# IMPLICIT REGULARIZATION OF MINI-BATCH TRAINING IN GRAPH NEURAL NETWORKS

**Anonymous authors**
Paper under double-blind review

## ABSTRACT

Mini-batch training of Graph Neural Networks (GNNs) is fundamentally different from training on i.i.d. data: sampling a subgraph alters the topology and introduces boundary effects, leading prior work to develop structure-aware samplers that preserve local connectivity and reduce embedding variance. Surprisingly, we demonstrate that the simplest possible scheme, Random Node Sampling (RNS) — training on the induced subgraph of uniformly sampled nodes — matches or outperforms full-graph training on most datasets at a fraction of the wall-clock time and memory. To explain this, we apply backward error analysis to graph mini-batch Stochastic Gradient Descent (SGD) and show that it implicitly minimizes the sampled loss plus a regularizer proportional to the mini-batch gradient variance, a quantity directly shaped by the sampler. Although RNS discards local structure, it produces mini-batches with a comparable loss bias but much lower per-batch gradient variance, yielding a better implicit objective. Our analysis reframes the choice of graph sampler as a form of implicit regularization, and identifies RNS as a strong, theoretically grounded method for scalable GNN training.

## 1 INTRODUCTION

Graph Neural Networks (GNNs) are commonly trained transductively, with the full graph available during optimization (Kipf & Welling, 2017; Veličković et al., 2018). While conceptually simple, full-graph training becomes costly at scale: message passing couples many node representations, increasing memory and compute requirements and limiting the number of optimization steps possible under a fixed budget. This has motivated sampling-based and mini-batch training methods for GNNs (Hamilton et al., 2017; Zeng et al., 2020).

Graph mini-batching, however, is fundamentally different from mini-batching i.i.d. examples. Since node predictions depend on neighboring features and edges, training on a sampled subgraph changes the interactions available to the model and introduces boundary effects. The sampler is therefore not merely an implementation detail: it changes the stochastic gradients, and hence the effective learning problem. This raises a basic question: when a GNN is trained on sampled subgraphs, what objective is actually optimized?

This paper studies *Random Node Sampling* (RNS) for transductive node classification. At each iteration, RNS samples nodes uniformly at random and trains on the induced subgraph. RNS has recently appeared in graph transformers (Wu et al., 2022; 2023; Deng et al., 2024) and message-passing GNNs (Luo et al., 2024), but mainly as a practical mechanism for reducing memory and runtime. As a result, its effect on predictive performance has remained entangled with architectural and optimization choices.

Beyond computational savings, graph mini-batching introduces a form of implicit regularization that has received comparatively little attention. Prior work has focused on the subgraph approximation effect — whether sampled subgraphs faithfully recover the full-graph objective — while the optimization effect induced by stochastic updates on structured batches has largely been overlooked. Using backward error analysis, we disentangle these two effects under SGD and show that training on sampled subgraphs implicitly minimizes a modified objective: the sampled loss augmented by a regularizer proportional to the variance of mini-batch gradients. We then show that RNS is distinctive in this regard — despite constructing batches without any knowledge of graph structure,



Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

it produces per-batch gradients whose statistics closely match those of the full graph, yielding a lower-variance, better-controlled implicit objective than structure-aware alternatives.

**Contributions.**

- **Theory.** We apply backward error analysis to GNN mini-batch training and prove that SGD over sampled subgraphs implicitly minimizes a modified objective combining the sampled loss with a gradient-variance regularizer shaped by the sampler. We show that RNS induces a well-controlled bias relative to $L_{\text{full}}$ and a gradient-variance regularization term orders of magnitude smaller than that of structure-based samplers, yielding a more stable implicit objective.
- **Benchmark.** Across five sampling strategies, three large-scale graphs, and four GNN architectures, RNS leads on both performance and speed. On the more realistic GraphLand datasets, it remains competitive with full-graph training on most benchmarks.
- **Practical impact.** RNS is a drop-in replacement for full-graph training requiring only a single integer hyperparameter $m$. It achieves $2\times$ to $12\times$ wall-clock speedups and up to $3\times$ lower peak GPU memory, making it a strong default for transductive GNN training.

## 2 Related work

**GNNs and large-scale training.** Scaling GNNs to large graphs relies on mini-batch training methods. Neighborhood sampling recursively samples local neighborhoods (Hamilton et al., 2017; Ying et al., 2018), while layer-wise methods such as FastGCN (Chen et al., 2018), LADIES (Zou et al., 2019), and LABOR (Balin & Çatalyürek, 2023) reduce cost by sampling nodes at each layer. Subgraph-based methods such as ClusterGCN (Chiang et al., 2019) and GraphSAINT (Zeng et al., 2020) process graph partitions or induced subgraphs as mini-batches. Random node sampling has recently appeared as a practical mechanism in graph transformers (Wu et al., 2022; 2023; Deng et al., 2024) and message-passing GNNs (Luo et al., 2024), but its effect on optimization dynamics has not been systematically analyzed. These methods are primarily motivated by efficiency; how the resulting mini-batch noise shapes the effective training objective remains an open question.

**Optimization dynamics and positioning.** A complementary line of work studies stochastic optimization in deep learning via continuous-time approximations, modeling SGD using stochastic differential equations (Mandt et al., 2017; Smith & Le, 2018; Jastrzebski et al., 2018; Chaudhari & Soatto, 2018) or backward analysis (Hairer et al., 2006; Barrett & Dherin, 2021; Cattaneo et al., 2024; Ghosh et al., 2023). Smith et al. (2021) applies backward analysis to SGD and shows that its iterates track the gradient flow of a modified objective, which can be interpreted as the original loss augmented with an implicit regularizer. These frameworks clarify how discretization and stochasticity induce implicit regularization, but they are typically developed under i.i.d. data assumptions. In contrast, we analyze the *structured* stochasticity induced by graph mini-batch training, and we characterize how Random Node Sampling modifies the effective training objective of GNNs.

## 3 Setting, notation, and sampling strategies

We study *transductive* node-level learning on a graph $G = (V, E)$, where each node $v \in V$ has features $x_v \in \mathbb{R}^d$ and a label $y_v$. The node set is split into disjoint subsets

$$V = V^{\text{train}} \dot\cup V^{\text{val}} \dot\cup V^{\text{test}}.$$

In the transductive regime, the learner has access to the full graph structure $E$ and all node features $X = \{x_v\}_{v \in V}$ during training, but the supervised loss is evaluated only on training nodes.

Let $f(w, ., .)$ denote a graph neural network (GNN) with parameters $w$. For a node $v$, we write $f(w, X, E)[v]$ the model output at node $v$ when evaluated on the full graph. Using a generic per-node loss $\ell(\cdot, \cdot)$, *full-graph training* minimizes

$$L_{\text{full}}(w) \;=\; \frac{1}{|V^{\text{train}}|} \sum_{v \in V^{\text{train}}} \ell\big(f(w, X, E)[v],\, y_v\big). \tag{1}$$

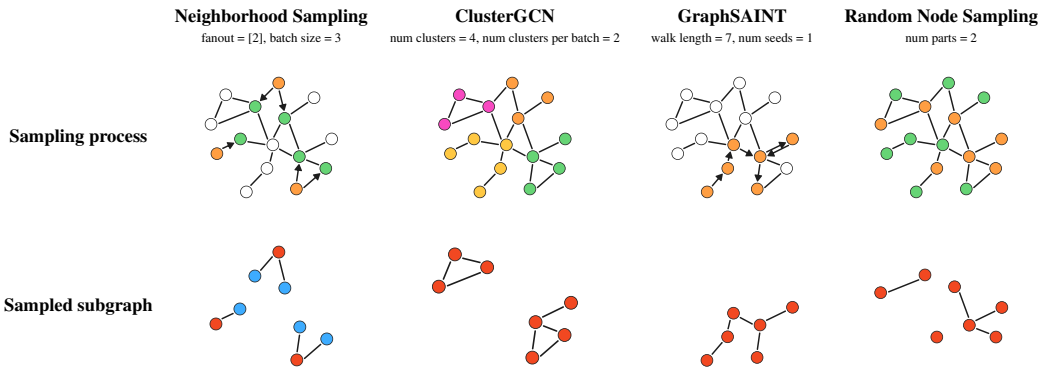



Figure 1: **Batch sampling strategies for GNN training.** Top: mini-batch construction for each method. Bottom: resulting sampled subgraph. Red nodes are targets ($B^{\text{train}}$), while blue nodes are auxiliary nodes used only for message passing ($B \setminus B^{\text{train}}$). See Appendix F.1.1 for details.

**Mini-batch subgraph training.** To scale to large graphs, one often replaces the full-graph forward pass by a sampled subgraph. A mini-batch is a subgraph

$$G_B = (B, E_B), \qquad X_B = \{x_v\}_{v \in B},$$

together with a set of *supervised target* nodes $B^{\text{train}} \subseteq B \cap V^{\text{train}}$ on which the loss is computed. The corresponding mini-batch objective is

$$\hat{L}^B_{\text{batch}}(w) \;=\; \frac{1}{|B^{\text{train}}|} \sum_{v \in B^{\text{train}}} \ell\big(f(w, X_B, E_B)[v],\, y_v\big). \tag{2}$$

**Random node sampling (RNS).** In this work we focus on *Random Node Sampling* (RNS). At the start of each epoch, we randomly split the node set $V$ into $m$ disjoint subsets $B_0, \ldots, B_{m-1}$ of (approximately) equal size. Each training step then processes one subset $B_k$: we form the mini-batch graph as the induced subgraph $G[B_k]$, and compute the supervised loss only on the labeled training nodes within that batch, i.e., $B_k^{\text{train}} = B_k \cap V^{\text{train}}$ (Alg. 1).

**Other sampling strategies.** Common alternatives include Neighborhood Sampling (Hamilton et al., 2017), ClusterGCN (Chiang et al., 2019), and GraphSAINT (Zeng et al., 2020) (see Figure 1 and Appendix F.1.1). These approaches construct mini-batches using local expansions, graph partitions, or subgraph sampling. We refer to the original papers for details.

Unlike other samplers, RNS does not explicitly preserve local structure: nodes typically have incomplete neighborhoods compared to $G$. One might therefore expect its stochastic gradients to differ substantially from those induced by structure-based mini-batches. We show, however, that despite this local sparsification, RNS yields mini-batch losses that closely approximate the full-graph loss, with low bias and variance.

# 4 On the implicit learning objective of GNN training with backward analysis

## 4.1 Backward analysis of GNN training

Backward analysis interprets a discrete optimization algorithm not as an approximation of a continuous-time gradient flow, but as the *exact* discretization of a *modified* flow. The modified system captures the cumulative effect of discretization errors and reveals implicit biases induced by the algorithm.

**Gradient descent as a discretized ODE.** Gradient descent with step size $\epsilon > 0$ on a loss $L(w)$,

$$w_{k+1} = w_k - \epsilon \nabla L(w_k), \tag{3}$$

is the forward Euler discretization of the gradient flow $\dot{w}(t) = -\nabla L(w(t))$. For finite $\epsilon$, discretization errors accumulate and the discrete trajectory no longer follows the exact flow. Backward analysis seeks a modified vector field $\tilde{f}$ such that the iterates $\{w_k\}$ lie exactly on the flow of $\dot{w} = \tilde{f}(w)$, up to a given order in $\epsilon$. Assuming an asymptotic expansion $\tilde{f}(w) = f(w) + \epsilon\, g(w) + O(\epsilon^2)$ with $f(w) = -\nabla L(w)$, the correction $g$ encodes an *implicit regularization* effect induced by the finite step size (Hairer et al., 2006; Smith et al., 2021). When $\tilde{f}$ is conservative, i.e. $\tilde{f} = -\nabla \tilde{L}$, gradient descent can be interpreted as minimizing a modified loss $\tilde{L}$ rather than $L$.

**Known results in standard deep learning.** For the empirical risk $L(w) = \frac{1}{N}\sum_v \ell(f(w, x_v), y_v)$ defined over i.i.d. samples, backward error analysis shows that, up to terms of order $O(\epsilon^3)$, full-batch Gradient Descent (GD) and mini-batch Stochastic Gradient Descent (SGD) respectively optimize the following modified losses (Barrett & Dherin, 2021):

$$\tilde{L}_{\mathrm{GD}}(w) = L(w) + \tfrac{\epsilon}{4}\,\|\nabla L(w)\|^2, \tag{4}$$

$$\tilde{L}_{\mathrm{SGD}}(w) = L(w) + \tfrac{\epsilon}{4}\,\|\nabla L(w)\|^2 + \tfrac{\epsilon}{4m}\sum_{k=0}^{m-1}\|\nabla \hat{L}_k(w) - \nabla L(w)\|^2, \tag{5}$$

where $\hat{L}_k$ denotes the loss on mini-batch $k$. The first correction penalizes large gradients and favors flatter regions of the loss landscape, which generally correspond to parameters with better generalization performance (Hochreiter & Schmidhuber, 1997). The second reflects gradient variance across batches and vanishes in the full-batch limit.

**Full-graph GNN training.** We extend the backward analysis of Barrett & Dherin (2021) from i.i.d. empirical-risk minimization to full-graph GNN training. The modified-loss construction is agnostic to the source of coupling in the loss, relying only on the additivity of the training objective and gradient-descent dynamics, both of which are preserved under GNN training. Thus, backward analysis applies directly to gradient descent on $L_{\mathrm{full}}$. Eq. 1 yields the modified objective derived in Appendix A.3:

$$\tilde{L}_{\mathrm{full}}(w) = L_{\mathrm{full}}(w) + \tfrac{\epsilon}{4}\left\|\nabla L_{\mathrm{full}}(w)\right\|^2. \tag{6}$$

The functional form matches the i.i.d. result, but the geometry of $L_{\mathrm{full}}$ is shaped by the graph: node losses are coupled through the topology and the receptive fields of the GNN.

**Sampled-subgraph mini-batch training.** The same argument applies to any mini-batch GNN training scheme based on subgraph sampling: the resulting modified loss does not depend on the specific sampling rule used to construct each batch, only on the per-batch losses it induces. We therefore state the result generically, denoting by $\hat{L}^{B_k}_{\mathrm{Sampl}}$ the loss computed on batch $B_k$ obtained from an arbitrary sampler $\mathrm{Sampl}$ (e.g., RNS, neighbor sampling, Cluster-GCN, GraphSAINT). Processing $m$ batches $\{B_0, \ldots, B_{m-1}\}$ once constitutes one epoch, with update

$$w_{t+m} = w_t - \epsilon \sum_{k=0}^{m-1} \nabla \hat{L}^{B_k}_{\mathrm{Sampl}}(w_{t+k}). \tag{7}$$

Define the epoch-averaged partial loss $\bar{L}_{\mathrm{Sampl}}(w) = \frac{1}{m}\sum_{k=0}^{m-1} \hat{L}^{B_k}_{\mathrm{Sampl}}(w)$. In expectation over the batch order in the epoch, the effective modified objective becomes (Appendix A.4)

$$\tilde{L}_{\mathrm{Sampl}}(w) = \bar{L}_{\mathrm{Sampl}}(w) + \tfrac{\epsilon}{4}\left\|\nabla \bar{L}_{\mathrm{Sampl}}(w)\right\|^2 + \underbrace{\tfrac{\epsilon}{4}\cdot\tfrac{1}{m}\sum_{k=0}^{m-1}\left\|\nabla \hat{L}^{B_k}_{\mathrm{Sampl}}(w) - \nabla \bar{L}_{\mathrm{Sampl}}(w)\right\|^2}_{R(w)}. \tag{8}$$

**Interpretation.** In both cases, a finite step size does not only optimize the nominal loss, it introduces an implicit bias acting as a regularizer. In the full-graph case, the dynamics are biased toward small-gradient regions of $L_{\mathrm{full}}$. In the sampled-subgraph case, two effects compound. First, the optimized objective is itself biased, since in general $\bar{L}_{\mathrm{Sampl}} \neq L_{\mathrm{full}}$: training minimizes an empirical average over sampled subgraphs rather than the full-graph loss. Second, the additional term $R(w)$ is

the variance of batch gradients, and arises specifically from chaining SGD steps on distinct batches within an epoch. It favors parameters whose updates are stable under graph sampling.

Equation 8 yields two testable predictions. First, removing chained per-batch updates should suppress $R(w)$ while preserving the sampling bias in $\bar{L}_{\mathrm{Sampl}}$. Second, because the result is sampler-agnostic, different samplers should differ through the bias and variance terms induced by their batches. We test the first prediction in Section 4.2 and quantify the second across samplers in Section 4.3.

### 4.2 Empirical evaluation of the implicit regularization with RNS

Table 1: **Test accuracy on OGBN-PRODUCTS (in %).** Mean $\pm$ 95% CI over 5 seeds. Both implicit regularization sources are necessary: sampling bias alone (RNS no batch) does not suffice, and neither does chained updating alone (full graph).

| Training regime | SGD (%) | SGD+M (%) | Adam (%) |
|---|---|---|---|
| Full graph | $72.04 \pm 0.11$ | $78.89 \pm 0.13$ | $80.08 \pm 0.26$ |
| RNS (no batch) | $70.94 \pm 0.15$ | $76.84 \pm 0.40$ | $83.01 \pm 0.26$ |
| RNS | $\mathbf{77.70 \pm 0.29}$ | $\mathbf{82.60 \pm 0.20}$ | $\mathbf{83.15 \pm 0.16}$ |

We test the first prediction by training GraphSAGE on OGBN-PRODUCTS under three regimes:

- **Full graph**: standard full-graph training, implicitly minimizing $L_{\mathrm{full}} + \frac{\epsilon}{4}\|\nabla L_{\mathrm{full}}\|^2$.
- **RNS (no batch)**: RNS batches are sampled, but their gradients are accumulated into one update, implicitly minimizing $\bar{L}_{\mathrm{RNS}} + \frac{\epsilon}{4}\|\nabla \bar{L}_{\mathrm{RNS}}\|^2$ and removing $R(w)$.
- **RNS**: standard mini-batch training with one update per batch, exposing both the sampling bias and $R(w)$ as in equation 8.

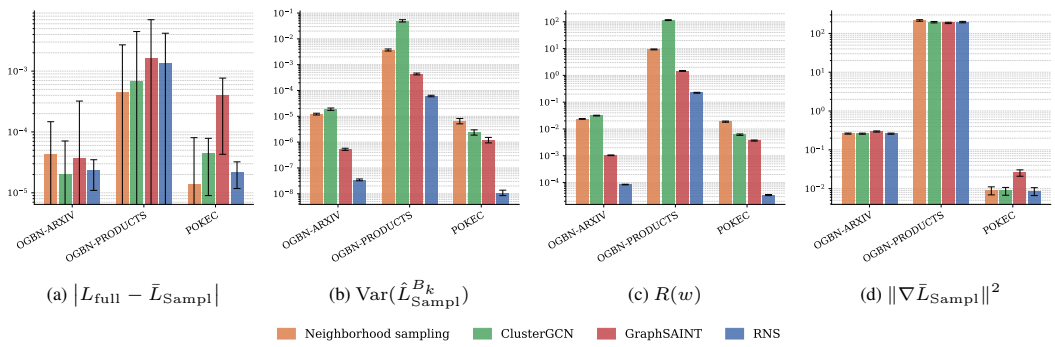
We run each regime with SGD, SGD with momentum (SGD+M), and Adam. Details are in Appendix F.2.

Table 1 reports test accuracy across regimes and optimizers. Replacing $L_{\mathrm{full}}$ with $\bar{L}_{\mathrm{RNS}}$ changes the optimization problem by removing edges between nodes assigned to different batches, thereby injecting sampling noise into training. With Adam, this perturbation improves accuracy, suggesting that the induced noise acts as a useful regularizer and that Adam's adaptive gradient normalization may help exploit it. In contrast, for SGD and SGD+M, RNS (no batch) underperforms the full-graph baseline, indicating that the same sampling-induced noise can hinder optimization when updates are not adaptively preconditioned. RNS training recovers this loss and exceeds full-graph accuracy across optimizers. This shows that the regularization effect predicted by equation 8 becomes beneficial when training performs chained mini-batch updates. Overall, the results indicate that both sources of implicit regularization, the sampling bias in $\bar{L}_{\mathrm{RNS}}$ and the chaining effect captured by $R(w)$, are empirically visible in the performance and jointly necessary for the strong performance of RNS. The theory–practice gap for Adam is further discussed in Section 6.1.

### 4.3 On the particularity of RNS implicit regularization

We measure the implicit regularization induced by mini-batch samplers at random initialization, before optimization can shape it. We use each sampler's best hyperparameters (Table 7) and compute statistics from 50 sampled batches over 100 random seeds. Figure 2 shows three consistent patterns: all samplers induce a similar loss bias $|L_{\mathrm{full}} - \bar{L}_{\mathrm{Sampl}}|$ while per-batch loss variance and gradient variance $R(w)$ differ by orders of magnitude, with RNS lowest in both and the averaged-objective gradient norm $\|\nabla \bar{L}_{\mathrm{Sampl}}\|^2$ is similar across methods.

These differences follow from how each method constructs its batches. For RNS, the bias of the implicit objective contains no node-subsampling term. What remains is an edge-removal term, which for message-passing GNNs we bound by a product of a homophily term, a density term, and a factor $(1 - 1/m)$ measuring the fraction of edges cut (Appendix B). We also derive a bound on the batch gradient variance $R$ that is linear in $m - 1$ (Appendix C), so both sources of error are controlled by




(a) $\left|L_{\text{full}} - \bar{L}_{\text{Sampl}}\right|$ (b) $\text{Var}(\hat{L}^{B_k}_{\text{Sampl}})$ (c) $R(w)$ (d) $\|\nabla \bar{L}_{\text{Sampl}}\|^2$

Figure 2: Implicit regularization terms induced by different mini-batch samplers at random initialization. Statistics computed from 50 batches. Mean $\pm$ 95% CI over 100 seeds.

the single parameter $m$. Finally, each RNS mini-batch preserves the power-law degree tail of the full graph (Appendix D).

Since the best configurations lead to different batch sizes across samplers, we also compare them over a sweep of sampling budgets (Appendix G.4). This also asks whether a sampler can deliver low variance and faithful structure together, since each is easy to obtain alone. RNS attains the lowest $R$ in every configuration we tested while matching the full-graph degree tail on the larger graphs. The baselines sit in the opposite regime, reaching comparable structural fidelity only at substantially higher variance, each for its own reason: METIS partitions make ClusterGCN's clusters systematically dissimilar, Neighborhood Sampling confines supervision to seed nodes, and GraphSAINT over-samples high-degree nodes.

Two consequences follow. First, the implicit objective is sampler-dependent: bias, loss variance, and gradient variance depend strongly on the sampler, so different samplers solve different learning problems. Second, because evaluation uses the full graph, useful batch gradients should approximate those of $L_{\text{full}}$. Among the methods considered, RNS does this most directly, combining relatively low bias, low gradient variance, and unbiased per-batch estimates of full-graph node statistics. In the next section, we empirically benchmark these sampling methods to assess the practical consequences of their different implicit biases.

## 5 Experiments and Results

### 5.1 Benchmark against other sampling-based training

We benchmark on three commonly used large-scale graph datasets (OGBN-ARXIV, OGBN-PRODUCTS, and POKEC), all using a GraphSAGE backbone. We tune hyperparameters for each dataset and each method separately with Bayesian search. Full implementation details and hyperparameters are reported in Appendix F.1. As shown in Table 2, RNS achieves the best test accuracy on all three datasets while training faster and using comparable GPU memory.

On OGBN-PRODUCTS, all sampling methods match or slightly exceed full-graph training, and the gaps between samplers are small. On OGBN-ARXIV and POKEC, the baseline samplers fall behind full-graph training, while RNS exceeds it. Graph structure offers a natural explanation for this split. OGBN-PRODUCTS is strongly homophilic and locally dense, so a sampled subgraph retains most of the label-relevant signal, and topology-aware samplers are well suited to it. OGBN-ARXIV and POKEC are less homophilic, so restricting message passing to a sampled subgraph can discard useful long-range information and bias the gradients. RNS avoids this by construction. Like any subgraph sampler it removes edges, but it removes them uniformly at random rather than along structurally meaningful boundaries, so no node is systematically deprived of long-range context, and the partition is redrawn at every epoch.

[1]All methods are trained for 1000 epochs (2000 for POKEC). Neighborhood Sampling for 20, as its CPU-based sampling and small batches make each epoch an order of magnitude slower. (See Appendix F).

Table 2: **Full-graph vs. sampling-based training.** Accuracy (%, mean ± 95% CI over 5 seeds), time, and peak memory [1]. RNS is faster and more accurate, with competitive memory use.

| Dataset | Method | Accuracy (%) | Runtime (s) | GPU (GB) | RAM (GB) |
|---|---|---|---|---|---|
| OGBN-ARXIV | Full | *72.28 ± 0.38* | *2m00s* | 3.94 | 1.53 |
| | Neighborhood sampling | 71.08 ± 0.47 | 6m49s | 1.61 | 1.67 |
| | ClusterGCN | 71.67 ± 0.12 | 9m00s | 1.60 | 1.68 |
| | SAINT | 71.38 ± 0.09 | 6m00s | 1.60 | 1.59 |
| | LADIES | 68.86 ± 0.26 | 7m07s | 2.43 | 1.60 |
| | RNS | **72.51 ± 0.19** | **1m40s** | 2.38 | 1.51 |
| OGBN-PRODUCTS | Full | 80.42 ± 0.50 | *45m50s* | 62.06 | 1.58 |
| | Neighborhood sampling | 80.56 ± 0.28 | 3h32m43s | 24.59 | 10.51 |
| | ClusterGCN | 81.51 ± 0.57 | 55m50s | 19.89 | 11.55 |
| | SAINT | *82.71 ± 0.18* | 1h37m30s | 19.79 | 4.32 |
| | LADIES | 78.33 ± 0.48 | 4h28m40s | 40.90 | 3.09 |
| | RNS | **83.11 ± 0.17** | **11m30s** | 20.35 | 1.93 |
| POKEC | Full | *83.40 ± 0.23* | 37m10s | 55.43 | 1.63 |
| | Neighborhood sampling | 80.70 ± 0.13 | 4h03m15s | 29.92 | 4.85 |
| | ClusterGCN | 80.97 ± 0.13 | **27m10s** | 10.69 | 6.55 |
| | SAINT | 78.30 ± 0.06 | 1h21m50s | 11.04 | 2.63 |
| | LADIES | 56.09 ± 0.09 | 3h47m38s | 19.96 | 1.70 |
| | RNS | **83.98 ± 0.08** | *29m30s* | 28.66 | 2.17 |

Table 3: **GraphLand results.** Full-graph vs. RNS performance under random-low (RL) and random-high (RH) regimes. Multiclass: accuracy; binary: average precision (%). Mean ± 95% CI.

| Dataset | RL | | | RH | | |
|---|---|---|---|---|---|---|
| | Full | RNS | Δ | Full | RNS | Δ |
| **Multiclass classification** | | | | | | |
| HM-CATEGORIES | $\mathbf{57.41 \pm 0.34}$ | $57.05 \pm 0.56$ | $-0.36$ | $\mathbf{72.86 \pm 0.34}$ | $72.33 \pm 0.26$ | $-0.53$ |
| POKEC-REGIONS | $37.12 \pm 1.09$ | $\mathbf{41.25 \pm 0.22}$ | $+4.13$ | $38.20 \pm 0.56$ | $\mathbf{47.79 \pm 0.50}$ | $+9.59$ |
| WEB-TOPICS | $47.39 \pm 0.06$ | $\mathbf{47.76 \pm 0.08}$ | $+0.37$ | $49.29 \pm 0.11$ | $\mathbf{50.01 \pm 0.05}$ | $+0.72$ |
| **Binary classification** | | | | | | |
| TOLOKERS-2 | $\mathbf{54.41 \pm 0.19}$ | $54.41 \pm 0.34$ | $-0.00$ | $58.83 \pm 1.01$ | $\mathbf{61.34 \pm 0.44}$ | $+2.51$ |
| CITY-REVIEWS | $77.87 \pm 0.09$ | $\mathbf{78.05 \pm 0.05}$ | $+0.18$ | $80.65 \pm 0.08$ | $\mathbf{80.84 \pm 0.21}$ | $+0.19$ |
| ARTNET-EXP | $42.47 \pm 0.77$ | $\mathbf{42.78 \pm 0.42}$ | $+0.31$ | $48.50 \pm 0.25$ | $\mathbf{49.28 \pm 0.31}$ | $+0.78$ |
| WEB-FRAUD | $\mathbf{12.51 \pm 0.04}$ | $11.44 \pm 0.28$ | $-1.07$ | $\mathbf{20.39 \pm 0.32}$ | $20.29 \pm 0.18$ | $-0.09$ |

### 5.2 SPEEDUP

We evaluate wall-clock efficiency using validation time-to-accuracy targets. Across all datasets, targets, and five seeds, both full-graph training and RNS reach every target, and RNS consistently reduces time-to-target, with speedups of about $2\times$ to $12\times$ depending on the dataset and whether startup overhead is included (full decomposition in Appendix G.2).

The speedup has two sources. First, RNS makes epochs cheaper. For a fixed message-passing architecture, the dominant cost is neighborhood aggregation, which scales with the number of edges processed. Full-graph training processes all edges each epoch, whereas RNS partitions nodes into $m$ random batches and runs message passing on the induced subgraphs. An edge falls in a given batch with probability about $1/m^2$, so a batch holds about $|E|/m^2$ edges in expectation and an RNS epoch about $|E|/m$. Message-passing cost thus drops by about a factor $m$, though realized epoch speedups are smaller due to validation, logging, data-loading, optimizer, and initialization overheads.

Second, one RNS epoch contains $m$ optimizer steps versus one for full-graph training, so epochs-to-target speedup reflects faster progress per epoch, not necessarily fewer parameter updates. Ap-

Table 4: **Architecture-wise Full-graph vs. RNS comparison.** Test accuracy (%, mean ± 95% CI over 5 seeds) by dataset and architecture. RNS benefits all message passing architectures. We briefly comment on GAT OOM in Appendix F.4.

| | OGBN-ARXIV | | OGBN-PRODUCTS | | POKEC | |
|---|---|---|---|---|---|---|
| Model | Full | RNS | Full | RNS | Full | RNS |
| GCN | $73.10 \pm 0.22$ | $\mathbf{73.26 \pm 0.28}$ | $77.55 \pm 0.30$ | $\mathbf{80.81 \pm 0.25}$ | $84.62 \pm 0.05$ | $\mathbf{84.98 \pm 0.12}$ |
| GraphSAGE | $72.28 \pm 0.38$ | $\mathbf{72.51 \pm 0.19}$ | $80.42 \pm 0.50$ | $\mathbf{83.11 \pm 0.17}$ | $83.40 \pm 0.23$ | $\mathbf{83.98 \pm 0.08}$ |
| GAT | $72.65 \pm 0.10$ | $\mathbf{72.74 \pm 0.26}$ | OOM | OOM | OOM | OOM |
| SGFormer | $\mathbf{72.54 \pm 0.07}$ | $72.42 \pm 0.16$ | $72.42 \pm 0.25$ | $\mathbf{74.64 \pm 0.15}$ | $\mathbf{77.23 \pm 0.13}$ | $75.38 \pm 0.18$ |

pendix G.2 reports both epoch-time and epochs-to-target speedups. RNS improves both across all targets. Appendix G.3 further compares time and memory with other sampling methods.

### 5.3 Evaluation on GraphLand datasets

To assess whether RNS gains generalize, we further evaluate it on seven node-classification datasets from GraphLand (Bazhenov et al., 2024). These datasets span a wide range of sizes (from 12K to 2.9M nodes), densities, homophily levels, and label spaces, and each comes with two official splits: *Random Low* (RL), with a 10%/10%/80% train/validation/test partition, and *Random High* (RH), with a 50%/25%/25% partition, corresponding to scarce and abundant supervision regimes. We closely follow the original GraphLand protocol, with the number of RNS parts treated as an additional hyperparameter in $\{2, 3, 5, 7, 10\}$. Full details on the training pipeline and hyperparameter search are given in Appendix F.3, and results are reported in Table 3.

Across the 14 (dataset, regime) configurations, we declare a difference only when the 95% confidence intervals of the two training modes are disjoint. Under this criterion, RNS wins in 7 configurations, ties in 6, and loses in 1. Its gains are larger under abundant supervision (RH) than scarce supervision (RL), where performance is mainly limited by label scarcity rather than by the training procedure. The single regression is WEB-FRAUD RL ($-1.07$), a highly imbalanced task with fewer than 1% positive labels, where random partitioning splits an already scarce positive class across batches. The remaining non-improvements are concentrated on HM-CATEGORIES, the smallest and densest graph in the suite, where RNS neither helps nor hurts at either supervision level. Otherwise, RNS matches or improves on full-graph training across varying homophily levels, densities, and graph sizes, supporting its use as a robust drop-in alternative to full-graph training.

### 5.4 Does RNS generalize across architectures?

Having fixed the backbone in the previous experiments, we now test whether RNS transfers across architectures. We compare full-graph training to RNS for GCN (Kipf & Welling, 2017), GraphSAGE (Hamilton et al., 2017), GAT (Veličković et al., 2018), and SGFormer (Wu et al., 2023), performing an independent hyperparameter search per (architecture, dataset, training mode) triple. GAT runs out of memory on OGBN-PRODUCTS and POKEC under both regimes. We briefly comment on that in Appendix F.4.

Table 4 shows that RNS matches or improves full-graph training for every message-passing backbone on every dataset. The only exception is SGFormer, which underperforms full-graph training on OGBN-ARXIV and on POKEC. We attribute this to SGFormer's global attention branch, whose all-pairs interactions are truncated when only a random subset of nodes is observed at each step. Overall, RNS is essentially architecture-agnostic for standard message-passing GNNs, while its interaction with global-attention models might require further investigation.

## 6 Discussion and limitations

### 6.1 From SGD to Adam: a theory-practice gap

Our backward error analysis is derived for vanilla SGD in order to keep the modified-equation expansion tractable and to make the role of sampling-induced perturbations explicit. In practice, Adam

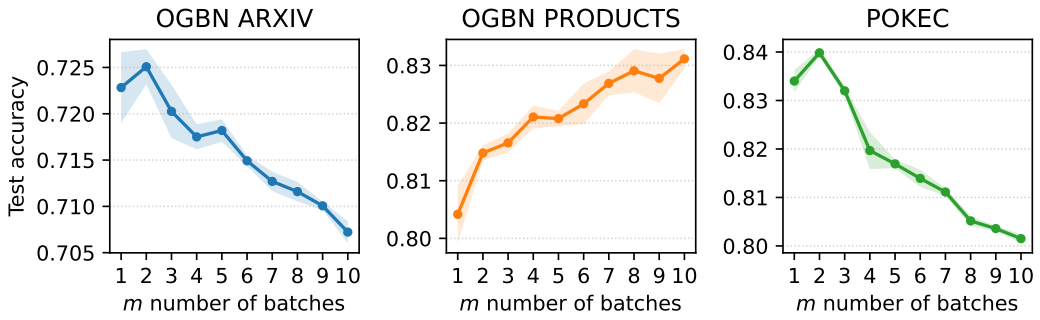



Figure 3: **Impact of the number of batches $m$.** Test accuracy vs. $m$ (batches per epoch) on OGBN-ARXIV, OGBN-PRODUCTS, and POKEC, shaded areas indicate 95% CI over 5 seeds. We fix the architecture and tune training hyperparameters separately for each value of $m$.

consistently achieves higher test accuracy than SGD across all regimes we consider (Table 1), so a natural question is whether the implicit objective of equation 8 extends to adaptive optimizers. Prior work partially closed this gap. Ghosh et al. (2023) show that SGD with Heavy-ball momentum, where $\beta$ denotes the momentum coefficient, admits a modified loss whose implicit regularizer is amplified by a factor $\frac{1+\beta}{1-\beta}$ relative to SGD. This provides a principled explanation for the generalization gains of momentum and motivates our SGD+M baseline. Cattaneo et al. (2024) derive an ODE approximation for Adam and show the existence of an implicit regularizer that penalizes a perturbed $\ell_1$-norm of the loss gradient, but their analysis is restricted to the full-batch setting. Extending this analysis to stochastic Adam remains, to our knowledge, open, as coordinate-wise adaptive normalization renders the modified ODE nonlinear. Empirically, our experiments suggest that the implicit regularization mechanism identified for SGD persists qualitatively under Adam, but extending the formal analysis is left for future work.

### 6.2 Impact of number of batches per epoch

Figure 3 shows that accuracy is sensitive to $m$ even with the architecture fixed and training hyperparameters retuned, so $m$ must be treated as an important hyperparameter. It is a single one, however, where competing methods couple two or more sampling parameters, and it is easy to tune in practice: we start at $m = 2$ and increase it until validation accuracy stops improving. Still, $m$ cannot be increased indefinitely. Since RNS is agnostic to connectivity, large $m$ produces increasingly structure-free subgraphs, which bounds the accuracy it can reach; structure-aware samplers, being designed to preserve structure, are less sensitive to the memory budget. Figure 4 rules out sparsity as the cause: edge count, average degree and component structure evolve almost identically across datasets. Only subgraph diameter tracks the degradation, growing with $m$ on OGBN-ARXIV and POKEC but not on OGBN-PRODUCTS. We leave this correlation to future work. Finally, gains are largest on our two biggest graphs, which suggests the trade-off improves with scale.

## 7 Conclusion

RNS provides a simple yet effective alternative to sophisticated GNN mini-batching methods. Our analysis shows that sampling does not merely approximate full-graph training: it optimizes a modified implicit objective with a sampled loss term and a batch gradient variance penalty. Compared with other sampling methods, this batch gradient variance regularization term is several orders of magnitude smaller, making training more stable by producing sampled subgraphs that remain closer to the full graph.

Empirically, RNS delivers a strong compute–accuracy trade-off on large transductive benchmarks. It generally achieves better test performance than full-graph training while reducing memory usage. Overall, our results establish RNS as both a practical baseline for scalable GNN training and a theoretically grounded form of implicit regularization.

## AI USE STATEMENT

In this work, we used generative AI tools to help develop theoretical arguments, formulate and refine mathematical claims, develop proof ingredients and assist with writing proofs, refine hypotheses, provide feedback on experimental methodology, implement methods, and interpret numerical results.

We have not used generative AI tools to generate synthetic datasets, assist with translation, clean or reformat datasets, or support qualitative or thematic data analysis.

Additionally, we used generative AI tools to draft and edit portions of the manuscript and improve its structure and readability.

We verified AI-assisted mathematical claims and proofs, including their assumptions, derivations, and conclusions. We also checked AI-assisted experimental descriptions and interpretations against the implementation, measurement protocol, and numerical results.

We take responsibility for the final content of this work, including text, claims, or artifacts produced with the aid of generative AI.

## ETHICS STATEMENT

This work focuses on the theoretical and empirical analysis of Random Node Sampling for graph learning. We have not identified additional ethical concerns specific to the contributions presented here. General considerations regarding data privacy, fairness, and responsible use of graph-learning methods remain relevant to downstream applications.

## REPRODUCIBILITY STATEMENT

We provide the assumptions and complete proofs of our theoretical results in the appendices: the implicit regularization analysis for non-i.i.d. data in Appendix A, the RNS bias bound in Appendix B, the RNS variance bound in Appendix C, and the degree-tail preservation result in Appendix D. Appendix F documents the experimental settings, hyperparameter-tuning protocol, and selected hyperparameters. The complete source code needed to reproduce our experiments is included in the supplementary material, and we will release it in a public repository upon acceptance.

## REFERENCES


Muhammed Fatih Balin and Ümit Çatalyürek. Layer-neighbor sampling—defusing neighborhood explosion in gnns. *Advances in Neural Information Processing Systems*, 36:25819–25836, 2023.

David Barrett and Benoit Dherin. Implicit gradient regularization. In *International Conference on Learning Representations, ICLR*, 2021.

Gleb Bazhenov, Oleg Platonov, and Liudmila Prokhorenkova. Graphland: Evaluating graph machine learning models on diverse industrial data. *arXiv preprint arXiv:2409.14500*, 2024.

Matias D. Cattaneo, Jason M. Klusowski, and Boris Shigida. On the implicit bias of adam. In *International Conference on Machine Learning, ICML*, 2024.

Pratik Chaudhari and Stefano Soatto. Stochastic gradient descent performs variational inference, converges to limit cycles for deep networks. In *International Conference on Learning Representations, ICLR*, 2018.

Jie Chen, Tengfei Ma, and Cao Xiao. Fastgcn: Fast learning with graph convolutional networks via importance sampling. In *International Conference on Learning Representations, ICLR*, 01 2018. doi: 10.48550/arXiv.1801.10247.

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 25th ACM SIGKDD international conference on knowledge discovery & data mining*, pp. 257–266, 2019.

Aaron Clauset and Cosma Rohilla Shalizi. Power-law distributions in empirical data. *SIAM review*, 51(4):661–703, 2009.

Chenhui Deng, Zichao Yue, and Zhiru Zhang. Polynormer: Polynomial-expressive graph transformer in linear time. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=hmv1LpNfXa.

Matthias Fey and Jan E. Lenssen. Fast graph representation learning with PyTorch Geometric. In *ICLR Workshop on Representation Learning on Graphs and Manifolds*, 2019.

Matthias Fey, Jinu Sunil, Akihiro Nitta, Rishi Puri, Manan Shah, Blaž Stojanovič, Ramona Bendias, Barghi Alexandria, Vid Kocijan, Zecheng Zhang, Xinwei He, Jan E. Lenssen, and Jure Leskovec. PyG 2.0: Scalable learning on real world graphs. In *Temporal Graph Learning Workshop @ KDD*, 2025.

Avrajit Ghosh, He Lyu, Xitong Zhang, and Rongrong Wang. Implicit regularization in heavy-ball momentum accelerated stochastic gradient descent. In *The Eleventh International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=ZzdBhtEH9yB.

Ernst Hairer, Christian Lubich, and Gerhard Wanner. *Geometric Numerical Integration: Structure-Preserving Algorithms for Ordinary Differential Equations*, volume 31 of *Springer Series in Computational Mathematics*. Springer-Verlag Berlin, 2006.

Will Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. *Advances in neural information processing systems*, 30, 2017.

Sepp Hochreiter and Jürgen Schmidhuber. Flat minima. *Neural computation*, 9(1):1–42, 1997.

Weihua Hu, Matthias Fey, Marinka Zitnik, Yuxiao Dong, Hongyu Ren, Bowen Liu, Michele Catasta, and Jure Leskovec. Open graph benchmark: Datasets for machine learning on graphs. *Advances in neural information processing systems*, 33:22118–22133, 2020.

Stanislaw Jastrzebski, Zachary Kenton, Devansh Arpit, Nicolas Ballas, Asja Fischer, Yoshua Bengio, and Amos J. Storkey. Three factors influencing minima in sgd. In *Artificial Neural Networks and Machine Learning, ICANN*, 2018.

George Karypis and Vipin Kumar. A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM Journal on scientific Computing*, 20(1):359–392, 1998.

Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. In *International Conference on Learning Representations*, 2017. URL https://openreview.net/forum?id=SJU4ayYgl.

Yuankai Luo, Lei Shi, and Xiao-Ming Wu. Classic gnns are strong baselines: Reassessing gnns for node classification. *Advances in Neural Information Processing Systems*, 37:97650–97669, 2024.

Stephan Mandt, Matthew D. Hoffman, and David M. Blei. Stochastic gradient descent as approximate bayesian inference. In *Journal of Machine Learning Research, ICML*, 2017.

Yu Rong, Wenbing Huang, Tingyang Xu, and Junzhou Huang. Dropedge: Towards deep graph convolutional networks on node classification. In *International Conference on Learning Representations, ICLR*, 2020. URL https://openreview.net/forum?id=Hkx1qkrKPr.

Samuel L. Smith and Quoc V. Le. A bayesian perspective on generalization and stochastic gradient descent. In *International Conference on Learning Representations, ICLR*, 2018.

Samuel L. Smith, Benoit Dherin, David G. T. Barrett, and Soham De. On the origin of implicit regularization in stochastic gradient descent. In *International Conference on Learning Representations, ICLR*, 2021.

Lubos Takac and Michal Zabovsky. Data analysis in public social networks. In *International scientific conference and international workshop present day trends of innovations*, volume 1, 2012.

Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph attention networks. In *International Conference on Learning Representations, ICLR*, 2018. URL https://openreview.net/forum?id=rJXMpikCZ.

Qitian Wu, Wentao Zhao, Zenan Li, David P Wipf, and Junchi Yan. Nodeformer: A scalable graph structure learning transformer for node classification. *Advances in Neural Information Processing Systems*, 35:27387–27401, 2022.

Qitian Wu, Wentao Zhao, Chenxiao Yang, Hengrui Zhang, Fan Nie, Haitian Jiang, Yatao Bian, and Junchi Yan. Sgformer: Simplifying and empowering transformers for large-graph representations. *Advances in Neural Information Processing Systems*, 36:64753–64773, 2023.

Rex Ying, Ruining He, Kaifeng Chen, Pong Eksombatchai, William L Hamilton, and Jure Leskovec. Graph convolutional neural networks for web-scale recommender systems. In *Proceedings of the 24th ACM SIGKDD international conference on knowledge discovery & data mining*, pp. 974–983, 2018.

Hanqing Zeng, Hongkuan Zhou, Ajitesh Srivastava, Rajgopal Kannan, and Viktor Prasanna. Graph-SAINT: Graph sampling based inductive learning method. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=BJe8pkHFwS.

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 IMPLICIT OBJECTIVE DERIVATION VIA BACKWARD ERROR ANALYSIS

Our analysis follows the backward-error framework of Smith et al. (2021), to which we refer for additional background and intuition. The presentation is organised so that the full-graph case (Section A.3) and the mini-batch case under subgraph sampling (Section A.4) are two instances of the same pair of expansions: a discrete $m$-step expansion (Lemma 5) and a continuous-time expansion (Lemma 6), matched at second order (Proposition 7).

## A.1 SETTING AND ASSUMPTIONS

Let $d \geq 1$ and let $\varepsilon > 0$ denote the step size. Given an ordered family of vector fields $\mathcal{F} = (f_0, \dots, f_{m-1})$, $f_k : \mathbb{R}^d \to \mathbb{R}^d$, we call *$m$-step scheme* the recursion

$$w_{t+k+1} = w_{t+k} + \varepsilon\, f_k(w_{t+k}), \qquad k = 0, \dots, m-1, \tag{9}$$

and we write

$$f(w) := \frac{1}{m} \sum_{k=0}^{m-1} f_k(w) \tag{10}$$

for the associated mean vector field. Full-batch gradient descent is the special case $f_0 = \cdots = f_{m-1}$; mini-batch training over one epoch is the case where $f_k$ is the negative gradient of the loss of the $k$-th mini-batch.

**Assumption 1.** Let $K \subset \mathbb{R}^d$ be compact and convex, contained in an open set $U$.

(A1) *(Regularity.)* Each $f_k$ is $C^2$ on $U$ and there is $C > 0$ with

$$\sup_{w \in K} \|f_k(w)\| \leq C, \qquad \sup_{w \in K} \|\nabla f_k(w)\| \leq C, \qquad \sup_{w \in K} \|\nabla^2 f_k(w)\| \leq C \qquad \text{for all } k.$$

(A2) *(Confinement.)* The iterates $w_t, \dots, w_{t+m}$ of equation 9 and the trajectory of every modified flow considered below remain in $K$ on the finite time horizon under study.

(A3) *(Gradient fields.)* There are $C^3$ functions $\hat{L}_k$ with $f_k = -\nabla \hat{L}_k$; in particular each Jacobian $\nabla f_k(w)$ is symmetric.

(A4) *(Random reshuffling.)* The batch contents $\{B_0, \dots, B_{m-1}\}$ are fixed and the batches are visited in an order given by a permutation $\varsigma$ drawn uniformly on $\mathfrak{S}_m$, independently of $w_t$.

*Remark* 2 (Piecewise-smooth activations). For ReLU networks $f_k$ is only piecewise smooth: it fails to be differentiable on the measure-zero set $\mathcal{N}$ of activation-switching parameters. Assumption (A1) is then understood as holding on a compact $K$ with $K \cap \mathcal{N} = \emptyset$, i.e. along a finite horizon on which no activation pattern switches. All bounds below are uniform on $K$ and the constant $C$ exists by compactness.

**Definition 3** ($\varepsilon$-modified field and modified loss)**.** An *$\varepsilon$-modified vector field* for the family $\mathcal{F}$ is a field of the form

$$\tilde{f}_\varepsilon(w) = f(w) + \varepsilon\, g(w) + O(\varepsilon^2), \tag{11}$$

with $f$ and $g$ independent of $\varepsilon$. $g$ encodes the first-order effect of the finite step size. If moreover $\tilde{f}_\varepsilon = -\nabla \tilde{L}$ for some $\tilde{L}$, we call $\tilde{L}$ the *implicit loss* and $\dot{w} = -\nabla \tilde{L}(w)$ the modified flow.

## A.2 EXPANSIONS

**Lemma 4** (Trajectory deviation)**.** *Under Assumption 1, for every $0 \leq k \leq m$,*

$$w_{t+k} - w_t = \varepsilon \sum_{l=0}^{k-1} f_l(w_{t+l}), \qquad \textit{hence} \qquad \|w_{t+k} - w_t\| \leq C\, \varepsilon\, k.$$

*Moreover*

$$w_{t+k} - w_t = \varepsilon \sum_{l=0}^{k-1} f_l(w_t) + e_k, \qquad \|e_k\| \leq \tfrac{1}{2}\, C^2 \varepsilon^2 k^2.$$

*Proof.* The first identity is obtained by telescoping equation 9, and the bound follows from $\|f_l\| \le C$ on $K$. For the second, subtract the two expressions: $e_k = \varepsilon \sum_{l<k} \big(f_l(w_{t+l}) - f_l(w_t)\big)$. Each $f_l$ is $C$-Lipschitz on the convex set $K$ by (A1), so $\|f_l(w_{t+l}) - f_l(w_t)\| \le C\|w_{t+l} - w_t\| \le C^2 \varepsilon l$ by the first part, and $\varepsilon \sum_{l<k} C^2 \varepsilon l \le \frac{1}{2} C^2 \varepsilon^2 k^2$. □

**Lemma 5** (Discrete $m$-step expansion)**.** *Under Assumption 1, with $f$ as in equation 10 and*

$$\xi(w) := \sum_{k=0}^{m-1} \sum_{l<k} \nabla f_k(w)\, f_l(w), \tag{12}$$

*the $m$-step scheme equation 9 satisfies*

$$w_{t+m} = w_t + \varepsilon\, m\, f(w_t) + \varepsilon^2\, \xi(w_t) + R, \qquad \|R\| \le \tfrac{1}{3}\, C^3 \varepsilon^3 m^3.$$

*Proof.* Summing equation 9 gives $w_{t+m} = w_t + \varepsilon \sum_{k=0}^{m-1} f_k(w_{t+k})$. Taylor's theorem with second-order remainder on the convex set $K$ yields, for each $k$,

$$f_k(w_{t+k}) = f_k(w_t) + \nabla f_k(w_t)\big(w_{t+k} - w_t\big) + R_k, \qquad \|R_k\| \le \tfrac{1}{2} \|\nabla^2 f_k\|_{\infty,K}\, \|w_{t+k} - w_t\|^2 \le \tfrac{1}{2} C^3 \varepsilon^2 k^2,$$

where the last bound uses Lemma 4. Inserting the second identity of Lemma 4 into the linear term gives

$$\nabla f_k(w_t)(w_{t+k} - w_t) = \varepsilon \sum_{l<k} \nabla f_k(w_t) f_l(w_t) + \nabla f_k(w_t) e_k, \qquad \|\nabla f_k(w_t) e_k\| \le \tfrac{1}{2} C^3 \varepsilon^2 k^2.$$

Summing over $k$ and multiplying by $\varepsilon$ produces the announced first two terms and a remainder bounded by $\varepsilon \sum_{k=0}^{m-1} C^3 \varepsilon^2 k^2 \le \frac{1}{3} C^3 \varepsilon^3 m^3$. □

**Lemma 6** (Continuous-time expansion)**.** *Let $\tilde{f}_\varepsilon = f + \varepsilon g + O(\varepsilon^2)$ be $C^2$ with $f, g$ and their first two derivatives bounded by $C$ on $K$, and let $w(\cdot)$ solve $\dot{w} = \tilde{f}_\varepsilon(w)$ with $w(t) = w_t$ and $w(s) \in K$ for $s \in [t, t+\tau]$. Then, for $\tau = m\varepsilon$,*

$$w(t+\tau) = w_t + m\varepsilon\, f(w_t) + m\varepsilon^2 g(w_t) + \frac{m^2 \varepsilon^2}{2}\, \nabla f(w_t) f(w_t) + O(m^3 \varepsilon^3),$$

*the constant in $O(\cdot)$ depending only on $C$.*

*Proof.* Along the trajectory, $\ddot{w} = \nabla \tilde{f}_\varepsilon(w) \tilde{f}_\varepsilon(w)$, which is bounded and Lipschitz on $K$; hence the third-order Taylor expansion of $s \mapsto w(s)$ at $s = t$ gives $w(t+\tau) = w_t + \tau \tilde{f}_\varepsilon(w_t) + \frac{\tau^2}{2} \nabla \tilde{f}_\varepsilon(w_t) \tilde{f}_\varepsilon(w_t) + O(\tau^3)$. Substituting $\tilde{f}_\varepsilon = f + \varepsilon g + O(\varepsilon^2)$ yields $\tau \tilde{f}_\varepsilon = \tau f + \tau \varepsilon g + O(\tau \varepsilon^2)$ and $\frac{\tau^2}{2} \nabla \tilde{f}_\varepsilon \tilde{f}_\varepsilon = \frac{\tau^2}{2} \nabla f\, f + O(\tau^2 \varepsilon)$. With $\tau = m\varepsilon$ all discarded terms are $O(m^3 \varepsilon^3)$, since $m\varepsilon^3 \le m^2 \varepsilon^3 \le m^3 \varepsilon^3$. □

**Proposition 7** (Second-order matching)**.** *Let $\mathcal{F}$ satisfy Assumption 1. There is exactly one field $g$ for which the time-$m\varepsilon$ flow of $\tilde{f}_\varepsilon = f + \varepsilon g + O(\varepsilon^2)$ agrees with the $m$-step map equation 9 up to $O(m^3 \varepsilon^3)$ uniformly on $K$, namely*

$$g(w) = \frac{1}{m}\, \xi(w) - \frac{m}{2}\, \nabla f(w) f(w). \tag{13}$$

*Proof.* Both expansions share the $O(\varepsilon)$ term $m\varepsilon f(w_t)$. Comparing Lemma 5 with Lemma 6, agreement up to $O(m^3 \varepsilon^3)$ holds if and only if the $\varepsilon^2$ coefficients coincide, i.e. $m\, g + \frac{m^2}{2} \nabla f\, f = \xi$, which is equation 13. Uniqueness is immediate since $g$ appears linearly with the nonzero factor $m$. □

*Remark* 8 (Choice of horizon). The horizon $\tau = m\varepsilon$ is the cumulative step size of one pass of equation 9, which is what makes the $m$-step discrete map directly comparable to the continuous flow. Equivalently one may keep $\tau = \varepsilon$ fixed and set the per-step size to $\varepsilon/m$, as in Smith et al. (2021); the two conventions give the same $g$.

**Lemma 9** (Gradient fields)**.** *If $h = -\nabla H$ with $H \in C^3$, then $\nabla h(w) h(w) = \frac{1}{2} \nabla \|h(w)\|^2$.*

*Proof.* $\frac{1}{2} \nabla \|h\|^2 = (\nabla h)^\top h$, and $\nabla h = -\nabla^2 H$ is symmetric. □

### A.3 FULL-GRAPH TRAINING

**Theorem 10** (Implicit objective of full-batch gradient descent)**.** *Let $L_{\text{full}} \in C^3$, let $f = -\nabla L_{\text{full}}$ satisfy Assumption 1, and consider gradient descent $w_{t+1} = w_t - \varepsilon \nabla L_{\text{full}}(w_t)$. Then the unique $g$ of Proposition 7 is*

$$g(w) = -\tfrac{1}{2}\,\nabla f(w) f(w) = -\tfrac{1}{4}\,\nabla \|\nabla L_{\text{full}}(w)\|^2,$$

*independently of $m$, and the iterates are tracked up to $O(m^3\varepsilon^3)$ over $m$ steps by the modified flow $\dot{w} = -\nabla \tilde{L}_{\text{full}}(w)$ with*

$$\boxed{\tilde{L}_{\text{full}}(w) = L_{\text{full}}(w) + \frac{\varepsilon}{4}\big\|\nabla L_{\text{full}}(w)\big\|^2.} \tag{14}$$

*Proof.* Here $f_k = f$ for all $k$, so $f$ is also the mean field and $\xi = \sum_{k=0}^{m-1}\sum_{l<k} \nabla f\, f = \frac{m(m-1)}{2}\nabla f\, f$ by equation 12. Proposition 7 gives $g = \frac{m-1}{2}\nabla f\, f - \frac{m}{2}\nabla f\, f = -\frac{1}{2}\nabla f\, f$, which is independent of $m$ as it must be. With $f = -\nabla L_{\text{full}}$, Lemma 9 yields $g = -\frac{1}{2}\nabla^2 L_{\text{full}} \nabla L_{\text{full}} = -\frac{1}{4}\nabla\|\nabla L_{\text{full}}\|^2$. Hence $\tilde{f}_\varepsilon = f + \varepsilon g = -\nabla\big(L_{\text{full}} + \frac{\varepsilon}{4}\|\nabla L_{\text{full}}\|^2\big)$, which is equation 14. □

### A.4 MINI-BATCH GNN TRAINING UNDER SUBGRAPH SAMPLING

**Setting.** Let $N = |V|$, fix a batch size $B$ with $N$ divisible by $B$, and set $m := N/B$. Let $\{B_0, \dots, B_{m-1}\}$ be a partition of $V$ produced by a subgraph sampler $\mathrm{Sampl}$ (e.g. RNS, neighbour sampling, Cluster-GCN, GraphSAINT), and let $B_k^{\text{train}} := B_k \cap V^{\text{train}}$. For batch $B_k$ define the partial loss and its vector field

$$\hat{L}^{B_k}_{\mathrm{Sampl}}(w) := \frac{1}{|B_k^{\text{train}}|} \sum_{v \in B_k^{\text{train}}} \ell\big(f_w(X_{B_k}, E_{B_k})[v],\, y_v\big), \qquad f_k := -\nabla \hat{L}^{B_k}_{\mathrm{Sampl}}. \tag{15}$$

Only equation 15 depends on the sampler, so every statement below holds for any $\mathrm{Sampl}$. One epoch of SGD is the $m$-step scheme equation 9 applied to $(f_{\varsigma(0)}, \dots, f_{\varsigma(m-1)})$ with $\varsigma$ as in (A4). We write $\bar{L}_{\mathrm{Sampl}} := \frac{1}{m}\sum_k \hat{L}^{B_k}_{\mathrm{Sampl}}$, so that the mean field equation 10 is $f = -\nabla \bar{L}_{\mathrm{Sampl}}$.

**Lemma 11** (Expected interaction term)**.** *Let $\xi_\varsigma(w) := \sum_{k=0}^{m-1}\sum_{l<k} \nabla f_{\varsigma(k)}(w) f_{\varsigma(l)}(w)$. Under (A3)–(A4),*

$$\mathbb{E}_\varsigma\big[\xi_\varsigma(w)\big] = \frac{1}{2}\sum_{k\neq l} \nabla f_k(w) f_l(w) = \frac{m^2}{2}\,\nabla f(w) f(w) - \frac{1}{4}\,\nabla \sum_{k=0}^{m-1} \big\|f_k(w)\big\|^2.$$

*Proof.* For fixed $k \neq l$, the pair $(k, l)$ contributes $\nabla f_k f_l$ to $\xi_\varsigma$ exactly on the event that $l$ is visited before $k$, which has probability $1/2$ by uniformity of $\varsigma$; diagonal pairs never contribute. Hence $\mathbb{E}_\varsigma[\xi_\varsigma] = \frac{1}{2}\sum_{k\neq l} \nabla f_k f_l$. Completing the double sum,

$$\frac{1}{2}\sum_{k\neq l} \nabla f_k f_l = \frac{1}{2}\Big(\sum_k \nabla f_k\Big)\Big(\sum_l f_l\Big) - \frac{1}{2}\sum_k \nabla f_k f_k = \frac{m^2}{2}\nabla f\, f - \frac{1}{4}\nabla \sum_k \|f_k\|^2,$$

using $\sum_k f_k = m f$, $\sum_k \nabla f_k = m \nabla f$ and Lemma 9 applied to each $f_k$ (legitimate by (A3)). □

*Remark* 12 (Order dependence). In general $\sum_{l<k} \nabla f_k f_l \neq \sum_{l>k} \nabla f_k f_l$: the weights reached after one epoch genuinely depend on the batch ordering. Lemma 11 describes the bias of the epoch map *in expectation over orderings*, the batch contents being held fixed.

**Theorem 13** (Implicit objective of mini-batch training)**.** *Under Assumption 1, the expected epoch map of SGD with sampler* $\mathrm{Sampl}$ *is tracked up to $O(m^3\varepsilon^3)$ by the modified flow $\dot{w} = -\nabla \tilde{L}_{\mathrm{Sampl}}(w)$, where*

$$\boxed{\tilde{L}_{\mathrm{Sampl}}(w) = \frac{1}{m}\sum_{k=0}^{m-1} \hat{L}^{B_k}_{\mathrm{Sampl}}(w) + \frac{\varepsilon}{4m}\sum_{k=0}^{m-1} \big\|\nabla \hat{L}^{B_k}_{\mathrm{Sampl}}(w)\big\|^2.} \tag{16}$$

*Equivalently, the unique matched field of Proposition 7 is $g(w) = -\frac{1}{4m}\nabla \sum_k \|f_k(w)\|^2$.*

*Proof.* Apply Proposition 7 with $\xi$ replaced by $\mathbb{E}_\varsigma[\xi_\varsigma]$ (Lemma 11), which is licit because $\varsigma$ is independent of $w_t$ by (A4) and the remainder bound of Lemma 5 is uniform in $\varsigma$:

$$g = \frac{1}{m}\mathbb{E}_\varsigma[\xi_\varsigma] - \frac{m}{2}\nabla f\, f = \frac{1}{m}\Big(\frac{m^2}{2}\nabla f\, f - \frac{1}{4}\nabla \sum_k \|f_k\|^2\Big) - \frac{m}{2}\nabla f\, f = -\frac{1}{4m}\nabla \sum_k \|f_k\|^2.$$

Since $f = -\nabla \bar{L}_{\mathrm{Sampl}}$ and $f_k = -\nabla \hat{L}^{B_k}_{\mathrm{Sampl}}$,

$$\tilde{f}_\varepsilon = f + \varepsilon g = -\nabla\Big(\bar{L}_{\mathrm{Sampl}} + \frac{\varepsilon}{4m}\sum_k \big\|\nabla \hat{L}^{B_k}_{\mathrm{Sampl}}\big\|^2\Big),$$

which is equation 16. □

**Corollary 14** (Decomposition of the implicit regularizer)**.** *With* $\bar{L}_{\mathrm{Sampl}} = \frac{1}{m}\sum_k \hat{L}^{B_k}_{\mathrm{Sampl}}$,

$$\tilde{L}_{\mathrm{Sampl}}(w) = \bar{L}_{\mathrm{Sampl}}(w) + \frac{\varepsilon}{4}\big\|\nabla \bar{L}_{\mathrm{Sampl}}(w)\big\|^2 + \frac{\varepsilon}{4m}\sum_{k=0}^{m-1}\big\|\nabla \hat{L}^{B_k}_{\mathrm{Sampl}}(w) - \nabla \bar{L}_{\mathrm{Sampl}}(w)\big\|^2.$$

*The first regularizer is the full-batch term of Theorem 10 applied to* $\bar{L}_{\mathrm{Sampl}}$*; the second is the empirical variance of the per-batch gradients and is the term created by sampling. It vanishes if and only if all per-batch gradients coincide, in particular when* $m = 1$*.*

*Proof.* Apply the bias–variance identity $\frac{1}{m}\sum_k \|a_k\|^2 = \|\bar{a}\|^2 + \frac{1}{m}\sum_k \|a_k - \bar{a}\|^2$ with $a_k = \nabla \hat{L}^{B_k}_{\mathrm{Sampl}}(w)$ and $\bar{a} = \nabla \bar{L}_{\mathrm{Sampl}}(w) = \frac{1}{m}\sum_k a_k$, then insert it into equation 16. □

*Remark* 15 (Regime of validity). Lemmas 5 and 6 retain terms of order $m^2\varepsilon^2$ while discarding terms of order $m^3\varepsilon^3$, so the expansion is informative precisely when $m\varepsilon$ is small, the sense in which the step size is "small but finite" in backward error analysis; see the extended discussion in Smith et al. (2021). This requirement is mild in our setting, because the number of mini-batches per epoch stays small: all our experiments use $m \leq 15$, larger $m$ degrading the topology of the sampled subgraphs too severely.

*Remark* 16 (No graph quantity appears directly). All terms above are optimization quantities: gradients in $w$ of scalar losses. The graph enters only at the input of the network and is absorbed into the value of $\nabla_w \hat{L}^{B_k}_{\mathrm{Sampl}}(w)$; making it surface would require structural assumptions we avoid (e.g. a linear GNN). Theorem 10 is thus the full-batch result of Smith et al. (2021) instantiated at $L_{\mathrm{full}}$, the algebra going through because the loss is a pointwise average and $\nabla_w$ is linear.

*Remark* 17 (Difference with the i.i.d. case). In the i.i.d. setting of Smith et al. (2021), which mini-batch a sample belongs to does not affect the final expression. Under subgraph sampling, $v \in B_k$ is scored on the sampled subgraph via $f_w(X_{B_k}, E_{B_k})[v]$, so $\bar{L}_{\mathrm{Sampl}} \neq L_{\mathrm{full}}$ and the batch-averaged loss survives in equation 16 as an object of its own.

## B Bias of Random Node Sampling

At fixed model parameters, we show that the bias of the RNS objective admits an upper bound proportional to $1 - 1/m$, where $m$ is the number of partition blocks. The proof proceeds in two steps. First, we show that the bias is controlled by sensitivity to edge removal, separating the effect of random partitioning from the model-dependent sensitivity. We then bound this sensitivity for a simplified mean-aggregation GNN.

### B.1 Setting and assumptions

Let $G = (V, E)$ be a simple undirected graph with $N := |V| \geq 2$, and let $V^{\mathrm{tr}} \subseteq V$ be a fixed training set of size $N_t := |V^{\mathrm{tr}}| \geq 1$. Write

$$\tau := \frac{N_t}{N}, \qquad \bar{d} := \frac{2|E|}{N}.$$

Assume that $m$ divides $N$, and set $n := N/m$. Random node sampling (RNS) draws a uniformly random ordered partition $P = (B_0, \ldots, B_{m-1})$ of $V$ into blocks of size $n$. Throughout, $B$ denotes

one fixed-index block and $B^{\mathrm{tr}} := B \cap V^{\mathrm{tr}}$. All expectations are over the partition, with model parameters $w$ fixed.

We consider an $L$-layer mean-aggregation GNN, with $L \geq 1$. Let $d_v := |N(v)|$, $\hat{d}_v := 1 + d_v$, $\hat{A} := A + I$, and $S := \hat{D}^{-1}\hat{A}$, where $\hat{D} := \mathrm{diag}(\hat{d})$. The full-graph forward pass is

$$h_v^{(0)} = x_v, \qquad z_v^{(l)} = \sum_{u \in V} S_{vu} h_u^{(l)}, \qquad h_v^{(l+1)} = \varphi\Big(W^{(l)} z_v^{(l)}\Big), \qquad 0 \leq l < L.$$

Write $f(w, X, E)[v] := h_v^{(L)}$. On a block $B$, the same model runs on the induced subgraph $G[B]$. For $v \in B$, define

$$R_v := N(v) \cap B, \qquad Q_v := N(v) \setminus B, \qquad \hat{d}_v^B := 1 + |R_v|,$$

and let $S_B$ be the row-stochastic matrix on $B$ given by

$$(S_B)_{vu} := \begin{cases} 1/\hat{d}_v^B, & u \in R_v \cup \{v\}, \\ 0, & \text{otherwise.} \end{cases}$$

Self-loops make this definition valid even at isolated nodes of $G[B]$. The resulting representations are denoted by $h_v^{B,(l)}$. For $v \in B$, set

$$\Delta_v^{(l)} := \|h_v^{(l)} - h_v^{B,(l)}\|_2, \qquad \delta_v := \Delta_v^{(L)}, \qquad \rho_v := \frac{|Q_v|}{\hat{d}_v} \in [0, 1), \qquad \partial B := \sum_{v \in B} |Q_v|.$$

Here $\partial B$ counts edges with exactly one endpoint in $B$, and $\rho := (\rho_v)_{v \in B}$.

**Assumption 18.** The loss is $C_\ell$-Lipschitz in its first argument. The activation acts entrywise, is $C_\varphi$-Lipschitz, and satisfies $\varphi(0) = 0$. The layer maps have no bias vectors and satisfy $\|W^{(l)}\|_2 \leq C_W$ for every layer. Finally, $\|x_v\|_2 \leq \beta$ for every node. Set $a := C_\varphi C_W$.

A Lipschitz readout can be incorporated into the loss, with its Lipschitz constant absorbed into $C_\ell$. Define the full-graph feature scale

$$\theta^{(l)} := \min\left\{ \max_{\{u,v\} \in E} \|h_u^{(l)} - h_v^{(l)}\|_2, \max_{u \in V} \|h_u^{(l)}\|_2 \right\}, \qquad \theta := \max_{0 \leq l < L} \theta^{(l)}, \tag{17}$$

where the edge maximum is zero when $E = \varnothing$. These quantities are deterministic at fixed parameters: they depend only on the full-graph forward pass, not on the partition. The first term measures worst-edge representation oscillation; the second provides a magnitude fallback. Finally, write

$$q := \frac{N - n}{N - 1} = \left(1 - \frac{1}{m}\right) \frac{N}{N - 1}, \qquad A_L := \sum_{j=1}^{L} a^j, \qquad r := \min\left\{1, \frac{\bar{d}}{\tau} q\right\}. \tag{18}$$

### B.2 Sampling facts

**Lemma 19** (Inclusion probabilities and cut budget)**.** *The block $B$ is uniform over the $n$-subsets of $V$. Consequently, for any distinct nodes $u, v$,*

$$\Pr(v \in B) = \frac{1}{m}, \qquad \Pr(u \notin B \mid v \in B) = q.$$

*In particular,*

$$\mathbb{E}[|Q_v| \mid v \in B] = q\, d_v, \qquad \mathbb{E}[\rho_v \mid v \in B] = q\, \frac{d_v}{1 + d_v} \leq q,$$

*and*

$$\mathbb{E}[\partial B] = \frac{2|E|}{m} q, \qquad \mathbb{E}[|B^{\mathrm{tr}}|] = \frac{N_t}{m}, \qquad \frac{m}{N_t} \mathbb{E}\big[\min\{|B^{\mathrm{tr}}|, \partial B\}\big] \leq r.$$

*Proof.* For each fixed $n$-set, the number of ordered partitions with that set in the chosen block position is $(N-n)!/(n!)^{m-1}$, independent of the set. Thus $B$ is uniform. Conditional on $v \in B$, the

remaining $n-1$ nodes are uniform among the other $N-1$ nodes, so a given neighbour is excluded with probability $(N-n)/(N-1) = q$. Summing neighbour indicators gives the conditional expectations. Hence

$$\mathbb{E}[\partial B] = \sum_{v \in V} \Pr(v \in B)\, \mathbb{E}[|Q_v| \mid v \in B] = \frac{q}{m} \sum_{v \in V} d_v = \frac{2|E|}{m} q.$$

The training-count identity follows in the same way. Finally, $\mathbb{E}[\min\{X, Y\}] \le \min\{\mathbb{E}X, \mathbb{E}Y\}$ and $2|E|/N_t = \bar{d}/\tau$ give the last inequality. □

### B.3 Separating node sampling from edge removal

Let

$$\ell_v^{\text{full}}(w) := \ell(h_v^{(L)}, y_v), \qquad \ell_v^B(w) := \ell(h_v^{B,(L)}, y_v),$$

and define

$$L_{\text{full}}(w) := \frac{1}{N_t} \sum_{v \in V^{\text{tr}}} \ell_v^{\text{full}}(w), \qquad \hat{L}^B(w) := \frac{m}{N_t} \sum_{v \in B^{\text{tr}}} \ell_v^B(w).$$

The second objective uses the deterministic expected training count for normalization. We address normalization by the actual training count below.

**Definition 20** (Mean edge-removal sensitivity)**.** Define

$$\sigma(w) := \frac{1}{N_t} \sum_{v \in V^{\text{tr}}} \mathbb{E}[\delta_v \mid v \in B] = \frac{m}{N_t} \mathbb{E}\left[ \sum_{v \in B^{\text{tr}}} \delta_v \right]. \tag{19}$$

The equality follows from $\Pr(v \in B) = 1/m$.

**Theorem 21** (Bias is controlled by edge-removal sensitivity)**.** *If the loss is $C_\ell$-Lipschitz, then*

$$\left| \mathbb{E}[\hat{L}^B(w)] - L_{\text{full}}(w) \right| \le C_\ell \sigma(w).$$

*Proof.* Consider the same sampled nodes evaluated with full-graph representations:

$$\tilde{L}^B(w) := \frac{m}{N_t} \sum_{v \in B^{\text{tr}}} \ell_v^{\text{full}}(w).$$

The full-graph losses are partition-independent, so the inclusion probabilities give $\mathbb{E}[\tilde{L}^B(w)] = L_{\text{full}}(w)$. Therefore

$$\begin{aligned} \left| \mathbb{E}[\hat{L}^B(w)] - L_{\text{full}}(w) \right| &\le \mathbb{E}\left[ |\hat{L}^B(w) - \tilde{L}^B(w)| \right] \\ &\le \frac{m C_\ell}{N_t} \mathbb{E}\left[ \sum_{v \in B^{\text{tr}}} \delta_v \right] = C_\ell \sigma(w). \end{aligned}$$

Thus uniform node subsampling introduces no bias when full-graph representations are used; the discrepancy here is due to the induced-subgraph forward pass. □

The same bound holds for the fixed-parameter epoch average

$$\bar{L}(w) := \frac{1}{m} \sum_{k=0}^{m-1} \hat{L}^{B_k}(w),$$

because every block has the same marginal distribution. This statement concerns objectives evaluated at one fixed parameter vector, not the trajectory of parameter updates within an epoch.

### B.4 CONTROLLING THE REPRESENTATION PERTURBATION

**Lemma 22** (Centred perturbation recursion)**.** *Under Assumption 18, for every partition and every $v \in B$,*

$$\delta_v \le 2 \sum_{l=0}^{L-1} a^{L-l} \theta^{(l)} (S_B^{L-1-l} \rho)_v. \tag{20}$$

*Moreover,*

$$\delta_v \le 2 \min\{a^L \beta, \theta A_L\}. \tag{21}$$

*Proof.* First, row-stochasticity and Assumption 18 imply, by induction,

$$\max_{u \in V} \|h_u^{(l)}\|_2 \le a^l \beta, \qquad \max_{u \in B} \|h_u^{B,(l)}\|_2 \le a^l \beta.$$

Indeed, averaging does not increase the maximum Euclidean norm, and each subsequent layer map increases it by at most a factor $a$. Consequently, $\theta^{(l)} \le a^l \beta$ and $\delta_v \le 2a^L \beta$.

For the perturbation estimate, extend the row $(S_B)_{v\cdot}$ by zero outside $B$, and write $\gamma_u := S_{vu} - (S_B)_{vu}$. Both rows sum to one, and their difference is supported on $N(v) \cup \{v\}$. The removed neighbours contribute $|Q_v|/\hat{d}_v$ to its absolute row sum, and the renormalized retained entries contribute the same amount. Hence

$$\sum_{u \in V} \gamma_u = 0, \qquad \sum_{u \in V} |\gamma_u| = 2\rho_v.$$

Subtracting $h_v^{(l)}$ from every full-graph feature, or instead leaving them centred at zero, gives the two bounds whose minimum defines $\theta^{(l)}$. Thus

$$\left\| \sum_{u \in V} \gamma_u h_u^{(l)} \right\|_2 \le 2\rho_v \theta^{(l)}.$$

Using

$$z_v^{(l)} - z_v^{B,(l)} = \sum_{u \in V} \gamma_u h_u^{(l)} + \sum_{u \in B} (S_B)_{vu} \big(h_u^{(l)} - h_u^{B,(l)}\big),$$

Lipschitzness of the layer map yields the componentwise recursion

$$\Delta^{(l+1)} \le 2a\theta^{(l)} \rho + a S_B \Delta^{(l)}, \qquad \Delta^{(0)} = 0.$$

Since $S_B$ is nonnegative, iterating this inequality proves equation 20. Finally, every power of $S_B$ is row-stochastic and $0 \le \rho_v < 1$, so

$$\delta_v \le 2 \sum_{l=0}^{L-1} a^{L-l} \theta^{(l)} \le 2\theta A_L.$$

Combining this with the feature-norm bound proves equation 21. □

**Lemma 23** (Propagation is controlled by the cut budget)**.** *For every partition and every integer $k \ge 0$,*

$$\sum_{v \in B^{\mathrm{tr}}} (S_B^k \rho)_v \le \min\{|B^{\mathrm{tr}}|, \partial B\}. \tag{22}$$

*Proof.* The induced adjacency matrix with self-loops is symmetric. Therefore its degree vector is a left fixed vector of the row-normalized shift:

$$(\hat{d}^B)^\top S_B = (\hat{d}^B)^\top.$$

Since $\mathbf{1} \le \hat{d}^B$ componentwise and $S_B^k \rho \ge 0$,

$$\begin{aligned} \sum_{v \in B^{\mathrm{tr}}} (S_B^k \rho)_v &\le \mathbf{1}^\top S_B^k \rho \\ &\le (\hat{d}^B)^\top S_B^k \rho = (\hat{d}^B)^\top \rho \\ &= \sum_{u \in B} \hat{d}_u^B \frac{|Q_u|}{\hat{d}_u} \le \partial B. \end{aligned}$$

The alternative bound follows from $(S_B^k \rho)_v \le 1$. Both arguments also hold for $k = 0$. □

### B.5 SENSITIVITY AND OBJECTIVE BIAS

**Theorem 24** (End-to-end bias bound)**.** *Under Assumption 18, for every fixed parameter vector and every $L \geq 1$,*

$$\left|\mathbb{E}[\hat{L}^B(w)] - L_{\text{full}}(w)\right| \leq C_\ell \sigma(w),$$
$$\sigma(w) \leq 2 \min\left\{ a^L \beta, a\theta^{(L-1)} q + r \sum_{l=0}^{L-2} a^{L-l} \theta^{(l)} \right\}, \tag{23}$$

*where the sum is empty when $L = 1$. Independently, the following simpler bound holds:*

$$\left|\mathbb{E}[\hat{L}^B(w)] - L_{\text{full}}(w)\right| \leq 2C_\ell \min\{a^L \beta, \theta A_L r\}. \tag{24}$$

*When $a \leq 1$, its right-hand side is at most $2C_\ell \min\{\beta, L\theta r\}$. For one layer,*

$$\left|\mathbb{E}[\hat{L}^B(w)] - L_{\text{full}}(w)\right| \leq 2C_\ell a \theta^{(0)} q. \tag{25}$$

*All these bounds vanish when $m = 1$.*

*Proof.* Sum equation 20 over the training nodes in the block. Isolating the term with $S_B^0 = I$ and applying Lemma 23 to the remaining terms gives

$$\sum_{v \in B^{\text{tr}}} \delta_v \leq 2a\theta^{(L-1)} \sum_{v \in B^{\text{tr}}} \rho_v + 2 \min\{|B^{\text{tr}}|, \partial B\} \sum_{l=0}^{L-2} a^{L-l} \theta^{(l)}. \tag{26}$$

The inclusion probabilities imply

$$\frac{m}{N_t} \mathbb{E}\left[ \sum_{v \in B^{\text{tr}}} \rho_v \right] = \frac{q}{N_t} \sum_{v \in V^{\text{tr}}} \frac{d_v}{1 + d_v} \leq q.$$

Multiplying equation 26 by $m/N_t$, taking expectations, and using Lemma 19 therefore yields

$$\sigma(w) \leq 2 \min\left\{ a^L \beta, a\theta^{(L-1)} q + r \sum_{l=0}^{L-2} a^{L-l} \theta^{(l)} \right\}, \tag{27}$$

where the first branch follows from equation 21. Theorem 21 now proves equation 23.

For the simpler bound, apply Lemma 23 to every term of equation 20, including the term with exponent zero. This gives

$$\sum_{v \in B^{\text{tr}}} \delta_v \leq 2 \min\{|B^{\text{tr}}|, \partial B\} \sum_{l=0}^{L-1} a^{L-l} \theta^{(l)}.$$

Taking expectations as above yields

$$\sigma(w) \leq 2 \min\left\{ a^L \beta, r \sum_{l=0}^{L-1} a^{L-l} \theta^{(l)} \right\} \leq 2 \min\{a^L \beta, \theta A_L, \theta A_L \frac{\bar{d}}{\tau} q\},$$

which proves equation 24 without any restriction on graph density. If $a \leq 1$, use $a^L \leq 1$ and $A_L \leq L$. For $L = 1$, the first bound has no upstream terms, giving equation 25. Finally, $m = 1$ implies $B = V$ and $q = r = 0$, so the two forward passes coincide. □

*Remark* 25 (Implemented normalization). The implementation normalizes by the observed training count:

$$T := |B^{\text{tr}}|, \qquad \hat{L}_{\text{r}}^B(w) := \frac{1}{T} \sum_{v \in B^{\text{tr}}} \ell_v^B(w), \qquad T > 0.$$

Conditionally on any positive training count, the sampled training set is uniform among training subsets of that size. Consequently,

$$\mathbb{E}\left[\frac{1}{T}\sum_{v\in B^{\mathrm{tr}}}\ell_v^{\mathrm{full}}(w)\,\middle|\,T>0\right]=L_{\mathrm{full}}(w).$$

Thus node subsampling remains unbiased under this normalization, and any remaining objective bias arises from induced-subgraph message passing. Moreover, the hypergeometric training count satisfies

$$\Pr(T=0)\le e^{-N_t/m},$$

so empty blocks are exponentially unlikely in the expected training count. The quantitative bounds above concern the deterministically normalized objective and are not asserted unchanged for the ratio objective.

**Interpretation and scope.** Uniform node subsampling is unbiased when losses use full-graph representations; the discrepancy analyzed here arises from induced-subgraph message passing. The bound connects this discrepancy to layerwise edgewise feature oscillations and edge-removal statistics. Small oscillations can reduce the bound even when many edges are removed. However, the upstream cut factor $r$ saturates when $(\bar{d}/\tau)q \ge 1$. Since every nontrivial equal-block partition has $q > 1/2$, this happens for all $m \ge 2$ whenever $\bar{d}/\tau \ge 2$; the layer-resolved bound still retains the output-layer factor $q$. The result may therefore be loose for dense graphs or small training splits. It applies to fixed parameters and row-normalized mean aggregation, and does not establish monotonicity of the actual bias in the number of blocks.

### B.6 Connection to DropEdge and oversmoothing

The analysis above shows that the bias of the RNS implicit objective comes entirely from edge removal: at each epoch, RNS partitions the nodes into $m$ disjoint parts and drops every edge whose endpoints fall in different parts, i.e., about a fraction $1 - 1/m$ of the edges. This resembles DropEdge (Rong et al., 2020), which removes edges uniformly at random at each training epoch. DropEdge acts as a data augmenter, and Rong et al. (2020) prove that it either slows the convergence of oversmoothing or relieves the information loss it causes. The removal patterns differ, however: DropEdge drops edges uniformly, while RNS keeps each batch as the full induced subgraph of its node set. We do not prove an analogous result for RNS, but we conjecture that its edge removal provides part of the same benefit.

## C Variance of the RNS Batch Gradient

At fixed model parameters, we show that the variance of RNS batch gradient admits an upper bound proportional to $m - 1$. The proof proceeds in two steps. First, we establish a single-swap variance inequality for uniform fixed-size sampling. We then use locality and bounded per-node gradients to control the effect of a swap through the average squared training-node influence. Finally, a variance decomposition connects this bound to the expected empirical gradient dispersion $R$ across partition blocks.

We keep the graph, training set, and local $L$-layer mean-aggregation model of Appendix B. The message-passing graph is undirected, and its full-graph distance is denoted by $\mathrm{dist}_G$. In particular, let $N := |V| \ge 2$, $N_t := |V^{\mathrm{tr}}| \ge 1$, and $\tau := N_t/N$. For an integer $m$ dividing $N$, a uniformly random ordered partition divides the graph into $m$ blocks of size $n = N/m$. We write $B$ for one fixed-index block, which is uniform among the $n$-subsets of $V$, and set $B^{\mathrm{tr}} := B \cap V^{\mathrm{tr}}$.

Throughout this section, the parameter vector $w$ is fixed. We study the gradient of the deterministically normalized objective:

$$\hat{g}_B := \nabla_w \hat{L}^B(w) = \frac{m}{N_t}\sum_{v\in B^{\mathrm{tr}}}\nabla_w \ell_v^B(w) = \frac{1}{n\tau}\sum_{v\in B^{\mathrm{tr}}}\nabla_w \ell_v^B(w). \tag{28}$$

Its variance is the trace of its covariance matrix:

$$\mathcal{V}_m(w) := \mathrm{Tr}\,\mathrm{Cov}[\hat{g}_B] = \mathbb{E}\|\hat{g}_B - \mathbb{E}\hat{g}_B\|_2^2, \tag{29}$$

where the expectation is over the random block. The result concerns this estimator, not the estimator normalized by the random training count $|B^{\mathrm{tr}}|$.

### C.1 Assumptions and local influence

**Assumption 26** (Uniformly bounded per-node gradients)**.** For every admissible block $B$ and every $v \in B^{\mathrm{tr}}$, the sampled loss is differentiable at $w$ and

$$\|\nabla_w \ell_v^B(w)\|_2 \le G_\nabla.$$

The same constant is used for all block sizes under consideration.

For example, this assumption holds with $G_\nabla = C_J C_\ell$ if the sampled output Jacobians have operator norm at most $C_J$ and the output-space loss gradients have norm at most $C_\ell$, since

$$\nabla_w \ell_v^B(w) = (J_v^B)^\top \nabla_p \ell(h_v^{B,(L)}, y_v).$$

No Lipschitz assumption on the loss gradient or activation derivative is needed for the variance bound below.

For a graph node $u$, define its closed full-graph receptive field and its training-node influence count by

$$\mathcal{N}_L(u) := \{v \in V : \mathrm{dist}_G(u,v) \le L\}, \qquad h_u := |\mathcal{N}_L(u) \cap V^{\mathrm{tr}}|.$$

By symmetry of graph distance, $h_u$ counts the training nodes whose full-graph receptive fields contain $u$. Define the average squared influence

$$\mathcal{H}_L := \frac{1}{N} \sum_{u \in V} h_u^2. \tag{30}$$

This quantity depends on the graph, training set, and depth, but not on the number of layers.

### C.2 A single-swap variance inequality

**Lemma 27** (Variance under uniform fixed-size sampling)**.** *Let $1 \le n < N$, let $B$ be a uniform $n$-subset of $[N]$, and let $F(B)$ take values in a finite-dimensional Euclidean space. Conditionally on $B$, choose $i \in B$ and $j \notin B$ independently and uniformly from their respective sets, and put $B' := (B \setminus \{i\}) \cup \{j\}$. Then*

$$\mathrm{Tr}\,\mathrm{Cov}[F(B)] \le \frac{n(N-n)}{2N} \mathbb{E}\|F(B) - F(B')\|_2^2. \tag{31}$$

*Proof.* Both sides of equation 31 are sums of the corresponding scalar quantities over coordinates, so it suffices to prove the scalar case.

For $i \in S$ and $j \notin S$, write $S^{i \to j} := (S \setminus \{i\}) \cup \{j\}$. Define

$$V_F := \mathrm{Var}[F(B)], \qquad \mathcal{D}(F) := \mathbb{E}_B \sum_{i \in B} \sum_{j \notin B} \big(F(B) - F(B^{i \to j})\big)^2.$$

By the conditional distribution of the swap,

$$\mathcal{D}(F) = n(N-n)\, \mathbb{E}\big[(F(B) - F(B'))^2\big]. \tag{32}$$

It is therefore enough to show

$$V_F \le \frac{\mathcal{D}(F)}{2N}. \tag{33}$$

We prove equation 33 by induction on $N$, simultaneously for all subset sizes $0 \le n \le N$. The degenerate sizes are included because they occur in the conditional distributions used below. If $n = 0$ or $n = N$, the set $B$ is deterministic and both sides are zero. This also settles $N = 1$. For $N = 2$, the only remaining case is $n = 1$. Writing $F_1 := F(\{1\})$ and $F_2 := F(\{2\})$, we have

$$V_F = \frac{1}{4}(F_1 - F_2)^2, \qquad \mathcal{D}(F) = (F_1 - F_2)^2,$$

so equation 33 holds with equality.

Now suppose $N \geq 3$ and $0 < n < N$, and assume the claim for $N-1$ and every subset size. Put

$$X := F(B) - \mathbb{E}F(B), \qquad p := \frac{n}{N}, \qquad a := p(1-p) > 0,$$

and, for $v \in [N]$, define

$$I_v := \mathbf{1}\{v \in B\}, \qquad Z_v := I_v - p, \qquad c_v := \mathbb{E}[XZ_v].$$

Uniform fixed-size sampling gives

$$\mathbb{E}[Z_v^2] = a, \qquad \mathbb{E}[Z_v Z_u] = -\frac{a}{N-1} \quad (u \neq v).$$

Indeed, $\mathbb{E}[I_v] = n/N$ and $\mathbb{E}[I_v I_u] = n(n-1)/(N(N-1))$ for $u \neq v$. Moreover, $\sum_v Z_v = 0$, and hence $\sum_v c_v = 0$.

Set $Q := \sum_v c_v^2$ and $Y := \sum_v c_v Z_v$. The preceding identities imply

$$\mathbb{E}[Y^2] = \frac{aN}{N-1}Q, \qquad \mathbb{E}[XY] = Q.$$

If $Q > 0$, Cauchy–Schwarz yields

$$Q^2 \leq \mathbb{E}[X^2]\mathbb{E}[Y^2] = V_F \frac{aN}{N-1}Q,$$

so $Q \leq aNV_F/(N-1)$. The same inequality is immediate when $Q = 0$.

Because $I_v$ is binary and $X$ is centered,

$$\mathbb{E}[X \mid I_v] = \frac{c_v}{a}Z_v, \qquad \mathrm{Var}(\mathbb{E}[X \mid I_v]) = \frac{c_v^2}{a}.$$

Consequently,

$$\frac{1}{N}\sum_v \mathrm{Var}(\mathbb{E}[X \mid I_v]) = \frac{Q}{Na} \leq \frac{V_F}{N-1}.$$

Averaging the law of total variance over $v$ gives

$$\frac{1}{N}\sum_v \mathbb{E}[\mathrm{Var}(X \mid I_v)] \geq \frac{N-2}{N-1}V_F. \tag{34}$$

We next upper-bound the same conditional variances. For a fixed $v$, define the swap sum omitting $v$:

$$S_v(B) := \sum_{i \in B\setminus\{v\}} \sum_{j \in [N]\setminus(B\cup\{v\})} \big(F(B) - F(B^{i\to j})\big)^2.$$

Conditionally on $I_v = 1$, the set $B\setminus\{v\}$ is a uniform $(n-1)$-subset of $[N]\setminus\{v\}$. Apply the induction hypothesis to the function $A \mapsto F(A \cup \{v\})$. Conditionally on $I_v = 0$, the set $B$ is a uniform $n$-subset of $[N]\setminus\{v\}$, so the induction hypothesis applies directly. In both cases, the admissible swaps are exactly those counted by $S_v(B)$. Averaging over the two values of $I_v$ therefore gives

$$\mathbb{E}[\mathrm{Var}(X \mid I_v)] \leq \frac{1}{2(N-1)}\mathbb{E}[S_v(B)].$$

Each ordered swap pair $(i, j)$ is included in $S_v(B)$ for exactly $N-2$ choices of $v$, namely those outside $\{i, j\}$. Thus, pointwise in $B$,

$$\sum_v S_v(B) = (N-2)\sum_{i\in B}\sum_{j\notin B}\big(F(B) - F(B^{i\to j})\big)^2.$$

It follows that

$$\frac{1}{N}\sum_v \mathbb{E}[\mathrm{Var}(X \mid I_v)] \leq \frac{N-2}{2N(N-1)}\mathcal{D}(F). \tag{35}$$

Combining equation 34 and equation 35, and dividing by the positive factor $(N-2)/(N-1)$, proves equation 33. Equation equation 32 and the coordinatewise reduction then give equation 31. □

### C.3 A VARIANCE UPPER BOUND LINEAR IN THE NUMBER OF BLOCKS

**Theorem 28** (Linear variance upper bound)**.** *Under Assumption 26, for the local mean-aggregation model of Appendix B,*

$$\boxed{\mathcal{V}_m(w) \le \frac{8G_\nabla^2 \mathcal{H}_L}{\tau^2} \frac{m-1}{N}.} \tag{36}$$

*Equivalently,*

$$\mathcal{V}_m(w) \le 8G_\nabla^2 \frac{\mathcal{H}_L}{\tau} \frac{m-1}{N_t}.$$

*For a fixed graph, training set, depth, and parameter vector, the upper bound is linear in $m-1$, with a coefficient independent of $m$. In particular, $\mathcal{V}_1(w) = 0$.*

*Proof.* When $m = 1$, the only block is $V$, so the estimator is deterministic and $\mathcal{V}_1(w) = 0$. Suppose henceforth that $m > 1$, so $1 \le n < N$, and draw $(B, i, j, B')$ as in Lemma 27.

*Locality of the gradient change.* For a node $v$ common to $B$ and $B'$, the $L$-layer forward computation depends only on its induced computation neighborhood, which is contained in $\mathcal{N}_L(v)$. If neither $i$ nor $j$ belongs to $\mathcal{N}_L(v)$, the two sampled computations coincide as functions of the shared parameters. Their per-node loss gradients at $w$ are therefore equal.

By symmetry of graph distance, only common training nodes in $\mathcal{N}_L(i) \cup \mathcal{N}_L(j)$ can have changed gradients. Each such gradient changes by at most $2G_\nabla$. A removed or added training contribution has norm at most $G_\nabla$, and its node also belongs to this union because the receptive fields are closed. The triangle inequality therefore gives

$$\left\| \sum_{v \in B^{\mathrm{tr}}} \nabla_w \ell_v^B(w) - \sum_{v \in B'^{\mathrm{tr}}} \nabla_w \ell_v^{B'}(w) \right\|_2 \le 2G_\nabla (h_i + h_j).$$

Using the deterministic normalization in equation 28, we obtain

$$\|\hat{g}_B - \hat{g}_{B'}\|_2^2 \le \frac{4G_\nabla^2}{n^2\tau^2}(h_i + h_j)^2. \tag{37}$$

*Averaging the swap influence.* Unconditionally, $(i, j)$ is uniform over ordered pairs of distinct nodes. Indeed, for distinct $u, v \in V$,

$$\mathbb{P}(i = u, j = v) = \frac{\mathbb{P}(u \in B,\ v \notin B)}{n(N-n)} = \frac{1}{N(N-1)}.$$

In particular, both endpoints are marginally uniform, although they are not independent. Hence

$$\mathbb{E}[(h_i + h_j)^2] \le 2\mathbb{E}[h_i^2 + h_j^2] = 4\mathcal{H}_L.$$

Taking expectations in equation 37 yields

$$\mathbb{E}\|\hat{g}_B - \hat{g}_{B'}\|_2^2 \le \frac{16G_\nabla^2 \mathcal{H}_L}{n^2\tau^2}.$$

*Applying the single-swap inequality.* Lemma 27 now gives

$$\begin{aligned} \mathcal{V}_m(w) &\le \frac{n(N-n)}{2N} \mathbb{E}\|\hat{g}_B - \hat{g}_{B'}\|_2^2 \\ &\le \frac{8G_\nabla^2 \mathcal{H}_L}{\tau^2} \frac{N-n}{nN} \\ &= \frac{8G_\nabla^2 \mathcal{H}_L}{\tau^2} \frac{m-1}{N}, \end{aligned}$$

where the last equality uses $n = N/m$. This proves equation 36; its equivalent form follows from $N_t = \tau N$. □

**Interpretation and connection to $R$.** The theorem provides a linear-in-$m-1$ upper bound at fixed parameters, accounting for shared receptive fields and fixed-size sampling dependence. It also controls the expected empirical gradient variance across blocks. Indeed, for a random partition $\mathcal{P} = (B_0, \dots, B_{m-1})$, write

$$\bar{g}_{\mathcal{P}} := \frac{1}{m}\sum_{k=0}^{m-1} \hat{g}_{B_k} = \nabla \bar{L}_{\mathrm{Sampl}}(w), \qquad R_{\mathcal{P}} := \frac{1}{m}\sum_{k=0}^{m-1} \|\hat{g}_{B_k} - \bar{g}_{\mathcal{P}}\|_2^2,$$

where the block gradients use the deterministic normalization in equation 28. Since all blocks have the same marginal distribution, the variance decomposition gives

$$\mathbb{E}_{\mathcal{P}}[R_{\mathcal{P}}] = \mathcal{V}_m(w) - \mathbb{E}_{\mathcal{P}}\big\|\bar{g}_{\mathcal{P}} - \mathbb{E}_{\mathcal{P}}\bar{g}_{\mathcal{P}}\big\|_2^2 \le \frac{8G_\nabla^2 \mathcal{H}_L}{\tau^2}\frac{m-1}{N}.$$

Thus the same bound applies to the expected empirical dispersion $R$, even though the block gradients are dependent. The influence coefficient $\mathcal{H}_L$ captures graph structure and training-node placement; on heavy-tailed graphs with large receptive fields, the resulting envelope may be conservative.

## D TAIL DEGREE DISTRIBUTION UNDER RANDOM NODE SUBSAMPLING

We show that RNS preserves the power-law exponent of the expected degree distribution over the admissible degree range. We first express the subsampled distribution using an exact hypergeometric kernel. We then show that the dominant contribution comes from original degrees near $mk$, while contributions from outside the power-law window are negligible. Combining these estimates yields the claimed preservation of the exponent.

Throughout, we fix $\alpha > 1$ and an integer $m \ge 2$. Let $\mathcal{N} \subseteq m\mathbb{N}$ be an infinite index set. For each $N \in \mathcal{N}$, let $G_N = (V_N, E_N)$ be a simple graph on $N$ vertices, with empirical degree distribution

$$\nu_N(d) := \frac{\#\{v \in V_N : \deg_{G_N}(v) = d\}}{N}, \qquad 0 \le d \le N-1.$$

Set $b := N/m$ and $q := 1/m$, and partition $V_N$ uniformly at random into $m$ batches of size $b$. Let $U$ be uniform on $V_N$ and independent of the partition, let $B$ be its batch, and put

$$D := \deg_{G_N}(U), \qquad \widetilde{D} := \deg_{G_N[B]}(U), \qquad \widetilde{p}_N(k) := \mathbb{P}(\widetilde{D} = k).$$

Thus $\widetilde{p}_N$ is the expected empirical degree distribution of the induced graph on a uniformly selected batch.

**Assumption A (uniform power law on a growing window).** There exist a constant $C > 0$, an integer $d_0 \ge 1$, integer-valued cutoffs $d_1(N) \in \{d_0, \dots, N-1\}$ tending to infinity, and numbers $\delta_N \in [0, 1)$ tending to zero such that, for every $N \in \mathcal{N}$,

$$\sup_{\substack{d \in \mathbb{Z} \\ d_0 \le d \le d_1(N)}} \left| \frac{\nu_N(d)}{C\, d^{-\alpha}} - 1 \right| \le \delta_N. \tag{A}$$

No upper envelope is assumed outside this window. All implied constants below may depend on $\alpha, m, C$, but not on $N$ or the admissible degree $k$.

**Proposition 29.** *Let $k = k(N)$ be an integer sequence tending to infinity and satisfying*

$$2mk \le d_1(N).$$

*Then, as $N \to \infty$ along $\mathcal{N}$,*

$$\widetilde{p}_N(k) = C\, m^{1-\alpha} k^{-\alpha}\left[1 + O\left(\delta_N + \frac{1}{k}\right)\right].$$

*In particular,*

$$\widetilde{p}_N(k) = C\, q^{\alpha-1} k^{-\alpha}\big(1 + o(1)\big).$$

The exponent is unchanged, and the amplitude is multiplied by $m^{1-\alpha}$. The conclusion holds along every diverging integer degree sequence in the certified range $k \le d_1(N)/(2m)$. It concerns the expected empirical distribution; no concentration statement for a realized batch is asserted.

THE SUBSAMPLING KERNEL

Conditional on $U$, the remaining $b-1$ members of its batch form a uniform $(b-1)$-subset of $V_N \setminus \{U\}$. Hence

$$H_{N,d}(k) := \mathbb{P}(\widetilde{D} = k \mid D = d) = \frac{\binom{d}{k}\binom{N-1-d}{b-1-k}}{\binom{N-1}{b-1}}, \qquad \widetilde{p}_N(k) = \sum_{d=k}^{N-1} \nu_N(d) H_{N,d}(k). \tag{38}$$

Here binomial coefficients are zero when the lower argument is outside its usual range. In particular, $H_{N,d}(k) = 0$ for $d < k$.

The window condition implies

$$2mk \le d_1(N) \le N - 1 < mb,$$

and therefore $k < b/2$.

**Lemma 30.** *For every integer $0 \le k < b$, the following hold.*

*(i) The kernel satisfies*

$$\sum_{d=k}^{N-1} H_{N,d}(k) = m = \frac{1}{q}.$$

*(ii) Let $X_1 < \cdots < X_b$ be a uniform $b$-subset of $\{1, \dots, N\}$, and define $Y_k := X_{k+1} - 1$. Then*

$$\mathbb{P}(Y_k = d) = q\, H_{N,d}(k), \qquad k \le d \le N-1.$$

*Consequently,*

$$\widetilde{p}_N(k) = m\, \mathbb{E}[\nu_N(Y_k)]. \tag{39}$$

*(iii) The mean and variance satisfy*

$$\mathbb{E}[Y_k] = km + (m-1)\left(1 - \frac{k+1}{b+1}\right) \in [km, km+m]$$

*and*

$$\mathrm{Var}(Y_k) \le (k+1)m(m-1).$$

*Proof.* For (i), Vandermonde's identity gives

$$\sum_{d=k}^{N-1} \binom{d}{k}\binom{N-1-d}{b-1-k} = \binom{N}{b},$$

while

$$\binom{N}{b} = \frac{N}{b}\binom{N-1}{b-1} = m\binom{N-1}{b-1}.$$

For (ii), the order-statistic mass function is

$$\mathbb{P}(X_{k+1} = d+1) = \frac{\binom{d}{k}\binom{N-1-d}{b-k-1}}{\binom{N}{b}} = q\, H_{N,d}(k).$$

Substituting this identity into equation 38 proves equation 39.

For (iii), the order-statistic formulas for sampling without replacement give

$$\mathbb{E}[X_{k+1}] = \frac{(k+1)(N+1)}{b+1}, \qquad \mathrm{Var}(X_{k+1}) = \frac{(k+1)(b-k)(N+1)(N-b)}{(b+1)^2(b+2)}.$$

Using $N + 1 = m(b+1) - (m-1)$ in the mean yields the stated expression. Its correction to $km$ lies between zero and $m-1$.

For the variance, use $b - k \le b$, $N + 1 \le m(b+1)$, and $N - b = b(m-1)$ to obtain

$$\mathrm{Var}(Y_k) \le \frac{(k+1)b \cdot m(b+1) \cdot b(m-1)}{(b+1)^2(b+2)} \le (k+1)m(m-1).$$

□

PROOF OF PROPOSITION 29

Write $Y := Y_k$ and $t := mk$. All asymptotics are along $\mathcal{N}$, with $k = k(N) \to \infty$. We may therefore assume $k \geq d_0$.

**Step 1 (averaging the ideal power law).** Lemma 30 (iii) gives

$$0 \leq \mathbb{E}[Y - t] \leq m$$

and

$$\mathbb{E}[(Y-t)^2] = \mathrm{Var}(Y) + (\mathbb{E}[Y] - t)^2 \leq (k+1)m(m-1) + m^2 = O(k).$$

Moreover, $Y \geq k$ almost surely.

Apply Taylor's theorem to $f(x) := x^{-\alpha}$ around $t$. Both $Y$ and $t$ are at least $k$, and

$$\sup_{x \geq k} |f''(x)| = \alpha(\alpha+1)k^{-\alpha-2}.$$

Thus

$$Y^{-\alpha} = t^{-\alpha} - \alpha t^{-\alpha-1}(Y-t) + R, \qquad |R| \leq \frac{\alpha(\alpha+1)}{2} k^{-\alpha-2}(Y-t)^2.$$

Taking expectations yields

$$\begin{aligned} \mathbb{E}[Y^{-\alpha}] &= t^{-\alpha} - \alpha t^{-\alpha-1}\mathbb{E}[Y-t] + O\big(k^{-\alpha-2}\mathbb{E}[(Y-t)^2]\big) \\ &= t^{-\alpha} + O(k^{-\alpha-1}) \\ &= t^{-\alpha}\left(1 + O(k^{-1})\right). \end{aligned} \tag{40}$$

**Step 2 (exponentially small mass beyond the window).** Set

$$L := 2mk, \qquad Z := \#\{i : X_i \leq L\}.$$

The hypothesis ensures $L \leq d_1(N) \leq N - 1$. Since the sampled subset is uniform, $Z$ is hypergeometric with population size $N$, number of marked elements $L$, and sample size $b$. In particular,

$$\mu := \mathbb{E}[Z] = \frac{bL}{N} = 2k.$$

If $Y > d_1(N)$, then the first $L$ positions contain at most $k$ sampled elements. Hence

$$\{Y > d_1(N)\} \subseteq \{Z \leq k\}.$$

The multiplicative lower-tail Chernoff inequality for hypergeometric sampling states that

$$\mathbb{P}\big(Z \leq (1-\eta)\mu\big) \leq \exp\left(-\frac{\eta^2\mu}{2}\right), \qquad 0 < \eta < 1.$$

Applying it with $\eta = 1/2$ gives

$$\mathbb{P}(Y > d_1(N)) \leq \mathbb{P}(Z \leq k) \leq e^{-k/4}. \tag{41}$$

**Step 3 (transferring the empirical approximation).** Let

$$A := \{Y \leq d_1(N)\}.$$

Since $Y \geq k \geq d_0$, Assumption A applies on $A$, giving

$$|\nu_N(Y) - CY^{-\alpha}| \leq C\delta_N Y^{-\alpha}.$$

On the complementary event, we use only

$$0 \leq \nu_N(Y) \leq 1, \qquad Y^{-\alpha} \leq k^{-\alpha}.$$

It follows from equation 41 that

$$\begin{aligned} \big|\mathbb{E}[\nu_N(Y)] - C\mathbb{E}[Y^{-\alpha}]\big| &\leq \mathbb{E}\big[|\nu_N(Y) - CY^{-\alpha}|\big] \\ &\leq C\delta_N \mathbb{E}\big[Y^{-\alpha}\mathbf{1}_A\big] + (1 + Ck^{-\alpha})\mathbb{P}(A^c) \\ &\leq C\delta_N k^{-\alpha} + (1 + Ck^{-\alpha})e^{-k/4}. \end{aligned}$$

Combining this estimate with equation 39 and equation 40, we obtain

$$\begin{aligned}\widetilde{p}_N(k) &= m\,\mathbb{E}[\nu_N(Y)] \\ &= mC\,t^{-\alpha}\left(1+O(k^{-1})\right) + O(\delta_N k^{-\alpha}) + O(e^{-k/4}) \\ &= C\,m^{1-\alpha}k^{-\alpha}\left[1+O\Big(\delta_N + k^{-1} + k^{\alpha}e^{-k/4}\Big)\right].\end{aligned}$$

Because $\alpha$ is fixed, $k^{\alpha}e^{-k/4} = O(k^{-1})$. Therefore

$$\widetilde{p}_N(k) = C\,m^{1-\alpha}k^{-\alpha}\left[1+O\left(\delta_N + \frac{1}{k}\right)\right].$$

Finally, $\delta_N \to 0$ and $k \to \infty$, which also proves the stated asymptotic equivalence. □

# E DATASETS

We evaluate our method on a diverse collection of large-scale transductive node-classification benchmarks. Our main evaluation uses three widely adopted large graphs: OGBN-ARXIV, OGBN-PRODUCTS Hu et al. (2020), and POKEC Takac & Zabovsky (2012). These datasets are standard in the scalable graph learning literature and cover complementary regimes: OGBN-ARXIV is a comparatively small large-scale citation graph, OGBN-PRODUCTS is substantially larger while remaining computationally tractable, and POKEC is a large social network with different structural and feature properties from the two OGB datasets.

We further extend our evaluation with seven node-classification datasets from GraphLand Bazhenov et al. (2024). These benchmarks increase the diversity of graph sizes, label spaces, homophily levels, and structural properties considered in our experiments. We include three multi-class classification datasets, HM-CATEGORIES, POKEC-REGIONS, and WEB-TOPICS, and four binary classification datasets, TOLOKERS-2, CITY-REVIEWS, ARTNET-EXP, and WEB-FRAUD. We refer readers to the original GraphLand paper for details on the construction of these datasets. Unless otherwise stated, all datasets are processed as undirected graphs. Summary statistics are reported in Table 5.

**OGBN-ARXIV.** OGBN-ARXIV is a directed citation network of Computer Science arXiv papers indexed by MAG Hu et al. (2020). Nodes correspond to papers, and directed edges represent citation relationships. Each node is associated with a 128-dimensional feature vector obtained by averaging word embeddings from the paper title and abstract. We use the official OGB time-based split, where papers published up to 2017 are used for training, papers published in 2018 for validation, and papers published from 2019 onward for testing.

**OGBN-PRODUCTS.** OGBN-PRODUCTS is an undirected, unweighted Amazon co-purchasing graph Hu et al. (2020). Nodes represent products, and edges connect products that are frequently bought together. Node features are constructed from bag-of-words representations of product descriptions and reduced to 100 dimensions using PCA. We use the official sales-rank split, where the top 8% most popular products are used for training, the next 2% for validation, and the remaining products for testing. This split reflects a realistic setting in which labels are first collected for popular items.

**POKEC.** POKEC is a large-scale social network from Slovakia Takac & Zabovsky (2012). We consider the binary node-classification task of predicting user gender. Following Luo et al. (2024), we use a random 1/1/8 train/validation/test split. Since the exact split used by Luo et al. (2024) is not publicly released, we resample the split at the beginning of each training run.

**GraphLand datasets.** GraphLand Bazhenov et al. (2024) provides a collection of large-scale node-classification datasets spanning diverse graph structures, label spaces, and homophily levels. We use the seven GraphLand node-classification datasets listed in Table 5. Each dataset is provided with two official random split regimes, Random Low (RL) and Random High (RH), which differ in the amount of labeled data available for training. In the RL regime, nodes are randomly partitioned into train/validation/test sets with proportions 10%/10%/80%, whereas in the RH regime the corresponding proportions are 50%/25%/25%. We report results on both regimes.

Table 5: Graph statistics for the datasets used in our experiments. Hom. denotes homophily. Diameter (Diam.) and average shortest-path distance (Avg. dist.) are respectively the maximum and the average shortest-path length between two nodes in the same connected component, estimated using BFS sampling. Edges are counted twice since all datasets are processed as undirected graphs.

| Dataset | Nodes | Edges | Density | Avg. deg. | # Isol. | # CC | Hom. | Diam. | Avg. dist. |
|---|---|---|---|---|---|---|---|---|---|
| OGBN-ARXIV | 169K | 1.17M | $4.10\times10^{-5}$ | 13.8 | 0 | 1 | 0.65 | 24 | 6.12 |
| OGBN-PRODUCTS | 2.45M | 61.9M | $1.03\times10^{-5}$ | 50.2 | 48K | 52K | 0.81 | 29 | 5.35 |
| POKEC | 1.63M | 30.6M | $1.15\times10^{-5}$ | 27.3 | 0 | 1 | 0.42 | 14 | 4.68 |
| HM-CATEGORIES | 46.5K | 10.7M | $4.95\times10^{-3}$ | 460.9 | 0 | 1 | 0.38 | 13 | 2.45 |
| POKEC-REGIONS | 1.63M | 30.6M | $1.15\times10^{-5}$ | 27.3 | 0 | 1 | 0.98 | 14 | 4.68 |
| WEB-TOPICS | 2.9M | 12.4M | $1.47\times10^{-6}$ | 8.6 | 0 | 1 | 0.55 | 36 | 3.08 |
| TOLOKERS-2 | 11.8K | 2.33M | $1.67\times10^{-2}$ | 88.3 | 0 | 1 | 0.10 | 11 | 2.79 |
| CITY-REVIEWS | 148.8K | 1.2M | $5.42\times10^{-5}$ | 15.7 | 0 | 1 | 0.69 | 19 | 4.91 |
| ARTNET-EXP | 50.4K | 560K | $2.21\times10^{-4}$ | 11.1 | 0 | 1 | 0.28 | 13 | 4.42 |
| WEB-FRAUD | 2.9M | 12.9M | $1.53\times10^{-6}$ | 8.6 | 0 | 1 | 0.32 | 36 | 3.08 |

# F EXPERIMENTAL DETAILS

This section describes the experimental protocols used throughout the paper. We provide implementation and evaluation details for each group of experiments, organized into four subsections: the sampling benchmark, the GraphLand benchmark, the ablation study on RNS, and the architecture benchmark.

## F.1 SAMPLING BENCHMARK

This section describes the experimental protocol used to benchmark sampling strategies on large-scale graphs. We consider three datasets, OGBN-ARXIV, OGBN-PRODUCTS, and POKEC, and compare several mini-batch training schemes under a common architecture and tuning protocol. Our goal is to isolate the effect of the *sampling strategy* itself, while ensuring that each method is evaluated under competitive optimization settings.

Unless otherwise stated, validation and test metrics are computed on the full graph. The only exception is neighbor sampling, for which we also use sampled neighborhoods at evaluation time, following the standard inference pipeline of that method.

The hyperparameter tuning phase was run on a heterogeneous GPU cluster containing NVIDIA H100 GPUs with 80GB of memory, L40S and A40 GPUs with 48GB of memory, and A100 GPUs with 40GB of memory. After tuning, all final training runs used for reporting results were run on H100 GPUs.

### F.1.1 IMPLEMENTATION OF SAMPLING STRATEGIES

All sampling procedures are implemented in PyTorch Geometric (PyG) Fey & Lenssen (2019); Fey et al. (2025). For all methods except Random Node Sampling (RNS), we use the corresponding PyG data loaders. In all cases, the sampled mini-batch adjacency is converted to a sparse representation before being passed to the model.

**Neighbor sampling Hamilton et al. (2017).** We use the standard neighbor sampler, parameterized by a fanout vector

$$\mathbf{f} = (f_1, \ldots, f_L),$$

where $f_\ell$ is the maximum number of neighbors sampled at layer $\ell$, together with a batch size of target nodes. Each mini-batch is formed by first selecting a set of seed nodes, then recursively sampling up to $f_\ell$ neighbors per node. One epoch corresponds to iterating over the full set of seed nodes once.

**ClusterGCN Chiang et al. (2019).** ClusterGCN is parameterized by the number of graph partitions $C$ and the number of sampled partitions per mini-batch $B$. We first partition the graph into $C$

clusters using METIS Karypis & Kumar (1998). At each optimization step, we sample $B$ clusters uniformly at random and construct the mini-batch as the induced subgraph over the union of their nodes.

**GraphSAINT Zeng et al. (2020).** GraphSAINT is parameterized by the random-walk length $L_{\mathrm{w}}$ and the number of seed nodes $S$. For each mini-batch, we sample $S$ seeds uniformly at random and launch one random walk of length $L_{\mathrm{w}}$ from each seed. The mini-batch is the induced subgraph over the union of visited nodes.

To make GraphSAINT comparable to RNS in terms of sampled graph volume per epoch, we fix the number of optimization steps per epoch to

$$\left\lfloor \frac{|V|}{S \cdot L_{\mathrm{w}}} \right\rfloor,$$

so that the total number of sampled node visits per epoch is on the order of $|V|$, up to overlaps among walks.

**LADIES Zou et al. (2019).** We reimplement LADIES from the original GitHub repository, with a GPU implementation of the sampling procedure. LADIES is parameterized by a batch size of target nodes and a layer-wise sampling budget vector

$$\mathbf{s} = (s_1, \ldots, s_L),$$

where $s_\ell$ denotes the number of nodes sampled for layer $\ell$. For each mini-batch, LADIES first selects a set of target nodes of size given by the batch size, then samples nodes layer by layer using an importance distribution defined over the one-hop neighbors of the current upper-layer node set. Let $\mathcal{S}_{\ell+1}$ denote the sampled nodes at layer $\ell+1$, and let $\mathbf{A}_{:,\mathcal{S}_{\ell+1}}$ be the submatrix of the normalized adjacency matrix restricted to the columns indexed by $\mathcal{S}_{\ell+1}$. LADIES assigns each candidate node $i$ the importance score

$$q_i = \left\| \mathbf{A}_{i,\mathcal{S}_{\ell+1}} \right\|_2^2,$$

and samples $s_\ell$ nodes with probabilities

$$p_i = \frac{q_i}{\sum_j q_j}.$$

Thus, nodes with larger aggregate normalized connectivity to the current upper-layer nodes are more likely to be selected. One epoch corresponds to iterating over the full set of target nodes once.

**Random Node Sampling (RNS).** We implement RNS with a custom GPU-based data loader. Since RNS only requires uniformly sampling nodes and extracting the corresponding induced subgraphs, a GPU implementation is substantially faster in our setting than the existing CPU-based PyG routine. RNS is parameterized by the number of partitions $P$, which controls the mini-batch size.

At the beginning of each epoch, we draw a random permutation of all nodes and split it into $P$ disjoint blocks of size $\lfloor |V|/P \rfloor$. Each block defines one mini-batch through its induced subgraph. Pseudocode is provided in Algorithm 1.

### F.1.2 Fair hyperparameter tuning protocol

A central difficulty when comparing sampling methods is that performance depends not only on the sampler itself, but also on the interaction between the sampler, the optimizer, and regularization. To ensure a fair comparison, we use the following protocol.

First, we *fix the model architecture within each dataset* across all sampling methods. In particular, hidden dimension, number of layers, normalization type, input/output projection choices, and residual connections are taken from Luo et al. (2024). The resulting architectures are summarized in Table 6. By keeping the architecture fixed, we reduce the risk that differences in accuracy are driven by model design rather than by sampling.

Second, for *each sampling method and each choice of sampling hyperparameters*, we independently tune the *training-related hyperparameters*. Concretely, for every dataset and every candidate sampling configuration, we run a separate hyperparameter sweep over learning rate, dropout, and weight

**Algorithm 1** RNS data loading for one epoch

**Require:** Graph $G = (V, E)$, device, number of partitions $P$
$G \leftarrow G.\texttt{to}(\texttt{device})$
$n \leftarrow |V|$
$b \leftarrow \lfloor n/P \rfloor$
$\pi \leftarrow \texttt{torch.randperm}(n, \texttt{device} = \texttt{device})$
**for** $i = 0, 1, \ldots, P-1$ **do**
  $s \leftarrow i \cdot b$
  $t \leftarrow s + b$
  $S_i \leftarrow \pi[s:t]$
  $G_i \leftarrow \texttt{extract_subgraph_from_nodes}(G, S_i)$
  $G_i \leftarrow \texttt{to_sparse_tensor}(G_i)$
  **yield** $G_i$
**end for**

decay. This is important: we do *not* tune one global optimizer configuration per method and reuse it across all sampling settings. Instead, each sampling configuration is allowed to operate under its own best optimization hyperparameters.

Third, the *sampling hyperparameters themselves* are selected from method-specific search spaces motivated by the original papers. For each candidate sampling configuration, we perform 50 Bayesian-optimization runs with WEIGHTS & BIASES sweeps over the training hyperparameters. We then report, for each dataset and method, the configuration with the best validation performance.

This procedure separates the roles of the different hyperparameter classes:

- **Architecture hyperparameters** are fixed per dataset and shared across methods.
- **Sampling hyperparameters** are explored within a method-specific search space.
- **Training hyperparameters** are tuned independently for every candidate sampling configuration.

This layered protocol is designed to make the comparison as fair as possible across methods with very different sampling behaviors.

**Training hyperparameters.** Across all datasets and methods, we tune the same set of optimization hyperparameters:

- dropout rate,
- learning rate,
- weight decay.

The common search space is:

| Hyperparameter | Search space |
|---|---|
| Dropout | $\{0.0, 0.2, 0.3, 0.5, 0.7\}$ |
| Learning rate | $\{10^{-2}, 5 \times 10^{-3}, 10^{-3}, 5 \times 10^{-4}, 10^{-4}\}$ |
| Weight decay | $\{0, 10^{-4}, 5 \times 10^{-4}, 10^{-3}, 5 \times 10^{-3}\}$ |

**Architecture hyperparameters.** We use GraphSAGE as the backbone model for all experiments. The dataset-specific architectural choices inherited from Luo et al. (2024) are reported in Table 6. For LADIES, to follow the original paper, we employ a GCN architecture with the same hyperparameters except the number of layers which was fixed 5.

For neighbor sampling, we train for 20 epochs only. This choice accounts for its substantially larger number of mini-batches per epoch, resulting from the smaller batch size, and its considerably higher training cost relative to the other methods (see Appendix G.6).

Table 6: Architecture hyperparameters used for each dataset. These parameters are fixed across sampling methods. In proj. and out proj. denote input and output projections, respectively. Note that Neighborhood sampling is trained for 20 epochs.

| Dataset | Hid. | Layers | In proj. | Out proj. | Res. | Norm | Epochs |
|---|---|---|---|---|---|---|---|
| OGBN-ARXIV | 256 | 4 | No | Yes | Yes | BN | 1000 |
| OGBN-PRODUCTS | 256 | 5 | No | No | No | LN | 1000 |
| POKEC | 256 | 7 | No | Yes | Yes | BN | 2000 |

**Sampling hyperparameter search spaces.** We now describe the search spaces used for the method-specific sampling hyperparameters. For ClusterGCN and GraphSAINT, we adopt candidate configurations directly from the original papers Chiang et al. (2019); Zeng et al. (2020). For neighbor sampling, the original paper explores only relatively shallow fanout schedules Hamilton et al. (2017), so we use standard fanout values and additionally include one deeper configuration to match the depth of our models. For RNS, since there is no canonical search space in prior work, we vary the number of partitions over a broad range.

For neighbor sampling, we evaluate the fanout vectors

$$\mathbf{f} \in \{[30, 10], [30, 10, 10], [30, 10, 10, 10]\}.$$

The first two configurations are standard shallow fanout schedules. We additionally consider the depth-4 configuration $[30, 10, 10, 10]$ in order to study the effect of increased sampling depth on predictive performance. We do not consider deeper schedules because, in our implementation, PyG's neighbor sampler runs on the CPU, making the sampling cost too high beyond depth 4 (See Section G.6).

For ClusterGCN, following the configurations considered in the original paper Chiang et al. (2019), we evaluate

$$(C, B) \in \{(15000, 10), (1500, 20), (200, 1), (3000, 100)\}.$$

Since our hardware likely provides more available VRAM than was used in the original work, and to ensure a fair comparison with RNS, we additionally include a more memory-intensive ClusterGCN setting.

For GraphSAINT, following the configurations considered in the original paper Zeng et al. (2020), we evaluate

$$(L_{\mathrm{w}}, S) \in \{(2, 6000), (2, 2000), (4, 2000)\}.$$

For the same reason, we also include an additional, more memory-intensive GraphSAINT setting.

For LADIES, we evaluate two configurations of the layer-wise sampling budget vector $\mathbf{s}$ and batch size $B$:

$$(\mathbf{s}, B) \in \{([8192, 8192, 8192, 8192, 8192], 512), ([65536, 65536, 65536, 65536, 65536], 8192)\}.$$

The first configuration is chosen to remain close to the hyperparameter scale used in the original LADIES paper Zou et al. (2019). Since our experiments consider substantially larger graphs, we also evaluate a larger sampling budget and batch size to account for the increased graph scale.

For RNS, we evaluate

$$m \in \{2, 3, \ldots, 10\}.$$

#### F.1.3 SELECTED HYPERPARAMETERS

The best configurations selected by the above tuning protocol are reported in Table 7. In particular, the table should be read as the *result of the full fairness pipeline described above*: for each dataset and method, we first searched over sampling hyperparameters, and for each candidate sampling configuration we separately optimized learning rate, dropout, and weight decay.

### F.2 EMPIRICAL EVALUATION OF THE IMPLICIT REGULARIZATION WITH RNS

We use the same experimental setup and hyperparameters as in the sampling benchmark. In particular, the data preprocessing, train/validation/test splits, model architectures, and evaluation protocol are kept unchanged for OGBN-PRODUCTS.

Table 7: Best hyperparameters selected for each dataset and sampling method. Sampling parameters are shown together with their corresponding tuned training hyperparameters. For LADIES, $\mathbf{s} = [a]^5$ denotes a five-layer sampling budget with $s_\ell = a$ for all layers.

| Dataset | Method | Sampling params | LR | Dropout | WD |
|---|---|---|---|---|---|
| OGBN-ARXIV | Full | – | $5 \times 10^{-4}$ | 0.5 | $5 \times 10^{-4}$ |
| | Neighbor | $\mathbf{f} = [30, 10, 10]$ | $5 \times 10^{-4}$ | 0.2 | $10^{-3}$ |
| | ClusterGCN | $C = 3000,\ B = 100$ | $10^{-4}$ | 0.5 | $10^{-3}$ |
| | GraphSAINT | $L_{\text{w}} = 2,\ S = 6000$ | $10^{-4}$ | 0.5 | 0 |
| | LADIES | $\mathbf{s} = [65536]^5,\ B = 8192$ | $10^{-2}$ | 0.2 | 0 |
| | RNS | $m = 2$ | $5 \times 10^{-4}$ | 0.5 | 0 |
| OGBN-PRODUCTS | Full | – | $5 \times 10^{-4}$ | 0.3 | $5 \times 10^{-4}$ |
| | Neighbor | $\mathbf{f} = [30, 10, 10, 10]$ | $10^{-4}$ | 0.3 | 0 |
| | ClusterGCN | $C = 1500,\ B = 20$ | $10^{-3}$ | 0.7 | $10^{-4}$ |
| | GraphSAINT | $L_{\text{w}} = 2,\ S = 6000$ | $5 \times 10^{-4}$ | 0.5 | 0 |
| | LADIES | $\mathbf{s} = [8192]^5,\ B = 512$ | $10^{-3}$ | 0.0 | 0 |
| | RNS | $m = 10$ | $5 \times 10^{-4}$ | 0.3 | $10^{-4}$ |
| POKEC | Full | – | $10^{-3}$ | 0.7 | $5 \times 10^{-4}$ |
| | Neighbor | $\mathbf{f} = [30, 10, 10, 10]$ | $10^{-4}$ | 0.3 | 0 |
| | ClusterGCN | $C = 1500,\ B = 20$ | $10^{-4}$ | 0.3 | $5 \times 10^{-3}$ |
| | GraphSAINT | $L_{\text{w}} = 2,\ S = 6000$ | $10^{-4}$ | 0.5 | $5 \times 10^{-4}$ |
| | LADIES | $\mathbf{s} = [8192]^5,\ B = 512$ | $5 \times 10^{-4}$ | 0.5 | 0 |
| | RNS | $m = 2$ | $10^{-3}$ | 0.3 | $10^{-4}$ |

For RNS (no batch), we iterate over all mini-batches $B_k$ in the epoch, accumulate the corresponding losses gradients, and perform a single optimizer step after the full pass. This is equivalent to gradient accumulation over the entire epoch, and ensures that the update is computed from the same examples while removing the effect of stochastic mini-batch updates.

Unless otherwise stated, we do not use weight decay. Dropout is fixed to $0.5$ in all experiments. For SGD, we use a learning rate of $0.03$. For SGD with momentum, we use a learning rate of $0.03$ and momentum coefficient $0.9$. For Adam, we use a learning rate of $0.001$.

### F.3 GRAPHLAND BENCHMARK

We follow the experimental protocol of GraphLand Bazhenov et al. (2024) as closely as possible. In particular, we use the official Random Low and Random High splits, the same evaluation metrics, and the same training budget of 1000 epochs. For evaluation, we report accuracy on multi-class classification tasks and average precision on binary classification tasks. We also adopt the GraphSAGE architecture used in the GraphLand benchmark, which differs slightly from our default architecture: it consists of three graph convolutional layers, with two-layer MLPs inserted between successive graph convolutions. We use this architecture to ensure consistency with the original GraphLand benchmark.

For hyperparameter selection, we follow the same procedure as in the sampling benchmark. For each dataset and method, we run 50 trials using W&B Bayesian search over the same hyperparameter space for dropout, weight decay, and learning rate. For RNS, we additionally tune the number of parts over $\{2, 3, 5, 7, 10\}$. Hyperparameter search results are given in Table 8.

These experiments are run on NVIDIA L40S GPUs with 48GB of memory. For memory-intensive settings, we use activation checkpointing: specifically, for full-graph training on POKEC-REGIONS and for RNS training on WEB-FRAUD. For full-graph training on WEB-FRAUD and WEB-TOPICS, activation checkpointing alone is not sufficient; therefore, we use layer-parallel training across two L40S GPUs while keeping the training configuration otherwise unchanged.

Table 8: GraphLand hyperparameters. For RNS, $K$ denotes the number of parts.

| Dataset | Reg. | Full | | | RNS | | | |
|---|---|---|---|---|---|---|---|---|
| | | LR | Drop. | WD | LR | Drop. | WD | $K$ |
| ARTNET-EXP | RH | 0.0001 | 0.1 | 0.001 | 0.0001 | 0.1 | 0.001 | 2 |
| ARTNET-EXP | RL | 0.005 | 0.2 | 0.001 | 0.0001 | 0.3 | 0.001 | 2 |
| CITY-REVIEWS | RH | 0.0005 | 0.2 | 0.0001 | 0.0005 | 0.2 | 0.0005 | 3 |
| CITY-REVIEWS | RL | 0.0001 | 0.1 | 0.001 | 0.0001 | 0.1 | 0.001 | 2 |
| HM-CATEGORIES | RH | 0.0005 | 0.2 | 0.0001 | 0.0001 | 0.2 | 0.0005 | 3 |
| HM-CATEGORIES | RL | 0.0001 | 0.2 | 0.0005 | 0.0001 | 0.3 | 0.001 | 2 |
| POKEC-REGIONS | RH | 0.0005 | 0.0 | 0.0005 | 0.01 | 0.1 | 0.0 | 7 |
| POKEC-REGIONS | RL | 0.0005 | 0.0 | 0.0005 | 0.0005 | 0.0 | 0.0005 | 2 |
| TOLOKERS-2 | RH | 0.005 | 0.0 | 0.001 | 0.0005 | 0.7 | 0.0 | 3 |
| TOLOKERS-2 | RL | 0.0001 | 0.3 | 0.001 | 0.0005 | 0.7 | 0.001 | 3 |
| WEB-FRAUD | RH | 0.0005 | 0.1 | 0.0 | 0.0001 | 0.1 | 0.0 | 2 |
| WEB-FRAUD | RL | 0.0005 | 0.1 | 0.0 | 0.0005 | 0.2 | 0.0 | 2 |
| WEB-TOPICS | RH | 0.0005 | 0.1 | 0.0 | 0.0005 | 0.1 | 0.0 | 2 |
| WEB-TOPICS | RL | 0.0005 | 0.1 | 0.0 | 0.0005 | 0.2 | 0.0005 | 2 |

Table 9: Architectural hyperparameters for GCN, GraphSAGE, and GAT.

| Dataset | Model | Hidden | Layers | Output proj. | Residual | Norm |
|---|---|---|---|---|---|---|
| OGBN-ARXIV | GCN | 512 | 5 | yes | yes | BN |
| | GraphSAGE | 256 | 4 | yes | yes | BN |
| | GAT | 256 | 5 | yes | yes | BN |
| OGBN-PRODUCTS | GCN | 256 | 5 | no | yes | LN |
| | GraphSAGE | 256 | 5 | no | no | LN |
| | GAT | 256 | 5 | no | no | LN |
| POKEC | GCN | 256 | 7 | yes | yes | BN |
| | GraphSAGE | 256 | 7 | yes | yes | BN |
| | GAT | 256 | 7 | yes | yes | BN |

### F.4 Architecture Benchmark Parameters

For GraphSAGE, GCN, and GAT, we use the same architectural hyperparameters as Luo et al. (2024). However, Luo et al. (2024) does not report the SGFormer hyperparameters, so we instead adopt those from the original SGFormer paper Wu et al. (2023). Because OGBN-PRODUCTS is not included in Wu et al. (2023), we use for this dataset the hyperparameter configuration reported for Amazon2M, which is the closest large-scale benchmark considered in that work. This choice may partly explain discrepancies between our SGFormer results and those reported by Luo et al. (2024). Tables 9 and 10 summarize the resulting architectural hyperparameters.

As in the sampling benchmark experiments, we tune training-related hyperparameters separately for each dataset-architecture pair using 50 Bayesian optimization runs with Weights & Biases Sweeps. For RNS, we fix the number of parts to the best-performing configuration identified in the sampling benchmark for each dataset, namely 2 for OGBN-ARXIV, 10 for OGBN-PRODUCTS, and 2 for POKEC. The selected training hyperparameters are reported in Table 11.

All experiments are repeated over five random seeds: 42, 123, 456, 789, and 101112.

Unless otherwise specified, all models use `use_input_projection = false`. For GAT, the number of attention heads is 1 in all cases. For SGFormer, we use $\alpha = 0.5$, `use_graph = true`, `use_weight = true`, and `graph_weight = 0.5` for all datasets.

On OGBN-PRODUCTS and POKEC, GAT runs result in out-of-memory errors in our implementation. Although Luo et al. (2024) report GAT results on these datasets, their public repository does not

Table 10: Architectural hyperparameters for SGFormer.

| Dataset | Hidden | Layers | Heads | Residual | Norm | GNN branch |
|---|---|---|---|---|---|---|
| OGBN-ARXIV | 256 | 1 | 1 | yes | BN | GCN, hidden 256, layers 3 |
| OGBN-PRODUCTS | 256 | 1 | 2 | yes | BN | GCN, hidden 256, layers 3 |
| POKEC | 64 | 1 | 1 | yes | BN | GCN, hidden 64, layers 2 |

Table 11: Tuned training hyperparameters for each dataset, architecture, and training setting. Each entry is reported as (dropout, learning rate, weight decay). N/A indicates settings for which runs did not yield valid results.

| Dataset | Architecture | Full | RNS |
|---|---|---|---|
| OGBN-ARXIV | GCN | $(0.5, 5 \times 10^{-4}, 0)$ | $(0.5, 5 \times 10^{-4}, 0)$ |
| | GraphSAGE | $(0.5, 5 \times 10^{-4}, 5 \times 10^{-4})$ | $(0.5, 5 \times 10^{-4}, 0)$ |
| | GAT | $(0.3, 5 \times 10^{-4}, 0)$ | $(0.5, 5 \times 10^{-4}, 5 \times 10^{-4})$ |
| | SGFormer | $(0.5, 5 \times 10^{-4}, 10^{-4})$ | $(0.7, 5 \times 10^{-4}, 0)$ |
| OGBN-PRODUCTS | GCN | $(0.5, 3 \times 10^{-3}, 10^{-3})$ | $(0.3, 10^{-3}, 0)$ |
| | GraphSAGE | $(0.3, 5 \times 10^{-4}, 5 \times 10^{-4})$ | $(0.3, 5 \times 10^{-4}, 10^{-4})$ |
| | GAT | N/A | N/A |
| | SGFormer | $(0.0, 10^{-2}, 0)$ | $(0.7, 5 \times 10^{-3}, 0)$ |
| POKEC | GCN | $(0.2, 5 \times 10^{-4}, 10^{-5})$ | $(0.5, 10^{-3}, 0)$ |
| | GraphSAGE | $(0.7, 10^{-3}, 5 \times 10^{-4})$ | $(0.3, 10^{-3}, 10^{-4})$ |
| | GAT | N/A | N/A |
| | SGFormer | $(0.0, 10^{-2}, 10^{-4})$ | $(0.7, 10^{-3}, 0)$ |

appear to contain run artifacts or scripts that clarify how these results were obtained. One possible explanation is that their experiments relied on RNS for example during evaluation, whereas our benchmark uses a unified end-to-end implementation and evaluation protocol across all methods. We therefore omit these GAT results rather than report numbers obtained under a different or unclear setting.

For SGFormer, the difference between our results and those of Luo et al. (2024) may similarly stem from differences in the experimental setup. In particular, Luo et al. (2024) do not report the SGFormer architectural hyperparameters, and their public repository does not appear to include the code or run artifacts needed to reproduce their SGFormer results on OGBN-PRODUCTS. Consequently, we report SGFormer results using the architecture specified above and the same unified training protocol used for the other models.

# G Additional Results

## G.1 RNS sampled graphs properties

On Figure 4, we report connectivity metrics on sampled graphs as a function of the number of batches per epoch $m$.

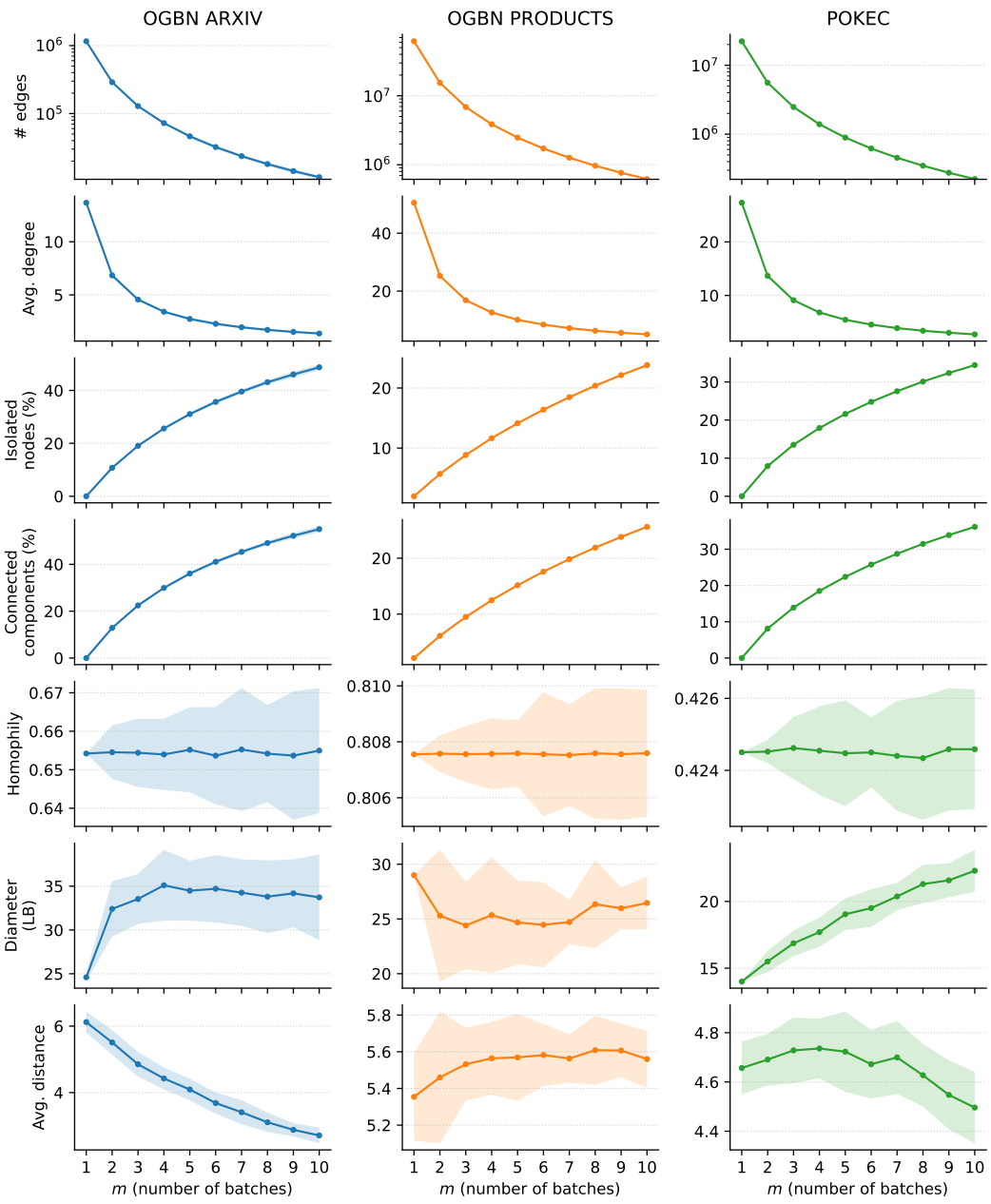



Figure 4: **Properties of sampled graphs as a function of the number of batches $m$.** Evolution of subsampled-graph metrics with respect to $m$ (number of batches per epoch). Shaded regions indicate $\pm$ one standard deviation over 5 epochs (full pass over all batches). The diameter lower bound corresponds to an estimate of the maximum distance between two nodes within the same connected component, computed using multi-sweep BFS. The average distance is an estimate of the expected shortest-path distance between two nodes within the same connected component, also computed using multi-sweep BFS.

### G.2 Speedup decomposition

For each dataset, method, seed, and target validation accuracy $a^\star$, we compute all quantities from validation logs. The per-epoch runtime is estimated by the median slope between consecutive logs,

$$\tau = \text{median}_j \frac{\text{runtime}_{j+1} - \text{runtime}_j}{\text{epoch}_{j+1} - \text{epoch}_j}.$$

Table 12: **RNS speedup decomposition.** Speedups over full-graph training for validation time-to-accuracy targets on a single NVIDIA H100 GPU. Both methods reach every target in all five seeds.

| Dataset | Target val. accuracy | Epoch time speedup | Epochs to target speedup | Time to target w/ startup | Time to target w/o startup |
|---|---|---|---|---|---|
| OGBN-ARXIV | 68.0% | 1.07× | 2.02× | 1.67× | 2.13× |
| | 69.0% | 1.07× | 1.97× | 1.68× | 2.09× |
| | 70.0% | 1.07× | 1.92× | 1.69× | 2.06× |
| OGBN-PRODUCTS | 88.0% | 1.77× | 6.88× | 4.95× | 12.00× |
| | 89.0% | 1.77× | 6.87× | 5.59× | 11.99× |
| | 90.0% | 1.77× | 6.72× | 6.32× | 11.74× |
| POKEC | 78.0% | 1.45× | 1.75× | 2.05× | 2.34× |
| | 79.0% | 1.45× | 1.51× | 1.88× | 2.03× |
| | 80.0% | 1.45× | 1.30× | 1.71× | 1.78× |

Table 13: Time to the target validation accuracy (lower is better). For each method and dataset, we report the shortest time obtained across the evaluated configurations and the corresponding sampling parameters. Bold indicates the shortest time in each dataset.

| | OGBN-ARXIV (72%) | | OGBN-PRODUCTS (92%) | | POKEC (76%) | |
|---|---|---|---|---|---|---|
| Method | Time | Params | Time | Params | Time | Params |
| Full | 22 s | – | 8 min 00 s | – | 1 min 18 s | – |
| Neighbor | 5 min 48 s | $\mathbf{f} = [30, 10, 10]$ | 17 min 02 s | $\mathbf{f} = [30, 10]$ | 12 min 53 s | $\mathbf{f} = [30, 10]$ |
| ClusterGCN | 2 min 35 s | $C = 1500,\ B = 20$ | 9 min 58 s | $C = 3000,\ B = 100$ | 1 min 18 s | $C = 200,\ B = 1$ |
| GraphSAINT | 44 s | $L_{\mathrm{w}} = 2,\ S = 6000$ | 17 min 33 s | $L_{\mathrm{w}} = 2,\ S = 6000$ | 1 min 03 s | $L_{\mathrm{w}} = 2,\ S = 6000$ |
| RNS | **19 s** | $m = 3$ | **1 min 02 s** | $m = 10$ | **18 s** | $m = 8$ |

Epochs-to-target $E(a^\star)$ are obtained by linear interpolation between the last validation point below $a^\star$ and the first point above it. Applying the same interpolation to runtime gives $T_{\mathrm{start}}(a^\star)$, and $T_{\mathrm{nostart}}(a^\star) = T_{\mathrm{start}}(a^\star) - t_0$ removes startup overhead, where $t_0$ is the first logged runtime.

We average run-level quantities over seeds and report speedups relative to full-graph training:

$$S_{\mathrm{epoch}} = \frac{\mathbb{E}_s[\tau_{\mathrm{full},s}]}{\mathbb{E}_s[\tau_{\mathrm{RNS},s}]}, \quad S_{\mathrm{epochs}} = \frac{\mathbb{E}_s[E_{\mathrm{full},s}(a^\star)]}{\mathbb{E}_s[E_{\mathrm{RNS},s}(a^\star)]}, \quad S_{\mathrm{time},\bullet} = \frac{\mathbb{E}_s[T_{\bullet,\mathrm{full},s}(a^\star)]}{\mathbb{E}_s[T_{\bullet,\mathrm{RNS},s}(a^\star)]},$$

for $\bullet \in \{\mathrm{start}, \mathrm{nostart}\}$. Wall-clock speedups are measured directly from runtime logs and therefore need not equal $S_{\mathrm{epoch}} S_{\mathrm{epochs}}$, due to validation, logging, data-loading, optimizer, and initialization overheads.

### G.3 More results on time and memory savings

We complement the main results with a comparison of time to target validation accuracy and whole-run peak GPU memory for all sampling methods.

**Comparison protocol.** We use dataset-specific validation-accuracy targets of 72% on OGBN-ARXIV, 92% on OGBN-PRODUCTS, and 76% on POKEC. Each target is attainable by all five methods in the reported experiments, enabling a comparison at a common accuracy level within each dataset. For time to target, we report the fastest evaluated hyperparameter setting for each method and dataset. The timing and memory tables report their respective configurations, which need not coincide: the reported time and memory values should therefore not be interpreted as jointly achieved by a single configuration. These comparisons characterize resource requirements at the chosen accuracy targets, rather than at each method's best attainable accuracy.

**Time to target.** Table 13 shows that RNS provides only marginal time-to-target gains over full-graph training on OGBN-ARXIV, but substantially larger gains on POKEC and OGBN-PRODUCTS. These two datasets contain roughly an order of magnitude more nodes than OGBN-ARXIV, suggesting that RNS may offer greater benefits on even larger graphs, although establishing this scaling

Table 14: Whole-run peak GPU memory in GB (lower is better) and associated sampling parameters. These configurations can differ from those in Table 13. Bold indicates the lowest reported peak in each dataset. Evaluation is full-graph for RNS, ClusterGCN, and GraphSAINT, but mini-batched for Neighbor; consequently, these measurements do not isolate training-memory usage.

| | OGBN-ARXIV (72%) | | OGBN-PRODUCTS (92%) | | POKEC (76%) | |
|---|---|---|---|---|---|---|
| Method | Peak (GB) | Params | Peak (GB) | Params | Peak (GB) | Params |
| Full | 4.01 | – | 58.48 | – | 52.48 | – |
| Neighbor | 1.86 | $\mathbf{f} = [30, 10, 10]$ | **2.14** | $\mathbf{f} = [30, 10]$ | **2.15** | $\mathbf{f} = [30, 10]$ |
| ClusterGCN | **1.69** | $C = 1500,\ B = 20$ | 17.77 | $C = 200,\ B = 1$ | 10.17 | $C = 15000,\ B = 10$ |
| GraphSAINT | 1.73 | $L_{\mathrm{w}} = 2,\ S = 2000$ | 17.74 | $L_{\mathrm{w}} = 4,\ S = 2000$ | 10.17 | $L_{\mathrm{w}} = 2,\ S = 2000$ |
| RNS | 1.75 | $m = 7$ | 17.95 | $m = 9$ | 10.17 | $m = 10$ |

trend would require controlled experiments. The other sampling methods generally match or exceed the full-graph time to target. Their more elaborate, structure-aware sampling procedures and mini-batch construction can introduce overhead that offsets the computational savings from sampling. This provides a plausible explanation for their limited speed gains, although time to target also depends on convergence and does not isolate sampling overhead.

**Peak GPU memory.** Table 14 provides a direct comparison of whole-run peak GPU memory across the evaluated training-and-evaluation pipelines. For RNS, ClusterGCN, and GraphSAINT, peak memory is dominated by full-graph evaluation, explaining their closely matched memory consumption despite different training procedures. Full-graph training additionally incurs the memory cost of retaining activations for backpropagation over the entire graph, resulting in substantially larger peaks. Neighbourhood sampling, in contrast, uses sampled mini-batches during validation and testing as well as training, avoiding the full-graph evaluation bottleneck. This gives it a clear memory advantage on OGBN-PRODUCTS and POKEC, where its peaks are substantially lower than those of the other methods. This advantage reflects its end-to-end mini-batched implementation, rather than demonstrating that its training sampler alone is more memory-efficient.

### G.4 Power-law preservation and batch gradient variance $R(w)$

We run the following experiment on OGBN-ARXIV, OGBN-PRODUCTS, and POKEC at random initialization, over 100 seeds and 50 batches per seed. For each seed and sampler, we estimate the power-law exponent $\alpha_{\mathrm{Sampl}}$ by averaging Clauset MLE estimates Clauset & Shalizi (2009) across the 50 sampled subgraphs, and compute the gradient variance $R(w)$ from the same batches. Each sampler with its parameters yields one point in the $\big(|\alpha_{\mathrm{Sampl}} - \alpha_{\mathrm{Full}}|,\ R(w)\big)$ plane. We used the architecture parameters found in the sampling benchmark for the estimations (See Appendix F.1).

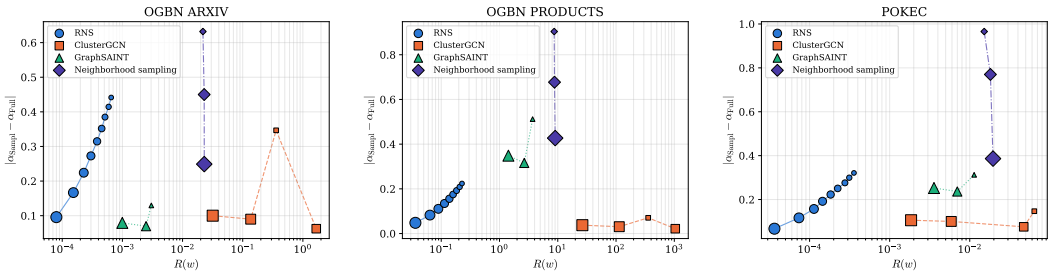



Figure 5: **$\alpha$-preservation error and $R$ across samplers and sampling budgets.** Each point corresponds to one sampler at a given budget; marker size is proportional to the budget (sizes are comparable within a sampler but *not* across samplers). The same trend holds on all three graphs, RNS delivers a much lower $R$ while preserving the power law degree distribution for larger datasets.

Two observations stand out. First, on the variance axis the ordering is unambiguous: RNS yields values of $R(w)$ two to four orders of magnitude smaller than every competitor on all three graphs, and

this gap persists even at the largest sampling budgets we consider ($m = 10$), i.e. it is not an artifact of RNS operating in a low-budget regime. Since $\alpha$ summarizes the heavy tail of the degree distribution, we read $|\alpha_{\text{Sampl}} - \alpha_{\text{Full}}|$ as a proxy for how faithfully a sampler preserves the *global* structure of the graph. Under this lens, RNS distorts the power law substantially on the smallest graph (OGBN-ARXIV), where the degree tail is short and random node thinning removes a non-negligible fraction of the hubs that carry it. The distortion shrinks on the larger graphs (OGBN-PRODUCTS, POKEC), where the tail is long enough for the sampled subgraphs to recover the exponent almost exactly. The two diagnostics are therefore not interchangeable, as the other samplers make clear. ClusterGCN sits at the opposite corner: its partitions are structurally coherent and preserve $\alpha$ well, yet it attains the largest $R(w)$ of all methods (up to $10^3$ on OGBN-PRODUCTS), because individual partitions are mutually dissimilar and hence produce strongly heterogeneous gradients across batches. GraphSAINT is an in-between for the two regimes, trading some structure preservation for a reduced $R(w)$, but the trade-off it achieves is dominated by RNS. In short, degree preservation diagnoses *intra*-batch representativeness, whereas $R(w)$ additionally captures *inter*-batch heterogeneity. Overall, RNS is the only sampler that jointly preserves the global structure of the larger graphs and attains a markedly lower $R(w)$, linking the graph-structural guarantee of Appendix D to the optimization term $R(w)$.

### G.5 Validation metrics through training

Figure 6 reports validation accuracy over 200 training epochs for full-graph training and RNS. The curves show that the clear speed advantage of RNS in runtime.

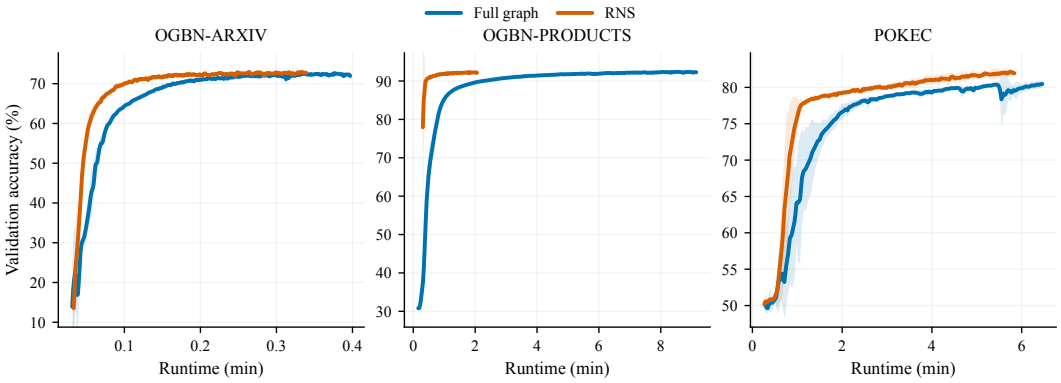



Figure 6: Validation accuracy over 200 epochs for full-graph training and RNS.

### G.6 Neighborhood sampling depth

We further analyze the effect of the neighbor sampling depth in Table 15. Although neighbor sampling is not the main focus of our work, this ablation helps explain the fanout schedules used in our benchmark and why we do not consider configurations deeper than four layers (see Appendix F.1 for experimental details). Increasing the depth can improve accuracy on larger graphs, especially POKEC, but this comes at a substantial computational cost. In our implementation, PyG's neighbor sampler runs on the CPU, making deeper schedules prohibitively slow for large-scale experiments.

### G.7 Empirical results on RNS bias and variance

We examine the sensitivity, bias, and variance of batch gradients of RNS, varying the number of partition blocks over $m \in \{2, \ldots, 10\}$. All models are randomly initialized and remain untrained, with parameters fixed during resampling. Each figure contains one panel per dataset; curves show averages across initializations, and shaded bands indicate 95% confidence intervals for these averages. The experimental losses use per-batch training-count normalization, so comparisons with the deterministically normalized theoretical bounds are qualitative.

Table 15: **Neighbor sampling fanout ablation.** Test accuracy, time per epoch, and peak memory usage for different fanout vectors. Results are reported as mean ± 95% CI over 5 seeds. Increasing the sampling depth generally improves accuracy on the larger graphs, but substantially increases sampling cost and memory usage.

| Dataset | Fanout **f** | Accuracy (%) | Time / Epoch (s) | GPU (GB) | RAM (GB) |
|---|---|---|---|---|---|
| OGBN-ARXIV | [30, 10] | 71.02 ± 0.27 | 9.53 | 0.84 | 1.67 |
| | [30, 10, 10] | **71.08 ± 0.47** | 20.43 | 1.61 | 1.67 |
| | [30, 10, 10, 10] | 70.95 ± 0.22 | 36.38 | 2.56 | 1.70 |
| OGBN-PRODUCTS | [30, 10] | 80.21 ± 0.35 | 58.72 | 1.85 | 10.07 |
| | [30, 10, 10] | 80.44 ± 0.34 | 196.78 | 8.44 | 10.38 |
| | [30, 10, 10, 10] | **80.56 ± 0.28** | 638.15 | 24.59 | 10.51 |
| POKEC | [30, 10] | 77.75 ± 0.09 | 40.56 | 1.89 | 4.67 |
| | [30, 10, 10] | 80.07 ± 0.06 | 200.94 | 9.26 | 4.70 |
| | [30, 10, 10, 10] | **80.70 ± 0.13** | 729.76 | 29.92 | 4.85 |

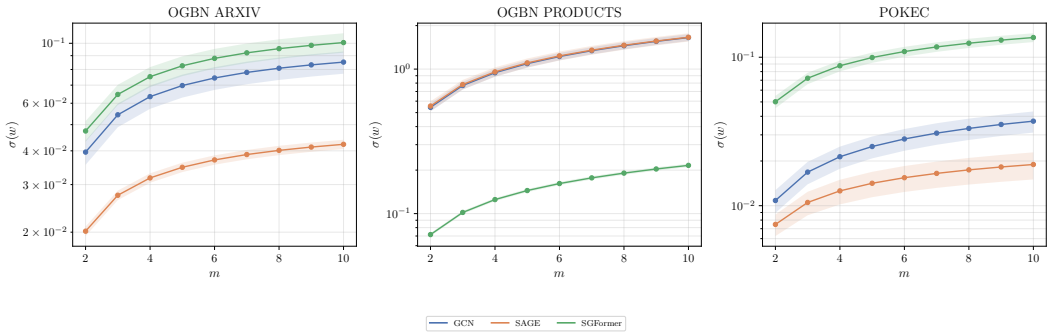



Figure 7: RNS sensitivity versus the number of partition blocks $m$ at random initialization. Each curve corresponds to an architecture. Results average five independent initializations, using 25 complete RNS partitions per initialization. Shaded bands indicate 95% confidence intervals across initializations.

**Sensitivity.** Figure 7 shows a saturating trend across datasets and architectures, including SGFormer. The elevated sensitivity observed for SGFormer on OGBN-ARXIV and POKEC suggests that sensitivity to subsampling may contribute to its weaker RNS performance on these datasets.

**Bias.** Figure 8 reports estimates of the objective bias. On OGBN-ARXIV and POKEC, the curves gradually saturate as $m$ increases though slower than sensitivity. This behavior is qualitatively consistent with the factor $1 - 1/m$ in the theoretical upper bound. The analysis provides an upper envelope, however, and does not assert that the bias itself is proportional to $1 - 1/m$.

**Batch gradient variance.** Figure 9 reports the empirically measured $R$, computed over the same 50 mini-batches used for the bias estimates. Its dependence on $m$ is approximately linear across all three datasets over the tested range. This empirical trend is consistent with the linear-in-$m-1$ upper bound for expected gradient dispersion.

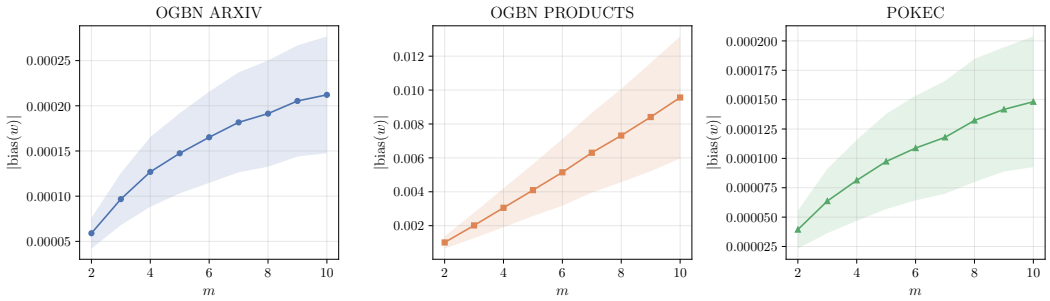



Figure 8: RNS bias versus the number of partition blocks

$m$

. Mean absolute gap to the full-graph objective over 20 initializations (50 mini-batches each). Bands are 95% CIs.

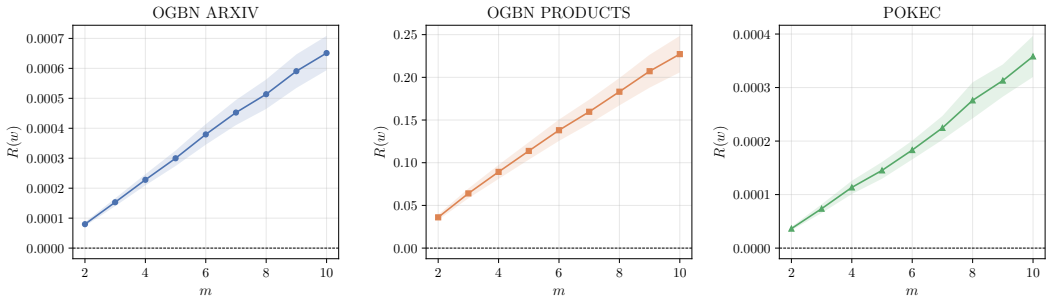



Figure 9: $R$ versus the number of partition blocks $m$ at random initialization, averaged over 20 initializations (50 RNS mini-batch gradients each); bands are 95% CIs.