IncAgg: Efficient Memory-Enhanced Graph Learning via Incremental Aggregation

Xingyue Shi[1] **Zhichao Hou**[1] **Jiahao Zhang**[2] **Suhang Wang**[2]
Tong Zhao[4] **Neil Shah**[3] **Xiaorui Liu**[1]

[1]North Carolina State University [2]Pennsylvania State University [3]Snap Inc. [4]Scale AI
{xshi22, zhou4, xliu96}@ncsu.edu, {jiahao.zhang, szw494}@psu.edu,
tong.zhao@scale.com, nshah@snap.com

Abstract

Scaling Graph Neural Network (GNN) training to large graphs commonly relies on mini-batch sampling, but incomplete neighborhoods can degrade approximation quality and training stability. Memory-enhanced methods mitigate this issue by reusing historical node embeddings to approximate full-neighborhood aggregation; however, they reconstruct pseudo full-neighborhood messages inside every training iteration, repeatedly fetching out-of-batch memories and aggregating over large neighborhoods. In this work, we identify this under-examined inner-loop bottleneck and propose IncAgg, an efficient memory-enhanced GNN training framework based on incremental aggregation. IncAgg periodically pre-aggregates historical embeddings into a global memory baseline, and during mini-batch training propagates only the current in-batch residual before fusing it with the stored global context. This design preserves historical full-neighborhood information at a configurable refresh frequency while avoiding per-iteration out-of-batch memory fetching and aggregation, without modifying the underlying GNN architecture. We provide theoretical analyses showing that IncAgg reduces gradient-carrying aggregation and memory-access costs while retaining the residual-based stability principle of memory-enhanced estimators. Extensive experiments on four large-scale benchmarks and four representative GNN backbones show that IncAgg matches or improves the accuracy of strong memory-enhanced baselines while achieving substantial training acceleration, with up to $18.6\times$ speedup in individual settings and up to $13.8\times$ geometric-mean speedup on large graphs.

1 Introduction

Graph Neural Networks (GNNs) have achieved remarkable success across a wide range of graph learning tasks, including node classification [8], graph classification [16], and link prediction [19], and have been widely studied and applied in numerous applications. Despite their effectiveness, scaling GNN training to large graphs and deep models remains challenging because full-neighborhood message passing requires repeatedly aggregating information over neighborhoods that grow exponentially with model depth. A common remedy is sampling-based mini-batch training, which limits the aggregation scope from the node-, layer-, or subgraph-wise sampling [5, 9, 14], but at the cost of discarding out-of-batch neighbor information, substantial approximation error, and unstable training.

Memory-enhanced methods [2, 7, 14, 15, 17] offer a promising path to reduce sampling variance and approximation error by storing and reusing historical node embeddings to approximate full-neighborhood aggregation during mini-batch training. Although effective at preserving accuracy and stabilizing training, existing memory-enhanced methods still face a significant efficiency bottleneck, one that, to our knowledge, has not been systematically characterized. We provide runtime profiling that reveals the source of this overhead: at each iteration, historical embeddings of out-of-batch

40th Conference on Neural Information Processing Systems (NeurIPS 2026).

neighbors are explicitly fetched from the memory cache and aggregated together with current in-batch embeddings to reconstruct pseudo full-neighborhood aggregation. This keeps expensive full-neighborhood aggregation and out-of-batch memory access inside the inner training loop, incurring substantial computation and data movement overhead at every mini-batch iteration.

This inefficiency points to an open question: *can we preserve the benefits of full-neighborhood context without explicitly fetching and aggregating out-of-batch histories at every iteration?* Our key insight is that the historical neighborhood contribution can be pre-aggregated into a global memory baseline and reused across iterations, while only the incremental changes introduced by the current batch need to be propagated at each training iteration. Based on this novel insight, we propose IncAgg, an efficient and versatile GNN training framework built on an innovative incremental aggregation mechanism. Specifically, IncAgg periodically refreshes a snapshot of node embeddings and computes a global memory baseline via full-neighborhood propagation over this snapshot. During mini-batch training, rather than reconstructing pseudo full-neighborhood aggregation from neighbor histories, IncAgg computes the embedding residual of the current batch, incrementally propagates it within the batch, and fuses it with the cached global baseline. This replaces repeated neighbor-wise memory fetching and aggregation with a cached baseline plus a lightweight in-batch incremental aggregation.

The proposed design directly eliminates the core inefficiency of prior memory-enhanced methods. By reusing pre-aggregated historical context rather than repeatedly materializing it from node-level histories, IncAgg removes per-iteration out-of-batch aggregation and substantially reduces memory access. The periodically refreshed global baseline meanwhile preserves full-neighborhood context, enabling stable and accurate training despite a much smaller iteration-level aggregation scope. IncAgg requires no architectural changes and is compatible with diverse GNN backbones.

To summarize, this work makes four key contributions. First, we identify a fundamental inefficiency in memory-enhanced GNN training: existing methods repeatedly fetch and aggregate out-of-batch histories at every iteration, incurring substantial computation and data movement overhead. Second, we propose IncAgg, a general incremental aggregation framework that incorporates full-neighborhood context while eliminating per-iteration out-of-batch memory fetching and aggregation, and it can be applied to diverse GNN backbones without architectural modifications. Third, we provide theoretical insights characterizing how stored global context and in-batch incremental aggregation together reduce training-path aggregation and memory-access complexity while preserving the residual-control principle of memory-enhanced estimators. Finally, extensive experiments across four representative datasets and four GNN backbones demonstrate that IncAgg achieves over $10\times$ training speedup over existing memory-enhanced baselines, while matching or improving accuracy.

2 Background and Related Work

Consider a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ with node set $\mathcal{V}$ and edge set $\mathcal{E}$. Let $\mathbf{P} \in \mathbb{R}^{|\mathcal{V}|\times|\mathcal{V}|}$ denote the normalized propagation matrix. We denote the neighborhood of node u by $\mathcal{N}(u)$ and a mini-batch of target nodes by $\mathcal{B} \subset \mathcal{V}$. Node embeddings at layer k are denoted by $\mathbf{H}^{(k)} \in \mathbb{R}^{|\mathcal{V}|\times d_k}$. We use set notation for neighborhood aggregation (e.g., $\mathcal{N}(u)$) and matrix notation (e.g., $\mathbf{P}$, $\mathbf{H}$) for batch-level expressions.

In GNNs, node embeddings are updated by aggregating their neighborhoods' embedding as $\mathbf{h}_u^{(k+1)} = AGG(\{\mathbf{h}_v^{(k)} \mid v \in \mathcal{N}(u)\})$, where $\mathbf{h}_v^{(k)}$ is the embedding of neighbor v at layer k and AGG denotes the aggregation function. The neighbor aggregation enables expressive propagation of structural information in the neighborhood [11]. As graph size and model depth increase, the number of involved neighbors grows rapidly, making neighborhood aggregation computationally prohibitive [8].

To scale GNN training to large graphs, mini-batch sampling methods restrict computation to a subset of target nodes and their sampled neighbors [9, 18]. Existing approaches span node-wise sampling [9, 12, 20], layer-wise sampling [3, 21], and subgraph-based training [5, 7, 14, 18]. By limiting aggregation to sampled neighbors, these methods alleviate the neighbor explosion problem and improve scalability but at the cost of incomplete neighborhoods, introducing high estimator variance/bias and degrade aggregation accuracy and training stability [2].

To mitigate the information loss induced by sampling, memory-enhanced methods maintain historical node embeddings to approximate missing out-of-batch information [2, 7, 14, 15, 17]. Their aggregation follows a pseudo full-neighborhood aggregation: $\mathbf{h}_u^{(k+1)} = AGG\{\mathbf{h}_v^{(k)}\}_{v\in\mathcal{N}(u)\cap\mathcal{B}} \cup \{\mathbf{m}_v^{(k)}\}_{v\in\mathcal{N}(u)\setminus\mathcal{B}}$, where $\mathcal{B}$ denotes the nodes in the sampled batch, $\mathcal{N}(u) \cap \mathcal{B}$ and $\mathcal{N}(u) \setminus \mathcal{B}$ denote

the in-batch and out-of-batch neighborhood of node u respectively. The stored historical embedding $\mathbf{m}_v^{(k)}$ will be refreshed after each iteration by pushing the current batch embeddings back to memory.

These memory-enhanced methods significantly reduce sampling variance and stabilize training by approximately recovering full-neighborhood context without recomputing all neighbor representations from scratch. However, our runtime profiling reveals the hidden overhead: each iteration must fetch out-of-batch memories and perform aggregation over them, introducing substantial redundant computation and data movement (see Section 3.1) The goal of this work is to eliminate this bottleneck: preserving the variance-reduction benefits of memory-enhanced aggregation while restoring the efficiency that sampling-based training promises.

3 Incremental Aggregation for Efficient Graph Learning

We begin with a motivating runtime profiling that exposes the efficiency bottleneck in memory-enhanced GNN training. We then describe the algorithm design and understanding of the proposed algorithm. We finally present variance and efficiency analyses to establish its theoretical advantages.

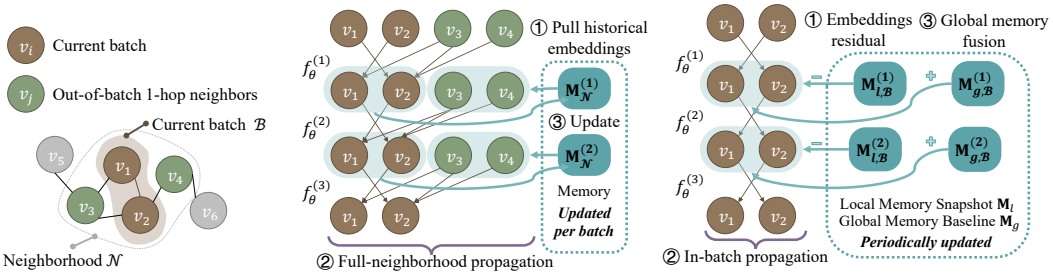
3.1 Motivation with Runtime Profiling

To complement Section 2, we profile GNNAutoScale (GAS) [7] using a GCN backbone on the Reddit dataset (batch size 5%) and report the runtime of major components within one training epoch. As memory transfers execute asynchronously, the reported values reflect accumulated transfer time, with part of the memory access cost already hidden from the main computation stream.

Table 1: Runtime breakdown of one training epoch for GNNAutoScale (GAS); times are accumulated over a complete epoch. Qualitatively similar bottlenecks are observed across other memory-enhanced GNN training methods.

Major Component	Time (ms)
Memory movement	1673.7
Aggregation (forward)	837.3
Linear transformation (forward)	689.2
Aggregation (backward)	776.7
Linear transformation (backward)	918.5

The results in Table 1 reveal two dominant sources of overhead. Even with asynchronous overlap, memory movement alone exposes a substantial data transfer bottleneck. Aggregation operations (forward and backward) further dominate the remaining computation cost, reflecting the burden of per-iteration pseudo full-neighborhood aggregation. Together, these findings empirically confirm that memory-enhanced training incurs significant computation and data movement overhead, motivating the design of incremental aggregation.



(a) Mini-batch training with partitioned subgraphs

(b) Full-neighborhood propagation with historical embeddings

(c) IncAgg: In-batch-only propagation with incremental aggregation

Figure 1: Overview. (a) Mini-batch training on partitioned subgraphs leads to missing out-of-batch neighbors. (b) Memory-enhanced methods like GAS [7] and GraphFM [17] address this by pulling historical embeddings of out-of-batch nodes and aggregating over full neighborhoods, incurring heavy data movement. (c) IncAgg avoids out-of-batch access by propagating the embedding delta within the batch and fusing periodically updated global memory to restore full-neighborhood context.

3.2 IncAgg: Incremental Aggregation

Figure 1 illustrates the core distinction between vanilla sampling, memory-enhanced methods, and IncAgg. As shown in Figure 1 (a), mini-batch training reduces computation but excludes out-of-batch neighbors in message passing. Memory-enhanced methods address this by pulling historical embeddings of missing neighbors and performing pseudo full-neighborhood aggregation at each layer, as shown in Figure 1 (b), restoring neighborhood context at the cost of tightly coupling every training iteration with out-of-batch memory access and full-neighborhood propagation.

The proposed IncAgg takes a different approach. Rather than reconstructing full-neighborhood aggregation from historical neighbor embeddings at every iteration, IncAgg maintains a periodically refreshed global baseline and propagates only the fresh embedding residual within the current batch (Figure 1 (c)). This recasts training-time message passing as an incremental update to a stored global context, eliminating per-iteration out-of-batch aggregation and reducing memory movement without any changes to the underlying GNN architecture. We describe the core components in IncAgg below.

Periodic Memory Refresh. IncAgg maintains two layer-wise memory states. The *local memory snapshot* $\mathbf{M}_L^{(k)}$ stores the most recently refreshed embeddings of all nodes at layer k. Based on this snapshot, the *global memory baseline* $\mathbf{M}_G^{(k)}$ is computed by full-neighborhood propagation: $\mathbf{M}_G^{(k)} = \mathbf{P}\mathbf{M}_L^{(k)}$, capturing the aggregated neighborhood context at the time of the last refresh. Unlike memory-enhanced methods that update historical embeddings after every batch, IncAgg refreshes $\mathbf{M}_L^{(k)}$ and $\mathbf{M}_G^{(k)}$ only periodically at a configurable frequency (e.g., once per epoch). This moves expensive full-neighborhood propagation out of the inner training loop. During mini-batch training, IncAgg only accesses the node embedding stored in memory corresponding to the current batch, rather than fetching historical embeddings for all out-of-batch neighbors. Both memories can be stored in CPU RAM, keeping per-iteration memory traffic small while maintaining scalable storage.

Incremental In-Batch Aggregation. Given a mini-batch $\mathcal{B}$, IncAgg performs message passing through three steps (Figure 1 (c)): (1) Embedding Residual: For the current batch embeddings $\mathbf{H}_\mathcal{B}^{(k)}$, IncAgg computes their deviation from the local memory snapshot: $\Delta\mathbf{H}_\mathcal{B}^{(k)} = \mathbf{H}_\mathcal{B}^{(k)} - \mathbf{M}_{L,\mathcal{B}}^{(k)}$. This residual captures the fresh information introduced by the current iteration relative to the last memory refresh in $\mathbf{M}_L^{(k)}$. (2) In-batch Propagation: Rather than aggregating over the full neighborhood, IncAgg propagates the residual within the batch only: $\mathbf{R}_\mathcal{B}^{(k)} = \mathbf{P}_{\mathcal{B},\mathcal{B}}\Delta\mathbf{H}_\mathcal{B}^{(k)}$, where $\mathbf{P}_{\mathcal{B},\mathcal{B}}$ is the in-batch propagation matrix. This confines iteration-level aggregation to the batch, avoiding any fetching of out-of-batch historical embeddings. (3) Global Memory Fusion: IncAgg fuses the propagated residual with the global memory baseline: $\widetilde{\mathbf{H}}_\mathcal{B}^{(k+1)} = \mathbf{R}_\mathcal{B}^{(k)} + \mathbf{M}_{G,\mathcal{B}}^{(k)}$. The resulting representation is then passed to the corresponding layer transformation $f_\theta^{(k)}$ to produce the next-layer embeddings.

3.3 IncAgg as Efficient Memory-Enhanced Aggregation

The proposed IncAgg can be viewed as an efficient formulation of memory-enhanced aggregation. Let's denote $\mathcal{B}$ and $\bar{\mathcal{B}}$ as in-batch nodes and out-of-batch neighbors respectively, $\widetilde{\mathbf{H}}^{(k)}$ as the mixed embedding matrix, where $\widetilde{\mathbf{H}}_\mathcal{B}^{(k)} = \mathbf{H}_\mathcal{B}^{(k)}$ (current batch) and $\widetilde{\mathbf{H}}_{\bar{\mathcal{B}}}^{(k)} = \mathbf{M}_{L,\bar{\mathcal{B}}}^{(k)}$ (historical local embedding), and $\mathbf{P}_{\mathcal{R},\mathcal{C}}$ as the submatrix of $\mathbf{P}$ with rows and columns restricted to $\mathcal{R}$ and $\mathcal{C}$ respectively. The pseudo full-neighborhood propagation for batch $\mathcal{B}$ can be reformulated as incremental aggregation:

$$\mathbf{P}_{\mathcal{B},\mathcal{B}\cup\bar{\mathcal{B}}}\widetilde{\mathbf{H}}_{\mathcal{B}\cup\bar{\mathcal{B}}}^{(k)} = \mathbf{P}_{\mathcal{B},\mathcal{B}\cup\bar{\mathcal{B}}}\mathbf{M}_{L,\mathcal{B}\cup\bar{\mathcal{B}}}^{(k)} + \mathbf{P}_{\mathcal{B},\mathcal{B}\cup\bar{\mathcal{B}}}\big(\widetilde{\mathbf{H}}_{\mathcal{B}\cup\bar{\mathcal{B}}}^{(k)} - \mathbf{M}_{L,\mathcal{B}\cup\bar{\mathcal{B}}}^{(k)}\big) = \mathbf{M}_{G,\mathcal{B}}^{(k)} + \mathbf{P}_{\mathcal{B},\mathcal{B}}(\widetilde{\mathbf{H}}_\mathcal{B}^{(k)} - \mathbf{M}_{L,\mathcal{B}}^{(k)}).$$

The first equation is a simple reformulation of pseudo aggregation, and the second equation recasts the aggregation over all neighbors $\mathcal{B}\cup\bar{\mathcal{B}}$ into only in-batch nodes $\mathcal{B}$. This equivalence is due to zero residual for the out-of-batch neighbors: $\widetilde{\mathbf{H}}_{\bar{\mathcal{B}}}^{(k)} - \mathbf{M}_{L,\bar{\mathcal{B}}}^{(k)} = \mathbf{0}$. The historical out-of-batch information is not discarded but captured in the cached global memory baseline through periodic memory refresh:

$$\mathbf{M}_{G,\mathcal{B}}^{(k)} = \mathbf{P}_{\mathcal{B},\mathcal{B}\cup\bar{\mathcal{B}}}\mathbf{M}_{L,\mathcal{B}\cup\bar{\mathcal{B}}}^{(k)} = \mathbf{P}_{\mathcal{B},\mathcal{B}}\mathbf{M}_{L,\mathcal{B}}^{(k)} + \mathbf{P}_{\mathcal{B},\bar{\mathcal{B}}}\mathbf{M}_{L,\bar{\mathcal{B}}}^{(k)},$$

which caches full-neighborhood aggregation over historical embeddings at a configurable frequency outside the inner mini-batch training loop. As a result, IncAgg avoids explicitly fetching and aggregating historical embeddings of out-of-batch neighbors at every iteration; instead, it reuses the global memory baseline and updates it incrementally through in-batch residual propagation.

Scope. The identity above requires the propagation to be a fixed-coefficient linear map $\mathbf{PH}$, where $\mathbf{P}$ depends only on the graph structure and normalization rather than on current representations. This covers the GCN, APPNP, GCNII, and mean-aggregation GraphSAGE configurations in our experiments; feature transformations, activations, and residual connections may remain nonlinear. Attention-based GNNs such as GAT, whose coefficients depend on current representations, and non-additive aggregators such as max/min pooling fall outside this exact identity.

3.4 Efficiency and Variance Analysis

Training-path cost and memory refresh. Let $n := |\mathcal{V}|$, $m := |\mathcal{E}|$, K be the number of GNN layers, and $b := |\mathcal{B}|$ be the batch size. IncAgg separates GNN training into a differentiable mini-batch training path and a forward-only refresh of the global memory $\mathbf{M}_G = \mathbf{P}\mathbf{M}_L$. During training, IncAgg backpropagates through aggregation only on the in-batch subgraph. If $\mathcal{E}[\mathcal{B}_t]$ denotes the edges whose two endpoints both lie in batch $\mathcal{B}_t$, the per-epoch training-path aggregation cost is $O(K \sum_t |\mathcal{E}[\mathcal{B}_t]|)$. Under uniform random partitioning, each edge is retained as an in-batch edge with probability $O(b/n)$ over one epoch, yielding an expected cost of $O(Kmb/n)$. In contrast, GAS [7] performs pseudo full-neighborhood aggregation along the mini-batch training path, with $O(Km)$ aggregation cost per epoch. Thus, IncAgg reduces the repeated gradient-carrying aggregation cost by $\Theta(n/b)$ in expectation under random partitioning. For any realized partition, the training-path cost is exactly $O(K \sum_t |\mathcal{E}[\mathcal{B}_t]|)$, so degree and community skew enter through the actual in-batch edges and the cut ratio, and the realized reduction grows with the out-of-batch edge ratio (Tables 8 and 9).

This reduction is a training-path cost reduction rather than the elimination of full-graph propagation. If $\mathbf{M}_g$ is refreshed every τ epochs, the forward-only refresh contributes an amortized cost of $O(Km/\tau)$, giving total edge-touch complexity $O(Kmb/n + Km/\tau)$. When $\tau = 1$, IncAgg has the same asymptotic edge-touch order as GAS, but the full-neighborhood operation is moved outside the gradient-carrying iteration and reused across batches. In our implementation, this refresh is integrated with the routine validation pass, avoiding a standalone full-neighborhood propagation pass.

Memory footprint and data movement. Both IncAgg and GAS maintain a full-graph historical embedding table $\mathbf{M}_L \in \mathbb{R}^{n \times d}$ in host memory, while IncAgg additionally stores $\mathbf{M}_G \in \mathbb{R}^{n \times d}$. This introduces only a constant-factor increase in host memory. For per-iteration host–GPU transfer, GAS pushes the newly computed in-batch embeddings to the history table and pulls historical embeddings for out-of-batch neighbors required by pseudo full-neighborhood aggregation. Let q_t be the number of such out-of-batch nodes for batch $\mathcal{B}_t$ and $\rho := \mathbb{E}[q_t]/b$. The corresponding transfer scales as $O((b + q_t)d) = O((1 + \rho)bd)$ in expectation. By contrast, IncAgg only accesses the in-batch slices $\mathbf{M}_{L,\mathcal{B}_t}$ and $\mathbf{M}_{G,\mathcal{B}_t}$, resulting in $O(bd)$ per-iteration memory access. Therefore, IncAgg removes the out-of-batch history pulls required by GAS, with larger savings when ρ is large.

Estimator stability. Although IncAgg restricts training-time aggregation to in-batch neighborhoods, its estimator follows the same residual-control principle as memory-enhanced methods [2, 7]. GAS replaces out-of-batch neighbor embeddings with historical embeddings, making the stochastic component depend on residuals between current and historical embeddings. Appendix A.1 provides the full derivation, showing that IncAgg has the same residual-based estimator variance form as GAS and avoids the rescaling-induced variance amplification of VR-GCN.

4 Experiments

We evaluate IncAgg from five complementary perspectives. First, we assess node classification accuracy under matched backbone settings to verify that the proposed efficiency improvements do not sacrifice predictive performance. Second, we measure training efficiency at both epoch level and operator level, attributing the observed speedups to reduced aggregation scope and memory movement. Third, we examine convergence behavior and GPU memory consumption to confirm that the wall-clock gains are not offset by slower optimization or increased device-memory usage. Fourth, we conduct sensitivity studies over batch size, partitioning strategy, and memory-refresh frequency to evaluate robustness and approximation quality under different training regimes. Finally, we compare IncAgg with representative memory-enhanced methods to contextualize its role as an efficiency-oriented alternative to approaches that primarily refine historical embeddings.

4.1 Experimental Setup

We evaluate IncAgg on four node classification benchmarks with diverse scales and densities: ogbn-arxiv [10], ogbn-products [10], Reddit [9], and Amazon-products [18]. Dataset statistics are summarized in Table 2. To assess its generality, we apply IncAgg to four representative GNN backbones: GCN [11], APPNP [8], GCNII [4], and SAGE [9]. Our primary comparison is against GAS [7], a commonly used memory-enhanced framework, with matched backbone and settings.

Table 2: Datasets. Deg. is average degree.

Dataset	Nodes	Edges	Deg.
ogbn-arxiv	169k	1.16M	**6.9**
ogbn-products	2.4M	61.9M	**25.8**
Amazon	1.6M	264.3M	**165.2**
Reddit	233k	23.2M	**99.6**

Table 3: Node classification accuracy (%). Results are averaged over 5 runs with standard deviation.

Framework	Backbone / Method	Arxiv	Products	Amazon	Reddit
-	GraphSAGE [9]	71.5±0.3	78.3±0.2	75.8±0.2	95.3±0.1
	GraphSAINT [18]	70.2±0.2	79.0±0.2	78.2±0.4	96.6±0.1
	Cluster-GCN [5]	65.3±0.2	79.0±0.3	75.9±0.8	95.4±0.1
VR-GCN [2]	GCN	72.2±0.1	69.9±0.2	75.9±0.3	94.5±0.2
GAS [7]	GCN	71.5±0.3	75.3±0.2	76.7±0.2	95.3±0.1
	APPNP	71.1±0.1	78.5±0.2	72.0±0.4	94.6±0.1
	GCNII	72.8±0.5	76.2±0.5	74.4±0.2	95.2±0.2
	SAGE	72.1±0.1	77.6±0.1	72.5±0.3	96.2±0.1
IncAgg (ours)	**GCN**	71.4±0.5	75.6±0.1	77.1±0.2	95.3±0.2
	APPNP	70.8±0.3	78.5±0.2	71.9±0.4	94.5±0.2
	GCNII	72.6±0.5	75.9±0.4	74.5±0.3	96.7±0.3
	SAGE	72.0±0.2	77.3±0.3	72.4±0.5	96.4±0.2

Table 4: Epoch time (ms) of baselines and IncAgg under best-performance settings. Each dataset block shows baseline time, IncAgg time, and the speedup factor (baseline / IncAgg). Last row reports the average (GeoMean) speedup per dataset across all backbones.

	Arxiv			Products			Amazon			Reddit		
Backbone	**GAS**	**IncAgg**	×	**GAS**	**IncAgg**	×	**GAS**	**IncAgg**	×	**GAS**	**IncAgg**	×
GCN	810	324	2.5	19 236	1430	13.5	12 694	1222	10.4	6667	727	9.2
APPNP	760	282	2.7	17 466	941	18.6	15 165	1283	11.8	3459	372	9.3
GCNII	1284	557	2.3	21 767	2204	9.9	13 617	1720	7.9	9761	2000	4.9
SAGE	874	372	2.4	18 448	1397	13.2	12 018	1153	10.4	3684	485	7.6
Avg.			**2.5**			**13.8**			**10.1**			**7.6**

4.2 Classification Accuracy

Table 3 reports node classification accuracy on four datasets. We include classic GNN baselines and representative memory-based models for reference. To enable a controlled comparison, we perform one-to-one evaluations between IncAgg and its GAS-trained counterparts using identical backbones. This comparison isolates the effect of the proposed training framework from architectural backbone differences. Across all datasets and backbones, IncAgg consistently matches or slightly improves upon the accuracy of GAS and achieves competitive performance among all. These results verify that the efficiency gains of IncAgg do not come at the cost of predictive performance. Small differences from GAS arise because, although the residual identity is exact for a fixed memory snapshot, the two methods update memory at different frequencies. GAS updates visited histories after every mini-batch, whereas IncAgg keeps its memories fixed between global refreshes, which leads to different staleness and selected models. Additional baselines are provided in Appendix D to contextualize IncAgg among recent memory-enhanced GNN methods, while our main comparison remains the matched GAS–IncAgg setting in Table 3.

4.3 Efficiency Analysis

Overall Training Speedup. Table 4 summarizes the epoch time reduction across different backbones and datasets, demonstrating that IncAgg consistently achieves substantial training acceleration in practice. Across all backbones, IncAgg attains average speedups of up to $13.8\times$, $10.1\times$, $7.6\times$, and $2.5\times$ on ogbn-products, Amazon, Reddit, and ogbn-arxiv, respectively. These results demonstrate that the efficiency gains of IncAgg are robust across both model architectures and graph scales.

The magnitude of speedup further depends on dataset characteristics. For example, ogbn-products, which contains a large number of nodes and edges, benefits the most from reduced aggregation and memory access, achieving speedups up to $13.8\times$. Meanwhile, Amazon-products, despite having fewer nodes, is relatively dense, and thus still exhibits notable speedups (up to $10.1\times$) due to the

Table 5: Main operations in an iteration of a GCN layer under GAS and IncAgg frameworks on Reddit. IncAgg restricts the scope from the full neighborhood (*full*) to the in-batch subgraph (*IB*).

GCN Operation		**Measured Runtime**			**Theoretical Analysis**
Component	**Op.**	**GAS (ms)**	**IncAgg (ms)**	**Speedup**	**Reduction**
GCNConv.lin (fwd)	GEMM	12.40	1.29	9.61	$N_{full}/N_{IB} \approx 17$
GCNConv.agg (fwd)	SpMM	11.81	1.56	7.57	$\text{nnz}_{full}/\text{nnz}_{IB} \approx 17$
GCNConv.agg (bwd)	SpMM	12.95	1.87	6.92	$\text{nnz}_{full}/\text{nnz}_{IB} \approx 17$
GCNConv.lin (bwd)	GEMM	15.31	1.68	9.11	$N_{full}/N_{IB} \approx 17$

substantial reduction in aggregation scope. Finally, we observe that simpler backbones such as GCN generally achieve higher speedups compared to more complex architectures. This is because aggregation constitutes a dominant portion of their training cost, and IncAgg directly targets this bottleneck by limiting aggregation scope and memory operations.

Time Consumption Breakdown. To account for the observed overall training speedup, we further break down the epoch time into fine-grained components during forward and backward propagation. As shown in Table 5, the dominant matrix multiplication operations in a GCN layer, including neighborhood aggregation (SpMM) and linear transformations (GEMM), are significantly accelerated under IncAgg compared to the baseline. By restricting aggregation from the full 1-hop neighborhood to the current batch, IncAgg substantially reduces the effective computation scope of these operations, leading to notable runtime savings. The measured speedups closely follow the theoretical reduction ratios derived from the relative sizes of the full and in-batch computation scopes. Although the empirical speedups are slightly lower than the theoretical estimates, this gap can be attributed to factors such as non-ideal GPU kernel scaling, among others. Overall, this fine-grained analysis confirms that reducing the computation scope of matrix multiplication operations is a primary contributor to the training acceleration of IncAgg.

Efficiency–Accuracy Trade-off. Figure 2 shows a two-dimensional comparison between accuracy and epoch time across batch sizes and partitioning strategies, where each arrow indicates the transition from the baseline to IncAgg under the same setting. IncAgg consistently shifts the accuracy–time trade-off toward the upper-left region, achieving substantial reductions in epoch time while maintaining comparable or even slightly improved accuracy. This trend holds across batch sizes and partitioning choices. It demonstrates that restricting aggregation to the in-batch subgraph does not impair model quality when combined with the proposed incremental aggregation, but instead enables a more favorable efficiency–accuracy balance.

Notably, IncAgg performs particularly well under extreme settings, such as random partitioning with very small batch sizes. In these scenarios, the baseline suffers from noticeable efficiency loss and accuracy degradation, whereas IncAgg maintains stable accuracy while achieving the largest relative speedups. This is consistent with our theoretical analysis, as smaller batch sizes and more aggressive partitioning increase the proportion of out-of-batch neighbors, thereby amplifying the benefits of reduced aggregation scope in IncAgg.

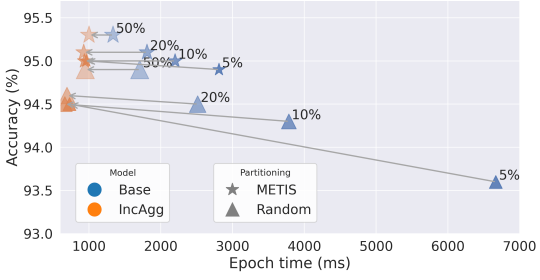


Figure 2: Efficiency–accuracy trade-off on Reddit dataset. "Base" denotes GAS.

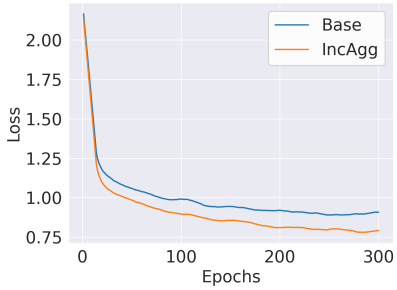


Figure 3: Training loss curves with GCN on ogbn-arxiv. "Base" is GAS.

Convergence Behavior. To verify the substantial efficiency gains of IncAgg does not come at the cost of slower or less stable optimization, we first compare its training loss curve with the GAS baseline using the same backbone GCN with matched settings on ogbn-arxiv. As shown in Figure 3, IncAgg follows a comparable convergence trajectory and reaches a similar final loss, suggesting that limiting aggregation to the current batch with periodically refreshed global memory does not destabilize training. We further evaluate time-to-accuracy by measuring how many epochs IncAgg needs to match the best accuracy achieved by GAS within 300 epochs. Table 6 reports the results on Reddit with random partitioning, averaged over 5 runs. IncAgg reaches the GAS target accuracy within substantially fewer than 300 epochs in all settings, and the required epochs decrease as the batch size becomes smaller. This indicates that the wall-clock gains of IncAgg are not offset by requiring more training epochs; instead, its per-epoch speedups in Table 8 translate into clear end-to-end training acceleration.

$\lvert\mathcal{B}\rvert/\lvert\mathcal{V}\rvert$	**Epochs**
50%	245.2
20%	67.3
10%	35.8
5%	16.4

Table 6: Epochs to match the GAS accuracy on Reddit.

GPU Memory Consumption. Although IncAgg maintains two versions of node embeddings, both are stored in RAM and therefore do not impose additional pressure on GPU memory. As shown in Table 7, IncAgg has GPU memory consumption comparable to the baseline, confirming that the efficiency improvements are achieved without increasing GPU memory overhead. Specifically, IncAgg tends to consume less reserved memory, especially significant with small batch sizes. This is because IncAgg avoids pulling out-of-batch neighbors into GPU and instead performs aggregation in-batch.

Table 7: GPU memory consumption on ogbn-arxiv under random partitioning. Δ denotes relative change, computed as $(\mathrm{IncAgg} - \mathrm{GAS})/\mathrm{GAS}$.

	Allocated (MB)			**Reserved (MB)**		
$\frac{\lvert\mathcal{B}\rvert}{\lvert\mathcal{V}\rvert}$	**GAS**	**IncAgg**	**Δ %**	**GAS**	**IncAgg**	**Δ %**
50%	1900	1850	**-3**	2800	2300	**-18**
25%	1180	1120	**-5**	2200	1600	**-27**
12.5%	830	780	**-6**	2000	950	**-53**
5%	610	560	**-8**	1700	820	**-52**
2.5%	520	480	**-8**	1350	700	**-48**

4.4 Sensitivity Analysis

Impact of Batch Size. We investigate the impact of batch size on both classification accuracy and training efficiency. Here, batch size is defined as the ratio $|\mathcal{B}|/|\mathcal{V}|$, representing the ratio of the number of subgraphs selected per training batch after graph partitioning. Across a wide range of batch sizes, IncAgg achieves performance comparable to or better than the GAS baseline while consistently reducing epoch time, as reported in Table 8. Notably, the speedup becomes increasingly pronounced as the batch size decreases. This trend is consistent with the edge statistics in Table 9: smaller batches induce a larger fraction of out-of-batch edges, which increases the aggregation cost avoided by IncAgg. These results show that IncAgg remains effective under small batch sizes, making it well suited for resource-constrained training settings.

Robustness to Subgraph Partitioning. Subgraph partitioning is a common technique for enabling mini-batch training on large graphs, with methods such as GAS typically relying on METIS to construct high-quality partitions. However, computing such partitions can be expensive in both time and memory, especially for large or dense graphs. We show that IncAgg is robust to the choice of partitioning strategy by comparing METIS and random partitioning in Table 8. Even under random partitioning, IncAgg preserves high classification accuracy while achieving larger speedups than under METIS. Table 9 further explains this behavior. Random partitioning creates substantially more out-of-batch edges than METIS; for example, at a 5% batch ratio, the out-of-batch edge ratio is 94.1% under random partitioning, compared with 36.2% under METIS. Despite this much harsher partitioning condition, IncAgg maintains competitive accuracy by combining in-batch residual aggregation with periodically refreshed global

Table 9: Inter-batch edge statistics under random and METIS partitioning. IB and OB denote in-batch and out-of-batch edges; OB% denotes $\mathrm{OB}/(\mathrm{IB} + \mathrm{OB})$.

Part.	**Type**	**100%**	**50%**	**20%**	**10%**	**5%**
Random	**IB**	23.5M	11.8M	4.9M	2.6M	1.4M
	OB	0.0M	11.6M	18.6M	20.9M	22.1M
	OB%	0.0	49.5	79.2	89.1	94.1
METIS	**IB**	23.5M	21.8M	18.4M	16.8M	15.0M
	OB	0.0M	1.7M	5.0M	6.6M	8.5M
	OB%	0.0	7.2	21.5	28.2	36.2

Table 8: Impact of batch size on accuracy and epoch time on Reddit under METIS and random partitioning. Speedup ($\times$) is computed as $\frac{\text{GAS}}{\text{IncAgg}}$.

	METIS Partitioning					Random Partitioning				
	Acc. (%)		Time (ms)			Acc. (%)		Time (ms)		
$\frac{\lvert\mathcal{B}\rvert}{\lvert\mathcal{V}\rvert}$	GAS	IncAgg	GAS	IncAgg	$\times$	GAS	IncAgg	GAS	IncAgg	$\times$
50%	95.0	95.3	1333	1000	1.33	94.9	94.9	1706	948	1.80
20%	95.1	95.1	1808	925	1.95	94.5	94.6	2513	692	3.63
10%	95.0	95.0	2199	936	2.35	94.3	94.5	3782	663	5.70
5%	94.9	95.0	2810	964	2.91	93.6	94.5	6667	727	9.17

memory. Meanwhile, the larger fraction of discarded out-of-batch aggregation also creates more room for computational savings. This indicates that IncAgg does not rely on costly high-quality graph partitioning to be effective.

Impact of Memory Update Frequency. Across settings, per-epoch updates already match or closely approach the accuracy of the GAS baseline, indicating that infrequent memory refreshes are sufficient to maintain accuracy in practice (Appendix C.2). To further examine this behavior, we measure the relative approximation error with respect to exact full-neighborhood aggregation: $\text{rel} = \|\text{out}_m - \text{out}_{\text{ref}}\|_F / (\|\text{out}_{\text{ref}}\|_F + 10^{-12})$, where out_m is the mini-batch forward output produced by GAS or IncAgg on the current batch targets, and out_{ref} is the exact L-hop forward output computed using full-neighborhood aggregation. This metric measures the error magnitude relative to the reference signal as smaller values indicate closer approximation. Table 10 reports representative settings, with the full results deferred to Appendix C.2. IncAgg achieves relative errors close to GAS across both METIS and random partitioning. Even in the challenging small-batch settings, the error remains well below 1 and differs from GAS only by a small margin. This suggests that although IncAgg uses periodically refreshed memory and avoids pseudo full-neighborhood aggregation during each training iteration, its global memory remains accurate enough to preserve the quality of the forward approximation. Together with the accuracy results, it supports per-epoch memory refresh as a strong default choice in practice.

Table 10: Representative relative approximation errors compared with exact full-neighborhood aggregation.

Dataset	Part.	$\lvert\mathcal{B}\rvert/\lvert\mathcal{V}\rvert$	GAS	IncAgg
Arxiv	random	25%	0.099	0.102
Arxiv	random	12.5%	0.132	0.151
Arxiv	METIS	12.5%	0.111	0.137
Reddit	random	12.5%	0.148	0.163

4.5 Comparison with Memory-Enhanced Methods

Existing memory-enhanced methods can be broadly viewed as following two design directions. Early and widely adopted baselines such as VR-GCN [2] and GAS [7] approximate full-neighborhood message by using historical embeddings for out-of-batch neighbors. Following this line, later methods mainly seek to improve the quality of such approximation. For example, GraphFM [17] introduces momentum-based history updates, LMC [14] further maintains historical gradient information, and REST [15] reduces feature staleness through additional forward-only memory refreshes. While these designs can improve the freshness or fidelity of historical states, they inherit the pseudo full-neighborhood recovery paradigm and introduce additional update, memory-access, or forward-computation overhead.

Table 11: Comparison with representative memory-enhanced methods on Reddit. Best results are in bold and the second best underlined.

Method	Acc. (%)	Epoch (ms)
IncAgg (ours)	<u>94.5</u>	**727**
GAS [7]	93.6	<u>6667</u>
LMC [14]	**94.7**	7204
VR-GCN [2]	<u>94.5</u>	14935
GraphFM [17]	93.1	25557

IncAgg explores a different and less studied direction. IncAgg asks whether such per-iteration pseudo full-neighborhood recovery is necessary for accurate training. By using periodically refreshed global memory as a stable baseline and propagating only in-batch residual updates, IncAgg deliberately reduces the reliance on out-of-batch historical embeddings during training. This design targets the efficiency bottleneck of memory-enhanced methods, aiming to substantially reduce aggregation and

data movement while minimizing performance sacrifice. The qualitative differences among these methods are summarized in Appendix E. We compare methods in terms of accuracy and epoch time under identical settings. The results in Table 11 show that IncAgg achieves the lowest epoch time while maintaining accuracy comparable to the strongest alternatives. LMC obtains slightly higher accuracy, but at nearly an order-of-magnitude higher epoch time, highlighting the different objective of IncAgg: improving efficiency while minimizing accuracy sacrifice. Appendix F further reports matched end-to-end comparisons with GAS and REST [15], a controlled comparison with neighbor sampling and LABOR [1], data-movement traces, and an analysis of memory staleness under different depths and refresh frequencies.

5 Conclusion

In this work, we revisit memory-enhanced GNN training from an efficiency perspective. Through profiling analysis, we show that a major bottleneck lies not only in maintaining historical embeddings, but also in reconstructing pseudo full-neighborhood aggregation inside every mini-batch iteration through repeated out-of-batch memory access. Motivated by this observation, we propose IncAgg, an incremental aggregation framework that decouples historical full-neighborhood context from iteration-level training. IncAgg periodically constructs a global memory baseline and propagates only the current in-batch residual along the differentiable training path, thereby reducing gradient-carrying aggregation and host–GPU data movement without modifying the backbone architecture. Experiments on multiple benchmarks demonstrate that IncAgg preserves predictive performance and provides substantial training speedups across datasets, backbones, batch sizes, partitioning strategies, and memory-refresh frequencies. These results suggest that, for scalable graph learning, when and how neighborhood context is incorporated can be as important as how accurately it is reconstructed, opening a practical direction for efficient memory-enhanced GNN training. This work focuses on transductive node classification with fixed-coefficient linear propagation; extending IncAgg to attention-based and non-additive aggregators, other graph learning tasks, and broader comparisons with sampling-based training beyond Reddit is left for future work.

Acknowledgments and Disclosure of Funding

Zhichao Hou and Xiaorui Liu are supported by the U.S. Department of Energy, Office of Science, Office of Advanced Scientific Computing Research, under Award Number DE-SC0026773; the National Science Foundation (NSF) under Grant Numbers IIS-2443182 and CMMI-2609644; and gift funding from Snap Inc.

References

[1] Muhammed Fatih Balın and Ümit V. Çatalyürek. Layer-neighbor sampling — defusing neighborhood explosion in GNNs. In *Proceedings of the 37th International Conference on Neural Information Processing Systems*, NIPS '23, pages 25819–25836, Red Hook, NY, USA, May 2024. Curran Associates Inc.

[2] Jianfei Chen, Jun Zhu, and Le Song. Stochastic training of graph convolutional networks with variance reduction. In *Proceedings of the International Conference on Machine Learning*, pages 942–950, 2018.

[3] Jie Chen, Tengfei Ma, and Cao Xiao. FastGCN: Fast learning with graph convolutional networks via importance sampling. In *Proceedings of the International Conference on Learning Representations*, 2018.

[4] Ming Chen, Zhewei Wei, Zengfeng Huang, Bolin Ding, and Yaliang Li. Simple and deep graph convolutional networks. In *Proceedings of the International Conference on Machine Learning*, pages 1725–1735, 2020.

[5] Wei-Lin Chiang, Xuanqing Liu, Si Si, Yang Li, Samy Bengio, and Cho-Jui Hsieh. Cluster-gcn: An efficient algorithm for training deep and large graph convolutional networks. In *Proceedings of the International Conference on Knowledge Discovery & Data Mining*, pages 257–266, 2019.

[6] Matthias Fey and Jan Eric Lenssen. Fast graph representation learning with pytorch geometric. *arXiv preprint arXiv:1903.02428*, 2019.

[7] Matthias Fey, Jan E Lenssen, Frank Weichert, and Jure Leskovec. Gnnautoscale: Scalable and expressive graph neural networks via historical embeddings. In *Proceedings of the International Conference on Machine Learning*, pages 3294–3304, 2021.

[8] Johannes Gasteiger, Aleksandar Bojchevski, and Stephan Günnemann. Predict then propagate: Graph neural networks meet personalized pagerank. In *Proceedings of the International Conference on Learning Representations*, 2018.

[9] William L. Hamilton, Rex Ying, and Jure Leskovec. Inductive representation learning on large graphs. In *Proceedings of the International Conference on Neural Information Processing Systems*, page 1025–1035, 2017.

[10] Weihua Hu, Matthias Fey, Hongyu Ren, Maho Nakata, Yuxiao Dong, and Jure Leskovec. Ogb-lsc: A large-scale challenge for machine learning on graphs. *arXiv preprint arXiv:2103.09430*, 2021.

[11] Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. In *Proceedings of the International Conference on Learning Representations*, 2017.

[12] Ziqi Liu, Zhengwei Wu, Zhiqiang Zhang, Jun Zhou, Shuang Yang, Le Song, and Yuan Qi. Bandit samplers for training graph neural networks. *Advances in Neural Information Processing Systems*, 33:6878–6888, 2020.

[13] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. *Advances in Neural Information Processing Systems*, 32, 2019.

[14] Zhihao Shi, Xize Liang, and Jie Wang. LMC: Fast training of GNNs via subgraph sampling with provable convergence. In *Proceedings of the International Conference on Learning Representations*, 2023.

[15] Rui Xue, Tong Zhao, Neil Shah, and Xiaorui Liu. Haste makes waste: A simple approach for scaling graph neural networks. In *Proceedings of the International Conference on Machine Learning*, 2025.

[16] Zhitao Ying, Jiaxuan You, Christopher Morris, Xiang Ren, Will Hamilton, and Jure Leskovec. Hierarchical graph representation learning with differentiable pooling. *Advances in Neural Information Processing Systems*, 31, 2018.

[17] Haiyang Yu, Limei Wang, Bokun Wang, Meng Liu, Tianbao Yang, and Shuiwang Ji. GraphFM: Improving Large-Scale GNN Training via Feature Momentum. In *Proceedings of the International Conference on Machine Learning*, pages 25684–25701, 2022.

[18] Hanqing Zeng, Hongkuan Zhou, Ajitesh Srivastava, Rajgopal Kannan, and Viktor Prasanna. GraphSAINT: Graph sampling based inductive learning method. In *Proceedings of the International Conference on Learning Representations*, 2020.

[19] Muhan Zhang and Yixin Chen. Link prediction based on graph neural networks. *Advances in Neural Information Processing Systems*, 31, 2018.

[20] Qingru Zhang, David Wipf, Quan Gan, and Le Song. A biased graph neural network sampler with near-optimal regret. *Advances in Neural Information Processing Systems*, 34:8833–8844, 2021.

[21] Difan Zou, Ziniu Hu, Yewen Wang, Song Jiang, Yizhou Sun, and Quanquan Gu. Layer-dependent importance sampling for training deep and large graph convolutional networks. *Advances in Neural Information Processing Systems*, 32, 2019.

A Theoretical Analysis and Complexity

A.1 Estimator Variance Analysis

Setup. We analyze a single message-passing layer and a single feature dimension. The extension to vector-valued embeddings follows coordinate-wise. For a node u, let $\mathcal{N}(u)$ be its neighborhood and $n_u = |\mathcal{N}(u)|$. We focus on the aggregated message computed from layer-k embeddings:

$$a_u^{(k)} = \sum_{v \in \mathcal{N}(u)} p_{uv} h_v^{(k)}. \tag{1}$$

This quantity is the neighborhood aggregation term used to construct the next layer representation, e.g., through $h_u^{(k+1)} = \sigma(a_u^{(k)} W^{(k)})$ for a GCN layer. Given a history snapshot $m_v^{(k)}$, define the residual

$$\Delta h_v^{(k)} = h_v^{(k)} - m_v^{(k)}. \tag{2}$$

Then the exact aggregated message admits the residual decomposition

$$a_u^{(k)} = \underbrace{\sum_{v \in \mathcal{N}(u)} p_{uv} m_v^{(k)}}_{\text{historical full-neighborhood context}} + \underbrace{\sum_{v \in \mathcal{N}(u)} p_{uv} \Delta h_v^{(k)}}_{\text{residual correction}}. \tag{3}$$

Sampling identity. Following the estimator-level analysis of VR-GCN [2], let $S_u \subseteq \mathcal{N}(u)$ contain D_u neighbors sampled uniformly without replacement, and define

$$C_u = 1 - \frac{D_u - 1}{n_u - 1}. \tag{4}$$

For any generic scalar message values $\{b_v\}_{v \in \mathcal{N}(u)}$, the following finite-population variance identity holds:

$$\mathrm{Var}_{S_u}\left[\frac{n_u}{D_u} \sum_{v \in S_u} b_v\right] = \frac{C_u}{2D_u} \sum_{v_1, v_2 \in \mathcal{N}(u)} (b_{v_1} - b_{v_2})^2. \tag{5}$$

For compactness, denote the right-hand side by

$$\mathcal{V}_u(b) = \frac{C_u}{2D_u} \sum_{v_1, v_2 \in \mathcal{N}(u)} (b_{v_1} - b_{v_2})^2. \tag{6}$$

In the following, we instantiate this identity with either $b_v = p_{uv} h_v^{(k)}$ for raw messages or $b_v = p_{uv} \Delta h_v^{(k)}$ for residual messages.

Neighbor sampling and VR-GCN. For neighbor sampling, the estimator is

$$\widehat{a}_{u,\mathrm{NS}}^{(k)} = \frac{n_u}{D_u} \sum_{v \in S_u} p_{uv} h_v^{(k)}, \tag{7}$$

and therefore

$$\mathrm{Var}\left[\widehat{a}_{u,\mathrm{NS}}^{(k)}\right] = \mathcal{V}_u\left(\{p_{uv} h_v^{(k)}\}_{v \in \mathcal{N}(u)}\right). \tag{8}$$

VR-GCN instead applies stochastic approximation to the residual term:

$$\widehat{a}_{u,\mathrm{VR}}^{(k)} = \sum_{v \in \mathcal{N}(u)} p_{uv} m_v^{(k)} + \frac{n_u}{D_u} \sum_{v \in S_u} p_{uv} \Delta h_v^{(k)}. \tag{9}$$

Thus,

$$\mathrm{Var}\left[\widehat{a}_{u,\mathrm{VR}}^{(k)}\right] = \mathcal{V}_u\left(\{p_{uv} \Delta h_v^{(k)}\}_{v \in \mathcal{N}(u)}\right). \tag{10}$$

Compared with neighbor sampling, VR-GCN replaces the variance of raw messages with the variance of residual messages, which is smaller when historical embeddings are close to current embeddings.

IB-only aggregation. A naive in-batch estimator keeps only the current in-batch messages:

$$\widehat{a}_{u,\mathrm{IB}}^{(k)} = \sum_{v \in S_u} p_{uv} h_v^{(k)}. \tag{11}$$

Since this estimator does not use the rescaling factor n_u/D_u, its sampling variance is

$$\mathrm{Var}\left[\widehat{a}_{u,\mathrm{IB}}^{(k)}\right] = \left(\frac{D_u}{n_u}\right)^2 \mathcal{V}_u\left(\{p_{uv} h_v^{(k)}\}_{v \in \mathcal{N}(u)}\right). \tag{12}$$

This smaller nominal variance should not be interpreted as a better approximation, since the out-of-batch messages are simply omitted.

GAS and IncAgg. For a unified comparison, we use the same notation S_u to denote the neighbors of u that are available in the current mini-batch. GAS uses current embeddings for these in-batch neighbors and historical embeddings for the remaining out-of-batch neighbors:

$$\widehat{a}_{u,\mathrm{GAS}}^{(k)} = \sum_{v \in S_u} p_{uv} h_v^{(k)} + \sum_{v \in \mathcal{N}(u) \setminus S_u} p_{uv} m_v^{(k)} \tag{13}$$

$$= \sum_{v \in \mathcal{N}(u)} p_{uv} m_v^{(k)} + \sum_{v \in S_u} p_{uv} \Delta h_v^{(k)}. \tag{14}$$

Conditioning on $|S_u| = D_u$, its sampling variance is

$$\mathrm{Var}\left[\widehat{a}_{u,\mathrm{GAS}}^{(k)}\right] = \left(\frac{D_u}{n_u}\right)^2 \mathcal{V}_u\left(\{p_{uv} \Delta h_v^{(k)}\}_{v \in \mathcal{N}(u)}\right). \tag{15}$$

IncAgg uses the same residual form, but materializes the historical full-neighborhood context as global memory:

$$g_u^{(k)} = \left(PM^{(k)}\right)_u = \sum_{v \in \mathcal{N}(u)} p_{uv} m_v^{(k)}. \tag{16}$$

Its estimator is

$$\widehat{a}_{u,\mathrm{IncAgg}}^{(k)} = g_u^{(k)} + \sum_{v \in S_u} p_{uv} \Delta h_v^{(k)}, \tag{17}$$

and hence

$$\mathrm{Var}\left[\widehat{a}_{u,\mathrm{IncAgg}}^{(k)}\right] = \left(\frac{D_u}{n_u}\right)^2 \mathcal{V}_u\left(\{p_{uv} \Delta h_v^{(k)}\}_{v \in \mathcal{N}(u)}\right). \tag{18}$$

Summary. The analysis shows that VR-GCN, GAS, and IncAgg share the same variance-reduction principle: their stochastic component depends on residual messages $p_{uv}\Delta h_v^{(k)}$ rather than raw messages $p_{uv} h_v^{(k)}$. Under a fixed history snapshot, GAS and IncAgg have the same residual-variance form. Their key distinction is computational: GAS constructs the historical contribution during each mini-batch iteration, whereas IncAgg materializes it as periodically refreshed global memory, removing repeated out-of-batch history aggregation and memory access.

A.2 Complexity Analysis

Notation. Let $n := |\mathcal{V}|$, $m := |\mathcal{E}|$, K be the number of GNN layers, d be the hidden dimension, and $b := |\mathcal{B}_t|$ be the mini-batch size. For a batch $\mathcal{B}_t$, we denote by

$$\mathcal{E}[\mathcal{B}_t] = \{(u, v) \in \mathcal{E} : u \in \mathcal{B}_t,\ v \in \mathcal{B}_t\} \tag{19}$$

the in-batch edges whose two endpoints both lie in $\mathcal{B}_t$. Let $\bar{\mathcal{B}}_t$ denote the out-of-batch neighbors that contribute messages to nodes in $\mathcal{B}_t$ under the full propagation matrix $\mathbf{P}$, and define the corresponding pseudo full-neighborhood edge set as

$$\mathcal{E}_{\mathrm{full}}(\mathcal{B}_t) = \{(u, v) \in \mathcal{E} : u \in \mathcal{B}_t,\ v \in \mathcal{B}_t \cup \bar{\mathcal{B}}_t\}. \tag{20}$$

GAS performs pseudo full-neighborhood aggregation over $\mathcal{E}_{\mathrm{full}}(\mathcal{B}_t)$ during each mini-batch training iteration. IncAgg instead uses the periodically refreshed local and global memories, $\mathbf{M}_L$ and $\mathbf{M}_G$, and performs differentiable aggregation only over $\mathcal{E}[\mathcal{B}_t]$ on the training path.

Training-path aggregation cost. We first isolate the gradient-carrying training path and omit backbone-specific linear transformations that are shared by both frameworks. For one layer and one mini-batch, the aggregation costs are

$$\text{GAS}: \quad O(|\mathcal{E}_{\text{full}}(\mathcal{B}_t)|d)\,, \tag{21}$$

$$\text{INCAGG}: \quad O(|\mathcal{E}[\mathcal{B}_t]|d)\,. \tag{22}$$

Across K layers, both costs are multiplied by K. Since $\mathcal{E}[\mathcal{B}_t] \subseteq \mathcal{E}_{\text{full}}(\mathcal{B}_t)$, IncAgg never performs more training-path aggregation than GAS in a mini-batch iteration. The reduction is largest when many neighbors of $\mathcal{B}_t$ lie outside the current batch.

For a partition of the graph into mini-batches $\{\mathcal{B}_t\}_{t=1}^{T}$, GAS has per-epoch training-path aggregation cost

$$O\left(Kd\sum_{t=1}^{T}|\mathcal{E}_{\text{full}}(\mathcal{B}_t)|\right) = O(Kmd), \tag{23}$$

up to constants depending on edge directionality and normalization. IncAgg has per-epoch training-path aggregation cost

$$O\left(Kd\sum_{t=1}^{T}|\mathcal{E}[\mathcal{B}_t]|\right). \tag{24}$$

Under uniform random partitioning with batch size b, a fixed edge is retained as an in-batch edge within one epoch with probability

$$\Pr[(u,v) \text{ is in-batch}] = \frac{b-1}{n-1} = \Theta\left(\frac{b}{n}\right). \tag{25}$$

Therefore,

$$\mathbb{E}\left[\sum_{t=1}^{T}|\mathcal{E}[\mathcal{B}_t]|\right] = \Theta\left(m\frac{b}{n}\right), \tag{26}$$

and the expected per-epoch training-path aggregation cost of IncAgg is

$$O\left(Kmd\frac{b}{n}\right). \tag{27}$$

Compared with the $O(Kmd)$ pseudo full-neighborhood aggregation cost of GAS, IncAgg reduces the repeated gradient-carrying aggregation cost by a factor of $\Theta(n/b)$ in expectation under random partitioning.

For graph partitioning methods such as METIS, the exact cost depends on the number of edges preserved within each partition. The same comparison still holds: IncAgg restricts training-path aggregation to in-batch edges, while GAS aggregates over pseudo full neighborhoods that include out-of-batch historical embeddings.

Cost of global memory refresh. The above reduction concerns the mini-batch training path rather than the elimination of full-graph propagation. IncAgg maintains a local memory snapshot $\mathbf{M}_L$ and a global memory baseline

$$\mathbf{M}_G^{(k)} = \mathbf{P}\mathbf{M}_L^{(k)} \tag{28}$$

for each layer k. Refreshing $\mathbf{M}_G$ over all K layers requires forward-only full-neighborhood propagation with cost $O(Kmd)$. This cost is not incurred at every mini-batch iteration. If the global memory is refreshed every τ epochs, its amortized edge-touch cost is

$$O\left(\frac{Kmd}{\tau}\right) \tag{29}$$

per epoch. Thus, under uniform random partitioning, the total edge-touch complexity of IncAgg is

$$O\left(Kmd\frac{b}{n} + \frac{Kmd}{\tau}\right). \tag{30}$$

When $\tau = 1$, IncAgg has the same asymptotic edge-touch order as GAS, but the full-neighborhood propagation is moved outside the gradient-carrying mini-batch training loop and reused across batches. In our implementation, the refresh can be integrated with the routine validation pass, avoiding a separate standalone full-neighborhood propagation pass on the main optimization path.

Algorithm 1 Mini-batch Training of IncAgg

Input: Propagation matrix $\mathbf{P} = \{p_{uv}\}_{(u,v)\in\mathcal{E}}$, feature matrix $\mathbf{X} = \{\mathbf{x}_v\}_{v\in\mathcal{V}}$, teleport parameter α, number of epochs N, number of layers K.
Output: Updated model parameters.
for epoch $e = 1$ to N **do**
  Construct I batches with partitioned subgraphs.
  for iteration $i = 1$ to I **do**
    Load batch feature matrix $\mathbf{X}_{\mathcal{B}}$, truncated propagation matrix $\mathbf{P}_{\mathcal{B},\mathcal{B}}$.
    Initialize $\mathbf{H}_{\mathcal{B}}^{(0)} = \mathbf{X}_{\mathcal{B}}$.
    for layer $k = 1$ to K **do**
      Pull memories $\mathbf{M}_{L,\mathcal{B}}^{(k-1)}$ and $\mathbf{M}_{G,\mathcal{B}}^{(k-1)}$.
      Backbone ops (before): Perform optional backbone-specific computation before propagation.
      Incremental update:

$$\mathbf{H}_{\mathcal{B}}^{(k)} = \mathbf{P}_{\mathcal{B},\mathcal{B}} \left(\mathbf{H}_{\mathcal{B}}^{(k-1)} - \mathbf{M}_{L,\mathcal{B}}^{(k-1)} \right) + \mathbf{M}_{G,\mathcal{B}}^{(k-1)}.$$

      Backbone ops (after): Perform optional backbone-specific computation after propagation.
    end for
    Compute loss and update optimizer.
  end for
  Validate model and update memories.
end for

Memory footprint and data movement. Both GAS and IncAgg maintain full-graph historical embeddings in host memory. Using the notation in the main text, GAS stores the local historical memory $\mathbf{M}_L \in \mathbb{R}^{n\times d}$, while IncAgg additionally stores the global memory baseline $\mathbf{M}_G \in \mathbb{R}^{n\times d}$. For K layer-wise memories, this changes the host-memory footprint from $O(Knd)$ to $O(2Knd)$, which is a constant-factor increase.

The benefit is reduced per-iteration host–GPU memory movement. Let q_t be the number of distinct out-of-batch neighbors whose historical embeddings are pulled by GAS for batch $\mathcal{B}_t$, and define $\rho := \mathbb{E}[q_t]/b$. GAS needs to access historical embeddings for both in-batch nodes and out-of-batch neighbors, giving per-iteration memory access

$$O((b + q_t)d) = O((1 + \rho)bd) \tag{31}$$

in expectation. By contrast, IncAgg only accesses the in-batch slices $\mathbf{M}_{L,\mathcal{B}_t}$ and $\mathbf{M}_{G,\mathcal{B}_t}$, resulting in

$$O(bd) \quad \text{to} \quad O(2bd) \tag{32}$$

per iteration depending on whether the two memory reads are materialized separately. Therefore, IncAgg removes the per-iteration out-of-batch history pulls required by GAS. The communication saving is more pronounced when ρ is large, such as under random partitioning or small batches, and smaller under high-quality graph partitions with few boundary neighbors.

B Backbone-Specific Algorithms

This section presents the adaptations of IncAgg training framework to different backbone architectures. We describe the training process for three representative GNN models—GCN, APPNP, and GCNII, and highlight the key modifications introduced by IncAgg to improve efficiency while maintaining performance.

B.1 General

Algorithm 1 outlines how IncAgg integrates with standard mini-batch training pipelines. Our framework is agnostic to specific GNN architectures and requires minimal modifications to the message-passing step.

Algorithm 2 Mini-batch Training Epoch of IncAgg-GCN

Input: Adjacency matrix $\mathbf{A} = \{a_{uv}\}_{(u,v)\in\mathcal{E}}$, feature matrix $\mathbf{X} = \{\mathbf{x}_v\}_{v\in\mathcal{V}}$, teleport parameter α, number of epochs N, number of layers K.
Compute normalized adjacency:

$$\tilde{\mathbf{A}} = \mathbf{I} + \mathbf{A}, \qquad \mathbf{P} = \tilde{\mathbf{D}}^{-\frac{1}{2}}\tilde{\mathbf{A}}\tilde{\mathbf{D}}^{-\frac{1}{2}}, \quad \tilde{\mathbf{D}}_{ii} = \sum_j \tilde{\mathbf{A}}_{ij}.$$

Output: Updated model parameters.
for epoch $e = 1$ to N **do**
  Construct I batches with partitioned subgraphs.
  for iteration $i = 1$ to I **do**
    Load batch feature matrix $\mathbf{X}_{\mathcal{B}}$.
    for layer $k = 1$ to K **do**
      Retrieve memories $\mathbf{M}_{L,\mathcal{B}}^{(k)}$, $\mathbf{M}_{G,\mathcal{B}}^{(k)}$.
      {Incremental embedding update}

$$\mathbf{H}_{\mathcal{B}}^{(k)} = \mathbf{P}_{\mathcal{B},\mathcal{B}}\left(\mathbf{H}_{\mathcal{B}}^{(k)} - \mathbf{M}_{L,\mathcal{B}}^{(k)}\right) + \mathbf{M}_{G,\mathcal{B}}^{(k)}.$$

      Apply linear transformation:

$$\mathbf{H}_{\mathcal{B}}^{(k)} = \mathrm{linear}\big(\mathbf{H}_{\mathcal{B}}^{(k)}\big).$$

      Apply activation:

$$\mathbf{H}_{\mathcal{B}}^{(k)} = \mathrm{ReLU}\big(\mathbf{H}_{\mathcal{B}}^{(k)}\big).$$

      Apply dropout:

$$\mathbf{H}_{\mathcal{B}}^{(k+1)} = \mathrm{dropout}\big(\mathbf{H}_{\mathcal{B}}^{(k)}\big).$$

    end for{Layer loop}
    Compute loss and update optimizer.
  end for{Iteration loop}
  Perform validation and update memories.
end for{Epoch loop}

B.2 IncAgg-GCN

Graph Convolutional Networks (GCN) follow a message-passing paradigm where node embeddings are updated using neighborhood information through a normalized adjacency matrix. As shown in Algorithm 2, IncAgg-GCN introduces an incremental embedding update mechanism that replaces the standard full-neighborhood aggregation. Instead of recomputing all neighbor embeddings in every iteration, the model retrieves stored historical embeddings and only updates relevant portions. After aggregation, the transformed embeddings undergo a linear transformation, followed by ReLU activation and dropout, ensuring stable training. This design reduces computational overhead while maintaining representation quality, making IncAgg-GCN more scalable for large graphs.

B.3 IncAgg-APPNP

Personalized PageRank-based Propagation (APPNP) introduces a teleportation mechanism, which preserves original node features across multiple propagation steps. The power iteration process in APPNP gradually refines node embeddings while maintaining a balance between raw features and aggregated neighborhood information. IncAgg-APPNP optimizes APPNP's propagation by integrating periodic memory updates, reducing redundant computation while retaining the effectiveness of personalized PageRank. As shown in Algorithm 3, the incremental embedding update ensures that only necessary updates are computed at each step, significantly lowering memory consumption. Additionally, the teleportation mechanism is preserved, where each update is weighted by a factor α, ensuring that node embeddings retain their initial features while being refined across multiple layers.

Algorithm 3 Mini-batch Training Epoch of IncAgg-APPNP

Input: Propagation matrix $\mathbf{P} = \{p_{uv}\}_{(u,v)\in\mathcal{E}}$, feature matrix $\mathbf{X} = \{\mathbf{x}_v\}_{v\in\mathcal{V}}$, teleport parameter α, number of epochs N, number of layers K.
Output: Updated model parameters.
for epoch $e = 1$ to N **do**
  Construct I batches with partitioned subgraphs.
  for iteration $i = 1$ to I **do**
    Load batch feature matrix $\mathbf{X}_{\mathcal{B}}$.
    $\mathbf{H}_{\mathcal{B}}^{(0)} = \text{MLP}(\mathbf{X}_{\mathcal{B}})$ {APPNP initial transformation}
    for layer $k = 1$ to K **do**
      Retrieve memories $\mathbf{M}_{L,\mathcal{B}}^{(k)}$ and $\mathbf{M}_{G,\mathcal{B}}^{(k)}$.
      {Incremental embedding update}

$$\mathbf{H}_{\mathcal{B}}^{(k)} = \mathbf{P}_{\mathcal{B},\mathcal{B}} \left(\mathbf{H}_{\mathcal{B}}^{(k)} - \mathbf{M}_{L,\mathcal{B}}^{(k)} \right) + \mathbf{M}_{G,\mathcal{B}}^{(k)}.$$

      {APPNP power iteration}

$$\mathbf{H}_{\mathcal{B}}^{(k+1)} = (1-\alpha)\mathbf{H}_{\mathcal{B}}^{(k)} + \alpha\,\mathbf{H}_{\mathcal{B}}^{(0)}.$$

    end for{Layer loop}
    Compute loss and update optimizer.
  end for{Iteration loop}
  Perform validation and update memories.
end for{Epoch loop}

B.4 IncAgg-GCNII

Graph Convolutional Networks with Initial Residual and Identity Mapping (GCNII) enhance GCN by enabling deeper architectures through residual connections and adaptive feature propagation. These improvements mitigate the common issue of oversmoothing in deep GNNs, making GCNII an effective backbone for graph learning tasks. In IncAgg-GCNII, IncAgg seamlessly integrates periodic memory updates into the propagation steps. Rather than recomputing full-neighborhood aggregation in each iteration, stored historical embeddings are retrieved and updated using incremental embedding updates. This adaptation maintains GCNII's deep feature propagation advantages while significantly improving computational efficiency. Epoch process is shown in Algorithm 4.

C Experimental Details and Additional Results

C.1 Implementation Details

All models are implemented with PyTorch [13] and PyTorch Geometric [6], and optimized using Adam. All experiments were conducted on a Linux server equipped with AMD Ryzen Threadripper PRO 7965WX 24-core CPU(48 hardware threads) and 250GB system memory, using one single NVIDIA RTX A6000 GPU with 48GB GPU memory. The server uses NVIDIA driver 550.163.01 with CUDA 12.4. Unless otherwise specified, hyperparameters are selected from a common search space with dropout rates $\{0, 0.1, 0.3, 0.5\}$ and learning rates $\{0.01, 0.005, 0.001, 0.0001\}$. Experiments on VR-GCN follow the settings in its official implementation.

C.2 Approximation Quality and Memory Refresh Frequency

This section provides additional evidence on the quality of the approximation introduced by IncAgg and the effect of memory refresh frequency. The goal is to understand whether periodically refreshed global memory can preserve the representation quality of memory-enhanced training while avoiding per-iteration out-of-batch history aggregation.

Approximation to exact full-neighborhood aggregation. We first measure the relative approximation error of GAS and IncAgg with respect to exact full-neighborhood aggregation. For a mini-batch

Algorithm 4 Mini-batch Training Epoch of IncAgg-GCNII

Input: Adjacency matrix $\mathbf{A} = \{a_{uv}\}_{(u,v)\in\mathcal{E}}$, feature matrix $\mathbf{X} = \{\mathbf{x}_v\}_{v\in\mathcal{V}}$, teleport parameter α, number of epochs N, number of layers K.
Output: Updated model parameters.
Compute

$$\tilde{\mathbf{A}} = \mathbf{I} + \mathbf{A}, \qquad \mathbf{P} = \tilde{\mathbf{D}}^{-\frac{1}{2}}\tilde{\mathbf{A}}\tilde{\mathbf{D}}^{-\frac{1}{2}}, \quad \tilde{\mathbf{D}}_{ii} = \sum_{j=1}^{|\mathcal{V}|}\tilde{\mathbf{A}}_{ij}.$$

for epoch $e = 1$ to N **do**
  Construct I batches with partitioned subgraphs.
  for iteration $i = 1$ to I **do**
    Load batch feature matrix $\mathbf{X}_{\mathcal{B}}$.
    $\mathbf{H}_{\mathcal{B}}^{(0)} = \text{MLP}(\mathbf{X}_{\mathcal{B}})$ {GCNII initial transformation}
    for layer $k = 1$ to K **do**
      Retrieve memories $\mathbf{M}_{L,\mathcal{B}}^{(k)}$ and $\mathbf{M}_{G,\mathcal{B}}^{(k)}$.
      {Incremental embedding update}

$$\mathbf{H}_{\mathcal{B}}^{(k)} = \mathbf{P}_{\mathcal{B},\mathcal{B}}\left(\mathbf{H}_{\mathcal{B}}^{(k)} - \mathbf{M}_{L,\mathcal{B}}^{(k)}\right) + \mathbf{M}_{G,\mathcal{B}}^{(k)}.$$

      {Remaining operations in GCNIIConv layer}
      $\mathbf{H}_{\mathcal{B}}^{(k)} = \text{ReLU}(\mathbf{H}_{\mathcal{B}}^{(k)})$
      $\mathbf{H}_{\mathcal{B}}^{(k+1)} = \text{dropout}(\mathbf{H}_{\mathcal{B}}^{(k)})$
    end for{Layer loop}
    Compute loss and update optimizer.
  end for{Iteration loop}
  Perform validation and update memories.
end for{Epoch loop}

output $\mathbf{H}_{\text{method}}$ and the exact full-neighborhood output $\mathbf{H}_{\text{exact}}$, we compute

$$e_{\text{exact}} = \frac{\|\mathbf{H}_{\text{method}} - \mathbf{H}_{\text{exact}}\|_F}{\|\mathbf{H}_{\text{exact}}\|_F + 10^{-12}}. \tag{33}$$

This metric evaluates how closely each scalable method approximates the exact message-passing result under the same trained model.

Table 12 reports the full results across datasets, partitioning strategies, and batch sizes. IncAgg yields approximation errors close to GAS in all settings, with only a small absolute increase. As expected, the error generally increases when the batch becomes smaller, because more neighbors fall outside the current mini-batch. METIS partitioning also tends to reduce the approximation error compared with random partitioning, consistent with its stronger locality.

Table 12: Relative approximation error compared with exact full-neighborhood aggregation. IncAgg remains close to GAS across partitioning strategies and batch sizes, while smaller batches lead to larger errors for both methods.

Dataset	Partitioning	$\lvert\mathcal{B}\rvert/\lvert\mathcal{V}\rvert$	GAS	IncAgg
ogbn-arxiv	METIS	50%	0.069	0.083
ogbn-arxiv	METIS	25%	0.081	0.093
ogbn-arxiv	METIS	12.5%	0.111	0.137
ogbn-arxiv	random	50%	0.074	0.089
ogbn-arxiv	random	25%	0.099	0.102
ogbn-arxiv	random	12.5%	0.132	0.151
reddit	random	12.5%	0.148	0.163

Accuracy under different memory refresh frequencies. We next evaluate how the memory refresh frequency f affects classification accuracy. Here, f denotes the number of global memory

refreshes per epoch. A larger f refreshes historical information more frequently and therefore provides fresher global context, while $f = 1$ corresponds to the most lightweight setting, where the global memory is refreshed once per epoch.

For ogbn-arxiv, following the GAS setup, the graph is partitioned into 80 subgraphs; for ogbn-products, the graph is partitioned into 7 subgraphs. For ogbn-arxiv, batch size refers to the number of subgraphs included in each mini-batch. Thus, batch sizes 40, 20, and 10 correspond to approximately 50%, 25%, and 12.5% of the graph, respectively. For ogbn-products, we report the setting where each mini-batch contains one subgraph. The "Target" column reports the corresponding GAS accuracy.

Table 13 summarizes the representative GCN results on ogbn-arxiv. Even the lowest refresh frequency $f = 1$ closely matches the GAS target across all batch sizes. Increasing f occasionally improves accuracy, but the gain is small relative to the additional refresh cost. This supports our default choice of $f = 1$ as a favorable efficiency–accuracy trade-off.

We further test the same conclusion across additional backbones and datasets. The results are reported in Tables 14 to 18. Across GCN, APPNP, and GCNII, IncAgg generally remains close to the corresponding GAS target even with $f = 1$. Larger refresh frequencies can slightly improve accuracy, especially for deeper backbones such as GCNII, but the improvements are usually marginal. The only relatively larger gap appears for APPNP on ogbn-products, where increasing f narrows but does not fully close the gap. Overall, these results suggest that IncAgg is not overly sensitive to frequent memory refreshes, which is important for obtaining its efficiency gains in practice.

Table 13: Impact of memory refresh frequency f on IncAgg-GCN accuracy (%) on ogbn-arxiv. Even one refresh per epoch is sufficient to closely match the GAS target across batch sizes.

$\lvert\mathcal{B}\rvert/\lvert\mathcal{V}\rvert$	$f=1$	$f=2$	$f=4$	$f=8$	**Target**
50%	71.4	71.5	–	–	71.5
25%	71.1	71.6	71.2	–	71.2
12.5%	70.5	71.0	70.7	71.1	70.6

Table 14: Impact of memory refresh frequency f on IncAgg-GCN accuracy (%) on ogbn-products. IncAgg matches or exceeds the GAS target under all tested refresh frequencies.

$\lvert\mathcal{B}\rvert/\lvert\mathcal{V}\rvert$	$f=1$	$f=3$	$f=7$	**Target**
14%	75.6	75.7	76.0	75.3

Table 15: Impact of memory refresh frequency f on IncAgg-APPNP accuracy (%) on ogbn-arxiv. Accuracy remains stable across refresh frequencies and closely follows the GAS target.

$\lvert\mathcal{B}\rvert/\lvert\mathcal{V}\rvert$	$f=1$	$f=2$	$f=4$	$f=8$	**Target**
50%	71.1	71.2	–	–	71.2
25%	70.8	71.1	71.1	–	71.1
12.5%	70.5	70.6	70.5	70.5	70.5

Table 16: Impact of memory refresh frequency f on IncAgg-APPNP accuracy (%) on ogbn-products. More frequent refreshes improve accuracy, but the gain is modest compared with the additional refresh cost.

$\lvert\mathcal{B}\rvert/\lvert\mathcal{V}\rvert$	$f=1$	$f=3$	$f=7$	**Target**
14%	75.6	75.7	76.0	76.8

Table 17: Impact of memory refresh frequency f on IncAgg-GCNII accuracy (%) on ogbn-arxiv. Increasing f slightly improves accuracy and can match the GAS target, while $f = 1$ remains close.

$\lvert\mathcal{B}\rvert/\lvert\mathcal{V}\rvert$	$f = 1$	$f = 2$	$f = 4$	$f = 8$	**Target**
50%	71.7	72.1	–	–	72.2
25%	72.6	72.6	72.8	–	72.8

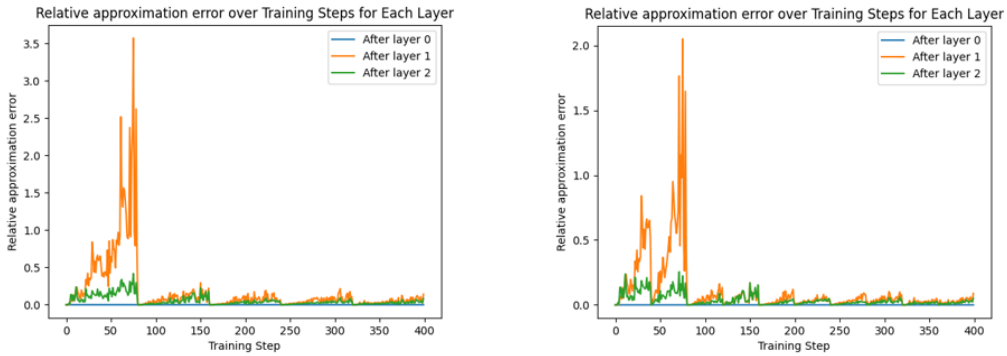
Table 18: Impact of memory refresh frequency f on IncAgg-GCNII accuracy (%) on ogbn-products. IncAgg remains close to the GAS target across refresh frequencies.

$\lvert\mathcal{B}\rvert/\lvert\mathcal{V}\rvert$	$f = 1$	$f = 3$	$f = 7$	**Target**
14%	75.9	76.1	76.0	76.2

Training dynamics of approximation error. The previous table compares each method against exact full-neighborhood aggregation. We further examine how closely IncAgg follows GAS during training. For each mini-batch, we measure the relative distance between the embeddings computed by IncAgg and GAS:

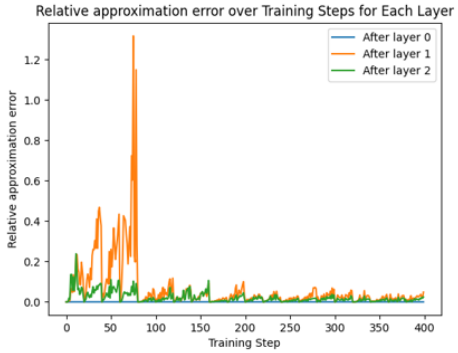
$$e_{\mathrm{GAS}} = \frac{\|\mathbf{H}_{\mathrm{IncAgg}} - \mathbf{H}_{\mathrm{GAS}}\|_F}{\|\mathbf{H}_{\mathrm{GAS}}\|_F + 10^{-12}}. \tag{34}$$

Experiments are conducted with GCN on ogbn-arxiv. The graph is partitioned into 80 subgraphs, and each mini-batch contains one subgraph. We vary the memory refresh frequency over $f \in \{1, 2, 4, 8, 16, 40\}$ and visualize the first five training epochs in Figure 4. Higher refresh frequency leads to smaller approximation error at the beginning of training, indicating that fresher memories better track GAS representations. More importantly, the error decreases across epochs for all refresh frequencies, suggesting that even infrequent memory refreshes can provide stable representation learning after the initial training phase.

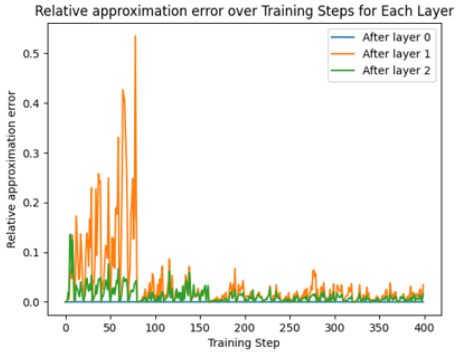


(a) $f = 1$

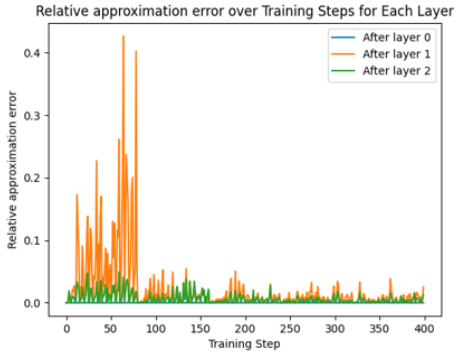
(b) $f = 2$



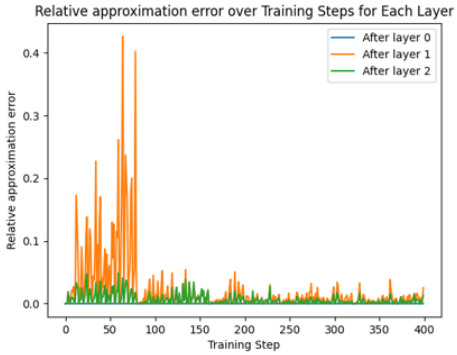
(c) $f = 4$



(d) $f = 8$



(e) $f = 16$



(f) $f = 40$

Figure 4: Relative embedding distance between IncAgg and GAS during the first five training epochs on ogbn-arxiv. More frequent memory refreshes reduce the initial discrepancy, while the error decreases over training even with infrequent refreshes.

C.3 Additional Sensitivity Analysis

We further study how IncAgg behaves under different mini-batch sizes and training hyperparameters. These experiments complement the main results by testing whether the efficiency gain of IncAgg is robust beyond the default configuration.

Impact of batch size. Batch size affects both training efficiency and approximation quality. Smaller batches reduce GPU memory usage but increase the proportion of out-of-batch neighbors. For memory-enhanced methods such as GAS, this leads to more frequent out-of-batch history aggregation and host–device memory access. IncAgg is designed to reduce this overhead by materializing the global historical aggregation in advance.

We evaluate APPNP and GCNII on ogbn-arxiv under different batch sizes. For APPNP, Table 19 shows that IncAgg consistently reduces epoch time across all batch sizes while maintaining accuracy close to GAS in most settings. The speedup is especially visible for batch sizes 20 and 10, where IncAgg reduces the epoch time from 327.90ms to 238.65ms and from 428.95ms to 301.88ms, respectively. Under the smallest batch size, both methods become slower, but IncAgg still keeps a clear efficiency advantage.

Table 19: Impact of batch size on APPNP accuracy (%) and epoch time (ms) on ogbn-arxiv. IncAgg reduces epoch time across all batch sizes while preserving accuracy in most settings.

Model	$\lvert\mathcal{B}\rvert/\lvert\mathcal{V}\rvert$	**Accuracy**	**Epoch time**
GAS-APPNP	50%	71.2	297.61
	25%	71.1	327.90
	12.5%	70.5	428.95
	5%	70.4	731.66
IncAgg-APPNP	50%	71.1	244.13
	25%	70.8	238.65
	12.5%	70.5	301.88
	5%	69.2	575.03

For GCNII, Table 20 shows a similar efficiency trend: IncAgg reduces epoch time from 194.86ms to 142.89ms at batch size 40 and from 339.11ms to 273.33ms at batch size 20. The accuracy gap is larger than in APPNP, indicating that deeper backbones can be more sensitive to the memory approximation under this additional sensitivity setting. This suggests that batch size and refresh frequency may need to be tuned jointly for deeper architectures, while the computational advantage of IncAgg remains consistent.

Table 20: Impact of batch size on GCNII accuracy (%) and epoch time (ms) on ogbn-arxiv. IncAgg consistently reduces epoch time, while deeper backbones can be more sensitive to the memory approximation.

Model	$\lvert\mathcal{B}\rvert/\lvert\mathcal{V}\rvert$	**Accuracy**	**Epoch time**
GAS-GCNII	50%	72.8	194.86
	25%	72.2	339.11
IncAgg-GCNII	50%	69.7	142.89
	25%	70.4	273.33

Impact of dropout. Dropout introduces stochasticity by masking embedding dimensions while leaving the graph structure unchanged. We evaluate whether IncAgg preserves the same dropout behavior as the GAS reference under different dropout rates. Experiments are conducted with a GCN backbone on ogbn-arxiv with $|\mathcal{B}|/|\mathcal{V}|$ taking 50%. The graph is partitioned into 80 subgraphs, each mini-batch contains 40 subgraphs, and results are averaged over 5 runs.

As shown in Table 21, IncAgg remains close to the GAS reference across all dropout rates. Without dropout, both GAS and IncAgg obtain lower accuracy than their dropout-regularized counterparts, while IncAgg with $f = 2$ still slightly outperforms the GAS reference. With dropout rates 0.3 and

Table 21: Impact of dropout rate on ogbn-arxiv with a GCN backbone. Results are test accuracy (%, mean $\pm$ std) over available random seeds. IncAgg remains close to the GAS reference across dropout rates, while the IB-only variant is less stable without historical full-neighborhood context.

Dropout	**GAS**	**IB-only**	**IncAgg,** $f = 1$	**IncAgg,** $f = 2$
0.0	70.25 ± 0.58	70.11 ± 0.38	70.43 ± 0.47	70.73 ± 0.63
0.3	71.22 ± 0.26	70.89 ± 0.35	70.93 ± 0.14	71.13 ± 0.26
0.5	71.54 ± 0.31	70.80 ± 0.43	71.40 ± 0.50	71.52 ± 0.20

0.5, IncAgg follows the same trend as GAS, and increasing the refresh frequency from $f = 1$ to $f = 2$ consistently narrows the gap. In contrast, the IB-only variant is generally below GAS and IncAgg, especially under dropout rate 0.5, suggesting that historical full-neighborhood context helps maintain stable performance across regularization settings.

D Additional Accuracy Results Reported in Prior Work

Table 22: Additional reported accuracy results (%) from prior work on shared datasets. These numbers are included as contextual references and are not controlled head-to-head comparisons with Table 3.

Reported setting	**Backbone / method**	**Arxiv**	**Products**	**Reddit**
Full batch	GCN	71.6±0.1	OOM	95.4±0.2
	APPNP	71.8±0.1	OOM	96.1±0.1
	GCNII	72.8±0.2	OOM	96.1±0.1
Memory-enhanced	GraphFM-GCN [17]	71.8±0.2	76.8±0.2	95.4±0.2
	GraphFM-GCNII [17]	72.9±0.1	77.4±0.3	96.8±0.1
	REST-GCN [15]	72.2±0.2	79.6±0.1	95.6±0.1
	REST-APPNP [15]	72.4±0.1	80.0±0.1	96.4±0.1
	REST-GCNII [15]	73.2±0.1	79.8±0.2	96.8±0.1
	REST-IS-GCN [15]	72.3±0.1	78.6±0.1	95.7±0.1
	REST-IS-APPNP [15]	72.4±0.2	80.5±0.2	96.5±0.1
	REST-IS-GCNII [15]	72.8±0.2	79.6±0.3	96.8±0.1

Note. OOM denotes out-of-memory. Results are copied from the corresponding papers and may differ in implementation, preprocessing, hyperparameter tuning, and hardware environment.

Table 22 summarizes accuracy numbers reported by prior work on datasets that overlap with our evaluation. We include these results only to provide broader context on the accuracy range of full-batch and memory-enhanced GNN training methods. Since these values are taken from separate papers and may use different implementations, preprocessing pipelines, hyperparameter choices, and hardware environments, they should not be interpreted as controlled comparisons with IncAgg.

Accordingly, our main empirical claims are based on the matched GAS–IncAgg comparisons in Table 3, where the backbone, dataset split, training protocol, and evaluation setting are controlled. The additional reported results suggest that full-batch training can become infeasible on large graphs such as ogbn-products, while existing memory-enhanced methods often improve predictive performance through increasingly refined memory maintenance. IncAgg pursues a complementary design objective: it reduces per-iteration aggregation and memory movement by restricting expensive message passing to the in-batch subgraph, while reusing periodically refreshed global context to preserve predictive performance.

E Design Comparison of Representative Memory-Enhanced GNN Training Methods

Table 23 further contrasts the design choices of representative memory-enhanced GNN training methods. Prior methods mainly improve mini-batch training by maintaining or refining historical representations for out-of-batch nodes. This line of work is effective for improving the quality

Table 23: Design comparison of representative memory-enhanced GNN training methods. IB denotes in-batch aggregation. The comparison highlights the per-iteration aggregation scope and memory-access pattern, which are central to IncAgg's efficiency objective.

Method	Per-iteration aggregation scope	Memory/context source	Update or refresh pattern	Main design emphasis
IncAgg (ours)	IB 1-hop subgraph; avoids per-iteration full-neighborhood aggregation.	Periodically refreshed global aggregated memory provides out-of-batch context.	Global context is refreshed infrequently, e.g., once per epoch in our implementation, and reused across iterations.	Reduce aggregation cost and memory movement while preserving the accuracy of memory-enhanced training.
GAS [7]	Full 1-hop neighborhood of the target batch.	Full-graph embedding histories are fetched for out-of-batch neighbors.	Histories are overwritten with fresh embeddings during training iterations.	Approximate full-neighborhood message passing in mini-batch training.
GraphFM [17]	Full 1-hop neighborhood of the target batch.	Feature memories are used to refine out-of-batch representations.	Memory is maintained at the iteration level, with additional refinement compared with GAS.	Improve the quality of memory-enhanced approximation, often with extra maintenance cost.
REST [15]	Full-neighborhood aggregation with additional memory/refinement steps.	Refined historical representations are used to improve stale-memory correction.	Memory/context is further updated or refined through extra propagation-style operations.	Strengthen predictive performance by improving memory quality.
VR-GCN [2]	Sampled mini-batch aggregation with historical control-variate correction.	Historical activations serve as control variates for variance reduction.	Histories are updated during training for visited nodes.	Reduce sampling variance in mini-batch GNN training.

of approximate message passing, but it often retains full-neighborhood aggregation or introduces additional memory-maintenance operations during training.

IncAgg takes a different route. Instead of further refining per-iteration out-of-batch histories, it changes the aggregation scope of each training iteration: expensive message passing is restricted to the in-batch subgraph, while global context is incorporated through a periodically refreshed memory. This design directly targets the computation and data-movement bottlenecks of memory-enhanced GNN training, making the objective of IncAgg complementary to prior memory-refinement approaches.

F Additional Comparisons and Staleness Analysis

F.1 Matched end-to-end comparison with recent memory-enhanced methods

We run matched Reddit-GCN experiments with random 5% batches, validation-only model selection, and three seeds, including the recent memory-enhanced method REST [15]. End-to-end (E2E) time includes loading, batching, training, refresh, validation, and checkpointing. Time to target (TTT) ends when validation accuracy first reaches the predeclared 94.31% historical GAS mean; it does not affect model selection. As shown in Table 24, IncAgg improves accuracy by 0.70/0.60 points and reduces E2E time by 5.71×/8.67× relative to GAS/REST, with the lowest device-level GPU peak.

F.2 Controlled comparison with modern sampling-based training

IncAgg is orthogonal to target-batch construction: random or METIS batching first selects a target set $\mathcal{B}$; neighbor sampling (NS) and LABOR [1] then expand multi-layer neighborhoods online, whereas

Table 24: Matched end-to-end comparison on Reddit-GCN (random 5% batches, three seeds, mean±sample SD). *GAS TTT uses the two seeds that reach the target.

Method	Test acc. (%) ↑	E2E (s) ↓	Reached / TTT (s) ↓	GPU allocator / device (GB) ↓	Host (GB) ↓
GAS	94.284±0.032	3108.3±51.5	2/3; 2645.0±50.8*	11.53 / 19.44	31.42
REST	94.375±0.029	4715.9±17.3	3/3; 3953.6±262.2	11.53 / 21.08	36.60
IncAgg	**94.980**±0.044	**544.1**±27.7	3/3; **87.7**±4.2	14.86 / **16.95**	36.61

IncAgg represents historical out-of-batch context with a cached global slice and propagates current residuals only through $\mathbf{P}_{\mathcal{B},\mathcal{B}}$. We use mean-aggregation GraphSAGE on Reddit. For each of three workload seeds, 20 fixed 5%-target batches cover all 232,965 nodes once, and each run is repeated three times. Table 25 reports median totals over the identical target stream, including IncAgg's refresh. All pipelines reach 96.4–96.5% accuracy; NS/LABOR materialize 16.91×/8.18× as many source-node instances and 16.39×/10.50× as many edges, and IncAgg processes the workload 8.90× faster than NS and 2.08× faster than LABOR. This comparison is limited to Reddit and reports logical transfer volume rather than peak memory.

Table 25: Controlled comparison with sampling-based pipelines on Reddit (mean-GraphSAGE). *Pipeline time includes construction, transfer, computation, minor overhead, and IncAgg's periodic refresh.

Pipeline	Test acc. (%)	Source nodes	Edges	Logical H2D / D2H (GB)	Pipeline time* (s)
IncAgg	96.4±0.2	232,965 (1.00×)	1,160,806 (1.00×)	4.391 / 3.817	**1.9675**
NS (10,10)	96.5±0.028	3,939,724 (16.91×)	19,023,111 (16.39×)	9.717 / 0	17.5052 (8.90×)
LABOR (10,10)	96.5±0.017	1,904,994 (8.18×)	12,193,714 (10.50×)	4.730 / 0	4.0865 (2.08×)

F.3 Data movement

We trace one complete seed-0 Reddit-GCN epoch with random 5% batches after one warm-up epoch (Table 26). GAS repeatedly fetches out-of-batch histories, whereas IncAgg fetches only batch-local slices of $\mathbf{M}_L$ and $\mathbf{M}_G$. Its refresh adds 1.908 GB of device-to-host (D2H) traffic but removes 50.122 GB of host-to-device (H2D) traffic, reducing total traffic by 46.5% and the transfer time exposed outside GPU computation by 42.9%.

Table 26: Data movement per traced epoch on Reddit-GCN (random 5% batches).

Per traced epoch	GAS	IncAgg
H2D traffic (GB)	99.844	49.722
D2H traffic (GB)	3.862	5.770
Total traffic (GB)	103.706	**55.492**
Non-overlapped transfer (s)	2.478	**1.416**

F.4 Operator-level timing

To explain why the linear transformation can take longer than aggregation in Table 5, we profile the middle layer of a three-layer Reddit GCN with random 5% batches (one mini-batch iteration, averaged over 20 synchronized repeats after five warm-ups). The dense transformation performs about 174× more arithmetic than the sparse aggregation (Table 27), which explains its longer time despite the irregular memory access of aggregation.

F.5 Depth, refresh frequency, and memory staleness

Depth can compound representation drift between global refreshes, while more frequent refreshes produce fresher memory. We evaluate GCNII on ogbn-arxiv at depths 4 and 8. The default $f = 1$ refreshes once per epoch; $f = 2, 4$ refresh more frequently. Global error is the normalized Frobenius distance between the cached global aggregation and its recomputation from current embeddings (lower is fresher). As shown in Table 28, at the default $f = 1$ IncAgg exceeds GAS by 1.32/2.94

Table 27: Operator profile of one middle GCN layer on Reddit.

Operator	Workload	FLOPs	Time (ms)
SpMM	1,179,125 nonzeros; hidden 1024	2.415 G	10.583
GEMM	$200{,}626 \times 1024$ by 1024×1024	420.743 G	21.754

points at depths 4/8. Increasing depth changes IncAgg's accuracy by only -0.02 points, versus -1.64 points for GAS. Raising f from 1 to 4 reduces global error by about 69%, while accuracy changes by only $+0.56$/$+0.48$ points. Thus the default remains accurate under the measured staleness, while the refresh frequency directly controls memory freshness.

Table 28: GCNII on ogbn-arxiv: accuracy (%, mean±sample SD over three seeds) under different depths and refresh frequencies f. *Global error is measured on seed 0.

Depth	GAS	IncAgg ($f = 1$)	IncAgg ($f = 2$)	IncAgg ($f = 4$)	Global error*, $f = 1 \to 4 \downarrow$
4	70.22±0.35	71.54±0.27	72.02±0.06	72.10±0.43	$0.396 \to 0.122$
8	68.58±1.40	71.52±0.29	71.54±0.28	72.00±0.49	$0.374 \to 0.113$

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: The abstract and introduction summarize the main contributions of the paper, including the proposed IncAgg framework, its efficiency motivation, and its empirical evaluation. The claims are supported by the method description, complexity analysis, and experimental results in the main paper and appendix.

   Guidelines:

   - The answer [N/A] means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A [No] or [N/A] answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: The Conclusion states the scope of this work and future directions; Section 3.2 states the scope of the exact residual identity; Appendix F.5 analyzes staleness.

   Guidelines:

   - The answer [N/A] means that the paper has no limitation while the answer [No] means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

Answer: [Yes]

Justification: The paper states the assumptions used in the theoretical analysis and provides the detailed derivations and proofs in the appendix. The main paper gives the intuition and key conclusions, while the appendix contains the complete technical details.

Guidelines:

- The answer [N/A] means that the paper does not include theoretical results.
- All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
- All assumptions should be clearly stated or referenced in the statement of any theorems.
- The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
- Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
- Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

Answer: [Yes]

Justification: The paper provides the datasets, baselines, model backbones, partitioning settings, batch-size settings, memory refresh settings, optimizer, and hyperparameter search space used in the experiments. Additional implementation details and sensitivity analyses are included in the appendix.

Guidelines:

- The answer [N/A] means that the paper does not include experiments.
- If the paper includes experiments, a [No] answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
- If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
- Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
- While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
  (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
  (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
  (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).

(d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

   Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

   Answer: [No]

   Justification: The experiments use publicly available benchmark datasets and standard open-source libraries, and the paper provides implementation details and hyperparameter settings. We plan to release the implementation and reproduction scripts after the review process, but an anonymized code package is not included with the initial submission.

   Guidelines:

   - The answer [N/A] means that paper does not include experiments requiring code.
   - Please see the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - While we encourage the release of code and data, we understand that this might not be possible, so [No] is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
   - The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
   - The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
   - At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
   - Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer) necessary to understand the results?

   Answer: [Yes]

   Justification: The experimental section and appendix specify the training and evaluation settings, including datasets, model backbones, baselines, optimizer, hyperparameter search space, batch sizes, partitioning strategies, and memory refresh frequencies.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
   - The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [Yes]

Justification: The main accuracy results report the mean and standard deviation over multiple runs. We state the source of variability and report these statistics for the experiments that support the main accuracy claims.

Guidelines:

- The answer [N/A] means that the paper does not include experiments.
- The authors should answer [Yes] if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
- The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
- The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
- The assumptions made should be given (e.g., Normally distributed errors).
- It should be clear whether the error bar is the standard deviation or the standard error of the mean.
- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g., negative error rates).
- If error bars are reported in tables or plots, the authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: The appendix reports the hardware and software environment used for the experiments, including the GPU/CPU resources and memory configuration. The paper also reports per-epoch runtime and speedup results for the main efficiency comparisons.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes]

   Justification: The research follows the NeurIPS Code of Ethics. The paper uses public benchmark datasets and standard evaluation protocols, and does not involve private user data, human-subject studies, or high-risk deployment.

   Guidelines:

   - The answer [N/A] means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer [No], they should explain the special circumstances that require a deviation from the Code of Ethics.

- The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [N/A]

    Justification: This work is a foundational algorithmic study on improving the efficiency of scalable GNN training and is not tied to a specific deployment or application domain. Any broader societal impacts are indirect and depend on downstream uses of graph learning systems.

    Guidelines:

    - The answer [N/A] means that there is no societal impact of the work performed.
    - If the authors answer [N/A] or [No], they should explain why their work has no societal impact or why the paper does not address societal impact.
    - Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
    - The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate Deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
    - The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
    - If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pre-trained language models, image generators, or scraped datasets)?

    Answer: [N/A]

    Justification: The paper does not release high-risk pretrained models, image generators, language models, or newly scraped datasets. The experiments are conducted on standard public graph benchmark datasets.

    Guidelines:

    - The answer [N/A] means that the paper poses no such risks.
    - Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
    - Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
    - We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

Answer: [Yes]

Justification: The paper uses publicly available benchmark datasets, open-source software libraries, and baseline implementations, and cites the corresponding original sources. We use these existing assets for academic research purposes according to their respective licenses and terms of use.

Guidelines:

- The answer [N/A] means that the paper does not use existing assets.
- The authors should cite the original paper that produced the code package or dataset.
- The authors should state which version of the asset is used and, if possible, include a URL.
- The name of the license (e.g., CC-BY 4.0) should be included for each asset.
- For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
- If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
- For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

Answer: [N/A]

Justification: The paper does not introduce a new dataset, benchmark, or pretrained model. The contribution is an algorithmic training framework evaluated on existing public benchmarks.

Guidelines:

- The answer [N/A] means that the paper does not release new assets.
- Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
- The paper should discuss whether and how consent was obtained from people whose asset is used.
- At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

Answer: [N/A]

Justification: The paper does not involve crowdsourcing, user studies, surveys, or research with human subjects.

Guidelines:

- The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.

- Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
- According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

    Answer: [N/A]

    Justification: The paper does not involve human-subject research, crowdsourcing, or collection of data from study participants.

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
    - Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
    - We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
    - For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

    Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does *not* impact the core methodology, scientific rigor, or originality of the research, declaration is not required.

    Answer: [N/A]

    Justification: The core method development in this research does not involve LLMs as important, original, or non-standard components. Any use of writing or editing assistance does not affect the methodology, scientific claims, or experimental results.

    Guidelines:

    - The answer [N/A] means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
    - Please refer to our LLM policy in the NeurIPS handbook for what should or should not be described.