A META-PARTITIONING ALGORITHM

Algorithm 2 demonstrates the detailed heterogeneous graph partitioning algorithm for Meta-Partitioning.

Algorithm 2: Meta-Partitioning of HetG

```
Input: Heterogeneous graph $G$, weighted metagraph $M$,
       target node type $root$, number of partitions $p$,
       number of HGNN layers $k$, (optional) user-defined
       metapaths
Output: Balanced partitions with minimized boundary
        nodes
1  if metapaths provided then
2      Construct a metatree $T$ with metapaths
3  else
4      Construct a metatree $T$ with $k$-depth BFS from $root$ in $M$
5  end
6  foreach child $c$ of root in $T$ do
7      Construct a sub-metatree $S_c$ starting from $root$ and
         containing $c$ and its descendants
8      Compute weight $w_c$ of $S_c$ as the sum of the weights of
         leaf vertices and links in $S_c$
9  end
10 Sort sub-metatrees in descending order of weights
11 Initialize a list of $p$ empty partitions $P = \{\emptyset, \dots, \emptyset\}$
12 Initialize an array of $p$ sums $sums = [0, 0, \dots, 0]$
13 foreach sub-metatree $S$ in the sorted list of sub-metatrees do
14     Find index $i$ of the partition with the smallest sum in
         $sums$
15     Add $S$ to the $i$-th partition in $P$
16     Add the weight of $S$ to $sums[i]$
17 end
18 foreach partition $P_i$ in $P$ do
19     Deduplicate relations and node features in $P_i$
20     Construct HetG partition based on $G$ and $P_i$
21 end
22 return $P$
```

B PROOFS

Proposition 1. Let v be an arbitrary target node. In the vanilla scheme, the embedding of $\mathbf{h}_v^{(\text{vanilla})}$ is computed as,

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

Under RAF, the embedding of $\mathbf{h}_v^{(\text{RAF})}$ is computed through two steps. First, by lines 2-6 in Algorithm 1, the intermediate embedding of the node from different relations is computed remotely and aggregated at the remote worker. This yields a set of partial aggregations,

$$\left\{\text{AGG}_r^{(l)}\left(\left\{\mathbf{h}_u^{(l-1)}, u \in N_r(v)\right\}\right), r \in \mathcal{R}\right\}$$

Then, these messages are sent to the worker with the target node, and the worker performs cross-relation aggregation $\text{AGG}_{\text{all}}^{(l)}$ on these partial aggregations (lines 8-11 in Algorithm 1). Since AGG(.) is order (permutation) invariant, we have that

$$\mathbf{h}_v^{(\text{RAF})} = \mathbf{h}_v^{(\text{vanilla})}$$

□

Proposition 2. Given an arbitrary partition G_1, G_2, we first focus our analysis on G_1. Let k denote the number of relations in the graphs, and recall that $\text{B}(G_1)$ is the number of boundary node in G_1. Let v be an arbitrary node in G_1 that requires communication from G_2. By definition, v must be a boundary node as it has a neighborhood from partition G_2. By lines 4-11 in Algorithm 1, only the intermediate representation of each relation r in v's neighborhood needs to be communicated. Therefore, the number of messages received for v in G_1 is at most k. Therefore, the number of messages received for G_1 from G_2 is at most

$$k\text{B}(G_1).$$

Since k is a constant for a given dataset or task and is independent of graph size, we have the communication complexity of worker G_1 is

$$\Theta(\text{B}(G_1)).$$

This completes the proof since the analysis for G_2 is symmetric.

□

Proposition 3. Again, let us focus on the analysis in G_1. By definition of boundary node, a boundary node in G_1 must be an endpoint of at least one cross-partition edge. However, there may be multiple cross-partition edges connecting to the node. Therefore, we have that

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

By a symmetric argument, we have that

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

Combining the result above, we can obtain that

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

□

Proposition 4. We prove the NP-hardness by reduction from the vertex-cut partitioning problem on metagraphs, which is known to be NP-hard [3]. Given a metagraph $M = (A, R)$ where A is the set of node types and R is the set of relations, the vertex-cut partitioning problem aims to partition the edges of M into p balanced subsets while minimizing the maximum number of vertices that are cut (duplicated across partitions).

Let us construct a mapping from the vertex-cut partitioning problem to our constrained partitioning problem:

- Each relation $r \in R$ in the metagraph corresponds to a mono-relation subgraph in the heterogeneous graph G.
- Assigning a relation r to partition i in the vertex-cut problem corresponds to assigning the corresponding mono-relation subgraph to partition G_i in our problem.
- A vertex (node type) $a \in A$ is cut in the vertex-cut problem if its incident edges (relations) are assigned to different partitions. In our problem, this corresponds to nodes of type a becoming boundary nodes when they participate in relations assigned to different partitions.

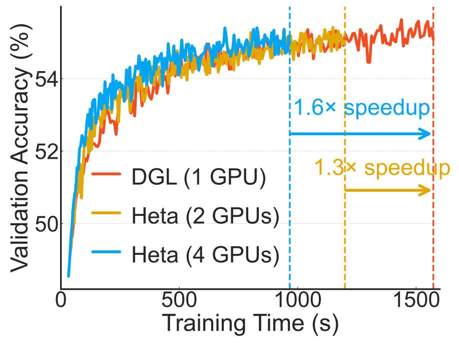

Figure 18: Validation accuracy vs. training time of training SeHGNN on the ogbn-mag dataset.

Given this mapping, minimizing the maximum number of boundary nodes in Eq. (2) is equivalent to minimizing the maximum number of vertex cuts in the vertex-cut partitioning problem. The balance constraint in both problems is also preserved. Since the vertex-cut partitioning problem is NP-hard, our constrained partitioning problem is also NP-hard. □

C EVALUATIONS ON ADDITIONAL HGNN MODELS

In addition to R-GCN, R-GAT, and HGT, *Heta* can support newer HGNN architectures as long as they follow the canonical form described in Eq. (1). Recent pre-computation-based HGNNs, such as Neighbor Averaging over Relation Subgraphs (NARS)[65] and Simple and Efficient Heterogeneous Graph Neural Network (SeHGNN)[63], have demonstrated superior performance through pre-computing relation-specific or metapath-based neighbor-aggregated features. During training, these models retrieve the pre-computed features for each mini-batch of target nodes and aggregate them using methods like multilayer perceptrons (MLPs) or transformers [53]. This approach enhances efficiency by reducing redundant computations—since neighbor aggregation is performed once during preprocessing rather than repeatedly during training—and effectively captures the complex semantics of heterogeneous graphs.

Heta naturally supports these pre-computation-based models through its core techniques:

- **For the preprocessing phase**: *Heta*'s meta-partitioning aligns naturally with relation-specific or metapath-based feature computation. Each partition maintains complete feature matrices for its assigned relations or metapaths, preserving the locality benefits of our partitioning strategy while supporting the pre-computation paradigm.
- **For the training phase**: *Heta*'s RAF paradigm remains effective by treating the pre-computed features as relation-specific inputs. Each worker processes its local feature matrices independently before aggregating intermediate activations across relations/metapaths, maintaining the communication efficiency benefits of RAF.

For SeHGNN specifically, we implement a hybrid parallelization strategy combining model and data parallelism. Under RAF, each worker first processes their local features (e.g., feature projection) independently. The partial aggregations are then gathered using all-gather operations and concatenated into a tensor of shape $[B, M, D]$, where B is the batch size, M is the number of metapaths, and D is the hidden dimension size. This tensor undergoes semantic fusion through a transformer followed by MLP layers for final prediction. We observe that after the RAF phase (model parallelism), the remaining computations (transformer, MLP) are duplicated across workers since they operate on identical inputs. Therefore, we apply data parallelism to these components by partitioning the concatenated tensor along the batch dimension. This hybrid approach allows SeHGNN to benefit from both RAF's communication efficiency in the feature processing phase and traditional data parallelism's computational scaling in the later stages. The implementation of this hybrid parallelization strategy for SeHGNN is available in our open-source repository (https://github.com/jasperzhong/heta).

We evaluated *Heta* with SeHGNN on the ogbn-mag dataset using default hyperparameters from their original implementation, without additional optimizations like extra embeddings or multi-stage training. For featureless nodes, we used randomly initialized feature vectors as in the original paper. We do not use GPU cache in this experiment. Experiments were conducted on one Amazon EC2 g4dn.metal instance for multi-GPU training. We train 200 epochs for each experiment. As shown in Figure 18, *Heta* maintains comparable validation accuracy to single-GPU training while achieving 1.3× and 1.6× speedups with 2 and 4 GPUs, respectively. The relatively modest speedups can be attributed to two main factors: (1) The large size of partial aggregations, with hidden dimension D=512 exceeding the original feature dimension (256), which reduces RAF's effectiveness by increasing communication volume. (2) RAF's benefits are limited to the pre-transformer computation stages, with the remaining model components relying on traditional data parallelism for acceleration. Optimizations for better handling complex HGNN architectures are left as future work (e.g., tensor parallelism for the transformer layers [50], or pipeline parallelism between the feature processing and transformer stages [58]).