NeuGraph

q1

EdgeNN不是应用于边的更新吗，为什么是更新Layer2上的点?

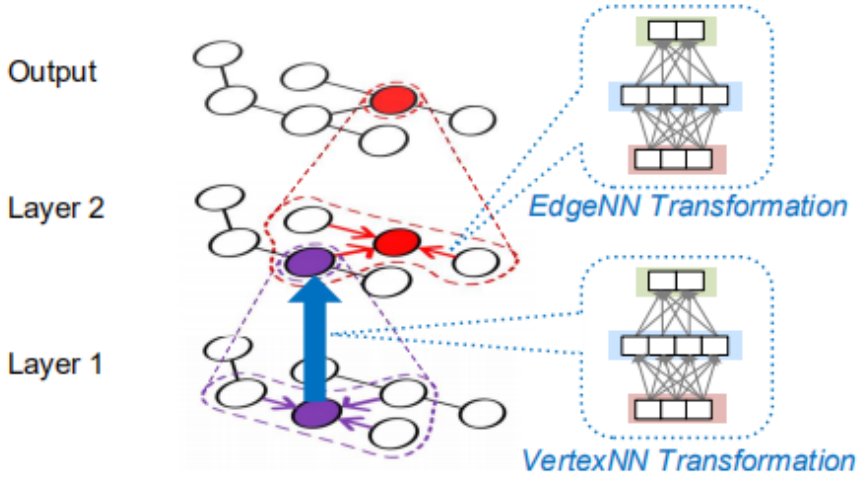


Figure 1: Feed-forward computation of a 2-layer GNN.

q2

流水线调度是如何根据统计执行时间来进行优化后续的迭代?

将vertex chunk分成k个sub-chunk，这能减少IO次数吗?

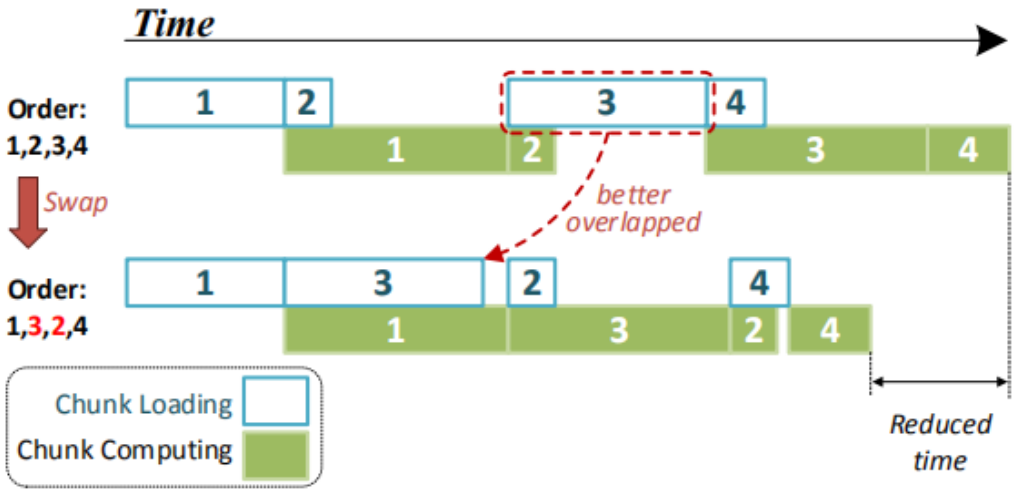


Figure 6: The swapping heuristic for a case of streaming two edge sub-chunks ($k = 2$).

Recall that different edge sub-chunks could have distinct data transfer and computation cost due to different sparsity levels. NeuGraph carefully makes a scheduling plan for streaming heterogeneous sub-chunks. Given a column of edge sub-chunks, the system first generates the initial schedule plan by assigning a random order for processing. Next, it repeatedly swaps the order of a pair sub-chunks such that a better schedule plan with less time can be obtained. This process stops when it converges or reaches maximum iterations.

Then, NeuGraph exploits the cyclic pattern inherent in GNNs: Both the computation time and data transfer time of each sub-chunk can be profiled in the first several iterations and used in refining the scheduling plan for processing in the following iterations. Specifically, the system simulates the

q3

多GPU执行过程不太理解?

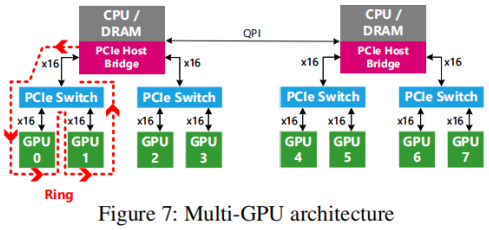


Figure 7: Multi-GPU architecture

dataflow subgraphs for different columns for cooperative processing.

However, with recent advances in hardware, modern multi-GPU systems introduce complex inter-connections among GPUs and across GPUs and CPUs, which presents new challenges to parallelize a dataflow graph. To illustrate this issue, Figure 7 shows the topology of a typical 8-GPU server, where GPUs are connected to CPU/DRAM (host memory) via a multi-level PCIe interface hierarchy. The upper level links that are shared by multiple communication paths can easily

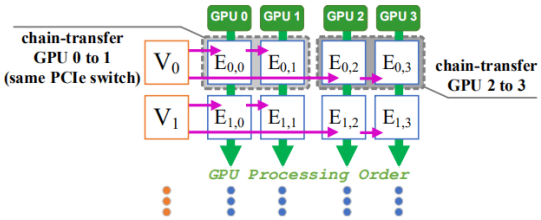


Figure 8: NeuGraph transfers vertex chunks along the chain.

and 2 start computing over chunk V_0, and also begin loading chunk V_1 from the host memory. Meanwhile, GPUs 1 and 3 start fetching chunk V_0 from GPUs 0 and 2, respectively. Next, GPUs 1 and 3 drop the data chunk V_0 after processing it locally as the chunk has already been consumed by all virtual GPUs. The whole process continues in such a pipelining fashion until all vertex data chunks have been loaded and processed.

AGL

q1

每个k-hop结构是围绕一个点生成的吗?

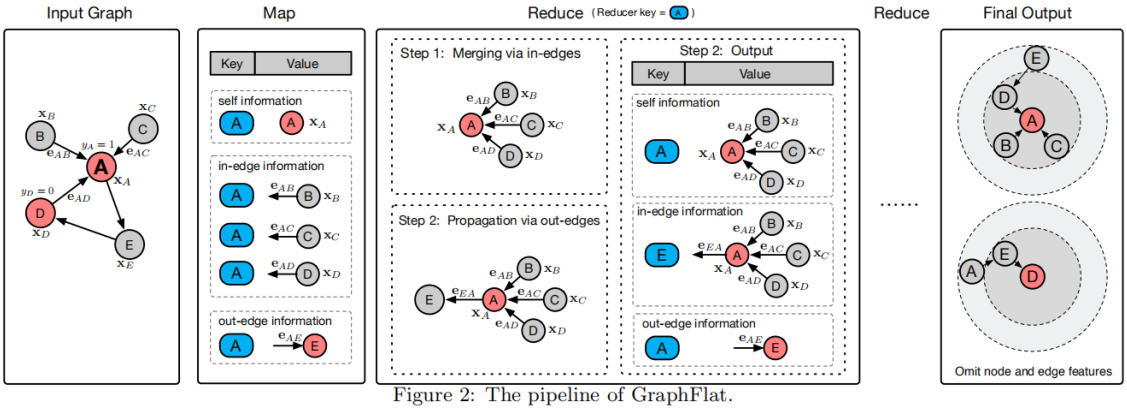


Figure 2: The pipeline of GraphFlat.

3.4 GraphInfer: distributed framework for GNN model inference

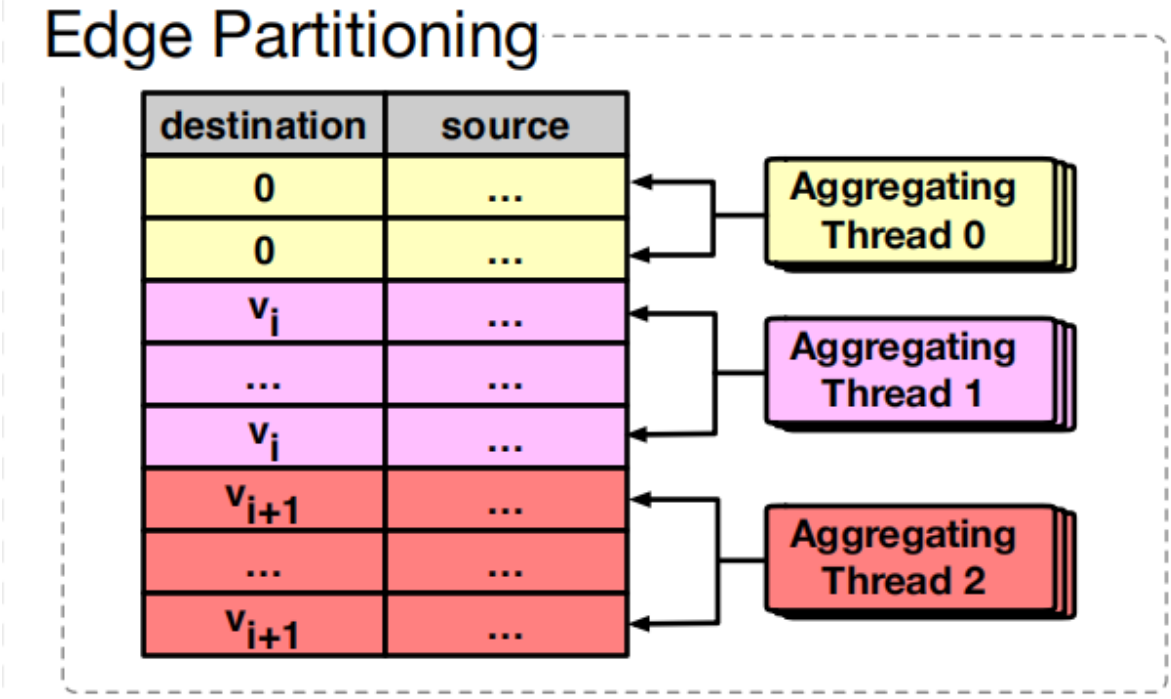
Performing GNN model inference over the industrial-scale graphs could be an intractable problem. On one hand, the data scale and use frequency of inference tasks could be quite higher than that of training tasks in industrial scenarios, which require a well-designed inference framework to boost the efficiency of inference tasks. On the other hand, since different k-hop neighborhoods described by *GraphFeatures* could overlap with each other, directly performing inference on *GraphFeatures* could lead to massive repetitions of embedding inference thus becomes time consuming.

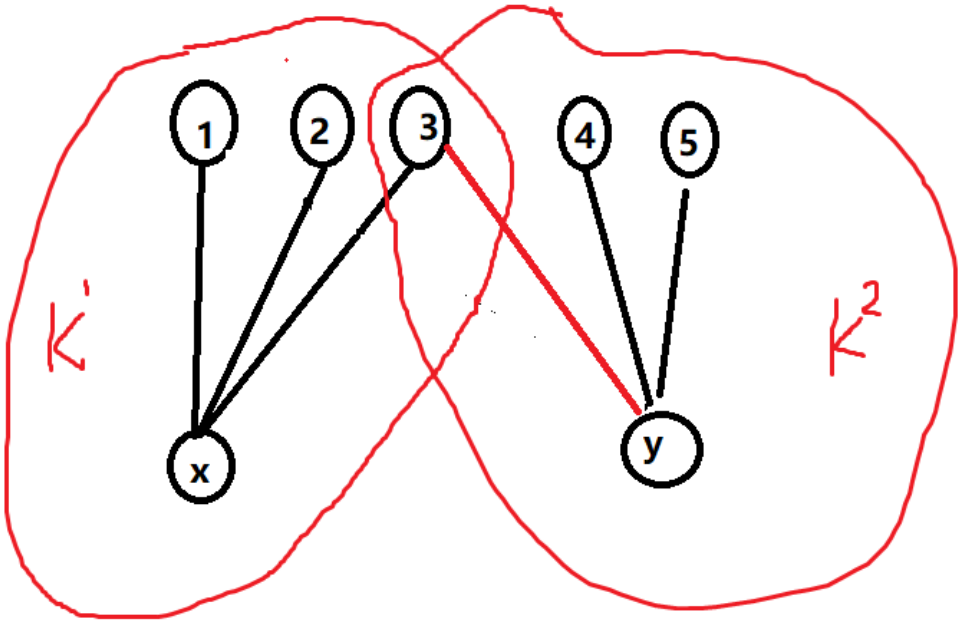
information as its new self information. Note that the new self information become the node's k-hop neighborhood. Next, the new self information is *propagated* to other destination nodes pointed along the out-edges, and is used to construct the new in-edge information w.r.t. the destination nodes. All of the out-edge information remain unchanged for the next *reduce* phase. At last, the reducer outputs the new data records, with the node ids and the updated information as the new *shuffle key* and *value* respectively, to the disk.

q2

edge partitioning将相同目标节点的边在同一个线程执行，这样彼此不冲突?

Tackling this problem, we propose an edge partitioning strategy to perform graph aggregation in parallel. The key insight is that a node only aggregates information along the edges pointing at it. If all edges with the same destination node can be handle with the same thread, the multi-thread aggregation could be very efficient since there will be no conflicts between any two threads. To achieve this goal, we partition the sparse adjacent matrix into t parts and ensure that the edges with the same destination node (i.e., the entries in the same row) fall in the same partition. The edge partitioning strategy is illustrated in the top of the middle part of Figure 4.





q3

处理hub时，在节点的key加后缀具体时如何实现的，是将一个点分成多个带后缀的点吗?

- **Re-indexing**. When the in-degree of a certain *shuffle key* (i.e., node id) exceeds a pre-defined threshold (like 10k), we will update *shuffle keys* by appending random suffixes, which is used to randomly partition the data records with the original *shuffle key* into smaller pieces.

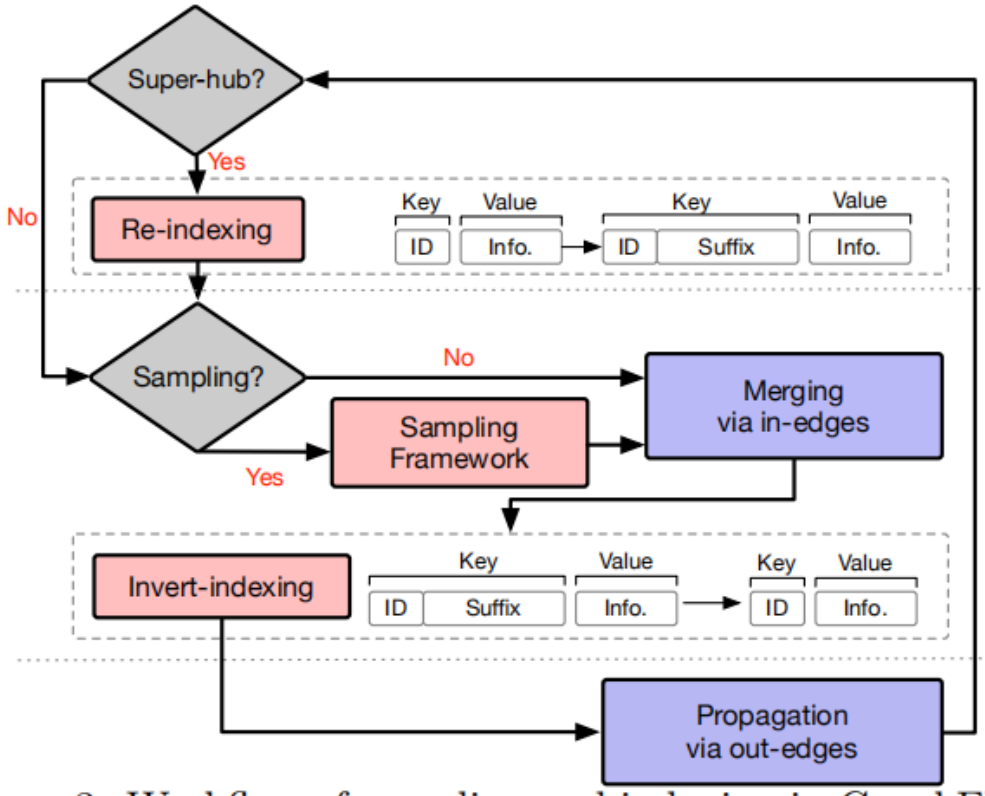
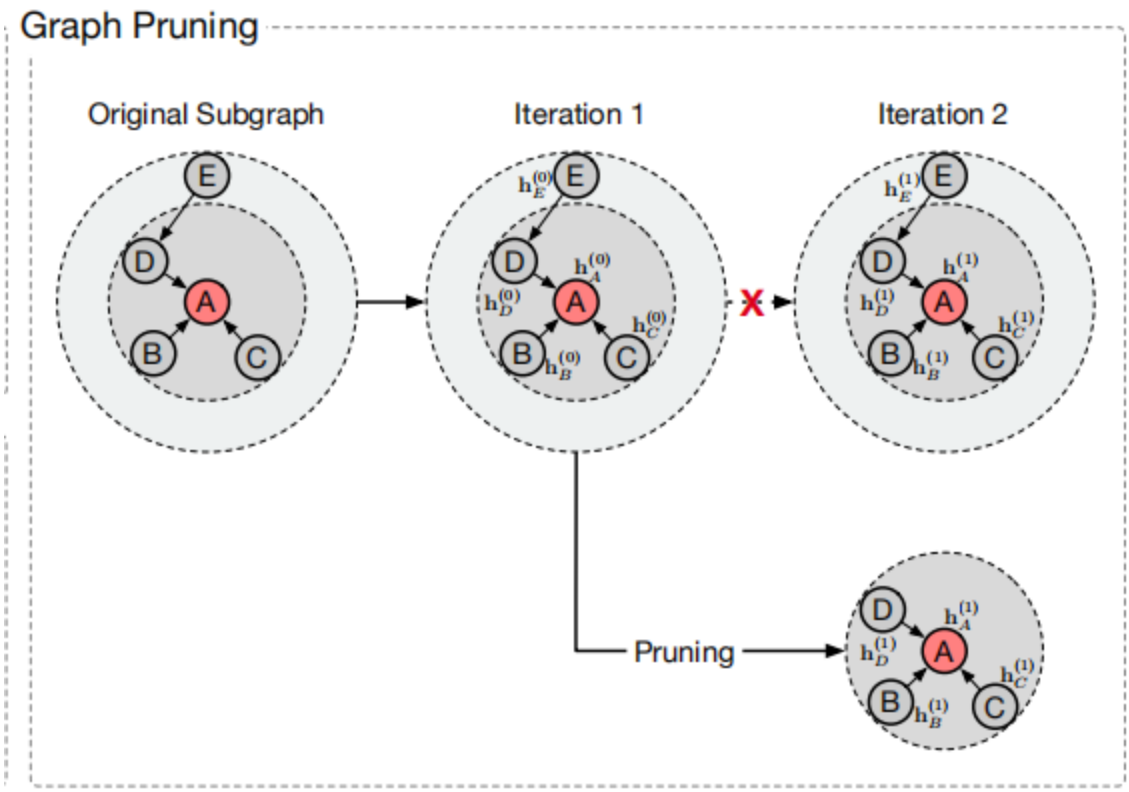


Figure 3: Workflow of sampling and indexing in GraphFlat.

q4

图的划分不太理解，每层只需要下一层的点，那每次执行完当前层，不是还要把下一层的结构加上，这样不会影响效率吗？



$$\mathbf{H}_{\mathcal{B}}^{(k+1)} = \Phi^{(k)}(\mathbf{H}_{\mathcal{B}}^{(k)}, \mathbf{A}_{\mathcal{B}}, \mathbf{E}_{\mathcal{B}}; \mathbf{W}_{\Phi}^{(k)}), \quad (2)$$

However, Equation 2 contains many unnecessary computations. On one hand, only the targeted nodes of $\mathcal{B}$ are labeled. Their embedding will be fed to the following part of the model. That means other embeddings in $\mathbf{H}_{\mathcal{B}}^{(K)}$ are unnecessary to the following part of the model. On the other hand, the three matrices $\mathbf{A}_{\mathcal{B}}$, $\mathbf{X}_{\mathcal{B}}$ and $\mathbf{E}_{\mathcal{B}}$ can provide sufficient and necessary information only for the targeted nodes. Thus other embeddings in $\mathbf{H}_{\mathcal{B}}^{(K)}$ could be generated incorrectly due to the lack of sufficient information.

Tackling this problem, we propose a *graph pruning* strat-

q5

GraphInfer将训练模型分为k+1块，为什么要这样做，这样做可以提高效率吗?

There is no repetitions of embedding inference in the above pipeline, which reduces the time cost in a great extent. Moreover, the pruning strategy similar to that in Graph-

3.4 GraphInfer: distributed framework for GNN model inference

Performing GNN model inference over the industrial-scale graphs could be an intractable problem. On one hand, the data scale and use frequency of inference tasks could be quite higher than that of training tasks in industrial scenarios, which require a well-designed inference framework to boost the efficiency of inference tasks. On the other hand, since different k-hop neighborhoods described by *GraphFeatures* could overlap with each other, directly performing inference on *GraphFeatures* could lead to massive repetitions of embedding inference thus becomes time consuming

Roc

q1

什么是有效地状态S?

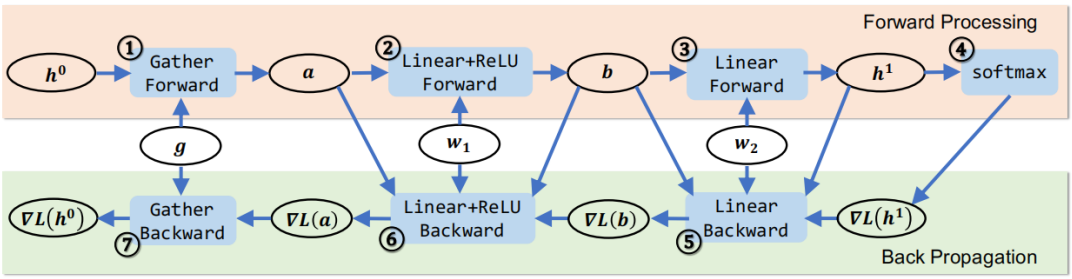


Figure 3. The computation graph of a toy 1-layer GIN architecture (Xu et al., 2019). A box represents an operation, and a circle represents a tensor. Arrows indicate dependencies between tensors and operations. The `gather` operation performs neighborhood aggregation. The `linear` and the following `ReLU` are fused into a single operation as a common optimization in existing frameworks. h^0 and g denote the input features and neighbors of all vertices, respectively. w_1 and w_2 are the weights of the two linear layers.

$\mathcal{G}$. A state is valid only if the operations it contains preserve all the data dependencies in $\mathcal{G}$, i.e., for any operation in $\mathcal{S}$, all its predecessor operations in $\mathcal{G}$ must be also in $\mathcal{S}$. Such

q2

不太理解line14?

```
5:  function COST(S, T)
6:      if (S, T) ∈ D then
7:          return D(S, T)
8:      if S is ∅ then
9:          return size(T)
10:     cost ← ∞
11:     for o_i ∈ S do
12:         if (S \ o_i) is a valid state then
13:             S' ← S \ o_i
14:             T' ← (T \ OUT(o_i)) ∩ A(S')
15:             xfer ← size(IN(o_i) \ T')
16:             if size(T ∪ IN(o_i) ∪ OUT(o_i)) ≤ cap then
17:                 cost = min{cost, COST(S', T') + xfer}
18:     D(S, T) ← cost
19:     return D(S, T)
```

q3

不太理解特征4和特征5?

The feature $x_3(v)$ is the number of consecutive blocks in v's neighbors, which is 3 in the example. In addition, $x_4(v)$ and $x_5(v)$ estimate the number of GPU memory accesses to load all neighbors and their input activations.

Table 2. The vertex features used in the current cost model. The semantics of the features are described in Section 4.1. `WS` is the number of GPU threads in a warp, which is 32 for the V100 GPUs used in the experiments.

	Definition	**Description**
x_1	1	the vertex itself
x_2	$\lvert\mathcal{N}(v)\rvert$	number of neighbors
x_3	$\lvert\mathcal{C}(v)\rvert$	continuity of neighbors
x_4	$\sum_i \lceil \frac{c_i(v)}{\texttt{WS}} \rceil$	# mem. accesses to load neighbors
x_5	$\sum_i \lceil \frac{c_i(v) \times d_{in}}{\texttt{WS}} \rceil$	# mem. accesses to load the activations of all neighbors

q4

动态分区，图推断为什么只用计算一次，图推断的过程是什么?

loads. *Dynamic* repartitioning (Venkataraman et al., 2013; Jia et al., 2017) exploits the iterative nature of many graph applications and rebalances the workload in each iteration based on the measured performance of previous iterations. This approach converges to a balanced workload distribution for GNN training, but is much less effective for inference which computes the GNN model only once for each new graph. ROC uses an online-linear-regression-based algorithm to achieve balanced partitioning for both GNN training and inference, through jointly learning a cost model to predict the execution time of the GNN model on arbitrary graphs.