Heta: Distributed Training of Heterogeneous Graph Neural Networks

Yuchen Zhong
The University of Hong Kong, China
yczhong@cs.hku.hk

Junwei Su*
The University of Hong Kong, China
jwsu@cs.hku.hk

Chuan Wu*
The University of Hong Kong, China
cwu@cs.hku.hk

Minjie Wang
AWS Shanghai AI Lab, China
minjiw@amazon.com

ABSTRACT

Heterogeneous Graphs (HetGs) that capture relationships among different types of nodes are ubiquitous in real-world applications such as academic networks and e-commerce. Although Heterogeneous Graph Neural Networks (HGNNs) have demonstrated superior performance in learning from these complex structures, distributed training of HGNNs on large-scale graphs with billions of edges faces substantial communication overhead. This challenge is exacerbated by heterogeneous characteristics such as varying feature dimensions across node types and featureless nodes requiring learnable parameters. Existing systems and communication reduction techniques designed for homogeneous graphs become suboptimal or even inapplicable for HetGs and HGNNs by overlooking both these heterogeneous characteristics and the inherent computational structure of HGNNs. We present *Heta*, a framework designed to address the communication bottleneck in distributed HGNN training. *Heta* leverages the key insight that HGNN aggregation is order-invariant and decomposable into relation-specific computations. Built on this insight, we introduce three key innovations: (1) a Relation-Aggregation-First (RAF) paradigm that conducts relation-specific aggregations within partitions and exchanges only partial aggregations across machines, proven to reduce communication complexity; (2) a meta-partitioning strategy that divides a HetG based on its graph schema and HGNN computation dependency while minimizing cross-partition communication and maintaining computation and storage balance; and (3) a heterogeneity-aware GPU cache system that accounts for varying miss-penalty ratios across node types. Through extensive evaluation of billion-edge heterogeneous graphs, we demonstrate that *Heta* achieves up to 5.3× and 4.4× speedup over state-of-the-art systems DGL and GraphLearn while maintaining model accuracy.

PVLDB Reference Format:
Yuchen Zhong, Junwei Su, Chuan Wu, and Minjie Wang. Heta: Distributed Training of Heterogeneous Graph Neural Networks. PVLDB, 18(9): 2790 - 2803, 2025.
doi:10.14778/3746405.3746408

PVLDB Artifact Availability:
The source code, data, and/or other artifacts have been made available at https://github.com/jasperzhong/Heta.

*Corresponding author

Proceedings of the VLDB Endowment, Vol. 18, No. 9 ISSN 2150-8097.
doi:10.14778/3746405.3746408

1 INTRODUCTION

Heterogeneous Graphs (HetGs) capture diverse semantic relationships among different types of nodes and edges. Real-world heterogeneous graphs often operate at massive scale with billions of interactions. For instance, Amazon links billions of users, products, and sellers, while YouTube manages billions of interactions among users, videos, and channels at a comparable scale. Heterogeneous Graph Neural Networks (HGNNs) [3, 4, 11, 23, 46, 57, 59, 61, 63] are tailored to learn from such graphs by encoding rich semantic and structural information, consistently surpassing homogeneous GNNs in tasks like recommendation systems [67], social network analysis [13], and cybersecurity [20].

However, training HGNNs on large-scale HetGs with millions of nodes and billions of edges remains challenging [21]. Similar to distributed GNN training on homogeneous graphs, these graphs must be partitioned across multiple machines for processing due to their size, necessitating frequent communication between machines to gather neighbors and their node features along edges that cross partition boundaries [62]. This inter-machine communication becomes a crucial bottleneck in distributed HGNN training. Unlike homogeneous graphs, HetGs exhibit heterogeneous characteristics that complicate this process. Typically, homogeneous graphs have nodes sharing a common feature space with uniform feature dimensions [66]. In contrast, HetGs display significant variations in feature dimensions across different node types. Feature dimensions of node types can vary dramatically (e.g., 7 to 789 in Donor dataset [26]). Moreover, some node types lack node features entirely. In the MAG240M dataset [22], only the paper nodes have features, while other types of nodes (approximately half of all nodes) do not. A prevalent solution in HGNN learning is to assign trainable parameters (called *learnable features*) to such featureless nodes that are updated during training [19, 27, 60, 65]. This introduces additional complexity, as learnable features and their associated optimizer states require frequent reads and writes between GPU and DRAM, while read-only node features only require reading from DRAM to GPU. Both the varying feature dimensions across node types and the coexistence of static and learnable features further exacerbate the communication bottleneck.

These fundamental challenges - communication bottleneck and heterogeneous characteristics - are not well addressed by existing distributed GNN systems. Most systems are tailored for homogeneous graphs, primarily focusing on communication reduction. Transferring their solutions to heterogeneous graphs is either inapplicable or suboptimal. First, techniques like P^3[12] partition node features along the feature dimension, assuming uniform feature dimensions across all nodes – a method inapplicable to HetGs with varying feature dimensions. Next, prior works propose caching

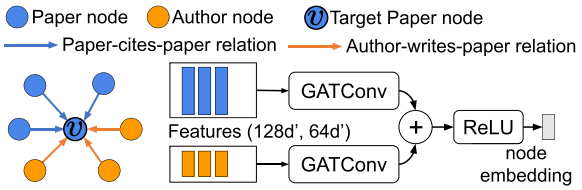


Figure 1: Illustration of relation-specific aggregation in R-GAT [3]. In an academic network, a target paper node aggregates features separately along two different relations.

features of frequently accessed nodes on GPUs to reduce fetching costs [25, 28, 35, 48, 58]. However, these caching methods only consider the "hotness" of nodes and overlook variations in the cache "miss penalty" across node types, such as different feature dimensions and whether features are learnable. Feature dimensions affect data retrieval cost, while cache misses for learnable features incur higher penalties as both reads and writes are needed. Consequently, these caching methods are suboptimal for heterogeneous graphs.

Moreover, current distributed GNN training frameworks such as DGL [66] and GraphLearn [49], which technically support distributed training of HGNNs, also achieve suboptimal performance by overlooking the unique characteristics of HGNNs. These systems train HGNNs in a manner similar to homogeneous GNNs, adhering to the *vanilla execution model* (detailed in §2.2), which employs data parallelism and edge-cut partitioning. DGL converts a HetG topology (which includes multiple adjacency matrices for different relations) into a single adjacency matrix and then applies edge-cut-minimizing algorithms like METIS [30] for graph partitioning. GraphLearn uses random edge-cut partitioning for nodes of each node type. The rest of the HGNN training process remains the same as homogeneous GNNs. This homogeneous-like treatment in partitioning and computation leads to inefficient data placement and processing, where nodes that should be processed together based on their types and relationships often end up scattered across partitions. As a result, these systems still suffer from significant communication overhead due to a large amount of remote data fetching and can be further optimized for HGNNs (§2.3).

To efficiently mitigate the communication bottleneck in distributed HGNN training on heterogeneous graphs, we propose *Heta*, a novel framework for efficient distributed HGNN training. The key insight behind *Heta* is that HGNN aggregation is order-invariant (i.e., aggregation order does not affect results) and can be decomposed into relation-specific aggregations, where each aggregation processes node features along one type of relation in the graph. As illustrated in Figure 1, in an academic network, a paper node aggregates features separately from its author nodes (through "Author-writes-paper" relation) and citation nodes (through "Paper-cites-paper" relation). These features are then processed through their respective homogeneous GNN layers (e.g., GATConv), and their outputs are combined and activated by ReLU to generate the node embedding of the target paper node. This decomposition applies to most HGNNs (§2.1) and has not been fully leveraged by current distributed GNN systems.

We begin by rethinking the fundamental HGNN computation model inspired by this insight, developing the *Relation-Aggregation-First (RAF) paradigm* to minimize communication overhead (§4). RAF employs model parallelism to conduct relation-specific aggregations within each partition, exchanging only partial aggregations and their gradients across partitions to compute final node embeddings. This approach substantially reduces communication overhead by avoiding direct feature movement and leveraging the smaller hidden dimensions of HGNN models. We prove that the communication complexity under RAF depends on the maximum number of boundary nodes (i.e., nodes with edges crossing partitions), which is lower than the number of cross-partition edges (communication complexity of the vanilla execution model used by existing systems) (Prop.2 &3). Built on this analysis, we develop a novel graph partitioning strategy called *meta-partitioning* (§5) that specifically minimizes the maximal number of boundary nodes. Drawing on the concept of a metagraph (a high-level abstraction of a HetG, detailed in §2.1), we devise a novel graph coarsening process tailored for efficiently partitioning HetG under RAF. Our graph coarsening process is based on the computation dependency of HGNN and assigns strongly connected relations to the same partition, achieving efficient partitioning and balanced storage and computation loads. To further optimize data movement, we design a *heterogeneity-aware GPU feature cache* (§6) that considers varying cache miss-penalty ratios across node types, influenced by both feature dimensions and whether features are learnable. Our primary contributions are summarized as follows:

- We introduce the Relation-Aggregation-First (RAF) paradigm, a novel computation method restructuring aggregation to prioritize relation-specific computations within each partition. We prove RAF achieves lower communication complexity by depending on boundary nodes rather than cross-partition edges.
- We present meta-partitioning, a graph partitioning strategy tailored for HetGs inspired by our communication complexity analysis. This approach minimizes the maximum number of boundary nodes across partitions, ensuring balanced computation and storage and maintaining constant communication complexity regardless of sampling fanout or the number of HGNN layers.
- We design a heterogeneity-aware GPU cache that accounts for the diverse characteristics of different node types in HetGs. Our cache system distributes cache resources based on both node hotness and their respective miss penalties, and employs a non-redundant cache design for data consistency.
- We implement these methods in *Heta* with DGL and PyTorch [41] and demonstrate its effectiveness via extensive evaluation on large-scale heterogeneous graphs with billions of edges. Results show that *Heta* achieves up to 5.3× and 4.4× speedup over state-of-the-art systems DGL and GraphLearn, respectively, when training representative HGNN models while maintaining model accuracy. Moreover, *Heta*'s meta-partitioning demonstrates superior efficiency in terms of time and memory footprint compared to state-of-the-art graph partitioning methods such as METIS. *Heta* is open sourced at https://github.com/jasperzhong/Heta.

2 BACKGROUND AND MOTIVATION

2.1 Heterogeneous Graph Neural Networks

Heterogeneous Graphs (HetGs). A HetG is denoted by $G = (V, E, A, R)$, where each node $v \in V$ and each edge $e \in E$ are associated with type mapping functions $\tau(v) : V \rightarrow A$ and $\phi(e) : E \rightarrow R$,

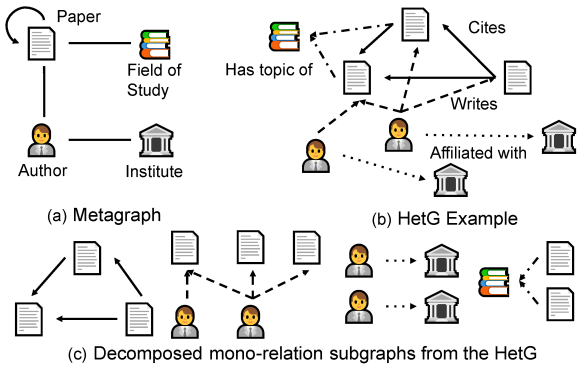


Figure 2: Metagraph and decomposed mono-relation subgraphs of the ogbn-mag dataset [22].

respectively. The set A represents the node types, while R represents the edge types that connect the nodes. A relation r on edge $e = (u, v)$ is the triple $(\tau(u), \phi(e), \tau(v))$, with $\mathcal{R}$ as the set of all such relations. The reverse relation r^{-1} for r is $(\tau(v), \bar{\phi}(e), \tau(u))$, where $\bar{\phi}(e)$ denotes the inverse edge type. A mono-relation subgraph can be defined for each relation, containing nodes of types $\tau(u)$ and $\tau(v)$ and edges of type $\phi(e)$. The HetG can be decomposed into a collection of mono-relation subgraphs by relations. The structure of a HetG can be further described by a metagraph $M = (A, R)$, and metapaths represent semantic relations, which are sequences characterized by the types of nodes and edges on the paths connecting the nodes. Some nodes have dense feature vectors; each node type may correspond to a feature vector of a different dimension; some node types lack features entirely. One node type, called the target node type, is of particular interest and is associated with labels; they are also referred to as training nodes.

Figure 2 shows a concrete HetG and the metagraph from the ogbn-mag dataset [22], consisting of four node types and four relations (plus three reverse relations). Figure 2 (c) shows the mono-relation subgraphs decomposed from the HetG. Only the node type "paper" is associated with node features, while other node types do not. The target node type is "paper" associated with venue labels, and the task of this dataset is to predict the venue of a given paper.

Heterogeneous Graph Neural Networks (HGNNs). Given a HetG and node v, the HGNN computation to generate node embedding $\mathbf{h}_v^{(l)}$ at its layer l is:

$$\begin{aligned} \mathbf{h}_{v,r}^{(l)} &= \mathrm{AGG}_r^{(l)}\left(\left\{\mathbf{h}_u^{(l-1)}, u \in N_r(v)\right\}\right) \\ \mathbf{h}_v^{(l)} &= \mathrm{AGG}_{\mathrm{all}}^{(l)}\left(\left\{\mathbf{h}_{v,r}^{(l)}, r \in \mathcal{R}\right\}\right) \end{aligned} \tag{1}$$

where $\mathrm{AGG}_r^{(l)}$ is a relation-specific aggregation function for relation r corresponding to a mono-relation subgraph in the HetG (typically a GNN like GCN [33]), $\mathbf{h}_{v,r}^{(l)}$ is the partial embedding of relation r using neighbors $N_r(v)$ (i.e., the neighbors of node v under relation r), and $\mathrm{AGG}_{\mathrm{all}}^{(l)}$ is a cross-relation aggregation function (typically a reduce function like summation). Both aggregation functions are typically order-invariant, meaning the order of aggregating neighbors or relations does not affect the final result. $\mathbf{h}_v^0$ is initialized using the feature vector of node v or its learnable feature vector $\mathbf{w}_v$ (if node features are not available) [66].

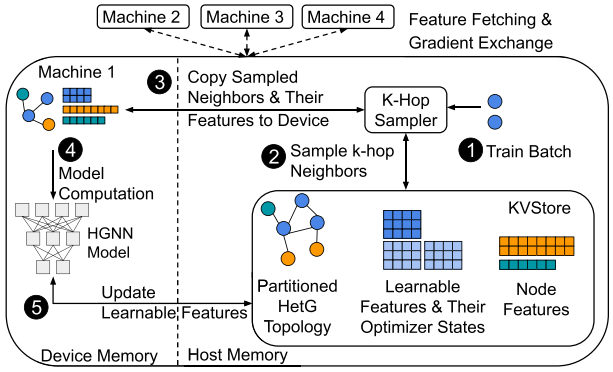


Figure 3: Vanilla execution model of existing distributed GNN training systems that support HGNN training.

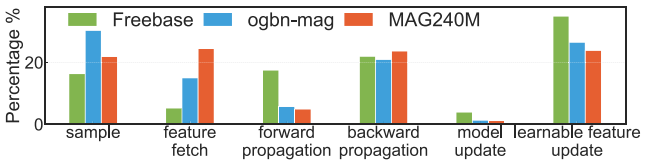


Figure 4: Percentage of the epoch time spent on each stage: training R-GCN on three datasets with DGL.

Eq. (1) suggests that the aggregation of a HGNN can be decomposed into individual relation-specific aggregations for each relation $r \in \mathcal{R}$, followed by a cross-relation aggregation. Two representative HGNNs, R-GCN [46] and R-GAT [3], apply GCN [33] and GAT [50], respectively, for relation-specific aggregation, and then sum the embeddings. These models have a set of weight matrices for each relation. HGT uses weight matrices for each node and edge type instead of each relation [23], allowing it to perform relation-specific aggregation using the parameters associated with corresponding node and edge types.

2.2 Distributed Training of HGNNs

Existing systems like DGL [66] and GraphLearn [49] use edge-cut partitioning and data-parallel training, termed *vanilla execution model*. DGL supports learnable features for featureless nodes, while GraphLearn does not. Optimizers (e.g., Adam [32]) keep states (moments and variances) equal in size to learnable features. All node-related data are partitioned according to their associated nodes.

In this model (Figure 3), each worker processes a mini-batch of training nodes (❶) and samples their k-hop neighbors (❷) [18]. Features of sampled nodes are fetched from the key-value store (KVStore) in host memory, incurring network communication for remote data. These features and topology are copied to the GPU (❸) for computation. After model computation (❹) and gradient synchronization, optimizer states are fetched from the local KVStore to the GPU, and both states and learnable features are updated and written back to the local KVStore (❺).

2.3 Opportunities & Challenges

Challenge 1: Communication Bottleneck. The vanilla execution model (§2.2) shows each training iteration requires constructing

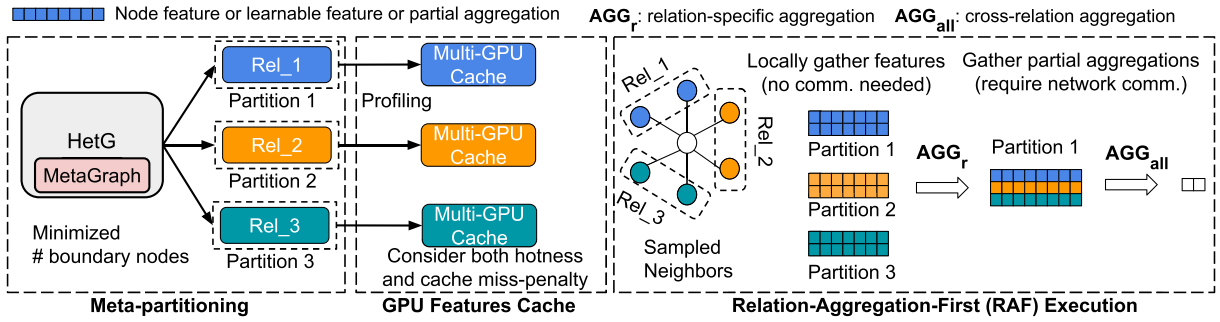


Figure 5: *Heta*'s design overview.

k-hop neighborhoods for input target nodes and collecting graph structures and feature vectors from different machines. HGNNs exacerbate this by needing extra communication to update learnable features for featureless nodes. Figure 4 shows that when training R-GCN on three large-scale datasets with DGL and METIS [30] on two Amazon EC2 g4dn.metal instances, communication-intensive operations (sampling, feature fetching, and learnable feature updates) consume over 60% of the epoch time. Existing communication reduction methods for homogeneous GNNs are insufficient: P^3 [12] assumes uniform feature dimensions, a condition absent in HetGs, while other approaches [10, 52–54] alter model computational equivalence.

Opportunity 1: Relation-Aware Model Parallelism. Eq. (1) indicates that HGNN computation is splittable by relation type r, allowing workers to aggregate embeddings within each relation before communicating. This amounts to model parallelism for HGNN training. We leverage this insight to propose the Relation-Aggregation-First (RAF) paradigm, which reduces communication overhead while preserving model mathematical equivalence (§4).

Challenge 2: Lack of Effective Partitioning Strategy for HetGs. The most common graph partitioning scheme in distributed GNN systems is edge-cut partitioning, especially METIS [30, 65]. These and other classical algorithms [2, 5, 14] target homogeneous graphs, treating all neighbors equally and ignoring that HGNNs process different types of neighbors separately based on their relations. For example, in an academic network, a paper node aggregates features from its authors and citing papers separately, as they represent distinct relations. This type-blind partitioning causes inefficient data placement where nodes that should be processed together often end up in different partitions. Thus, a largely unexplored frontier remains in graph partitioning for HetGs.

Opportunity 2: Relation-Aware Partitioning. By considering relations and HGNN computation dependency, relation-aware partitioning can better place connected node types and their links within the same partition. This requires new partitioning objectives and methods. We propose meta-partitioning, using the metagraph and HGNN computation dependency for HetG partitioning (§5).

Challenge 3: Feature Heterogeneity in GPU Caching. Prior works [35, 48, 58] suggest caching frequently accessed node features on GPUs but overlook the heterogeneity of features in HetGs. Different node types have varying feature dimensions that affect retrieval costs, and some have learnable features requiring frequent host-device communication, rendering existing methods suboptimal for HetGs.

Opportunity 3: Heterogeneity-Aware GPU Cache. Cache misses for different node types incur varying time penalties based on their feature dimensions and types (static vs. learnable). This variance highlights the need for careful cache resource allocation in HetGs. We propose a caching strategy that considers these heterogeneous traits to optimize efficiency (§6).

3 SYSTEM OVERVIEW

Heta is a distributed framework for efficiently training HGNN on HetGs. Figure 5 presents an overview of the design. *Heta* uses meta-partitioning (§5), a relation-aware scheme that decomposes the heterogeneous graph into balanced partitions via its metagraph and HGNN computation dependency. Each partition contains one or more complete mono-relation subgraphs (§2.1). Before training, *Heta* profiles node access frequencies and cache miss-penalty ratios for different node types, accordingly calculates cache sizes for each type, and initializes the GPU feature cache using these frequencies (§6). *Heta* uses mini-batch sampling-based, iterative GNN training and introduces the novel Relation-Aggregation-First (RAF) paradigm. This paradigm aggregates partial results to generate node embeddings across partitions without moving features (§4). After each training iteration's backpropagation, model parameters and learnable features are updated.

4 RAF HGNN COMPUTATION

As §2.1 shows, HGNN aggregation is inherently decomposable by relations, a property current distributed systems fail to exploit. This observation, and that HGNN communication overhead stems mainly from cross-partition feature collection, motivates a fundamental redesign of the distributed execution model. Rather than treating HGNNs as homogeneous GNNs, we propose aligning the system design with HGNN's computational structure by prioritizing relation-specific processing. Moreover, we prove our proposed method achieves lower communication complexity than the vanilla execution model of existing systems.

RAF Paradigm. We present the *Relation-Aggregation-First (RAF)* paradigm in Algorithm 1, which fundamentally reduces communication by exploiting HGNN's relation-specific computation structure. RAF operates on partitioned HetGs, with each partition holding one or more complete mono-relation subgraphs. In RAF, each worker

Algorithm 1: RAF Execution Paradigm

Input: Partitioned heterogeneous graph G based on relations $\{G_1, G_2, \ldots, G_p\}$; set of target nodes V_{target}; the k-layer HGNN; designated worker w_d

Output: Updated model parameters θ of the HGNN

1 **foreach** *batch* $B \subseteq V_{target}$ **do**
2 $S_B^L \leftarrow$ Sample k-hop neighbors for nodes in B
3 **for** $l = 1$ **to** k **do**
  `/* Relation-specific aggregation */`
4 **for** *each worker* i **do in parallel**
5 $\mathbf{h}_i^{(l)} \leftarrow$ Aggregate neighbor features/embeddings of S_B^l for the i-th partition's relations
6 Send partial aggregations $\mathbf{h}_i^{(l)}$ to worker w_d
7 **on worker** w_d **do**
  `/* Cross-relation aggregation */`
8 $\mathbf{h}^{(l)} \leftarrow \text{AGG}_{\text{all}}^{(l)}(\{\mathbf{h}_i^{(l)}\})$
9 **if** $l = k$ **then**
10 Compute loss and backpropagate
11 Send gradients $\nabla\mathbf{h}_i^{(l)}$ to workers
12 **for** *each worker* i **do in parallel**
13 **if** $l = k$ **then**
14 Backpropagate with $\nabla\mathbf{h}_i^{(l)}$ and update local model parameters θ_i

maintains an input HetG partition and holds only the model parameters for the relations in its partition. This enables local relation-specific aggregations (partial aggregations) without cross-machine feature fetching. A randomly assigned designated worker handles cross-relation aggregation. Specifically, each worker aggregates features or previous-layer embeddings from the sampled neighborhood of the target nodes for its local relations (line 5, Algorithm 1) without network communication. The designated worker then collects all partial aggregations from other workers (line 6, via network transmissions), combines embeddings from different relations (line 8), and performs loss computation and backpropagation (line 10). Gradients of partial aggregations are sent back to the corresponding workers (line 11). These workers then compute gradients and update their local model parameters (line 14). Learnable features in a worker's partition are part of its model parameters and are updated together.

Communication Reduction. RAF significantly reduces communication in two ways. *First*, it dramatically cuts the message count by eliminating inter-machine feature fetching. In the vanilla execution model of existing GNN systems, features of sampled k-hop neighbors must be fetched. In RAF, each partition holds complete mono-relation subgraphs, enabling local aggregation of outer-hop features. Only inner-hop partial aggregations might need communication if they span partitions. *Second*, RAF reduces message sizes by sending partial aggregations instead of high-dimensional features. These partial aggregations match the model's hidden layer dimensions, which are typically much smaller than feature dimensions.

To quantify these benefits, consider training a 2-layer R-GCN model [46] with a hidden dimension 64 and fanouts {25, 20} on the MAG240M dataset [21], where paper nodes have 768-dimensional features and other nodes use 64-dimensional learnable features, all in float16 format. Suppose a batch of 1024 training nodes is sampled and the graph partitioned onto two machines using METIS [30]. In total 289,986 neighbor nodes are sampled in layer 0 (2-hop), with 66,772 stored on remote machines. Fetching features of these remote nodes and their topology with the vanilla execution model requires transmitting 92.3 MB of data. RAF, however, only requires exchanging partial aggregations and their gradients of 14,568 nodes in layer 1 (1-hop) and 1024 nodes in layer 2 (input), totaling 8.0 MB.

Our meta-partitioning approach (§5) further optimizes communication by enabling the combination of aggregation messages from multiple relations within the same partition. This approach is highly effective in multi-hop or metapath-based sampling with hierarchical aggregations. In the previous example, only partial aggregations and their gradients of 1024 nodes in layer 2 need to be exchanged, not those in layer 1, as meta-partitioning limits cross-partition dependencies (i.e., boundary nodes) to the target nodes (i.e., "paper" nodes), reducing the communication volume to just 0.5 MB.

Theoretical Analysis. We now establish the theoretical foundations of RAF's effectiveness via three key propositions. Proofs are available in [70]. First, we prove that RAF maintains mathematical equivalence with the vanilla execution model, ensuring that our optimization preserves model accuracy:

Proposition 1 (Mathematical Equivalence). *Let* $\mathbf{h}_v^{(\text{vanilla})}$ *and* $\mathbf{h}_v^{(\text{RAF})}$ *be the embedding of a target node* v *obtained with the vanilla execution model and the RAF paradigm, respectively. It holds that* $\mathbf{h}_v^{(\text{RAF})} = \mathbf{h}_v^{(\text{vanilla})}$.

Next, we analyze communication complexity for a HetG divided into two partitions, G_1 and G_2. The results can be readily extended to multiple partitions.

Proposition 2 (Communication Complexity). *With RAF, the number of communication messages from the worker with partition* G_1 *to the worker with partition* G_2 *is proportional to the number of boundary nodes in* G_1 *(nodes with neighbors in the other partition), i.e.,* $\Theta(|\text{B}(G_1)|)$. *The communication complexity in the reverse direction is* $\Theta(|\text{B}(G_2)|)$. $\text{B}(G_i)$ *denotes the boundary nodes of partition* G_i.

While the vanilla execution model's communication scales with cross-partition edges $\text{E}(G_1, G_2)$, we show:

Proposition 3 (Communication Reduction).

$$\max\{|\text{B}(G_1)|, |\text{B}(G_2)|\} \leq \text{E}(G_1, G_2).$$

Proposition 3 proves RAF's boundary-node complexity is strictly better than the vanilla model's edge-count complexity. With equivalence guarantees, this shows RAF's communication efficiency without accuracy loss.

5 META-PARTITIONING OF HETG

Our analysis in §4 shows that RAF's communication complexity depends on the maximal number of boundary nodes across partitions. This insight leads to a new graph partitioning objective directly targeting this property. Suppose there are p machines used for HGNN training. The proposed p-way graph partitioning problem is:

$$\begin{aligned} \text{Minimize} \quad & \max\left\{|\text{B}(G_1)|, |\text{B}(G_2)|, \ldots, |\text{B}(G_p)|\right\} \\ \text{subject to} \quad & G \subseteq G_1 \bigcup G_2 \bigcup \ldots \bigcup G_p, \\ & \text{with balanced partitions.} \end{aligned} \tag{2}$$

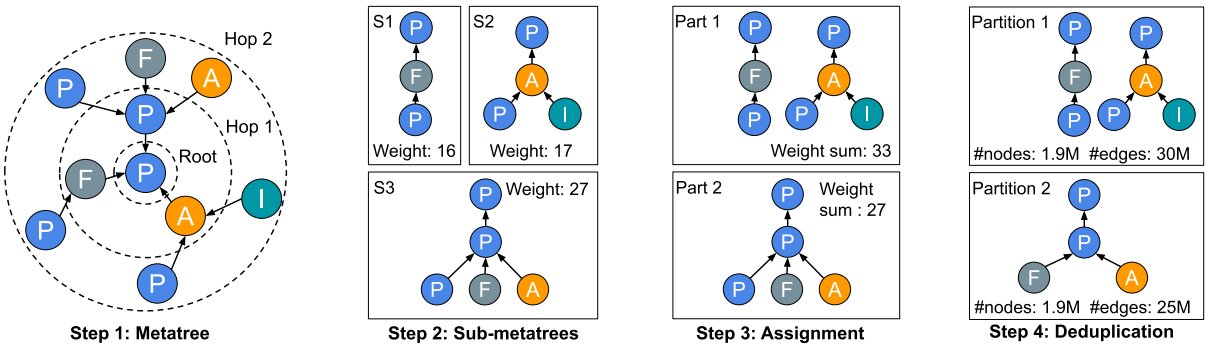


Figure 6: Workflow of meta-partitioning on the ogbn-mag dataset. 'P' - paper, 'A' - author, 'F' - field of study, 'I' - institute.

Proposition 4. *The partitioning problem defined in Eq. (2) is NP-hard.*

This formulation is theoretically sound: Proposition 3 proves that the boundary node count (our system's communication complexity) is always upper-bounded by the cross-edge count (communication complexity in existing systems). Proposition 4 establishes that the optimization problem is NP-hard (the proof is available in [70]), thus needing an efficient heuristic solution. Traditional graph partitioning often uses graph coarsening [30] to reduce complexity by merging nodes to form a smaller graph while preserving essential structural properties. For heterogeneous graphs, the metagraph (§2.1) offers a natural coarsening mechanism by grouping nodes and edges by type.[1] However, as we mention in §2, directly applying this approach faces two key challenges:

First, *HGNN computation dependencies* must be considered. Naively randomly assigning relations to partitions ignores that some relations have stronger computational dependencies. Our key insight is leveraging HGNN's computational structure: for any target node, computations across different relations in the same neighborhood are inherently parallel. Since HGNN training typically focuses on a specific node type as training targets (§2.1), we can represent the computation dependencies as a tree-structured graph—which we call a **metatree**—rooted at this target node type. Figure 6 (Step 1) illustrates this concept using the ogbn-mag dataset [22] with 2-hop sampling. The metatree abstraction effectively captures structural relationships and computation paths, enabling partitioning that aligns with HGNN processing.

Second, *balanced workload distribution* is vital for efficient parallel processing. In mini-batch training with neighborhood sampling, each target node has a fixed computational cost set by the sampling fanout (e.g., {25, 20}) [36]. Thus, balancing target nodes across partitions is key for computational load balance. Additionally, we need to consider storage requirements: we augment our metatree approach with weights where vertex weights represent node counts of each type and link weights capture edge counts of each type, allowing the partitioning process to achieve balanced storage distribution across workers.

This design transforms our original optimization problem defined in Eq. (2) into a weighted metatree partitioning problem that operates on a significantly smaller scale (based on node and edge types rather than individual nodes and edges). Below, we detail this partitioning procedure and illustrate it with a concrete example.

Algorithm. The meta-partitioning procedure (Detailed Algorithm is available in [70]) consists of four steps.

Step 1: Build metatree. We build the metatree via BFS from the target node type as the root vertex, which naturally aligns with multi-hop sampling patterns. For k-hop sampling, we use k-depth BFS, though users can also provide specific metapaths. Figure 6 (Step 1) shows the metatree for ogbn-mag built with 2-depth BFS from the target node type "paper" (P).

Step 2: Split metatree. We create sub-metatrees based on the root's children, each containing the root vertex, a child, and its descendants. This design ensures every partition contains all target nodes, restricting boundary nodes to target nodes only (since other nodes in each partition do not relate to nodes in other partitions). The weight of each sub-metatree is computed from its *leaf vertices (node types)* (i.e., vertices in the outmost hop of the metatree) and links to guide load balancing, as retrieving node features of the outmost hop from host memory to GPU and learnable feature updates incur substantial host-device data copies. When a leaf node type appears in multiple partitions, its features are duplicated to enable local access. This way, HGNN aggregations defined by the sub-metatrees can be performed within each partition. Splitting the metatree from the root vertex also helps balance the computational load since all partitions contain all target nodes, and the computation load for each worker is primarily determined by the number of target nodes in the partition. In Figure 6 (Step 2), the metatree splits into three sub-metatrees S_1, S_2, and S_3, with weights of 16, 17, and 27 million, respectively. Paper node features are duplicated since node type 'Paper' is the leaf node type in S_1, S_2, and S_3.

Step 3: Assignment. Assigning sub-metatrees to p partitions is a p-way number partitioning problem that minimizes the largest sum of weights among partitions to ensure load balance. We use a greedy longest-processing-time-first (LPT) approach [17], assigning "heavier" sub-metatrees first to the least loaded partitions. As shown in Figure 6 (Step 3), S_1 and S_2 are assigned to Partition 1, while S_3 goes to Partition 2.

Step 4: Deduplication. As the metatree may contain duplicated relations from metagraph cycles, we must remove duplicates within

[1] To avoid confusion, we use the terms vertex" and link" for the metagraph, and node" and edge" for the HetG.

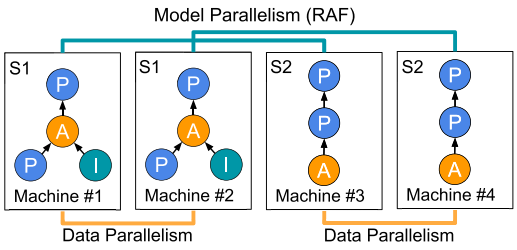


Figure 7: Hybrid parallelism strategy when partitions exceed sub-metatrees. With 4 machines and 2 sub-metatrees (S1, S2), we adopt data parallelism by duplicating each sub-metatree across two machines and splitting target nodes ('Paper' nodes). The partial aggregations of corresponding target nodes are combined using RAF.

each partition to improve storage efficiency. This deduplication preserves computational correctness by removing redundant copies while keeping all unique relation instances. Each final HetG partition thus contains the complete mono-relation subgraph for each unique relation, including all nodes of the two node types connected by that relation and all edges of that edge type. Figure 6 (Step 4), Partition 2's duplicate "Paper-cites-paper" relation is removed, resulting in final partitions: Partition 1 with five unique relations, 1.9 million nodes, and 30 million edges; and Partition 2 with three unique relations, 1.9 million nodes, and 25 million edges.

Complexity. Meta-partitioning operates on the metagraph (few vertices and links), not the full HetG. The BFS algorithm's complexity is at most $O(|R|)$ when traversing all links. With at most $|A|$ sub-metatrees (if all node types connect to the target node type), the time complexity of sorting and deduplication is $O(|A| \log |A|)$. Thus, the overall time complexity of the meta-partitioning algorithm is $O(|A| \log |A|) + O(|R|)$, far lower than graph partitioning algorithms in existing GNN training systems (at least $O(|V|)$, where $|V|$ is the number of nodes in the HetG [2]).

Large Mono-Relation Subgraph. All existing benchmark HetG datasets [19, 21, 22, 26] have mono-relation subgraphs fitting a single machine with 200 GB host memory, making our designs ideal for these cases. If a mono-relation subgraph is too large for one machine, we can use a heuristic to find a balanced vertex-cut [45], referring to the problem Eq. (2). Then, we can still adapt the RAF paradigm (§4) to work across these partitions: each partition performs local aggregations on its portion of the subgraph, exchanges partial aggregation results for the cut vertices, and combines the local and received partial results to complete the relation-specific aggregation. After processing all relations spanning multiple machines, we proceed with the cross-relation aggregation as in the original RAF paradigm.

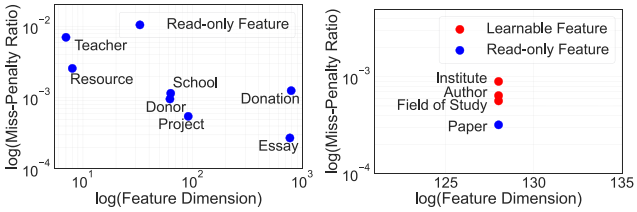
Balance Partitions and Sub-metatrees. When balancing sub-metatrees across available partitions, two scenarios arise: (1) If sub-metatrees substantially outnumber partitions, we assign multiple sub-metatrees to one partition via our weighted LPT algorithm (Step 3). This algorithm accounts for relation size differences by assigning larger relations (with more nodes and edges) first, ensuring storage balance across partitions. Each partition contains all target nodes, ensuring an even distribution of the computational workload in mini-batch training, as noted earlier. (2) If partitions significantly outnumber sub-metatrees (e.g., in graphs with few relations like co-author networks), we use a hybrid parallelism strategy combining model and data parallelism. As shown in Figure 7, with 4 machines and only 2 sub-metatrees, we can assign the first sub-metatree to machines 1 and 2, and the second sub-metatree to machines 3 and 4. Each data-parallel group (e.g., machine 1&2) processes the same relations but different subsets of target nodes ('Paper' nodes). Each model-parallel group (e.g., machine 1&3) processes the same subset of target nodes and aggregates their partial results using RAF. This hybrid approach maintains the optimization objective in Eq. (2) by minimizing boundary nodes while distributing the computational load. Note that the sub-metatree count is determined by the number of relations connected to the target node type, making it necessarily smaller than the total number of relations in the graph.

Extending to Multiple Target Node Types. While our presentation focused on single target node type scenarios, *Heta* can be extended to support multiple target node types. For such cases, we can adapt meta-partitioning in two ways: (1) build multiple metatrees, one rooted at each target node type, and assign them to partitions while maintaining load balance; or (2) add a virtual "super root" in the metagraph connected to all target node types and build a single metatree from it. The RAF paradigm naturally extends to multiple target types by processing relation-specific aggregations for each target type within each partition. Our communication complexity analysis remains valid as the boundary nodes would now include nodes of all target types. This extension preserves *Heta*'s key advantages of reduced communication overhead, enabling the system to handle more complex heterogeneous graph tasks.

6 HETEROGENEITY-AWARE GPU CACHE

GPU feature caching is crucial for efficient HGNN training. However, heterogeneous graphs pose unique challenges that traditional caching strategies fail to address: node types have varying feature dimensions (tens to thousands), and features can be static (read-only) or learnable (needing extra optimizer states and frequent updates). This heterogeneity demands a more sophisticated caching strategy considering both node access patterns and diverse cache miss costs. To address this, we allocate cache space using pre-profiled access frequencies and miss penalties. We propose a static cache allocation where cached features remain in GPU memory without replacement or updates during training. We choose this static design for two reasons: (1) The inherent randomness in neighbor sampling—essential for preventing overfitting [18]—creates non-deterministic feature access patterns, making exact access patterns impossible to pre-determine; (2) Dynamic replacements incur runtime overhead for metadata tracking and eviction, which can negate latency savings [36].

Miss-Penalty-Aware Cache Size Allocation. Our design's key insight is that cache effectiveness depends on both access frequency and the cost of cache misses, which varies across node types. We quantify this with a *miss-penalty ratio*, measuring the time penalty per unit cache size for a non-cached feature. This ratio varies with feature traits (dimension size, read-only vs. learnable) and system factors (e.g., PCIe transaction overhead).



(a) Donor dataset **(b) ogbn-mag dataset**

Figure 8: Miss-penalty ratio for different node types in Donor [26] and ogbn-mag [22] datasets.

To determine node access patterns, we adopt pre-sampling [58], tracking node access frequencies over one training epoch. To profile miss penalties, we measure DRAM-to-GPU transfer times for read-only features, and read/write times for learnable features and their optimizer states. This profiling has a modest overhead (13-57% of one epoch time) but is done once per hardware and sampling configuration, making it negligible over hundreds of training epochs.

Figure 8a shows that smaller feature dimensions often incur larger miss-penalty ratios due to fixed transfer overheads [48]. Figure 8b shows that learnable features have higher miss penalties from write operations and optimizer state management. These observations lead to our hierarchical caching strategy that balances both access patterns and miss penalties. Rather than computing individual node scores, which would introduce substantial overhead, we first allocate cache sizes to node types based on their aggregate statistics, then prioritize frequently accessed nodes within each type. Specifically, we allocate the cache size to each node type a on each partition in proportion to the product of the total visit count $count_a$ of nodes of type a (obtained from pre-sampling) and the miss-penalty ratio o_a: given the total cache size (e.g., 4GB per GPU as in our experiments), the percentage of the cache size allocated to node type a is $\frac{count_a \times o_a}{\sum_{a' \in A'} count_{a'} \times o_{a'}}$, where A' is the set of all node types in the HetG partition.

Cache Consistency. Learnable features introduce cache consistency challenges in multi-GPU settings. While replication seems appealing for read performance, it requires complex synchronization for updates. We instead adopt a non-replicative cache split strategy: each learnable feature resides in one GPU cache or host memory, eliminating consistency issues. Features are distributed across GPUs using the modular hashing scheme: the learnable feature of node nid is cached in GPU rank $nid \mod num\_of\_gpus$, leveraging efficient CUDA peer-to-peer operations [39] for cross-GPU access.

7 IMPLEMENTATION

Heta is implemented on DGL 1.1 and PyTorch 1.3.1 with 2.8K lines of Python code, including minor DGL modifications to enable GPU caching and load meta-partitioned graphs. We leverage PyTorch's distributed package[34] with Gloo [9] and NCCL [40] to implement the RAF paradigm. The meta-partitioning algorithm uses NetworkX [8], while the GPU cache is built using PyTorch tensors for node ID tracking and presence indication. In multi-GPU setups, each machine handles one partition with intra-machine workers performing data-parallel training on evenly split target nodes. *Heta* provides three main APIs: `Partition` for graph partitioning with optional metapath specification, `FetchFeature` for efficient node feature retrieval, and `HGNN` for model definition through relation-specific and cross-relation aggregation functions.

Table 1: Dataset information. # Node T. and # Edge T. represent the number of node types and edge types, respectively.

Attribute	ogbn-mag	Freebase	Donor	IGB-HET	MAG240M
# Nodes	1.9e6	1.2e7	9.7e6	2.6e7	2.4e8
# Node T.	4	8	7	4	3
# Edges	4.2e7	1.3e8	2.5e7	4.9e8	3.4e9
# Edge T.	7	64	14	7	5
# Node T. w/ Feat.	1	0	7	4	1
Feat. dim	128	N/A	7-789	1024	768
# Classes	349	8	2	2983	153
Storage (GB)	0.86	1.2	22	104	202

8 EVALUATION

8.1 Methodology

Testbed. All experiments use Amazon EC2 g4dn.metal instances, each with 8 NVIDIA 16 GB T4 GPUs, 96 vCPU cores, and 384 GB DRAM. Instances are connected with a 100 Gbps network. By default, each experiment uses two instances. We also use an Amazon EC2 X1.32xlarge instance with 2 TB DRAM for graph partitioning.

Datasets. We evaluate our system on five diverse heterogeneous graph datasets (Table 1) spanning various scales, domains, and structures. The ogbn-mag dataset [22] is a heterogeneous academic citation network (Figure 2), where only paper nodes have features. The Freebase dataset [19], extracted from Freebase [16], is a complex knowledge graph with 64 edge types but no initial node features, thus requiring learnable features for all nodes. The Donor dataset [26], built with methods from [6], models DonorsChoose.org project proposal approvals, capturing complex relationships between teachers, schools, and project applications. It has rich attributes across all node types (dimensions varying from 7 to 789) and aims to predict proposal approval. The IGB-HET dataset [31] is a large-scale citation network where all nodes have high-dimensional features (1024) and many labeled nodes for training across 2983 classes. The MAG240M dataset [21], derived from the Microsoft Academic Graph [55], is one of the largest known heterogeneous graphs (240M nodes, 3.4B edges). Like ogbn-mag, only paper nodes have features, but they are high-dimensional (768) and consume significant storage (about 175 GB in float16).

HGNNs. We train three representative HGNN models: R-GCN [46], R-GAT [3], and HGT [23]. Following OGB leaderboard settings [22], all models use 2 layers and 64 hidden neurons per layer. We also evaluate *Heta* on a more recent HGNN model - SeHGNN [59] (results and discussion are available in [70]). For nodes without initial features (e.g., all nodes in Freebase), we use learnable embeddings of dimension 64. We set the minibatch size to 1024 for all datasets and sample two-hop neighbors with fanout {25, 20} in all experiments. We allocate a 4 GB cache size per GPU in *Heta*, providing a good balance between cache size and GPU memory needed for training.

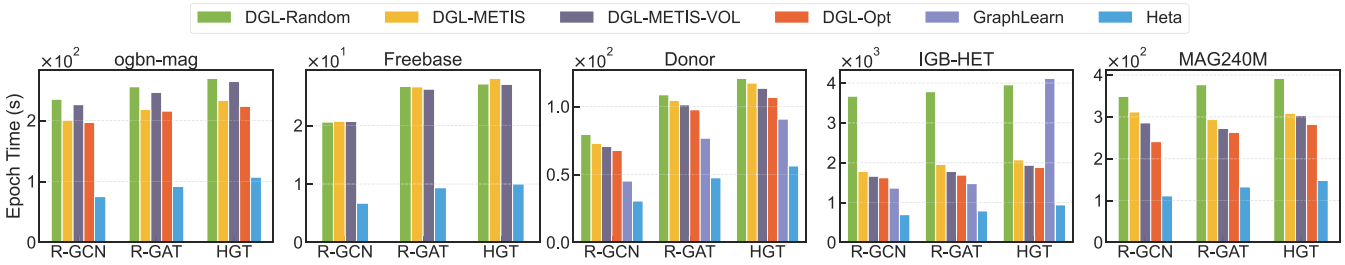


Figure 9: Overall performance on various HGNN models and datasets. *Heta* achieves significant speedup over baselines.

Baselines. We compare *Heta* with these baselines: (1) **DGL-Random**: uses random partitioning; (2) **DGL-METIS**: uses traditional edge-cut minimization; (3) **DGL-METIS-VOL** [29]: uses METIS 5.0+'s communication volume minimization (objtype='vol'). Unlike standard edge-cut, this heuristic directly minimizes the total communication volume by considering both boundary node count and the number of partitions each boundary node connects to; (4) **DGL-Opt**: our optimized version of the best-performing METIS variant (DGL-METIS/DGL-METIS-VOL) with GPU node feature caching.[2] It is inapplicable to featureless Freebase, as caching non-replicative learnable features gives DGL little benefit since remote workers still need network communication for fetching; (5) **GraphLearn** (PyTorch): uses its native caching and runs only on feature-rich datasets (Donor, IGB-HET) as it lacks learnable feature support. We apply the same 4 GB cache size per GPU and cache allocation method (§6) from *Heta* to DGL-Opt and GraphLearn.

8.2 Overall Performance

We first evaluate end-to-end distributed training performance by comparing epoch time (time for a complete training dataset pass) across systems. Figure 9 shows *Heta*'s superior performance in all scenarios, with speedups of 1.9× to 5.3× over DGL and 1.5× to 4.4× over GraphLearn. *Heta*'s advantages are especially clear against DGL variants. It achieves its highest speedup (up to 5.3×) over DGL-Random, which suffers from low data locality and frequent cross-partition communication from its random partitioning. While DGL-METIS improves this by minimizing edge-cuts to reduce communication, and DGL-Opt further boosts performance with node feature caching, *Heta* still holds significant advantages, with up to 3.1× and 2.6× speedup over these optimized versions, respectively. GraphLearn performs well among baselines on the Donor and IGB-Het datasets with its asynchronous graph sampling and feature cache. Still, *Heta* achieves up to 4.4× speedup over GraphLearn, mainly from its RAF computation paradigm and meta-partitioning, which effectively reduce cross-partition communication.

When examining performance across different model architectures, *Heta* demonstrates consistent advantages with average speedups of 2.8×, 2.6×, and 2.4× over baselines for R-GCN, R-GAT, and HGT, respectively. The variation in speedup factors can be attributed to the different computation-communication characteristics of these models. R-GCN shows the highest performance improvement as it is more communication-intensive. In contrast, R-GAT and HGT are more computation-bound due to their attention mechanisms, resulting in a higher ratio of computation over communication. This leaves a relatively smaller improvement space for *Heta* as the communication bottleneck is less pronounced in these models.

[2]DGL-Opt uses DGL-METIS-VOL for Donor/IGB-HET/MAG240M (superior performance) and DGL-METIS for others.

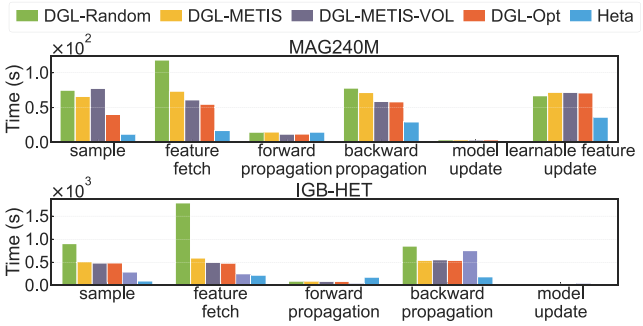


Figure 10: Time breakdown of training stages of R-GCN.

8.3 Training Time Breakdown

We analyze time distribution across training stages using the R-GCN model. Figure 10 details results on IGB-HET and MAG240M datasets. Other models and datasets showed similar patterns. The breakdown reveals *Heta*'s key innovations significantly reduce time in most training stages. Specifically, RAF and meta-partitioning cut cross-machine communication for sampling, feature fetching, and learnable feature updates by constraining operations to local partitions. The GPU cache further optimizes feature operations by holding frequent data in high-speed memory. While forward computation slightly increases due to aggregating relation-wise partial embeddings under RAF, backpropagation time is substantially reduced by avoiding cross-machine gradient synchronization, needing only fast intra-machine synchronization for multi-GPU cases. The model update stage also benefits from cross-machine model parallelism, as each machine holds only part of the model.

To quantify the communication benefits, Table 2 compares the inter-machine communication volume per epoch between *Heta* and baselines when training R-GCN on two machines. Other models show similar results. The results demonstrate that *Heta* dramatically reduces communication across all datasets, achieving total communication reductions ranging from 47.22% to 98.42%. This significant reduction stems from two key factors: *First*, meta-partitioning eliminates sampling and feature transfer communication by constraining cross-partition dependencies to only target nodes, as evidenced by

Table 2: Inter-machine communication volume comparison (GB per Epoch) when training R-GCN on two machines. *Heta* significantly reduces the communication by up to 98.42% compared to baselines from RAF and meta-partitioning.

Method	Inter-machine Comm. Breakdown				Comm. Reduction
	Sample	Feature Fetch/ Partial Aggr.	Grad. Sync.	Total Comm.	
ogbn-mag					
DGL-METIS	0.32	18.55	14.9	33.76	96.80%
DGL-METIS-VOL	0.38	22.38	19.9	42.66	**97.47%**
Heta	0	0.32	0.76	**1.08**	–
Freebase					
DGL-METIS	0.01	0.2	0.87	1.08	47.22%
DGL-METIS-VOL	0.01	0.2	1.25	1.45	**60.69%**
Heta	0	0.02	0.56	**0.57**	–
Donor					
DGL-METIS	0.04	4.12	2.64	6.8	92.50%
DGL-METIS-VOL	0.04	3.44	5.28	8.76	94.18%
GraphLearn	0.1	12.77	2.64	15.51	**96.71%**
Heta	0	0.25	0.25	**0.51**	–
IGB-HET					
DGL-METIS	2.72	1391.87	308.35	1702.93	95.29%
DGL-METIS-VOL	2.07	1061.66	616.69	1680.43	95.23%
GraphLearn	8.9	4557.14	308.35	4874.39	**98.36%**
Heta	0	3.07	77.09	**80.16**	–
MAG240M					
DGL-METIS	0.56	95.15	13.16	108.87	**98.42%**
DGL-METIS-VOL	0.36	61.69	17.4	79.45	97.84%
Heta	0	0.57	1.15	**1.72**	–

zero communication volume in these stages. *Second*, RAF's relation-specific processing requires only exchanging compact partial aggregations rather than high-dimensional features. For example, on the MAG240M dataset, while DGL-METIS transfers 95.15GB of features between machines, *Heta* only communicates 0.57GB of partial aggregations of target nodes. The benefits are particularly pronounced on datasets with large feature dimensions - for IGB-HET (1024-dimensional features), *Heta* reduces the total communication from 4874.39GB to just 80.16GB per epoch. Even for Freebase, which only has learnable features, *Heta* still achieves up to 60.69% reduction in communication volume. These results provide strong empirical validation of the theoretical communication complexity advantages of RAF and meta-partitioning. Note that despite drastically reduced inter-machine I/O cutting time in communication-heavy stages (Figure 10), stage-level speedups are not proportional, as local costs within each stage (e.g., host-device transfers via PCIe) become dominant when inter-machine I/O shrinks.

8.4 Meta-Partitioning Efficiency

Table 3 compares *Heta*'s meta-partitioning to DGL's random and METIS partitioning on the MAG240M and IGB-HET dataset (2 parts). We ignore METIS-VOL's results as they are similar to METIS. Both random and METIS demand hours for partitioning, mainly from splitting the original HetG and shuffling nodes/edges for contiguous ID ranges [66]. In contrast, our meta-partitioning can be done in just 20.6 minutes. GraphLearn assumes all nodes have features for partitioning, making it unsuitable for MAG240M. When partitioning IGB-HET into two parts, GraphLearn's random partitioning takes 611 seconds, while random and METIS partitioning of DGL take 1131 and 3014 seconds, respectively. *Heta* is the fastest, completing the task in 549 seconds: most of the time is spent saving partitioned graphs to disk while building/splitting/assigning metatrees takes less than 1 second. Moreover, meta-partitioning reduces peak memory usage by over 50% compared to traditional methods (406.3 GB vs. 917.3 GB on MAG240M) by avoiding the need to load and maintain the full graph structure in memory and eliminating intermediate data copies during node/edge shuffling. Instead, our approach only requires storing and manipulating the lightweight metatree structure during the partitioning process.

Table 3: Partitioning performance comparison on MAG240M and IGB-HET datasets.

Method	MAG240M		IGB-HET	
	Time	Peak Memory	Time	Peak Memory
Random	3.9 hr	917.3 GB	1131 s	253.0 GB
METIS	7.2 hr	847.7 GB	3014 s	261.9 GB
GraphLearn	-	-	611 s	175.4 GB
Meta-partitioning	20.6 min	406.3 GB	549 s	132.8 GB

Table 4: Partition storage comparison across all ddatasets. MP (Meta-Partitioning) consistently achieves the lowest topology storage through its relation-aware approach.

Dataset	Strategy	Storage Consumption (GB)		
		Topology	Features	Total
ogbn-mag	MP (2-hop)	**1.10**	0.70	**1.80**
	MP (3-hop)	1.60	0.70	2.30
	METIS	1.32	0.57	1.89
	Random	1.78	0.57	2.35
Freebase	MP (2-hop)	**1.95**	0	**1.95**
	MP (3-hop)	2.08	0	2.08
	METIS	4.70	1.07	5.77
	Random	5.50	0.02	5.52
Donor	MP (2-hop)	**0.46**	23.78	24.24
	MP (3-hop)	1.10	42.23	43.33
	METIS	0.86	23.17	24.03
	Random	1.23	21.95	23.18
	GraphLearn	1.13	33.37	34.50
IGB-HET	MP (2-hop)	**9.22**	198.15	207.37
	MP (3-hop)	18.63	198.15	216.78
	METIS	15.43	105.59	121.02
	Random	20.93	101.67	122.60
	GraphLearn	11.16	226.69	241.76
MAG240M	MP (2-hop)	**59.13**	348.55	407.68
	MP (3-hop)	98.49	348.55	447.04
	METIS	106.71	207.65	314.45
	Random	150.14	179.75	329.89

Table 4 details storage consumption comparison across partitioning strategies. We ignore METIS-VOL as its storage use is nearly identical to METIS. For graph topology, meta-partitioning (MP) with 2-hop consistently uses the least storage on all datasets, with

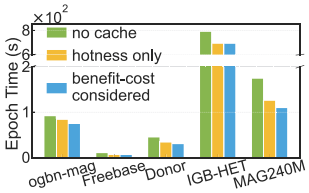


Figure 11: Comparison of GPU cache methods.

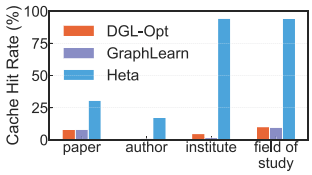


Figure 12: Cache hit rate comparison on IGB-HET.

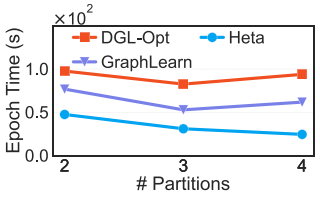


Figure 13: Performance under different GPU numbers. *Heta* scales well with more GPUs.

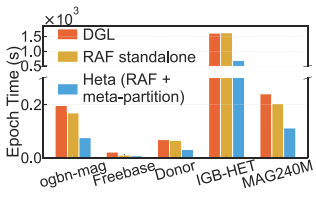


Figure 14: Meta-partitioning significantly enhances RAF's effectiveness.

reductions of up to 58.5% versus METIS (on Freebase) and 64.5% versus random partitioning (on Freebase). This efficiency comes from meta-partitioning's relation-aware approach, which eliminates extensive edge replication (1-hop replication [44]) across partitions—a common need in edge-cut methods like those used by DGL and GraphLearn. Moreover, unlike DGL, which converts heterogeneous graphs into homogeneous ones requiring additional feature storage for type information, or GraphLearn, which maintains costly partition assignment tables (nearly doubling their feature storage sizes), our approach eliminates such auxiliary storage overhead. While meta-partitioning may incur higher feature storage due to duplication of some node features across partitions to enable local access, this storage-communication tradeoff in our framework design leads to substantially reduced communication overhead during training compared to existing frameworks like DGL and GraphLearn (up to 98.42% reduction, Table 2). The total storage consumption remains competitive, with meta-partitioning even achieving lower overall storage than competitors (on ogbn-mag and Freebase). Furthermore, deeper models (3-hop vs. 2-hop) only increase total storage by 25% on average (up to 78%) because meta-partitioning deduplicates repeated relations (Step 4 in §5).

8.5 GPU Cache

Figure 11 shows the effectiveness of *Heta*'s cache design via epoch time measurements in R-GCN training across various datasets. Compared to a no-cache baseline, our approach considering both hotness and miss-penalty achieves up to 1.6× training speedup. This design also surpasses a simpler hotness-only allocation by up to 15%, with the largest gains on MAG240M (15%) and Donor (13%). These gains are most pronounced on datasets with varied feature dimensions and learnable features (as in MAG240M), whereas datasets with uniform feature dimensions like IGB-HET show modest gains. The influence of feature dimensions and learnable features on miss-penalty ratios directly impacts cache performance by affecting data retrieval efficiency (§6).

The superior performance of *Heta*'s caching strategy is further explained by the cache hit rates shown in Figure 12. When training R-GAT on IGB-HET, *Heta* achieves substantially higher cache hit rates across all node types compared to DGL-Opt and GraphLearn, a pattern that holds consistent across other datasets. This advantage stems from our meta-partitioning approach, which limits node types in each partition and allows GPU caches to focus exclusively on locally needed node types. In contrast, traditional approaches like DGL and GraphLearn must cache all node types due to their type-blind graph partitioning. The practical impact of this difference is striking: while DGL-Opt achieves only a 1.1× speedup over DGL-METIS (no cache DGL variant, as shown in Figure 9), *Heta* delivers a 1.6× speedup compared to no-cache configurations. This performance gap is directly attributable to cache efficiency - for instance, with the "paper" node type in MAG240M, *Heta* achieves a 67% cache hit rate compared to DGL-Opt's mere 11%, significantly reducing feature fetching overhead.

8.6 Scalability

We evaluate the training epoch time of R-GAT on the Donor dataset with 16, 24, and 32 GPUs across 2, 3, and 4 Amazon EC2 g4dn.metal instances (2, 3, and 4 partitions), keeping a global batch size of 1024. Figure 13 shows *Heta*'s robust scalability with more GPUs; other models showed similar results. In contrast, DGL-Opt and GraphLearn performance degrades when scaling from 24 to 32 GPUs. This divergence stems from different distributed data approaches: while all systems benefit from reduced per-GPU computation workload, DGL-Opt and GraphLearn face escalating communication overhead as graph data spreads across more machines, requiring more inter-machine data fetching. *Heta* overcomes this with its meta-partitioning strategy, which constrains boundary nodes to target nodes, maintaining constant communication complexity regardless of partition count.

8.7 Ablation Study

Component Ablation. To isolate RAF and meta-partitioning contributions, we run experiments using RAF with vanilla relation-based partitioning (without relation duplication) and compare it to both DGL (the best-performing variant) and full *Heta* (RAF + meta-partitioning) with R-GCN on various datasets. Other models show similar results. Figure 14 shows RAF alone achieves 1.2×-2× speedup over DGL on ogbn-mag, Freebase, and MAG240M datasets, but limited improvement (≤5%) on IGB-HET and Donor. This discrepancy stems from two key factors: (1) Without topology replication in relation-based partitioning, RAF requires intensive P2P communication (5× to 22× more messages than the full *Heta* system) for partial aggregations and their gradients, increasing forward/backward propagation time by 38%-100% compared to the ful *Heta* system; (2) Remote sampling operations for cross-partition edges increase sampling time by 3.4× to 9.9× compared to *Heta*'s local-only sampling. After integrating meta-partitioning, *Heta* eliminates all remote sampling operations through relation replication and reduces P2P messages by up to 22× by constraining boundary nodes to target types only. This synergistic combination delivers a 1.5×-2.3× speedup over standalone RAF, demonstrating that meta-partitioning is essential for maximizing RAF's potential by

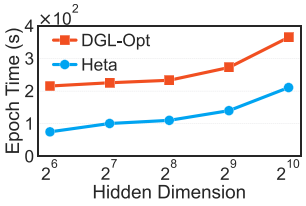


Figure 15: Performance under different hidden dimension sizes.

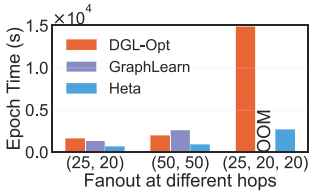
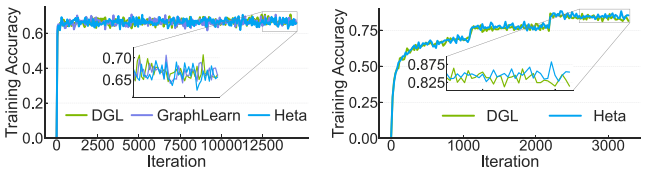


Figure 16: Performance under different sampling fanouts and hops.



(a) IGB-HET dataset **(b) MAG240M dataset**

Figure 17: *Heta* achieves the same model accuracy as DGL.

fundamentally reconfiguring the graph layout to align with HGNN computation patterns.

Hidden Dimension. In Figure 15, we evaluate the training epoch time of R-GCN on the ogbn-mag dataset while increasing the hidden dimension from 64 to 1024. As the hidden dimension increases, *Heta*'s epoch time grows due to increased network communication from larger partial aggregations. Nevertheless, *Heta* maintains a 1.7× speedup over DGL-Opt even at a hidden dimension of 1024, though it's worth noting that HGNN models typically use smaller hidden dimensions than feature dimensions [56].

Sampling. Figure 16 shows our experiments with varying sampling fanouts and neighborhood hops when training R-GCN on the IGB-HET dataset. *Heta* achieves 2.3× to 4.9× speedups over DGL-Opt with larger fanout and more neighborhood hops, while GraphLearn encounters out-of-memory issues with fanouts of {25, 20, 20} for three sampling hops. This performance advantage stems from *Heta*'s meta-partitioning, which maintains constant cross-partition communication, while baselines require increasing network communication when sampling larger neighborhoods.

8.8 Model Accuracy

Figure 17a and Figure 17b show training accuracy curves for R-GAT on IGB-HET and HGT on MAG240M, respectively. *Heta* matches training accuracies of baselines on both datasets. For test accuracy, *Heta* and DGL both reach 0.66 on MAG240M, matching the leaderboard results [1]. All three systems reach 0.68 on IGB-HET, consistent with its original paper's reported accuracy [31]. These results validate our theoretical analysis (§4) that *Heta* preserves the mathematical equivalence of the model.

9 RELATED WORK

Other Communication Reduction Methods in Distributed and Out-of-core GNN Training. BNS-GCN [53] drops a subset of boundary nodes to reduce communication. PipeGCN [54] introduces staleness to overlap communication and computation. AdaQP [52] quantizes transmitted features to low-precision integers. Sancus [42] and FreshGNN [24] cache historical node embeddings for reuse. Disk-based, single-machine solutions such as Marius-GNN [51] and OUTRE [47] reduce overhead by reusing sampled neighbors, reordering samples, and caching embeddings. All these techniques alter the computational equivalence, whereas *Heta*'s RAF maintains it (Prop.1 in §4). These methods are orthogonal and can be combined with *Heta* to reduce communication further.

Graph Partitioning in Graph Processing and GNN Systems. Early systems like Pregel [37] and GraphX [15] use edge-cut partitioning, while PowerGraph [14] applies vertex-cut, and PowerLyra [5] combines both. However, they treat property graphs homogeneously, ignoring node/edge types. For Resource Description Framework (RDF) graphs, Minimum Property-Cut (MPC) [44] and Minimum Motif-Cut (MMC) [43] improve partitioning to reduce inter-partition joins and optimize SPARQL queries. Modern GNN systems commonly adopt METIS [30] for partitioning [48, 53, 66, 68, 71]. DistGNN [38] uses vertex-cut. BGL [36] and ByteGNN [64] consider multi-hop connectivity to preserve local structures within partitions. Still, these approaches ignore relations and node/edge types during partitioning, leading to inefficient data placement for heterogeneous graphs. *Heta*'s meta-partitioning overcomes this by relation-aware partitioning tailored for HetGs (§5).

GPU Feature Cache. PaGraph [35] and GNNLab [58] implement GPU caching for read-only node features. BGL [36] uses FIFO-based dynamic node feature caching. Legion [48] unifies caching for node features and topology. GNNFlow [69] and TASER [7] extend dynamic caching to edge features. These approaches neither consider varying cache miss penalties across node types nor handle updatable learnable features. Our heterogeneity-aware cache design addresses both challenges (§6).

10 CONCLUSION

Heta is a distributed framework for efficient HGNN training on heterogeneous graphs. At its core, the RAF computation paradigm eliminates feature movement across machines, while meta-partitioning further reduces communication by minimizing boundary node count. The heterogeneity-aware GPU cache considers varying cache miss-penalty ratios across node types to further optimize data movements. Our comprehensive evaluation demonstrates that *Heta* achieves superior performance, delivering up to 5.3× speedup in epoch time without loss of accuracy compared to state-of-the-art systems DGL and GraphLearn. Notably, *Heta*'s performance advantages become more pronounced as the sampling neighborhood size increases, and its meta-partitioning demonstrates superior efficiency in both time and memory footprint compared to state-of-the-art graph partitioning methods such as METIS.

ACKNOWLEDGMENTS

We would like to thank the anonymous reviewers and area chairs for their helpful comments and suggestions. This work was supported in part by grants from Hong Kong RGC under the contracts 17207621, 17203522, and C7004-22G (CRF).

REFERENCES

[1] Open Graph Benchmark. 2024. OGB-LSC Leaderboards. https://ogb.stanford.edu/docs/lsc/leaderboards/ (Accessed: 2025-04-01).

[2] Aydın Buluç, Henning Meyerhenke, Ilya Safro, Peter Sanders, and Christian Schulz. 2016. *Recent Advances in Graph Partitioning*. Springer.

[3] Dan Busbridge, Dane Sherburn, Pietro Cavallo, and Nils Y Hammerla. 2019. Relational Graph Attention Networks. *arXiv preprint* (2019).

[4] Yukuo Cen, Xu Zou, Jianwei Zhang, Hongxia Yang, Jingren Zhou, and Jie Tang. 2019. Representation Learning for Attributed Multiplex Heterogeneous Network. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*.

[5] Rong Chen, Jiaxin Shi, Yanzhe Chen, and Haibo Chen. 2015. PowerLyra: Differentiated Graph Computation and Partitioning on Skewed Graphs. In *Proceedings of the Tenth European Conference on Computer Systems*.

[6] Milan Cvitkovic. 2020. Supervised Learning on Relational Databases with Graph Neural Networks. *arXiv preprint* (2020).

[7] Gangda Deng, Hongkuan Zhou, Hanqing Zeng, Yinglong Xia, Christopher Leung, Jianbo Li, Rajgopal Kannan, and Viktor Prasanna. 2024. TASER: Temporal Adaptive Sampling for Fast and Accurate Dynamic Graph Representation Learning. In *Proceedings of International Symposium on Parallel and Distributed Processing*.

[8] NetworkX Developers. 2023. NetworkX. https://networkx.org/ (Accessed: 2025-04-01).

[9] Facebook. 2023. Gloo. https://github.com/facebookincubator/gloo (Accessed: 2025-04-01).

[10] Matthias Fey, Jan E Lenssen, Frank Weichert, and Jure Leskovec. 2021. GNNAutoScale: Scalable and Expressive Graph Neural Networks via Historical Embeddings. In *Proceedings of International Conference on Machine Learning*.

[11] Xinyu Fu, Jiani Zhang, Ziqiao Meng, and Irwin King. 2020. MAGNN: Metapath Aggregated Graph Neural Network for Heterogeneous Graph Embedding. In *Proceedings of The Web Conference 2020*.

[12] Swapnil Gandhi and Anand Padmanabha Iyer. 2021. P^3: Distributed Deep Graph Learning at Scale. In *Proceedings of the 15th USENIX Symposium on Operating Systems Design and Implementation*.

[13] Liqun Gao, Haiyang Wang, Zhouran Zhang, Hongwu Zhuang, and Bin Zhou. 2022. HetInf: Social Influence Prediction with Heterogeneous Graph Neural Network. *Frontiers in Physics* (2022).

[14] Joseph E Gonzalez, Yucheng Low, Haijie Gu, Danny Bickson, and Carlos Guestrin. 2012. PowerGraph: Distributed Graph-Parallel Computation on Natural Graphs. In *Proceedings of the 10th USENIX Symposium on Operating Systems Design and Implementation*.

[15] Joseph E Gonzalez, Reynold S Xin, Ankur Dave, Daniel Crankshaw, Michael J Franklin, and Ion Stoica. 2014. GraphX: Graph Processing in a Distributed Dataflow Framework. In *Proceedings of the 11th USENIX Symposium on Operating Systems Design and Implementation*.

[16] Google. 2024. Freebase Data Dumps. https://developers.google.com/freebase/data (Accessed: 2025-04-01).

[17] Ronald L. Graham. 1969. Bounds on Multiprocessing Timing Anomalies. *SIAM journal on Applied Mathematics* (1969).

[18] Will Hamilton, Zhitao Ying, and Jure Leskovec. 2017. Inductive Representation Learning on Large Graphs. In *Proceedings of Advances in Neural Information Processing Systems*.

[19] Hui Han, Tianyu Zhao, Cheng Yang, Hongyi Zhang, Yaoqi Liu, Xiao Wang, and Chuan Shi. 2022. OpenHGNN: An Open Source Toolkit for Heterogeneous Graph Neural Network. In *Proceedings of the 31st ACM International Conference on Information & Knowledge Management*.

[20] Binbin Hu, Zhiqiang Zhang, Chuan Shi, Jun Zhou, Xiaolong Li, and Yuan Qi. 2019. Cash-out User Detection Based on Attributed Heterogeneous Information Network with a Hierarchical Attention Mechanism. In *Proceedings of the AAAI Conference on Artificial Intelligence*.

[21] Weihua Hu, Matthias Fey, Hongyu Ren, Maho Nakata, Yuxiao Dong, and Jure Leskovec. 2021. OGB-LSC: A Large-Scale Challenge for Machine Learning on Graphs. In *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks*.

[22] Weihua Hu, Matthias Fey, Marinka Zitnik, Yuxiao Dong, Hongyu Ren, Bowen Liu, Michele Catasta, and Jure Leskovec. 2020. Open Graph Benchmark: Datasets for Machine Learning on Graphs. In *Proceedins of Advances in Neural Information Processing systems*.

[23] Ziniu Hu, Yuxiao Dong, Kuansan Wang, and Yizhou Sun. 2020. Heterogeneous Graph Transformer. In *Proceedings of the World Web Conference*.

[24] Kezhao Huang, Haitian Jiang, Minjie Wang, Guangxuan Xiao, David Wipf, Xiang Song, Quan Gan, Zengfeng Huang, Jidong Zhai, and Zheng Zhang. 2024. FreshGNN: Reducing Memory Access via Stable Historical Embeddings for Graph Neural Network Training. In *Proceedings of the VLDB Endowment*.

[25] Zhihao Jia, Sina Lin, Mingyu Gao, Matei Zaharia, and Alex Aiken. 2020. Improving the Accuracy, Scalability, and Performance of Graph Neural Networks with ROC. In *Proceedings of Machine Learning and Systems*.

[26] Kaggle. 2024. DonorsChoose.org Application Screening. https://www.kaggle.com/competitions/donorschoose-application-screening/data (Accessed: 2025-04-01).

[27] Maria Kalantzi and George Karypis. 2021. Position-based Hash Embeddings for Scaling Graph Neural Networks. In *2021 IEEE International Conference on Big Data*.

[28] Tim Kaler, Alexandros Iliopoulos, Philip Murzynowski, Tao Schardl, Charles E Leiserson, and Jie Chen. 2023. Communication-Efficient Graph Neural Networks with Probabilistic Neighborhood Expansion Analysis and Caching. In *Proceedings of Machine Learning and Systems*.

[29] George Karypis. 2013. METIS: A Software Package for Partitioning Unstructured Graphs, Partitioning Meshes, and Computing Fill-Reducing Orderings of Sparse Matrices Version 5.1.0. https://karypis.github.io/glaros/files/sw/metis/manual.pdf (Accessed: 2025-04-01).

[30] George Karypis and Vipin Kumar. 1998. A Fast and High Quality Multilevel Scheme for Partitioning Irregular Graphs. *SIAM Journal on scientific Computing* (1998).

[31] Arpandeep Khatua, Vikram Sharma Mailthody, Bhagyashree Taleka, Tengfei Ma, Xiang Song, and Wen-mei Hwu. 2023. IGB: Addressing The Gaps In Labeling, Features, Heterogeneity, and Size of Public Graph Datasets for Deep Learning Research. In *Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*.

[32] Diederik P Kingma and Jimmy Ba. 2014. Adam: A Method for Stochastic Optimization. *arXiv preprint* (2014).

[33] Thomas N. Kipf and Max Welling. 2017. Semi-Supervised Classification with Graph Convolutional Networks. In *Proceedings of International Conference on Learning Representations*.

[34] Shen Li, Yanli Zhao, Rohan Varma, Omkar Salpekar, Pieter Noordhuis, Teng Li, Adam Paszke, Jeff Smith, Brian Vaughan, Pritam Damania, et al. 2020. PyTorch Distributed: Experiences on Accelerating Data Parallel Training. In *Proceedings of the VLDB Endowment*.

[35] Zhiqi Lin, Cheng Li, Youshan Miao, Yunxin Liu, and Yinlong Xu. 2020. PaGraph: Scaling GNN Training on Large Graphs via Computation-aware Caching. In *Proceedings of the 11th ACM Symposium on Cloud Computing*.

[36] Tianfeng Liu, Yangrui Chen, Dan Li, Chuan Wu, Yibo Zhu, Jun He, Yanghua Peng, Hongzheng Chen, Hongzhi Chen, and Chuanxiong Guo. 2023. BGL: GPU-efficient GNN Training by Optimizing Graph Data I/O and Preprocessing. In *Proceedings of the 20th USENIX Symposium on Networked Systems Design and Implementation*.

[37] Grzegorz Malewicz, Matthew H Austern, Aart JC Bik, James C Dehnert, Ilan Horn, Naty Leiser, and Grzegorz Czajkowski. 2010. Pregel: A System for Large-scale Graph Processing. In *Proceedings of the 2010 ACM SIGMOD International Conference on Management of Data*.

[38] Vasimuddin Md, Sanchit Misra, Guixiang Ma, Ramanarayan Mohanty, Evangelos Georganas, Alexander Heinecke, Dhiraj Kalamkar, Nesreen K Ahmed, and Sasikanth Avancha. 2021. DistGNN: Scalable Distributed Training for Large-scale Graph Neural Networks. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*.

[39] NVIDIA. 2023. CUDA Peer Device Memory Access. https://docs.nvidia.com/cuda/cuda-runtime-api/group__CUDART__PEER.html.

[40] NVIDIA. 2023. NCCL. https://github.com/NVIDIA/nccl (Accessed: 2025-04-01).

[41] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. 2019. PyTorch: An Imperative Style, High-performance Deep Learning Library. In *Proceedings of Advances in Neural Information Processing Systems*.

[42] Jingshu Peng, Zhao Chen, Yingxia Shao, Yanyan Shen, Lei Chen, and Jiannong Cao. 2022. Sancus: Staleness-aware Communication-avoiding Full-graph Decentralized Training in Large-scale Graph Neural Networks. In *Proceedings of the VLDB Endowment*.

[43] Peng Peng, Shengyi Ji, M Tamer Özsu, and Lei Zou. 2024. Minimum Motif-cut: a Workload-aware RDF Graph Partitioning Strategy. *The VLDB Journal* (2024).

[44] Peng Peng, M Tamer Özsu, Lei Zou, Cen Yan, and Chengjun Liu. 2022. MPC: Minimum Property-cut RDF Graph Partitioning. In *2022 IEEE 38th International Conference on Data Engineering (ICDE)*.

[45] Fabio Petroni, Leonardo Querzoni, Khuzaima Daudjee, Shahin Kamali, and Giorgio Iacoboni. 2015. Hdrf: Stream-based Partitioning for Power-law Graphs. In *Proceedings of the 24th ACM international on conference on information and knowledge management*.

[46] Michael Schlichtkrull, Thomas N Kipf, Peter Bloem, Rianne Van Den Berg, Ivan Titov, and Max Welling. 2018. Modeling Relational Data with Graph Convolutional Networks. In *Proceedings of The Semantic Web: 15th International Conference*.

[47] Zeang Sheng, Wentao Zhang, Yangyu Tao, and Bin Cui. 2024. OUTRE: An OUT-of-Core De-REdundancy GNN Training Framework for Massive Graphs within A Single Machine. *Proceedings of the VLDB Endowment* (2024).

[48] Jie Sun, Li Su, Zuocheng Shi, Wenting Shen, Zeke Wang, Lei Wang, Jie Zhang, Yong Li, Wenyuan Yu, Jingren Zhou, and Fei Wu. 2023. Legion: Automatically Pushing the Envelope of Multi-GPU System for Billion-Scale GNN Training. In *Proceedings of 2023 USENIX Annual Technical Conference*.

[49] GLT Team. 2023. GraphLearn for PyTorch. https://github.com/alibaba/graphlearn-for-pytorch (Accessed: 2025-04-01).

[50] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. 2018. Graph Attention Networks. In *Proceedings of International Conference on Learning Representations*.

[51] Roger Waleffe, Jason Mohoney, Theodoros Rekatsinas, and Shivaram Venkataraman. 2023. MariusGNN: Resource-efficient Out-of-core Training of Graph Neural Networks. In *Proceedings of the Eighteenth European Conference on Computer Systems*.

[52] Borui Wan, Juntao Zhao, and Chuan Wu. 2023. Adaptive Message Quantization and Parallelization for Distributed Full-Graph GNN Training. In *Proceedings of Machine Learning and Systems*.

[53] Cheng Wan, Youjie Li, Ang Li, Nam Sung Kim, and Yingyan Lin. 2022. BNS-GCN: Efficient Full-graph Training of Graph Convolutional Networks with Partition-parallelism and Random Boundary Node Sampling. In *Proceedings of Machine Learning and Systems*.

[54] Cheng Wan, Youjie Li, Cameron R Wolfe, Anastasios Kyrillidis, Nam Sung Kim, and Yingyan Lin. 2021. PipeGCN: Efficient Full-Graph Training of Graph Convolutional Networks with Pipelined Feature Communication. In *Proceedings of International Conference on Learning Representations*.

[55] Kuansan Wang, Zhihong Shen, Chiyuan Huang, Chieh-Han Wu, Yuxiao Dong, and Anshul Kanakia. 2020. Microsoft Academic Graph: When Experts are Not Enough. *Quantitative Science Studies* (2020).

[56] Xiao Wang, Deyu Bo, Chuan Shi, Shaohua Fan, Yanfang Ye, and S Yu Philip. 2022. A Survey on Heterogeneous Graph Embedding: Methods, Techniques, Applications and Sources. *IEEE Transactions on Big Data* (2022).

[57] Xiao Wang, Houye Ji, Chuan Shi, Bai Wang, Yanfang Ye, Peng Cui, and Philip S Yu. 2019. Heterogeneous Graph Attention Network. In *Proceedings of the world wide web conference*.

[58] Jianbang Yang, Dahai Tang, Xiaoniu Song, Lei Wang, Qiang Yin, Rong Chen, Wenyuan Yu, and Jingren Zhou. 2022. GNNLab: A Factored System for Sample-based GNN Training over GPUs. In *Proceedings of the 17th European Conference on Computer Systems*.

[59] Xiaocheng Yang, Mingyu Yan, Shirui Pan, Xiaochun Ye, and Dongrui Fan. 2023. Simple and Efficient Heterogeneous Graph Neural Network. In *Proceedings of the AAAI Conference on Artificial Intelligence*.

[60] Chunxing Yin, Da Zheng, Israt Nisa, Christos Faloutsos, George Karypis, and Richard Vuduc. 2022. Nimble GNN Embedding with Tensor-train Decomposition. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*.

[61] Lingfan Yu, Jiajun Shen, Jinyang Li, and Adam Lerer. 2020. Scalable Graph Neural Networks for Heterogeneous Graphs. *arXiv preprint* (2020).

[62] Hao Yuan, Yajiong Liu, Yanfeng Zhang, Xin Ai, Qiange Wang, Chaoyi Chen, Yu Gu, and Ge Yu. 2023. Comprehensive Evaluation of GNN Training Systems: A Data Management Perspective. In *Proceedings of the VLDB Endowment*.

[63] Chuxu Zhang, Dongjin Song, Chao Huang, Ananthram Swami, and Nitesh V Chawla. 2019. Heterogeneous Graph Neural Network. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*.

[64] Chenguang Zheng, Hongzhi Chen, Yuxuan Cheng, Zhezheng Song, Yifan Wu, Changji Li, James Cheng, Hao Yang, and Shuai Zhang. 2022. ByteGNN: Efficient Graph Neural Network Training at Large Scale. *Proceedings of the VLDB Endowment* (2022).

[65] Da Zheng, Chao Ma, Minjie Wang, Jinjing Zhou, Qidong Su, Xiang Song, Quan Gan, Zheng Zhang, and George Karypis. 2020. DistDGL: Distributed Graph Neural Network Training for Billion-scale Graphs. In *Proceedings of 2020 IEEE/ACM 10th Workshop on Irregular Applications: Architectures and Algorithms*.

[66] Da Zheng, Xiang Song, Chengru Yang, Dominique LaSalle, and George Karypis. 2022. Distributed Hybrid CPU and GPU Training for Graph Neural Networks on Billion-scale Heterogeneous Graphs. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*.

[67] Jiawei Zheng, Qianli Ma, Hao Gu, and Zhenjing Zheng. 2021. Multi-view Denoising Graph Auto-encoders on Heterogeneous Information Networks for Cold-start Recommendation. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*.

[68] Zheng, Da and Song, Xiang and Ma, Chao and Tan, Zeyuan and Ye, Zihao and Dong, Jin and Xiong, Hao and Zhang, Zheng and Karypis, George. 2020. DGL-KE: Training Knowledge Graph Embeddings at Scale. In *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval*.

[69] Yuchen Zhong, Guangming Sheng, Tianzuo Qin, Minjie Wang, Quan Gan, and Chuan Wu. 2023. GNNFlow: A Distributed Framework for Continuous Temporal GNN Learning on Dynamic Graphs. *arXiv preprint* (2023).

[70] Yuchen Zhong, Junwei Su, Chuan Wu, and Minjie Wang. 2024. *Heta: Distributed Training of Heterogeneous Graph Neural Networks*. Technical Report. https://arxiv.org/pdf/2408.09697.pdf (Accessed: 2025-06-11).

[71] Rong Zhu, Kun Zhao, Hongxia Yang, Wei Lin, Chang Zhou, Baole Ai, Yong Li, and Jingren Zhou. 2019. AliGraph: A Comprehensive Graph Neural Network Platform. In *Proceedings of the VLDB Endowment*.