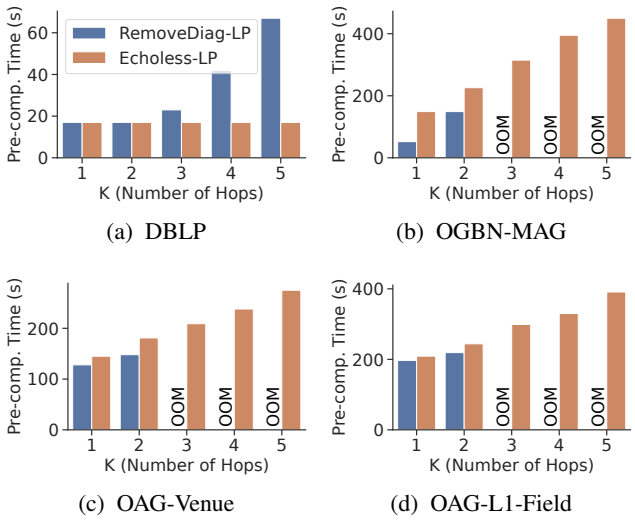


Figure 6: Ablation study. Using F1-Micro for DBLP, IMDB, and Freebase; Accuracy for OGBN-MAG; and NDCG for OAG-Venue (OAG-V) and OAG-L1-Field (OAG-L1).

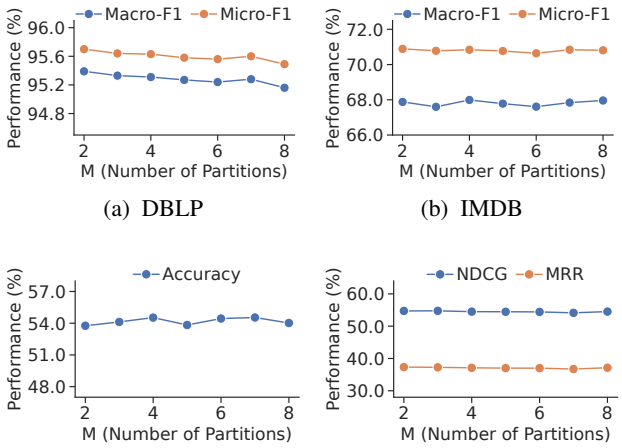
- The label-based pre-computation method RemoveDiag-LP can avoid training label leakage and improves most backbones (NARS and SeHGNN), showing that avoiding training label leakage is necessary for effective learning. However, it has compatibility issues and cannot integrate with the most powerful backbone RpHGNN, as it only supports linear message passing and is thus incompatible with RpHGNN, which involves complex message passing operations like feature normalization.
- Echoless-LP achieves the best performance and most of the second-best performance across datasets. Both Echoless-LP and RemoveDiag-LP can avoid training label leakage. However, RemoveDiag-LP is limited to 2 hops on large graphs due to costly memory overhead, while Echoless-LP can utilize label information within K hops ($K > 2$) without such overhead. Additionally, unlike RemoveDiag-LP, which is limited to linear message passing, Echoless-LP is compatible with the powerful backbone RpHGNN with complex message passing



(a) DBLP (b) OGBN-MAG

(c) OAG-Venue (d) OAG-L1-Field

Figure 7: Pre-computation time (s) vs. number of hops K.



(a) DBLP (b) IMDB

(c) OGBN-MAG (d) OAG-Venue

Figure 8: Impact of the number of partitions M.

K	OGBN-MAG		OAG-Venue	
	RemoveDiag-LP	Echoless-LP	RemoveDiag-LP	Echoless-LP
1	40GB	50GB	60GB	60GB
2	50GB	60GB	60GB	60GB
3	OOM (> 13TB)	70GB	OOM (> 4TB)	60GB
4	OOM (> 13TB)	80GB	OOM (> 4TB)	60GB
5	OOM (> 13TB)	90GB	OOM (> 4TB)	70GB

Table 3: Memory usage vs. number of hops K. Memory for non-OOM cases is rounded up to the nearest 10GB, and OOM cases show estimated memory requirements.

operations. As a result, Echoless-LP achieves superior overall performance, demonstrating its effectiveness.

Detailed Analysis

Ablation Study To verify the effectiveness of main components, we design several variants of our Echoless-LP: (1) *Echoless-LP w/o PFEP* removes PFEP. Instead, it directly uses feature-based pre-computation methods to collect neighbor label information by replacing the input node features with label vectors. (2) *Echoless-LP w/o APS* uses uniform random partitioning scheme, instead of our APS. (3) *Echoless-LP w/o PostAdjust* skips the PostAdjust operation after message passing. Results in Figure 6 compare their performance (using RpHGNN as the backbone).

Echoless-LP outperforms Echoless-LP w/o PFEP, benefiting from PFEP's ability to avoid training label leakage caused by echo. Our method also outperforms Echoless-LP w/o APS, showing that APS effectively reduces information loss caused by partitioning. Besides, Echoless-LP surpasses Echoless-LP w/o PostAdjust, confirming that alleviating distribution shifts between partitions improves performance.

Memory-Efficiency of Pre-computation We vary the number of hops K and report pre-computation time in Figure 7 using NARS as the backbone. While Echoless-LP costs moderately more time than RemoveDiag-LP when $K \leq 2$ due to separate message passing on M partitions, it demonstrates clear efficiency advantages when $K > 3$. RemoveDiag-LP becomes much slower on the small dataset (DBLP) and runs out of memory (OOM) on large datasets (OGBN-MAG, OAG-Venue, and OAG-L1-Field), whereas Echoless-LP maintains approximately linear time scaling. This efficiency gap is explained by memory usage shown in Table 3: when $K > 2$, RemoveDiag-LP requires explicit computation of the multi-hop propagation matrix for diagonal removal, causing terabyte-level memory usage, while Echoless-LP avoids such costly overhead.

Impact of Number of Partitions Although our APS eliminates information loss for unlabeled nodes (including test nodes), training nodes still experience information loss affected by the number of partitions M, where labels of any neighbor have a $1/M$ probability of being masked. Intuitively, a larger M should reduce this information loss and improve performance. However, results in Figure 8 show that a small M (e.g., $M \leq 4$) is typically sufficient, with $M = 2$ achieving optimal performance in most cases. One potential explanation is that, for training nodes, the moderate information loss introduced by smaller M values may act as a form of beneficial regularization, improving model robustness rather than hindering performance.

Conclusion

We propose Echoless Label-based Pre-computation (Echoless-LP) for memory-efficient heterogeneous graph learning. It eliminates training label leakage caused by echo with Partition-Focused Echoless Propagation (PFEP). PFEP is compatible with any message passing method, and message passing proceeds as usual without modification, avoiding costly memory overhead and ensuring memory efficiency. We introduce Asymmetric Partitioning Scheme (APS) and PostAdjust to address information loss from partitioning and distributional shifts across partitions, respectively. Experiments show that Echoless-LP achieves superior performance and maintains memory efficiency.

Acknowledgments

This research is supported by the National Research Foundation, Singapore and Infocomm Media Development Authority under its Trust Tech Funding Initiative, and the Ministry of Education, Singapore, under the Academic Research Fund Tier 2 (FY2025) (Grant MOE-T2EP20124-0009). Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not reflect the views of National Research Foundation, Singapore and Infocomm Media Development Authority.

References

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

Fu, X.; Zhang, J.; Meng, Z.; and King, I. 2020. MAGNN: Metapath Aggregated Graph Neural Network for Heterogeneous Graph Embedding. In *WWW '20: The Web Conference 2020, Taipei, Taiwan, April 20-24, 2020*, 2331–2341. ACM / IW3C2.

Hamilton, W. L.; Ying, Z.; and Leskovec, J. 2017. Inductive representation learning on large graphs. *Advances in Neural Information Processing Systems*, 30.

Hong, H.; Guo, H.; Lin, Y.; Yang, X.; Li, Z.; and Ye, J. 2020. An attention-based graph neural network for heterogeneous structural learning. In *Proceedings of the AAAI conference on artificial intelligence*, volume 34, 4132–4139.

Hu, J.; Hooi, B.; and He, B. 2024. Efficient Heterogeneous Graph Learning via Random Projection. *IEEE Transactions on Knowledge and Data Engineering*, 36(12): 8093–8107.

Hu, W.; Fey, M.; Zitnik, M.; Dong, Y.; Ren, H.; Liu, B.; Catasta, M.; and Leskovec, J. 2020a. Open graph benchmark: Datasets for machine learning on graphs. *Advances in Neural Information Processing Systems*, 33: 22118–22133.

Hu, Z.; Dong, Y.; Wang, K.; Chang, K.; and Sun, Y. 2020b. GPT-GNN: Generative Pre-Training of Graph Neural Networks. In *KDD '20: The 26th ACM SIGKDD Conference on Knowledge Discovery and Data Mining, Virtual Event, CA, USA, August 23-27, 2020*, 1857–1867. ACM.

Hu, Z.; Dong, Y.; Wang, K.; and Sun, Y. 2020c. Heterogeneous Graph Transformer. In *WWW '20: The Web Conference 2020, Taipei, Taiwan, April 20-24, 2020*, 2704–2710. ACM / IW3C2.

Kipf, T. N.; and Welling, M. 2016. Semi-Supervised Classification with Graph Convolutional Networks. In *International Conference on Learning Representations*.

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 & data mining*, 1150–1160.

Mao, Q.; Liu, Z.; Liu, C.; and Sun, J. 2023a. HINormer: Representation Learning On Heterogeneous Information Networks with Graph Transformer. In *Proceedings of the ACM Web Conference 2023, WWW 2023, Austin, TX, USA, 30 April 2023 - 4 May 2023*, 599–610. ACM.

Mao, Q.; Liu, Z.; Liu, C.; and Sun, J. 2023b. Hinormer: Representation learning on heterogeneous information networks with graph transformer. In *Proceedings of the ACM web conference 2023*, 599–610.

Schlichtkrull, M. S.; Kipf, T. N.; Bloem, P.; van den Berg, R.; Titov, I.; and Welling, M. 2018. Modeling relational data with graph convolutional networks. In *The Semantic Web: 15th International Conference, ESWC 2018, Heraklion, Crete, Greece, June 3–7, 2018, Proceedings 15*, 593–607. Springer.

Sinha, A.; Shen, Z.; Song, Y.; Ma, H.; Eide, D.; Hsu, B.-J.; and Wang, K. 2015. An overview of microsoft academic service (mas) and applications. In *Proceedings of the 24th international conference on world wide web*, 243–246.

Tang, J.; Zhang, J.; Yao, L.; Li, J.; Zhang, L.; and Su, Z. 2008. Arnetminer: extraction and mining of academic social networks. In *Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*, 990–998.

Velickovic, P.; Cucurull, G.; Casanova, A.; Romero, A.; Liò, P.; and Bengio, Y. 2018. Graph Attention Networks. In *International Conference on Learning Representations*.

Wang, X.; Ji, H.; Shi, C.; Wang, B.; Ye, Y.; Cui, P.; and Yu, P. S. 2019. Heterogeneous Graph Attention Network. In *The World Wide Web Conference, WWW 2019, San Francisco, CA, USA, May 13-17, 2019*, 2022–2032. ACM.

Yang, X.; Yan, M.; Pan, S.; Ye, X.; and Fan, D. 2023. Simple and efficient heterogeneous graph neural network. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 37, 10816–10824.

Yu, L.; Shen, J.; Li, J.; and Lerer, A. 2020. Scalable graph neural networks for heterogeneous graphs. *arXiv preprint arXiv:2011.09679*.

Yun, S.; Jeong, M.; Kim, R.; Kang, J.; and Kim, H. J. 2019. Graph transformer networks. *Advances in Neural Information Processing Systems*, 32.

Zhang, C.; Song, D.; Huang, C.; Swami, A.; and Chawla, N. V. 2019a. Heterogeneous graph neural network. In *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining*, 793–803.

Zhang, F.; Liu, X.; Tang, J.; Dong, Y.; Yao, P.; Zhang, J.; Gu, X.; Wang, Y.; Kharlamov, E.; Shao, B.; Li, R.; and Wang, K. 2022a. OAG: Linking Entities across Large-scale Heterogeneous Knowledge Graphs. *IEEE Transactions on Knowledge and Data Engineering*, 1–14.

Zhang, F.; Liu, X.; Tang, J.; Dong, Y.; Yao, P.; Zhang, J.; Gu, X.; Wang, Y.; Shao, B.; Li, R.; et al. 2019b. OAG: Toward linking large-scale heterogeneous entity graphs. In *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining*, 2585–2595.

Zhang, W.; Yin, Z.; Sheng, Z.; Li, Y.; Ouyang, W.; Li, X.; Tao, Y.; Yang, Z.; and Cui, B. 2022b. Graph attention multi-layer perceptron. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, 4560–4570.

Zhou, X.; and Shen, Z. 2023. A Tale of Two Graphs: Freezing and Denoising Graph Structures for Multimodal Recommendation. In *Proceedings of the 31st ACM International Conference on Multimedia, MM 2023, Ottawa, ON, Canada, 29 October 2023- 3 November 2023*, 935–943. ACM.

Zhu, S.; Zhou, C.; Pan, S.; Zhu, X.; and Wang, B. 2019. Relation structure-aware heterogeneous graph neural network. In *2019 IEEE international conference on data mining (ICDM)*, 1534–1539. IEEE.