SCAFFOLD: SUPPORT GRAPH THEORY BASED SPARSIFICATION FOR GRAPH NEURAL NETWORKS

Anonymous authors
Paper under double-blind review

ABSTRACT

Graph neural networks (GNNs) rely on message passing over graph edges, making their computational and memory costs strongly dependent on graph density. Graph sparsification offers a natural way to reduce these costs, but removing edges indiscriminately can distort important communication structure and degrade predictive performance. We introduce `Scaffold`, a topology-based, unsupervised graph sparsification framework derived from support graph theory preconditioners. `Scaffold` explicitly controls two complementary structural quantities: *dilation*, which measures the length of rerouting paths induced by removed edges, and *congestion*, which measures how strongly these rerouted paths concentrate on the retained support. By jointly controlling dilation and congestion, `Scaffold` preserves short communication paths while avoiding structural bottlenecks. To our knowledge, `Scaffold` is the first scalable GNN sparsification framework to use a joint supporting-path dilation-congestion criterion. Across 19 homophilic and heterophilic benchmarks spanning small to large graphs, `Scaffold` achieves the best aggregate rank among the evaluated sparsification and related methods. Using only 10%–50% of the original edges per sparse support, `Scaffold` recovers or closely approaches full-graph GNN performance while using less than half the memory of full-graph training and reducing end-to-end training time, including sparsification overhead. We provide an open-source software package at `anonymous.4open.science/r/Scaffold-B6DB`.

1 INTRODUCTION

Graph Neural Networks (GNNs) (Zhou et al., 2020) have been remarkably successful across a wide range of graph-learning applications. GNNs operate through two iterative steps, *aggregation* and *update*, where each node first aggregates information from its neighborhood and then updates its embedding using the aggregated information (Hamilton et al., 2017). Graphs used for GNN training are typically stored in sparse formats such as coordinate (COO), compressed sparse row (CSR), or compressed sparse column (CSC). COO stores edges explicitly as source-destination index pairs and is commonly used for edge-wise gather-scatter operations (Fey & Lenssen, 2019), where source-node features are gathered along edges and aggregated at destination nodes. CSR/CSC instead compress row or column indices and are commonly used for efficient sparse-matrix–dense-matrix multiplication (`SpMM`) (Wang et al., 2019). These sparse aggregation operations often dominate GNN execution, with `SpMM` alone accounting for approximately $\mathbf{70}\%$ of the total computation cost (Liu et al., 2023). Since their cost scales with the number of processed edges, GNN training can become expensive on larger and denser graphs (Chen et al., 2018; Zeng et al., 2019).

A common approach for scaling GNN training is *sampling* (Hamilton et al., 2017; Chiang et al., 2019), which reduces per-iteration computation by typically controlling the size of the sampled computation graph rather than explicitly preserving its global structure. Graph *sparsification* offers a complementary structure-aware sampling perspective by selecting a subset of edges while preserving properties of the original graph so that the resulting subgraph remains an effective support for message passing (Spielman & Teng, 2011; Loukas, 2019). However, not all sparsification objectives align with the communication requirements of GNNs. Equally sized sparse graphs can induce very different communication structures, forcing originally adjacent nodes to communicate through long detours or rerouting many communication paths on a small number of edges or hub nodes. This is particularly important in the *unsupervised* setting, where labels and downstream task informa-

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.

tion are unavailable for identifying task-specific edge importance. The overarching goal is therefore to construct sparse graphs that preserve structural properties and support effective communication while satisfying a fixed edge budget.

Among structural properties, *low dilation* and *low congestion* are particularly relevant to sparsification. In support graph theory, dilation measures how far communication is rerouted when an edge is removed, while congestion measures the crowdedness or density of rerouted paths on the retained support. These concepts originate from combinatorial preconditioners to solve systems of linear equations, and were later formalized through support theory and congestion-dilation analysis (Vaidya, 1991; Boman & Hendrickson, 2003; Bern et al., 2006; Maggs et al., 2005). Together, they characterize how well a sparse graph supports the original graph. We bring this perspective to graph representation learning through `Scaffold`, a practical and scalable sparsification framework that explicitly controls dilation and congestion under a prescribed edge budget. If removing an edge forces its endpoints to communicate through a long or highly congested supporting path, that edge is poorly represented by the current sparse graph and receives higher priority for retention. `Scaffold` therefore constructs sparse supports in which removed connections remain represented by short paths while avoiding heavily congested communication bottlenecks.

We develop five realizations of this principle for different scalability regimes: `greedy` fully recomputes support scores, `heap` uses lazy updates, `batch` scores sampled candidate batches, `fast` uses static tree-prefix scoring for large graphs, and `sample` converts structural scores into a distribution over sparse graph views. Together, these variants support both fixed sparsification and stochastic GNN sampling. These sparse supports also provide practical efficiency benefits. As illustrated in Figure 1 on REDDIT (232.97K nodes, 57.31M undirected edges), `Scaffold` approaches full-graph performance at low edge-retention ratios while substantially reducing end-to-end training time and peak memory for both gather–scatter and `SpMM`-based implementations. Our contributions are as follows:

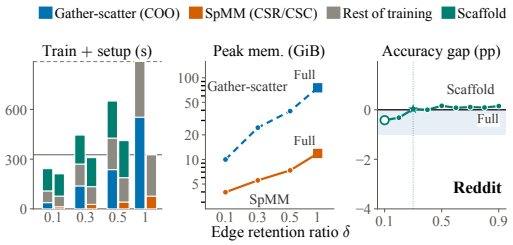

Figure 1: Runtime, memory, and accuracy trade-off.

(i) We introduce `Scaffold`, a topology-based unsupervised and theoretically rigorous sparsification framework that translates support-graph dilation and congestion from combinatorial preconditioners to graph representation learning (§2).

(ii) We develop a scalable family of `Scaffold` algorithms with different quality-cost trade-offs, supporting both fixed sparse graphs and stochastic sparse views for efficient GNN training and inference (§3).

(iii) We characterize how dilation and edge congestion bound the structural and communication distortion introduced by sparsification, and empirically study their implications for the GNN message passing framework (§3.4).

(iv) Across 19 datasets and 20 related baselines, `Scaffold` achieves strong predictive performance while reducing memory and end-to-end training cost. With each sparse support retaining only 30% and 50% of the original edges, `Scaffold` reaches within one percentage point of full-graph performance on $12/19$ and $16/19$ datasets, respectively (§5).

2 PRELIMINARIES

Sparse support and supporting paths. Let $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \boldsymbol{w})$ be an undirected graph with $n = |\mathcal{V}|$ nodes and $m = |\mathcal{E}|$ edges. For an edge-retention ratio $\delta \in (0, 1]$, let $q = \lceil \delta m \rceil$ denote the target edge budget. We seek a spanning subgraph $\mathcal{H} = (\mathcal{V}, \mathcal{E}_{\mathcal{H}}, \boldsymbol{w}_{\mathcal{H}}) \subseteq \mathcal{G}$ with $|\mathcal{E}_{\mathcal{H}}| = q$, and denote the omitted edges by $\mathcal{I} = \mathcal{E} \setminus \mathcal{E}_{\mathcal{H}}$. Following the support-graph view (Vaidya, 1991; Boman & Hendrickson, 2003; Bern et al., 2006), each omitted edge $e = (u, v) \in \mathcal{I}$ is represented by a supporting path P_e in $\mathcal{H}$ connecting its endpoints. In `Scaffold`, P_e is chosen as a shortest path in the current support. Figure 2 illustrates this view and the structural quantities used below.

Dilation and congestion. Classical support theory characterizes support quality using dilation and edge congestion (Vaidya, 1991; Boman & Hendrickson, 2003; Bern et al., 2006); we additionally consider node congestion for GNN message passing. For the unweighted graphs studied in this

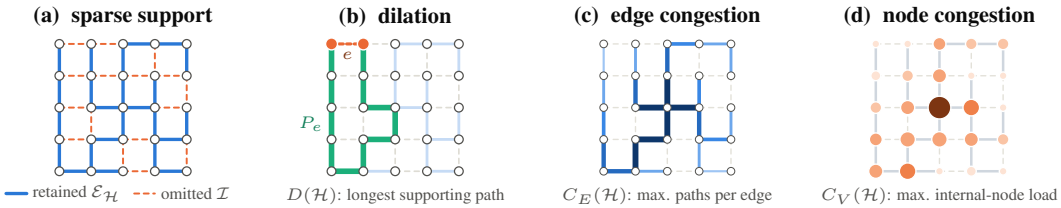

Figure 2: Support-graph features: (a) sparse support, (b) dilation, (c) edge congestion, and (d) node congestion. Darker edges and larger nodes indicate higher path loads; node loads exclude endpoints.

work, we define

$$D(\mathcal{H}) = \max_{e \in \mathcal{I}} |P_e|, \qquad C_E(\mathcal{H}) = \max_{a \in \mathcal{E}_{\mathcal{H}}} |\{e \in \mathcal{I} : a \in P_e\}|, \text{ and}$$

$$C_V(\mathcal{H}) = \max_{x \in \mathcal{V}} |\{e \in \mathcal{I} : x \in \mathrm{int}(P_e)\}|.$$

Thus, dilation measures how far an omitted connection is rerouted, while edge and node congestion measure how strongly rerouted paths concentrate on the retained support. Weighted extensions, further support-theoretic background, and controlled examples illustrating these quantities are provided in Appendix A.

Sparsification objective. Given an edge budget q, we seek a spanning support, $\mathcal{H}^\star$, in which omitted edges remain represented by short and lightly congested paths:

$$\Phi(\mathcal{H}) = \underbrace{(1 + D(\mathcal{H}))^{\alpha}}_{\text{dilation}} \underbrace{(1 + C_E(\mathcal{H}))^{\beta_E}}_{\text{edge congestion}} \underbrace{(1 + C_V(\mathcal{H}))^{\beta_V}}_{\text{node congestion}},$$
$$\mathcal{H}^\star \in \arg \min_{\substack{\mathcal{H} \subseteq \mathcal{G},\ |\mathcal{E}_{\mathcal{H}}| = q \\ \mathcal{H} \text{ spans each component}}} \Phi(\mathcal{H}). \tag{1}$$

Here, $\alpha, \beta_E, \beta_V \geq 0$ control the relative contribution of the three criteria. Classical support theory uses dilation and edge congestion, while C_V is an additional GNN-oriented extension; setting $\beta_V = 0$ recovers the core dilation–edge-congestion objective.

3 Proposed Method for Sparsification: Scaffold

Equation 1 defines the desired sparse support globally, but direct optimization is NP-hard, even in the dilation-only case (Appendix B.1). `Scaffold` therefore replaces the global objective with a candidate-wise structural surrogate. We first define this score, then present scalable realizations, describe their use within GNNs, and summarize their structural properties and computational cost.

3.1 From the Global Objective to a Greedy Surrogate

Let $\mathcal{H}_t$ be the current support and $\mathcal{I}_t = \mathcal{E} \setminus \mathcal{E}_{\mathcal{H}_t}$ the remaining candidates. An exact greedy step would select

$$e_t^{\text{exact}} = \arg\max_{e \in \mathcal{I}_t} \Delta_t(e), \qquad \Delta_t(e) = \Phi(\mathcal{H}_t) - \Phi(\mathcal{H}_t \cup \{e\}). \tag{2}$$

However, evaluating every marginal gain requires repeatedly recomputing supporting paths and congestion after hypothetical insertions, making exact greedy selection computationally expensive and impractical at scale.

Therefore, `Scaffold` instead scores each candidate from its current supporting path. Starting from a spanning-forest backbone $\mathcal{H}_0 = \mathcal{F}$, with one spanning tree per connected component, for $e = (u, v) \in \mathcal{I}_t$ let $P_e = \mathrm{ShortestPath}_{\mathcal{H}_t}(u, v)$. For the unweighted graphs studied in this work, its path dilation is

$$D_t^{\text{path}}(e) = |P_e|. \tag{3}$$

For a path P_f supporting $f = (u, v)$, let $\mathrm{int}(P_f) = V(P_f) \setminus \{u, v\}$ denote its internal vertices. For the current candidate paths, the edge- and node-congestion counts are

$$c_{E,t}(a) = |\{f \in \mathcal{I}_t : a \in P_f\}|, \qquad c_{V,t}(x) = |\{f \in \mathcal{I}_t : x \in \mathrm{int}(P_f)\}|.$$

We aggregate them along P_e using power means ($p_E, p_V \geq 1$):

$$C^{\text{path}}_{E,t}(e) = \left(\frac{\sum_{a \in P_e} c_{E,t}(a)^{p_E}}{|P_e|} \right)^{1/p_E}; \quad (4) \qquad C^{\text{path}}_{V,t}(e) = \left(\frac{\sum_{x \in \text{int}(P_e)} c_{V,t}(x)^{p_V}}{|\text{int}(P_e)|} \right)^{1/p_V}. \quad (5)$$

We set $C^{\text{path}}_{V,t}(e) = 1$, when $\text{int}(P_e) = \emptyset$; larger p_E or p_V values emphasize stronger bottlenecks.

Let $D^{\text{path}}_{t,\max}$, $C^{\text{path}}_{E,t,\max}$, and $C^{\text{path}}_{V,t,\max}$ denote the corresponding maxima over $\mathcal{I}_t$. The normalized `Scaffold` score is

$$s_t(e) = \underbrace{\left(\frac{D^{\text{path}}_t(e)}{D^{\text{path}}_{t,\max}} \right)^{\alpha}}_{\text{dilation}} \underbrace{\left(\frac{C^{\text{path}}_{E,t}(e)}{C^{\text{path}}_{E,t,\max}} \right)^{\beta_E}}_{\text{edge congestion}} \underbrace{\left(\frac{C^{\text{path}}_{V,t}(e)}{C^{\text{path}}_{V,t,\max}} \right)^{\beta_V}}_{\text{node congestion}}. \quad (6)$$

A zero exponent disables the corresponding component. For example, $(1, 1, 0)$ gives the dilation–edge-congestion criterion, while $\beta_V > 0$ additionally includes node congestion. The resulting rule replaces the exact global marginal gain with the local support score:

$$\underbrace{e^{\text{exact}}_t = \arg\max_{e \in \mathcal{I}_t} \Delta_t(e)}_{\text{exact marginal}} \quad \Longrightarrow \quad \underbrace{e^{\star}_t = \arg\max_{e \in \mathcal{I}_t} s_t(e)}_{\texttt{Scaffold}\text{ surrogate}}, \quad (7)$$

here $s_t(e)$ measures the current structural deficit of candidate e, whereas $\Delta_t(e)$ also accounts for how inserting e reroutes other candidates. `Scaffold` partially captures the cross-candidate effects by refreshing paths and scores as $\mathcal{H}_t$ grows. For our surrogate, obtaining a general approximation ratio with respect to the global objective is not straightforward, as discussed in Appendix B.2.

3.2 Family of Scaffold Algorithms

All `Scaffold` variants share three components: a support backbone $\mathcal{F}$, the support score in Eq. 6, and a candidate-selection policy. For feasible budgets, the spanning-forest backbone ensures that every omitted edge has a supporting path, while the selection policy determines how aggressively paths and scores are recomputed. In general, $\mathcal{F}$ may be deterministic, randomized, weight-aware, or low-stretch (Abraham et al., 2008; Abraham & Neiman, 2012). For weighted graphs, `Scaffold` uses the supplied edge weights through the path metric. A weighted formulation of `Scaffold` using weighted path lengths and routing loads is given in Appendix A.3. Since all benchmark graphs in this work are unweighted, we use a deterministic spanning forest (`SF`) together with randomized forests and fast low-stretch approximations. These algorithms are novel support-theoretic sparsifiers.

Greedy reference. `Scaffold-Greedy` is the full-recomputation realization of Eq. 7. Starting from $\mathcal{H}_0 = \mathcal{F}$, it recomputes supporting paths, congestion statistics, and candidate scores after every insertion, retains the highest-scoring edge, and repeats until $|\mathcal{E}_{\mathcal{H}}| = q$. Algorithm 1 gives the reference construction, while Figure 3 illustrates one run.

(a) Greedy reference

Algorithm 1 `Scaffold-Greedy`$(\mathcal{G}, \delta)$

Require: Graph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \boldsymbol{w})$, retention ratio δ
Ensure: Sparse support $\mathcal{H}$
1: $q \leftarrow \lceil \delta m \rceil$; choose spanning forest $\mathcal{F}$
2: $\mathcal{H}_0 \leftarrow \mathcal{F}$; $\mathcal{I}_0 \leftarrow \mathcal{E} \setminus \mathcal{E}_{\mathcal{F}}$; $t \leftarrow 0$
3: **while** $|\mathcal{E}_{\mathcal{H}_t}| < q$ **do**
4: Compute P_e, $D^{\text{path}}_t(e)$, $C^{\text{path}}_{E,t}(e)$, $C^{\text{path}}_{V,t}(e)$ for $e \in \mathcal{I}_t$
5: Compute $s_t(e)$ for $e \in \mathcal{I}_t$
6: $e^{\star}_t \leftarrow \arg\max_{e \in \mathcal{I}_t} s_t(e)$
7: $\mathcal{H}_{t+1} \leftarrow \mathcal{H}_t \cup \{e^{\star}_t\}$;
$\mathcal{I}_{t+1} \leftarrow \mathcal{I}_t \setminus \{e^{\star}_t\}$; $t \leftarrow t + 1$
8: **end while**
9: **return** $\mathcal{H}_t$

(b) Example construction

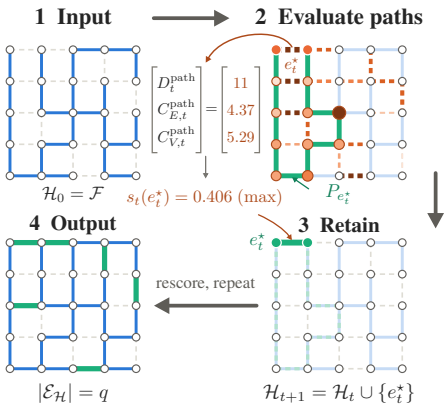

Figure 3: Starting from a spanning backbone, `Scaffold` repeatedly scores the omitted edges, retains the highest-scoring candidate, and refreshes the support until the target budget is reached.

Scalable variants. Full recomputation makes `Greedy` expensive, so we develop progressively cheaper realizations of the same scoring principle. `Heap` caches paths and refreshes only affected candidates; `Batch` scores sampled candidate batches and inserts their top-r edges between refreshes; `Fast` computes all scores once on the initial forest using Lowest Common Ancestor (`LCA`) based tree-prefix operations in $\mathcal{O}((m+n)\log n)$ time; and `Sample` converts scores from multiple backbones into reusable sampling weights, enabling repeated exact-budget draws without recomputing scores. Table 1 reports complexity, runtimes, and parameters that can be tuned for cost–quality trade-offs. Appendix C provides detailed algorithms, backbone choices, and runtime analysis.

Table 1: `Scaffold` variants and measured runtime ($\delta = 0.2$, 8 CPU workers).

Variant	Scoring / selection	Complexity	Graph (n, m)	Time (s)	Use
Greedy	dynamic, all; best one	$\mathcal{O}(TnC_{\text{sp}})$	(400, 3.1K)	0.78	reference
Heap	dynamic, affected; heap max.	$\mathcal{O}((n + T(\kappa + c))C_{\text{sp}} + m \log m)$	(4K, 39.9K)	25.16	small
Batch	dynamic, b sampled; top-r	$\mathcal{O}(\lceil T/r \rceil b C_{\text{sp}})$	(10K, 250K)	31.01	medium/large
Fast	static on $\mathcal{F}$; top-T	$\mathcal{O}((m+n)\log n)$	(200K, 2.4M)	2.30	large
Sample	static, R forests; budgeted draw	$\mathcal{O}(R(m+n)\log n)$ precomputation	(200K, 2.4M)	9.02	large

$T = q - |\mathcal{E}_{\mathcal{F}}|$; C_{sp} is one BFS/Dijkstra search. Times are medians of three complete calls including forest construction and selection. Sample uses $R = 8$; Batch uses $b = 512$, $r = 64$. Graph sizes differ across rows; see Appendix C.3.

3.3 SCAFFOLD FOR GNN TRAINING AND INFERENCE

Since `Scaffold` depends on graph topology, support graphs are generated independently of labels and GNN parameters, and directly replace $\mathcal{G}$ during message passing. In `Scaffold`-1, only one q-edge support is constructed once and used throughout training and inference. In `Scaffold`-K, a new q-edge support is generated every ρ epochs, while only the K strongest validation supports are retained in an online support bank. At inference, these supports are combined as $\mathcal{H}_{\text{union}} = \bigcup_{k=1}^{K} \mathcal{H}^{(k)}$ and evaluated in a single forward pass. Since $|\mathcal{E}_{\mathcal{H}_{\text{union}}}|$ may exceed q, its realized inference edge count is reported separately. Appendix C.2 provides further details.

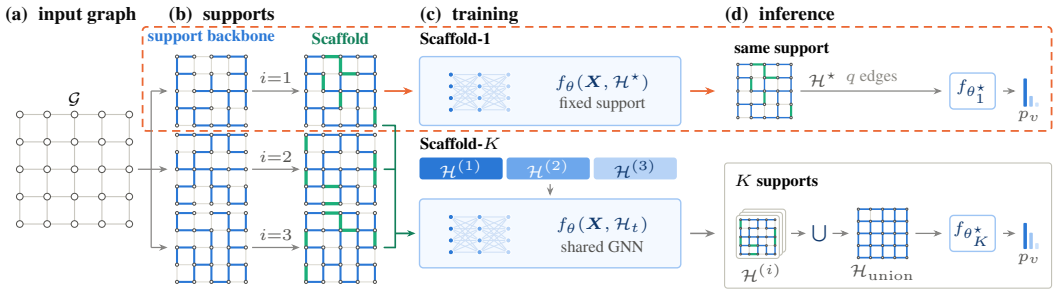

Figure 4: `Scaffold`-1 uses one fixed q-edge support for training and inference, whereas `Scaffold`-K periodically refreshes q-edge supports, retains the validation-best K, and performs one inference pass on their union.

3.4 STRUCTURAL PROPERTIES AND COMPLEXITY

Our theoretical analysis covers sparse-support properties rather than nonlinear GNN dynamics. In the dilation-only setting, selecting large-detour edges bounds the maximum dilation remaining after sparsification. For an unweighted graph $\mathcal{G}$ and a support $\mathcal{H} \subseteq \mathcal{G}$ preserving its connected components, dilation and edge congestion yield $\boldsymbol{L}_{\mathcal{H}} \preceq \boldsymbol{L}_{\mathcal{G}} \preceq \big(1 + D(\mathcal{H})C_E(\mathcal{H})\big)\boldsymbol{L}_{\mathcal{H}}$ (Boman & Hendrickson, 2003; Bern et al., 2006). This bounds effective-resistance distortion for connected graphs, while separate path arguments bound graph distances and edge-cut sizes. Node congestion gives an analogous neighborhood-concentration bound but is not required by the Laplacian bound. Complete statements, assumptions and proofs are provided in Appendix B. A general approximation guarantee for the full dilation–congestion objective remains open, and these structural bounds do not imply monotonic improvement in downstream GNN accuracy.

Runtime and memory. For an L-layer GNN with hidden dimension d, message passing on a q-edge `Scaffold` support costs $\mathcal{O}(Lqd)$ per epoch, compared with $\mathcal{O}(Lmd)$ on the full graph, while graph-dependent memory decreases from $\mathcal{O}(m)$ to $\mathcal{O}(q)$. If one support costs $C_{\texttt{Scaffold}}$ to generate, refreshing every ρ epochs gives amortized generation cost $C_{\texttt{Scaffold}}/\rho$. Support generation can additionally be performed asynchronously with GNN training. Detailed construction costs and scaling measurements are given in Appendix C.3.

4 Related Work

Support-graph theory (Vaidya, 1991; Boman & Hendrickson, 2003; Bern et al., 2006) was developed for combinatorial preconditioners, with support-tree preconditioners and low-stretch spanning trees constructing sparse backbones with short replacement paths (Maggs et al., 2005; Elkin et al., 2005; Abraham et al., 2008; Abraham & Neiman, 2012). Graph spanners (Althöfer et al., 1990) are closely related to the dilation component of `Scaffold`, whereas `Scaffold` follows the broader support-theoretic view by jointly controlling dilation and congestion under a fixed edge budget. To the best of our knowledge, `Scaffold` is the first GNN sparsifier to explicitly use this joint criterion to construct the graph used directly for graph learning. Broadly, sparsification approaches in GNNs can be grouped into: *topology-based* sparsifiers using spectral, degree, similarity, or traversal criteria (Spielman & Teng, 2011; Spielman & Srivastava, 2008; Hamann et al., 2016; Voudigari et al., 2016; Xu et al., 2007; Satuluri et al., 2011; Leskovec & Faloutsos, 2006), *semantic-based* methods that learn sparse structures from downstream signals (Chen et al., 2021; Zhang et al., 2024; 2025), and *sampling-based* methods that reduce the computation graph processed during training (Hamilton et al., 2017; Chen et al., 2018; Zeng et al., 2019; Chiang et al., 2019). `Scaffold` is a topology-based approach that scores how well omitted edges are represented by paths in the retained support. A broader discussion of related work is provided in Appendix D.

5 Experimental Results

Datasets and evaluation. We evaluate `Scaffold` on 19 node-classification benchmarks: eight homophilic, six heterophilic, and five large-scale graphs. All graphs are made undirected and self-loop-free before sparsification. We report Accuracy for single-label classification and ROC-AUC for Minesweeper, Questions, and ogbn-proteins. Dataset statistics, splits, metrics, and training configurations are given in Appendix E.

Baselines and protocol. We compare against topology- and semantic-based methods together with Full-Graph, Spanning Forest (`SF`) and No-Graph references. We use the per-dataset configurations from TunedGNN (Luo et al., 2024) and sparsifiers share the same splits, architecture, optimization, and training budget. Smaller graphs use $\delta \in \{0.3, 0.5, 0.7\}$ and large graphs use $\delta \in \{0.1, 0.3, 0.5\}$; below the spanning-forest floor ($q < n - \#\text{components}(\mathcal{G})$), construction stops at the target budget q. Unless specified, `Scaffold` uses `batch`, `fast`, and `sample` at $(\alpha, \beta_E, \beta_V) = (1, 1, 0)$. `Scaffold`-1 uses one fixed q-edge support with deterministic spanning forest (`SF`) throughout training and inference, whereas `Scaffold`-K uses random spanning forests (`RandSF`) as a starting backbone and periodically refreshes q-edge supports while asynchronously computing support to ensure the GPU is not blocked for sparse computation. Full settings are detailed in Appendix E.2[1].

5.1 Predictive Performance under Sparsification

Comparison with existing sparsifiers. We summarize key performance in Table 2. We compare sparsifiers at dataset-specific targets δ_d chosen at or above the spanning-forest floor, so the target budget is sufficient to preserve all connected components, with each method retaining exactly $q = \lceil \delta_d m \rceil$ edges. `Scaffold`-1 uses one fixed q-edge support throughout training and inference, whereas `Scaffold`-K refreshes q-edge supports during training and performs inference on the union of the K best validation supports. `Scaffold`-1 achieves the highest geometric mean among the matched-budget methods, while `Scaffold`-K can provide gains through support refresh. The main table reports the validation-selected best of `Batch`, `Fast`, and `Sample` for each dataset. Complete variant-wise results at other retention ratios are provided in Appendix F (Tables 9, 10, and 11). Figure 5 summarizes the average-rank comparison (Benavoli et al., 2016) over 57 dataset-budget evaluations, including `Scaffold` at its target-budget and best-ratio settings, the full graph and other baselines.

Performance across retention ratios. Figure 6 compares `Scaffold`-1 and `Scaffold`-K with the corresponding full-graph Tuned-GCN as the retention ratio δ increases, mapped along X-axis. Both variants approach full-graph performance as more edges are retained. At $\delta = 0.3$ and 0.5, `Scaffold`-1 reaches within one percentage point of full-graph performance on 9/19 and 13/19 datasets, respectively, while `Scaffold`-K reaches 12/19 and 16/19.

[1] `Scaffold` with GNN: `anonymous.4open.science/r/Scaffold-GNN-115E`

Wilcoxon–Holm comparison (Demšar, 2006; Benavoli et al., 2016) over 19 datasets at three retention ratios (57 tasks). Lower rank is better; bars join groups with no significant pairwise differences at $\alpha = 0.05$. **Best** uses `Scaffold`'s best ratio; **Target** uses the prescribed ratio.

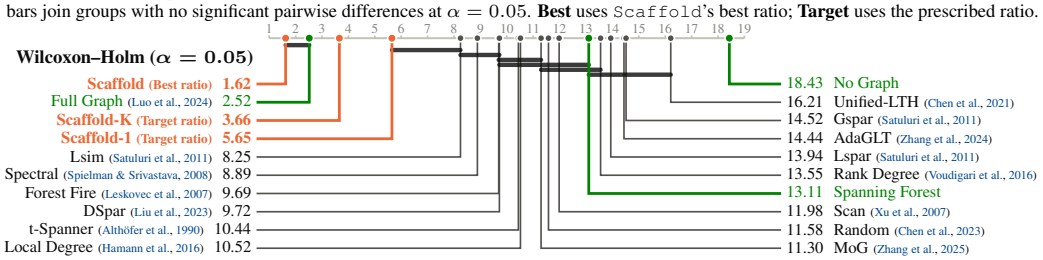

Figure 5: Wilcoxon–Holm average-rank comparison over 19 datasets and 3 edge-retention ratios.

Table 2: Accuracy/ROC-AUC at dataset-specific targets $\delta_d \geq 0.3$ above the spanning-forest floor. `Scaffold`-1 is ranked at the matched budget; `Scaffold`-K is unranked.

(a) Dataset group I: Homophilic and Heterophilic

	Method	Cora	CiteSeer	Am. Photo	WikiCS	Am. Comp.	Coauth. CS	PubMed	Coauth. Phys.	Chameleon	Squirrel	GM
	Target δ_d	0.7	0.7	0.3	0.3	0.3	0.5	0.5	0.3	0.3	0.3	
	$\lvert\mathcal{V}\rvert/\lvert\mathcal{E}\rvert$	2.71k/5.28k	3.33k/4.55k	7.65k/119.08k	11.70k/215.60k	13.75k/245.86k	18.33k/81.89k	19.72k/44.32k	34.49k/247.96k	0.89k/8.85k	2.22k/47.00k	
Ref.	Full-graph	84.56 ±0.63	72.30 ±0.47	95.95 ±0.33	80.27 ±0.44	93.80 ±0.20	95.72 ±0.04	81.22 ±1.11	97.50 ±0.12	45.46 ±4.42	44.77 ±2.36	76.44
Ref.	No graph	59.64 ±1.31	57.12 ±0.55	92.09 ±0.51	73.48 ±0.25	87.04 ±0.36	95.10 ±0.11	70.86 ±1.44	96.31 ±0.02	36.48 ±3.09	40.07 ±1.40	67.29
Ref.	S. Forest	80.10 ±0.26	69.42 ±0.56	93.97 ±0.08	76.65 ±0.22	90.45 ±0.25	95.47 ±0.19	79.48 ±0.46	96.90 ±0.02	36.73 ±4.52	37.59 ±1.74	71.87
Semantic	Uni.-LTH	79.72 ±2.26	69.18 ±0.95	91.94 ±0.36	75.49 ±0.17	86.88 ±0.74	94.96 ±0.21	76.70 ±1.05	96.49 ±0.00	37.61 ±3.67	38.17 ±2.09	71.21
Semantic	AdaGLT	80.20 ±0.95	69.46 ±0.43	94.16 ±0.19	76.48 ±0.13	91.53 ±0.42	95.05 ±0.18	77.22 ±1.10	96.65 ±0.00	38.03 ±4.12	42.30 ±3.91	72.82
Semantic	MoG	81.50 ±0.81	70.14 ±0.99	95.45 ±0.29	77.48 ±0.66	92.55 ±0.05	95.46 ±0.16	74.12 ±1.51	96.93 ±0.10	43.40 ±3.13	43.25 ±1.12	74.17
Topology	Random	81.10 ±0.72	70.16 ±0.53	94.38 ±0.16	77.04 ±0.63	91.05 ±0.15	95.32 ±0.15	75.60 ±0.88	96.66 ±0.05	37.85 ±3.35	40.90 ±1.70	72.59
Topology	Loc. Deg.	82.74 ±0.68	70.20 ±0.55	94.49 ±0.03	75.69 ±0.48	89.28 ±0.33	95.01 ±0.04	80.44 ±0.30	96.30 ±0.18	41.37 ±2.59	41.80 ±1.66	73.69
Topology	Rnk. Deg.	82.46 ±0.57	70.40 ±0.45	93.79 ±0.05	76.13 ±0.53	88.04 ±0.28	94.98 ±0.21	79.66 ±0.70	96.40 ±0.17	40.35 ±3.82	40.17 ±1.89	73.03
Topology	For. Fire	80.60 ±0.40	**71.50** ±0.48	95.40 ±0.22	79.20 ±0.58	92.57 ±0.12	95.22 ±0.24	75.50 ±0.97	96.91 ±0.03	41.28 ±1.96	43.15 ±1.90	74.12
Topology	SCAN	81.82 ±0.63	70.08 ±0.24	94.01 ±0.22	75.08 ±0.44	91.60 ±0.18	95.24 ±0.12	77.34 ±0.51	96.84 ±0.03	36.05 ±3.41	39.50 ±1.10	72.04
Topology	L-Spar	78.02 ±0.67	69.64 ±0.79	94.07 ±0.20	75.10 ±0.25	90.95 ±0.22	95.15 ±0.12	75.62 ±1.07	96.74 ±0.05	36.11 ±3.06	39.17 ±1.31	71.39
Topology	G-Spar	77.92 ±0.74	69.66 ±0.77	93.42 ±0.37	74.91 ±0.39	91.02 ±0.21	95.21 ±0.17	75.62 ±1.07	96.73 ±0.06	38.62 ±4.13	39.77 ±1.57	71.91
Topology	L-Sim	80.42 ±0.35	70.56 ±1.18	94.88 ±0.13	78.71 ±0.96	93.04 ±0.07	95.05 ±0.13	79.36 ±1.39	96.83 ±0.06	39.71 ±3.97	39.23 ±2.38	73.31
Topology	Spectral	82.78 ±0.72	70.34 ±0.82	95.24 ±0.29	78.80 ±0.47	92.13 ±0.11	95.33 ±0.15	78.86 ±0.99	96.74 ±0.08	42.98 ±3.60	43.94 ±1.18	74.86
Topology	t-Spanner	80.56 ±0.27	70.72 ±0.61	94.95 ±0.24	78.07 ±0.23	91.51 ±0.11	94.87 ±0.06	77.48 ±0.17	96.75 ±0.10	38.71 ±3.72	42.65 ±2.01	73.40
Topology	DSpar	81.20 ±0.86	70.44 ±0.37	**95.62** ±0.33	79.47 ±0.68	**93.17** ±0.08	95.49 ±0.24	79.10 ±1.09	96.84 ±0.05	39.73 ±3.87	43.48 ±1.56	74.28
	Scaffold-1	**83.80** ±0.92[b]	70.94 ±0.91[b]	95.14 ±0.23[s]	**79.48** ±0.32[s]	93.06 ±0.11[s]	**95.57** ±0.16[s]	**80.72** ±1.09[b]	**97.09** ±0.06[s]	**43.88** ±5.13[b]‡	**44.62** ±1.67[s]‡	**75.65**
	Scaffold-K	84.24 ±0.91[f] ↑	72.22 ±0.64[f] ↑	95.97 ±0.24[b] ↑	80.16 ±0.45[f] ↑	93.66 ±0.09[f] ↑	96.04 ±0.06[f] ↑	80.72 ±0.70[f]	97.22 ±0.14[s] ↑	43.89 ±2.40[s] ↑	43.99 ±2.12[f] ↓	75.94 ↑

(b) Dataset group II: Heterophilic and Large-scale

	Method	Minesweeper	Roman	Am. Ratings	Questions	Proteins	Arxiv	Reddit	Pokec	Products	GM
	Target δ_d	0.7	0.7	0.7	0.7	0.3	0.3	0.3	0.3	0.3	
	$\lvert\mathcal{V}\rvert/\lvert\mathcal{E}\rvert$	10.00k/39.40k	22.66k/32.93k	24.49k/93.05k	48.92k/153.54k	132.53k/39.56m	169.34k/1.17m	232.97k/57.31m	1.63m/22.30m	2.45m/61.86m	
Ref.	Full-graph	97.18 ±0.45	91.05 ±0.13	53.61 ±0.63	78.47 ±0.29	77.48 ±0.06	72.83 ±0.20	95.44 ±0.05	86.24 ±0.04	80.67 ±0.10	80.34
Ref.	No graph	51.49 ±0.86	66.40 ±0.33	50.01 ±0.06	70.29 ±0.28	72.91 ±0.17	55.60 ±0.16	69.66 ±0.09	62.31 ±0.02	45.46 ±0.19	59.69
Ref.	S. Forest	67.03 ±0.58	82.45 ±0.56	49.95 ±0.77	73.89 ±0.61	74.13 ±0.16	67.27 ±0.12	86.07 ±0.14	69.36 ±0.01	67.79 ±0.09	70.15
Semantic	Uni.-LTH	82.40 ±0.81	70.81 ±0.37	47.15 ±0.27	75.97 ±0.29	67.15 ±0.00	56.37 ±0.00	92.68 ±0.00	68.23 ±0.00	75.26 ±0.00	69.46
Semantic	AdaGLT	81.28 ±1.45	**90.08** ±0.00	51.64 ±0.33	73.36 ±0.59	52.31 ±0.00	55.93 ±0.00	85.09 ±0.00	64.42 ±0.00	66.74 ±0.00	67.65
Semantic	MoG	87.53 ±0.14	79.65 ±0.37	53.28 ±0.34	76.00 ±0.24	72.72 ±1.75	56.42 ±0.16	90.13 ±0.08	66.94 ±0.07	68.27 ±0.05	71.31
Topology	Random	85.18 ±0.50	77.92 ±0.41	52.26 ±0.09	75.65 ±0.28	76.83 ±0.08	67.77 ±0.18	94.69 ±0.02	76.33 ±0.05	79.08 ±0.06	75.33
Topology	Loc. Deg.	86.30 ±0.37	83.35 ±0.22	52.14 ±0.69	77.98 ±0.38	71.10 ±0.00	68.34 ±0.17	90.82 ±0.15	**80.07** ±0.09	77.62 ±0.16	75.55
Topology	Rnk. Deg.	86.26 ±0.81	82.73 ±0.19	51.59 ±0.35	77.15 ±0.34	66.01 ±0.78	62.59 ±0.07	77.28 ±0.04	67.29 ±0.10	63.13 ±0.02	69.64
Topology	For. Fire	85.26 ±0.54	78.76 ±0.17	52.08 ±0.32	75.37 ±0.65	**78.25** ±0.32	68.75 ±0.21	95.41 ±0.00	79.43 ±0.10	80.65 ±0.05	76.21
Topology	SCAN	90.20 ±0.33	80.49 ±0.74	52.34 ±0.12	77.17 ±0.68	70.62 ±0.42	67.36 ±0.37	90.28 ±0.01	68.56 ±0.02	72.56 ±0.15	73.51
Topology	L-Spar	90.43 ±0.22	79.97 ±0.27	52.60 ±0.20	73.56 ±0.69	70.80 ±0.39	65.82 ±0.05	90.06 ±0.01	67.42 ±0.02	72.64 ±0.02	72.81
Topology	G-Spar	**90.46** ±0.10	80.46 ±0.18	52.49 ±0.11	73.57 ±0.70	70.39 ±0.56	65.86 ±0.06	90.05 ±0.01	67.35 ±0.01	72.55 ±0.05	72.79
Topology	L-Sim	89.28 ±0.07	82.33 ±0.25	52.45 ±0.26	78.37 ±0.44	77.95 ±0.11	**70.69** ±0.08	94.35 ±0.05	76.56 ±0.06	79.20 ±0.09	77.01
Topology	Spectral	83.95 ±0.53	76.26 ±0.63	51.40 ±0.40	78.10 ±0.54	77.09 ±0.66	69.88 ±0.15	95.41 ±0.03	79.65 ±0.07	80.00 ±0.15	75.96
Topology	t-Spanner	85.22 ±0.72	78.17 ±0.19	52.65 ±0.61	76.61 ±0.69	77.43 ±0.12	67.51 ±0.09	94.72 ±0.01	76.38 ±0.07	79.30 ±0.01	75.59
Topology	DSpar	85.51 ±0.53	77.26 ±0.66	52.38 ±0.20	**78.47** ±0.16	51.94 ±1.16	55.19 ±0.13	95.22 ±0.01	69.39 ±0.03	77.43 ±0.08	69.90
	Scaffold-1	86.98 ±0.76[f]	88.01 ±0.59[f]	**53.36** ±0.81[f]	78.29 ±0.29[f]	77.80 ±0.63[s]	70.65 ±0.42[s]	**95.43** ±0.18[s]	80.00 ±0.01[s]	**81.07** ±0.25[s]	**78.15**
	Scaffold-K	93.35 ±0.34[f] ↑	88.26 ±0.88[f] ↑	54.73 ±0.20[b] ↑	78.67 ±0.28[f] ↑	76.91 ±0.56[s] ↓	72.58 ±0.01[s] ↑	95.36 ±0.19[s] ↓	81.53 ±0.08[f] ↑	80.46 ±1.12[s] ↓	79.29 ↑

Green/blue/amber mark the top three matched-budget methods; references are unranked. Values are mean ± std. (%); **GM**: geometric mean. For each dataset and `Scaffold` setting, validation selects the best of `Batch`/`Fast`/`Sample`, marked $^{b/f/s}$. `Scaffold`-K arrows compare against `Scaffold`-1. †ROC-AUC; ‡SLST on CHAMELEON/SQUIRREL (SF otherwise). Full results: Appendix F.

5.2 Runtime and Memory Efficiency

Figure 7 shows that reducing the retained-edge ratio directly lowers both message-passing time and peak GPU memory on REDDIT and OGBN-PRODUCTS. Even after including support graph construction, the reduced `Gather-Scatter/SpMM` workload yields lower end-to-end training cost at sparse budgets. The benefits are greatest at lower retention ratios, on large graphs, and over long runs, where per-epoch savings amortize sparsification overhead.

Figure 8 further compares standalone sparsification and total end-to-end costs across large graphs: `Scaffold` remains practical, while several more expensive structural or learned baselines incur substantially higher costs. We note that settings for `Scaffold` and sampling methods can be adapted to trade off runtime versus quality. Here, we use the same settings as in the preceding accuracy experiments. Detailed runtimes, memory measurements, worker scaling, and per-dataset results are reported in Appendix G.

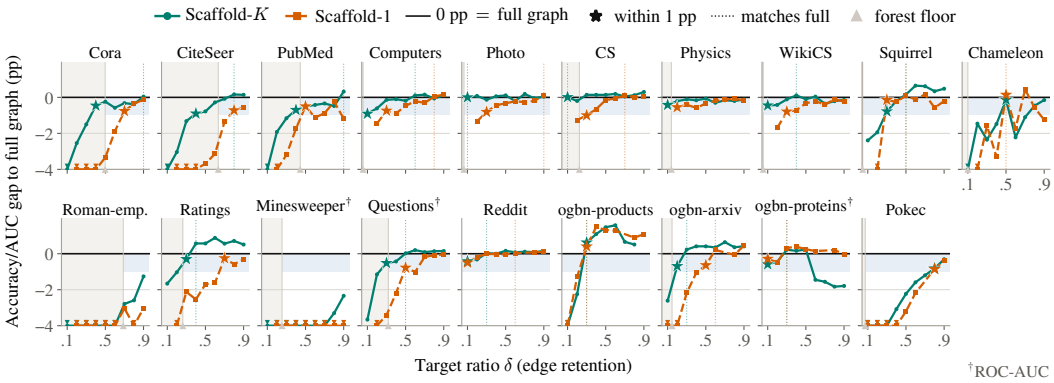

Figure 6: Performance gap of `Scaffold-1` and `Scaffold-`K to the full graph across δ. Blue marks results within 1%; stars mark the first such δ, and gray marks below the spanning-forest.

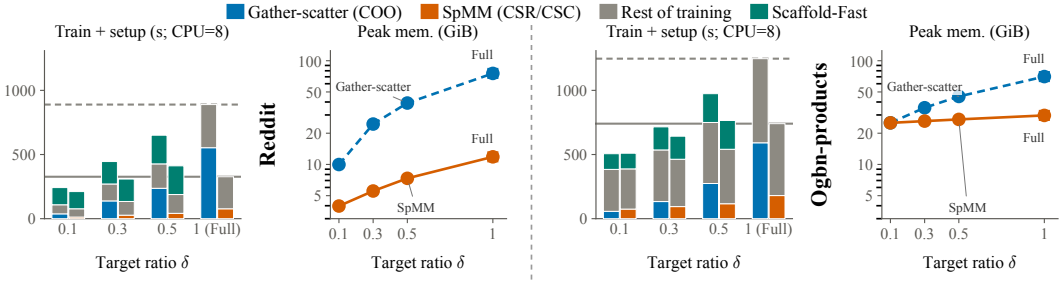

Figure 7: Runtime/memory profile of `Scaffold-Fast-1` with `Gather-Scatter/SpMM` aggregation in comparison to the Full-GCN.

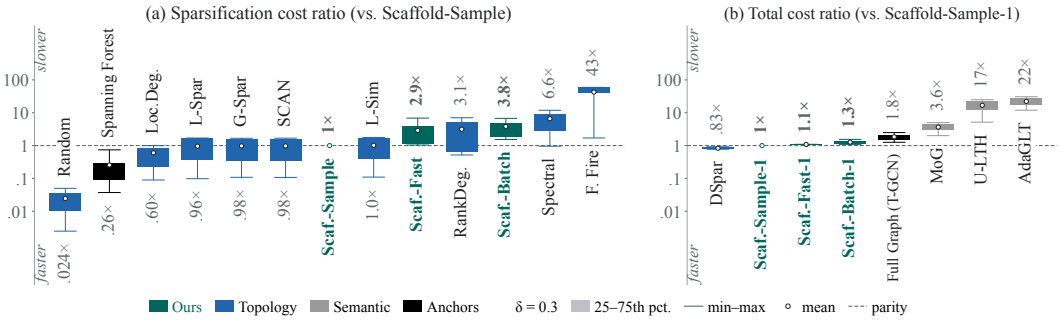

Figure 8: (a) Standalone sparsification cost relative to `Scaffold-Fast`, and (b) total end-to-end cost relative to `Scaffold-Sample-1` over 300 epochs at $\delta = 0.3$. More details are in Appendix G.

5.3 Ablation and Structural Analysis

Support-score components and structural quality. Figure 9 compares components of Equation 1 under the same support backbone (`SF`) and budget. Prioritizing low dilation improves over random filling ($\emptyset$), which highlights the importance of short communication paths. Edge or node congestion alone can favor longer routes and is less effective, whereas combining dilation and edge congestion gives the strongest predictive association with test performance. Node congestion captures a distinct theoretical failure mode in controlled examples (§A.4), but provides limited additional benefit on these GNN benchmarks. This pattern is not specific to `Scaffold`. In Figure 10, all sparsifiers share the same spanning forest and edge budget, isolating their edge-selection rules. Methods that better reduce dilation and edge congestion achieve higher GNN accuracy, with dilation showing the strongest association. This may partly explain why Random remains competitive once connectivity is guaranteed, as random edges can shorten detours and partially alleviate structural bottlenecks.

Scaffold variants and support backbones. Figure 11 shows a clear quality-cost trade-off: `Heap` and `Batch` provide stronger structural refinement at higher runtime, while `Fast` and `Sample` trade some structural quality for much lower construction cost. Predictive performance is not strictly determined by structural score, indicating that backbone choice and stochasticity also matter.

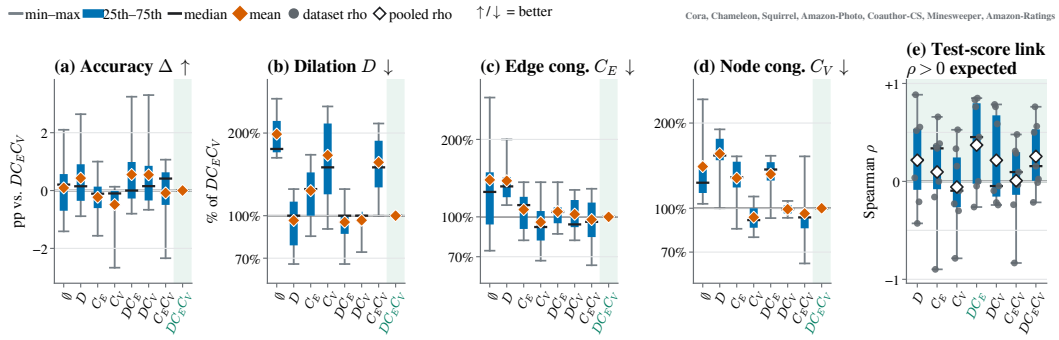

Figure 9: Dilation achieves the best individual performance, while combining dilation and edge congestion gives the strongest accuracy–structure trade-off and association with test performance.

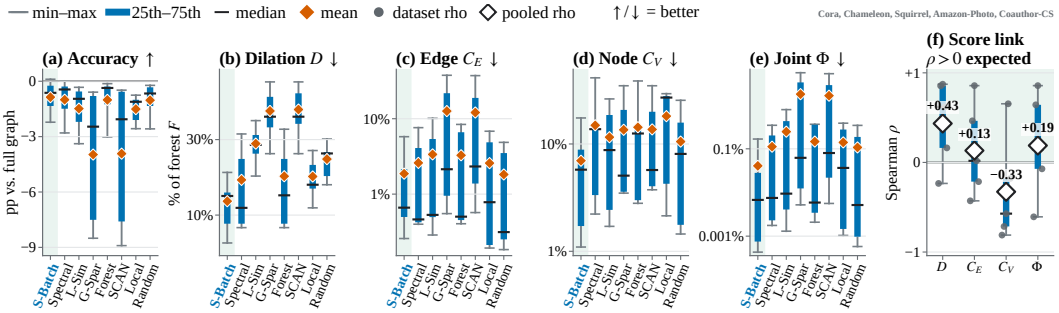

Figure 10: Support quality at $\delta = 0.7$ from a common spanning-forest backbone. Lower dilation and edge congestion generally correspond to higher GNN accuracy.

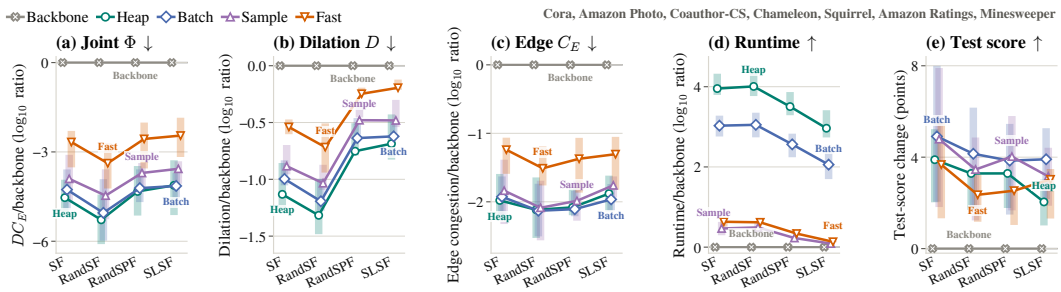

Figure 11: Quality–cost trade-offs across variants and backbones. `SF`: spanning forest; `RandSF`: random spanning forest; `RandSPF`: random shortest-path forest; `SLSF`: scalable low-stretch forest.

Additional Ablations and Analyses. Appendix H extends the main analysis with `Greedy` variant and backbone comparisons (§H.1), path-congestion aggregation choices (p_E, p_V) (§H.2), and analyses of inference supports K and full-graph evaluation (§H.3). We further study support-refresh frequency (§H.4), synchronous versus asynchronous support generation (§H.5), the interaction between training-time support refresh and inference topology (§H.6), and whether repeated structure-aware supports provide benefits beyond uniform random resampling (§H.7). Additional experiments examine structural implications (§H.8) and compatibility across GNN architectures (§H.9).

6 Conclusions

We introduced `Scaffold`, a novel unsupervised topology-based graph sparsification framework that uses the support-theoretic notions of dilation and congestion to construct sparse communication graphs for GNNs. Across GNN benchmarks, `Scaffold` maintains strong performance at low edge budgets while reducing memory and end-to-end training cost. These results suggest that, under a fixed edge budget, GNN performance depends on how well the retained graph represents the omitted connections. Our theory characterizes the resulting structural distortion, but a general approximation guarantee for the full dilation-congestion objective remains open. The current evaluation focuses on static, undirected, unweighted node-classification graphs. Extensions of `Scaffold` to task-aware, localized, and broader graph-learning settings are left for future work.

AI USE DISCLOSURE

Generative AI tools were used to assist with polishing the manuscript and supporting research execution, including software implementation and experimental workflows. They were not used for research ideation, formulation of the proposed method, or generation of scientific claims. All AI-assisted content and code were reviewed, revised, and validated by the authors. The authors independently verified the reported results and claims and take full responsibility for the final content of this work.

REPRODUCIBILITY STATEMENT

We provide two anonymous code repositories to support reproducibility. (i) `anonymous.4open.science/r/Scaffold-B6DB` contains the standalone, installable `Scaffold` package implementing the proposed methods. (ii) `anonymous.4open.science/r/Scaffold-GNN-115E` contains the integration of `Scaffold` with the GNN benchmarks used in this work, together with implementations of the evaluated baselines and the dataset-specific configurations and hyperparameter settings. Appendix E provides additional details on the datasets, training procedures, and evaluation protocols. All support, variant, and checkpoint selection is performed exclusively using validation data; test labels are never used for model or method selection.

REFERENCES

Ittai Abraham and Ofer Neiman. Using petal-decompositions to build a low stretch spanning tree. In *Proceedings of the forty-fourth annual ACM symposium on Theory of computing*, pp. 395–406, 2012.

Ittai Abraham, Yair Bartal, and Ofer Neiman. Nearly tight low stretch spanning trees. In *2008 49th Annual IEEE Symposium on Foundations of Computer Science*, pp. 781–790. IEEE, 2008.

Selahattin Akkas and Ariful Azad. Shapley-value-based graph sparsification for gnn inference. *arXiv preprint arXiv:2507.20460*, 2025.

Ingo Althöfer, Gautam Das, David Dobkin, and Deborah Joseph. Generating sparse spanners for weighted graphs. In *Scandinavian Workshop on Algorithm Theory*, pp. 26–37. Springer, 1990.

Joshua D Batson, Daniel A Spielman, and Nikhil Srivastava. Twice-ramanujan sparsifiers. In *Proceedings of the forty-first annual ACM symposium on Theory of computing*, pp. 255–262, 2009.

Alessio Benavoli, Giorgio Corani, and Francesca Mangili. Should we really use post-hoc tests based on mean-ranks? *The Journal of Machine Learning Research*, 17(1):152–161, 2016.

András A Benczúr and David R Karger. Randomized approximation schemes for cuts and flows in capacitated graphs. *SIAM Journal on Computing*, 44(2):290–319, 2015.

Marshall Bern, John R Gilbert, Bruce Hendrickson, Nhat Nguyen, and Sivan Toledo. Support-graph preconditioners. *SIAM Journal on Matrix Analysis and Applications*, 27(4):930–951, 2006.

Erik G Boman and Bruce Hendrickson. Support theory for preconditioning. *SIAM Journal on Matrix Analysis and Applications*, 25(3):694–717, 2003.

Leizhen Cai and Derek G Corneil. Tree spanners. *SIAM Journal on Discrete Mathematics*, 8(3): 359–387, 1995.

Jie Chen, Tengfei Ma, and Cao Xiao. Fastgcn: Fast learning with graph convolutional networks via importance sampling. *arXiv preprint arXiv:1801.10247*, 2018.

Tianlong Chen, Yongduo Sui, Xuxi Chen, Aston Zhang, and Zhangyang Wang. A unified lottery ticket hypothesis for graph neural networks. In *International conference on machine learning*, pp. 1695–1706. PMLR, 2021.

Yuhan Chen, Haojie Ye, Sanketh Vedula, Alex Bronstein, Ronald Dreslinski, Trevor Mudge, and Nishil Talati. Demystifying graph sparsification algorithms in graph properties preservation. *arXiv preprint arXiv:2311.12314*, 2023.

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.

Siddhartha Shankar Das, Naheed Anjum Arafat, Muftiqur Rahman, SM Ferdous, Alex Pothen, Mahantesh Halappanavar, and Danda B Rawat. Sgs-gnn: A supervised graph sparsifier for graph neural networks. In *Proceedings of the 32nd ACM SIGKDD Conference on Knowledge Discovery and Data Mining V. 2*, pp. 755–766, 2026.

Janez Demšar. Statistical comparisons of classifiers over multiple data sets. *Journal of Machine learning research*, 7(Jan):1–30, 2006.

Haipeng Ding, Zhewei Wei, and Yuhang Ye. Large-scale spectral graph neural networks via laplacian sparsification. In *Proceedings of the 31st ACM SIGKDD Conference on Knowledge Discovery and Data Mining V. 1*, pp. 213–223, 2025.

Michael Elkin, Yuval Emek, Daniel A Spielman, and Shang-Hua Teng. Lower-stretch spanning trees. In *Proceedings of the thirty-seventh annual ACM symposium on Theory of computing*, pp. 494–503, 2005.

Yuval Emek and David Peleg. Approximating minimum max-stretch spanning trees on unweighted graphs. *SIAM Journal on Computing*, 38(5):1761–1781, 2009.

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

Luca Franceschi, Mathias Niepert, Massimiliano Pontil, and Xiao He. Learning discrete structures for graph neural networks. In *International conference on machine learning*, pp. 1972–1982. PMLR, 2019.

Yuan Gao, Rasmus Kyng, and Daniel A Spielman. Robust and practical solution of laplacian equations by approximate elimination. *arXiv preprint arXiv:2303.00709*, 2023.

Johannes Gasteiger, Stefan Weißenberger, and Stephan Günnemann. Diffusion improves graph learning. *Advances in neural information processing systems*, 32, 2019.

Michael Hamann, Gerd Lindner, Henning Meyerhenke, Christian L Staudt, and Dorothea Wagner. Structure-preserving sparsification methods for social networks. *Social Network Analysis and Mining*, 6(1):22, 2016.

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

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.

Bo Hui, Da Yan, Xiaolong Ma, and Wei-Shinn Ku. Rethinking graph lottery tickets: Graph sparsity matters. *arXiv preprint arXiv:2305.02190*, 2023.

Wei Jin, Yao Ma, Xiaorui Liu, Xianfeng Tang, Suhang Wang, and Jiliang Tang. Graph structure learning for robust graph neural networks. In *Proceedings of the 26th ACM SIGKDD international conference on knowledge discovery & data mining*, pp. 66–74, 2020.

Jure Leskovec and Christos Faloutsos. Sampling from large graphs. In *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 631–636, 2006.

Jure Leskovec, Jon Kleinberg, and Christos Faloutsos. Graph evolution: Densification and shrinking diameters. *ACM transactions on Knowledge Discovery from Data (TKDD)*, 1(1):2–es, 2007.

Jiayu Li, Tianyun Zhang, Hao Tian, Shengmin Jin, Makan Fardad, and Reza Zafarani. Sgcn: A graph sparsifier based on graph convolutional networks. In *Pacific-Asia Conference on Knowledge Discovery and Data Mining*, pp. 275–287. Springer, 2020.

Zhixun Li, Liang Wang, Xin Sun, Yifan Luo, Yanqiao Zhu, Dingshuo Chen, Yingtao Luo, Xiangxin Zhou, Qiang Liu, Shu Wu, et al. Gslb: The graph structure learning benchmark. *Advances in Neural Information Processing Systems*, 36:30306–30318, 2023.

Ningyi Liao, Zihao Yu, Ruixiao Zeng, and Siqiang Luo. Unifews: You need fewer operations for efficient graph neural networks. *arXiv preprint arXiv:2403.13268*, 2024.

Angelica Liguori, Ettore Ritacco, Pietro Sabatino, and Annalisa Socievole. Spectral neural graph sparsification. *arXiv preprint arXiv:2510.27474*, 2025.

Derek Lim, Felix Hohne, Xiuyu Li, Sijia Linda Huang, Vaishnavi Gupta, Omkar Bhalerao, and Ser-Nam Lim. Large scale learning on non-homophilous graphs: New benchmarks and strong simple methods. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2021.

Yixin Liu, Yu Zheng, Daokun Zhang, Hongxu Chen, Hao Peng, and Shirui Pan. Towards unsupervised deep graph structure learning. In *Proceedings of the ACM web conference 2022*, pp. 1392–1403, 2022.

Zirui Liu, Kaixiong Zhou, Zhimeng Jiang, Li Li, Rui Chen, Soo-Hyun Choi, and Xia Hu. Dspar: An embarrassingly simple strategy for efficient gnn training and inference via degree-based sparsification. *Transactions on Machine Learning Research*, 2023.

Andreas Loukas. Graph reduction with spectral and cut guarantees. *Journal of Machine Learning Research*, 20(116):1–42, 2019.

Dongsheng Luo, Wei Cheng, Wenchao Yu, Bo Zong, Jingchao Ni, Haifeng Chen, and Xiang Zhang. Learning to drop: Robust graph neural network via topological denoising. In *Proceedings of the 14th ACM international conference on web search and data mining*, pp. 779–787, 2021.

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.

Bruce M Maggs, Gary L Miller, Ojas Parekh, R Ravi, and Shan Leung Maverick Woo. Finding effective support-tree preconditioners. In *Proceedings of the seventeenth annual ACM symposium on Parallelism in algorithms and architectures*, pp. 176–185, 2005.

Péter Mernyei and Cătălina Cangea. Wiki-CS: A Wikipedia-based benchmark for graph neural networks. *ICML Workshop on Graph Representation Learning and Beyond*, 2020.

Oleg Platonov, Denis Kuznedelev, Artem Babenko, and Liudmila Prokhorenkova. Characterizing graph datasets for node classification: Homophily–heterophily dichotomy and beyond. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2023a.

Oleg Platonov, Denis Kuznedelev, Michael Diskin, Artem Babenko, and Liudmila Prokhorenkova. A critical look at the evaluation of GNNs under heterophily: Are we really making progress? In *International Conference on Learning Representations (ICLR)*, 2023b.

Yu Rong, Wenbing Huang, Tingyang Xu, and Junzhou Huang. Dropedge: Towards deep graph convolutional networks on node classification. *arXiv preprint arXiv:1907.10903*, 2019.

Venu Satuluri, Srinivasan Parthasarathy, and Yiye Ruan. Local graph sparsification for scalable clustering. In *Proceedings of the 2011 ACM SIGMOD International Conference on Management of data*, pp. 721–732, 2011.

Prithviraj Sen, Galileo Namata, Mustafa Bilgic, Lise Getoor, Brian Galligher, and Tina Eliassi-Rad. Collective classification in network data. *AI Magazine*, 29(3):93–106, 2008.

Oleksandr Shchur, Maximilian Mumme, Aleksandar Bojchevski, and Stephan Günnemann. Pitfalls of graph neural network evaluation. *Relational Representation Learning Workshop, NeurIPS*, 2018.

Daniel A Spielman and Nikhil Srivastava. Graph sparsification by effective resistances. In *Proceedings of the fortieth annual ACM symposium on Theory of computing*, pp. 563–568, 2008.

Daniel A Spielman and Shang-Hua Teng. Spectral sparsification of graphs. *SIAM Journal on Computing*, 40(4):981–1025, 2011.

PM Vaidya. Solving linear equations with symmetric diagonally dominant matrices by constructing good preconditioners. unpublished manuscript uiuc 1990. In *IMA Workshop on Graph Theory and Sparse Matrix Computation*, 1991.

Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. Graph attention networks. *arXiv preprint arXiv:1710.10903*, 2017.

Elli Voudigari, Nikos Salamanos, Theodore Papageorgiou, and Emmanuel J Yannakoudakis. Rank degree: An efficient algorithm for graph sampling. In *2016 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining (ASONAM)*, pp. 120–129. IEEE, 2016.

Kun Wang, Yuxuan Liang, Pengkun Wang, Xu Wang, Pengfei Gu, Junfeng Fang, and Yang Wang. Searching lottery tickets in graph neural networks: A dual perspective. In *the eleventh international conference on learning representations*, 2022.

Minjie Wang, Da Zheng, Zihao Ye, Quan Gan, Mufei Li, Xiang Song, Jinjing Zhou, Chao Ma, Lingfan Yu, Yu Gai, et al. Deep graph library: A graph-centric, highly-performant package for graph neural networks. *arXiv preprint arXiv:1909.01315*, 2019.

Siyao Wang and Miles E Lopes. Empirical error estimates for graph sparsification. *arXiv preprint arXiv:2503.08031*, 2025.

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.

Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? *arXiv preprint arXiv:1810.00826*, 2018.

Xiaowei Xu, Nurcan Yuruk, Zhidan Feng, and Thomas AJ Schweiger. Scan: a structural clustering algorithm for networks. In *Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 824–833, 2007.

Yang Ye and Shihao Ji. Sparse graph attention networks. *IEEE Transactions on Knowledge and Data Engineering*, 35(1):905–916, 2021.

Hanqing Zeng, Hongkuan Zhou, Ajitesh Srivastava, Rajgopal Kannan, and Viktor Prasanna. Graphsaint: Graph sampling based inductive learning method. *arXiv preprint arXiv:1907.04931*, 2019.

Guibin Zhang, Kun Wang, Wei Huang, Yanwei Yue, Yang Wang, Roger Zimmermann, Aojun Zhou, Dawei Cheng, Jin Zeng, and Yuxuan Liang. Graph lottery ticket automated. In *International Conference on Learning Representations*, volume 2024, pp. 31044–31068, 2024.

Guibin Zhang, Xiangguo Sun, Yanwei Yue, Chonghe Jiang, Kun Wang, Tianlong Chen, and Shirui Pan. Graph sparsification via mixture of graphs. In *International Conference on Learning Representations*, volume 2025, pp. 92735–92763, 2025.

Shiqi Zhang, Renchi Yang, Jing Tang, Xiaokui Xiao, and Bo Tang. Efficient approximation algorithms for spanning centrality. In *Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pp. 3386–3395, 2023.

Cheng Zheng, Bo Zong, Wei Cheng, Dongjin Song, Jingchao Ni, Wenchao Yu, Haifeng Chen, and Wei Wang. Robust graph representation learning via neural sparsification. In *International conference on machine learning*, pp. 11458–11468. PMLR, 2020.

Jie Zhou, Ganqu Cui, Shengding Hu, Zhengyan Zhang, Cheng Yang, Zhiyuan Liu, Lifeng Wang, Changcheng Li, and Maosong Sun. Graph neural networks: A review of methods and applications. *AI open*, 1:57–81, 2020.

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.

APPENDIX CONTENTS

A Supplementary Preliminaries **15**

A.1 Notations 15

A.2 Support-Theoretic Background 15

A.3 Weighted Formulation of Scaffold 16

A.4 Why the Support Quantities Matter for GNNs 16

B Proofs and Structural Guarantees **17**

B.1 Optimization of the Global Objective is NP-Hard 17

B.2 Dilation Reduction is Non-Submodular 18

B.3 Guarantee for Dilation-Only Selection 18

B.4 Dilation and Edge Congestion Bound Structural Distortion 19

B.5 Effective-Resistance, Distance, and Congestion Consequences 19

B.6 Support Quality Under Edge Unions 20

C Method Details **20**

C.1 Scaffold Algorithms, Backbones, and Variants 20

C.2 Scaffold-1 and Scaffold-K Training and Inference 23

C.3 Runtime and Scalability 24

C.4 Possible Extension: Localized or Partitioned Supports 26

D Additional Related Work **26**

E Experimental Settings **27**

E.1 Datasets and Shared Training Settings 27

E.2 Baseline Training Settings 27

F Detailed Experimental Results **28**

F.1 Results on Homophilic Graphs 28

F.2 Results on Heterophilic Graphs 28

F.3 Results on Large-scale Graphs 28

F.4 Case study of performance of different datasets 32

F.5 Realized Inference Union Size of Scaffold-K 32

G Runtime and Memory Details with GNN Benchmarks **33**

G.1 Sparsification time and Scaling of Scaffold 33

G.2 Training profile breakdown of Scaffold-Fast-1 and Full-GCN. 33

G.3 Sparsification and end-to-end runtime 34

H Additional Ablation and Structural Analysis **35**

H.1 Scaffold Variants and Support Backbones with Greedy 35
H.2 Effect of the path node and edge congestion parameter p_E, p_V 35
H.3 Sensitivity to the number of supports K 36
H.4 Support-Refresh Frequency 37
H.5 Impact of Synchronous vs. Asynchronous Support Generation 37
H.6 Training Support Refresh and Inference Topology 37
H.7 Scaffold-K vs Random sparse sample at every epoch 38
H.8 Structural implications of Scaffold 38
H.9 Compatibility of Scaffold across GNN architectures 39

A Supplementary Preliminaries

This section provides additional support-theoretic background, weighted-path conventions, and examples illustrating the role of support quantities in GNN message passing.

A.1 Notations

Table 3: Core notation used in Scaffold.

Symbol	Meaning
$\mathcal{G} = (\mathcal{V}, \mathcal{E}, \boldsymbol{w}); n, m$	Input graph with $n = \lvert\mathcal{V}\rvert$ nodes and $m = \lvert\mathcal{E}\rvert$ edges.
$\mathcal{H}_t$; $\mathcal{F}$	Current sparse support and initial spanning-forest backbone.
δ; $q = \lceil \delta m \rceil$	Edge-retention ratio and target edge budget.
$\mathcal{I}_t = \mathcal{E} \setminus \mathcal{E}_{\mathcal{H}_t}$	Candidate edges omitted from the current support.
P_e	Supporting path of omitted edge e in $\mathcal{H}_t$.
$D(\mathcal{H})$, $C_E(\mathcal{H})$, $C_V(\mathcal{H})$	Maximum dilation, edge congestion, and node congestion.
$D_t^{\text{path}}(e)$, $C_{E,t}^{\text{path}}(e)$, $C_{V,t}^{\text{path}}(e)$	Candidate path dilation and edge/node congestion scores.
α, β_E, β_V	Weights of the three support-score components.
$s_t(e)$	Normalized Scaffold score of candidate edge e.
$\mathcal{H}_{\text{union}}$	Union of retained supports used for multi-support inference.

A.2 Support-Theoretic Background

Support graph theory was developed for constructing sparse combinatorial preconditioners for symmetric diagonally dominant linear systems (Vaidya, 1991; Boman & Hendrickson, 2003; Bern et al., 2006). For graph Laplacians $\boldsymbol{L}_{\mathcal{G}}$ and $\boldsymbol{L}_{\mathcal{H}}$, the support number is defined as

$$\sigma(\boldsymbol{L}_{\mathcal{G}}, \boldsymbol{L}_{\mathcal{H}}) = \min\{\tau : \boldsymbol{L}_{\mathcal{G}} \preceq \tau \boldsymbol{L}_{\mathcal{H}}\},$$

where smaller τ indicates that the sparse graph $\mathcal{H}$ provides a closer spectral support for $\mathcal{G}$.

For graph Laplacians, this algebraic relation also admits a combinatorial interpretation. An edge omitted from $\mathcal{H}$ is represented by a supporting path in $\mathcal{H}$ connecting the same pair of endpoints. The quality of this representation depends on two complementary properties: (i) how long the supporting path becomes relative to the omitted edge, and (ii) how strongly supporting paths overlap on the retained graph. These effects are captured by *dilation* and *congestion*, respectively. Classical congestion–dilation arguments relate their joint effect to the quality of the support graph (Vaidya, 1991; Boman & Hendrickson, 2003; Bern et al., 2006).

Scaffold builds directly on this perspective. Rather than using the sparse support $\mathcal{H}$ as a preconditioner for solving a linear system, we use it as the communication graph for GNN message passing. An omitted edge is therefore well supported when its endpoints remain connected by a short path and when that path does not pass through heavily shared parts of the retained graph. This motivates the dilation–congestion criteria used throughout Scaffold.

A.3 Weighted Formulation of Scaffold

The main text uses the unweighted formulation used in our experiments. More generally, let $\ell_e > 0$ denote the path length of edge e and $\omega_e > 0$ its routing load. For an omitted edge $e = (u, v)$, let

$$P_e = \mathrm{ShortestPath}_{\mathcal{H}}(u, v; \ell), \qquad D_\ell(\mathcal{H}) = \max_{e \in \mathcal{I}} \frac{\sum_{a \in P_e} \ell_a}{\ell_e}.$$

Weighted edge and node congestion are

$$C_E^\omega(\mathcal{H}) = \max_{a \in \mathcal{E}_{\mathcal{H}}} \sum_{\substack{e \in \mathcal{I} \\ a \in P_e}} \omega_e, \qquad C_V^\omega(\mathcal{H}) = \max_{x \in \mathcal{V}} \sum_{\substack{e \in \mathcal{I} \\ x \in \mathrm{int}(P_e)}} \omega_e.$$

The corresponding weighted objective is

$$\Phi_{\ell,\omega}(\mathcal{H}) = (1 + D_\ell(\mathcal{H}))^\alpha (1 + C_E^\omega(\mathcal{H}))^{\beta_E} (1 + C_V^\omega(\mathcal{H}))^{\beta_V}.$$

Setting $\ell_e = \omega_e = 1$ recovers Equation 1.

The candidate-wise surrogate extends analogously. For the current candidate set $\mathcal{I}_t$, define

$$D_t^{\mathrm{path},\ell}(e) = \frac{\sum_{a \in P_e} \ell_a}{\ell_e}, \qquad c_{E,t}^\omega(a) = \sum_{\substack{f \in \mathcal{I}_t \\ a \in P_f}} \omega_f, \qquad c_{V,t}^\omega(x) = \sum_{\substack{f \in \mathcal{I}_t \\ x \in \mathrm{int}(P_f)}} \omega_f.$$

The path-congestion scores are obtained from these weighted loads using the same power-mean aggregation as Equations 4–5, and Equation 6 is applied unchanged after normalization. Thus, $\ell_e = \omega_e = 1$ recovers the unweighted `Scaffold` surrogate used throughout this work.

The spanning-forest formulation requires $q \geq n - c(\mathcal{G})$ to preserve every connected component. If $q < n - c(\mathcal{G})$, no support can simultaneously satisfy the requested budget and preserve all components; in this case, we enforce the budget and do not invoke guarantees requiring a spanning support.

A.4 Why the Support Quantities Matter for GNNs

Equation 1 combines dilation, edge congestion, and node congestion in the `Scaffold` objective. Classical support theory directly motivates the first two quantities, while node congestion is included to capture concentration at intermediate vertices during message passing. Although node congestion provides limited additional benefit on our node-classification benchmarks, it represents a distinct communication failure mode that can arise independently of edge congestion.

Figure 12 gives controlled node-classification examples that isolate the three quantities. In each row, the detrimental and useful supports retain the same number of edges and are matched on the other two support quantities, so the structural difference is restricted to the quantity being studied. Each target must predict the ± 1 bit associated with its source, giving chance accuracy of 50%.

For *dilation*, deleting a shortcut increases the source–target detour from $D = 2$ to $D = 8$ while both congestion measures remain fixed. The source then lies outside the two-layer receptive field, reducing accuracy from 100% to chance. This illustrates finite-hop under-reaching caused by long supporting paths.

For *edge congestion*, the two supports have the same dilation and node congestion, but the detrimental support routes more omitted relations through a shared edge, increasing C_E from 5 to 7. The resulting bottleneck makes the paired targets indistinguishable, whereas separating the routes recovers perfect prediction.

For *node congestion*, dilation and edge congestion remain fixed while the detrimental support routes multiple relations through a common hub, increasing C_V from 3 to 12. The shared intermediate representation causes the targets to become indistinguishable, whereas private relays avoid this node-level bottleneck.

These constructions show that dilation, edge congestion, and node congestion capture complementary support-level communication failures, even when their empirical importance differs across benchmark families.

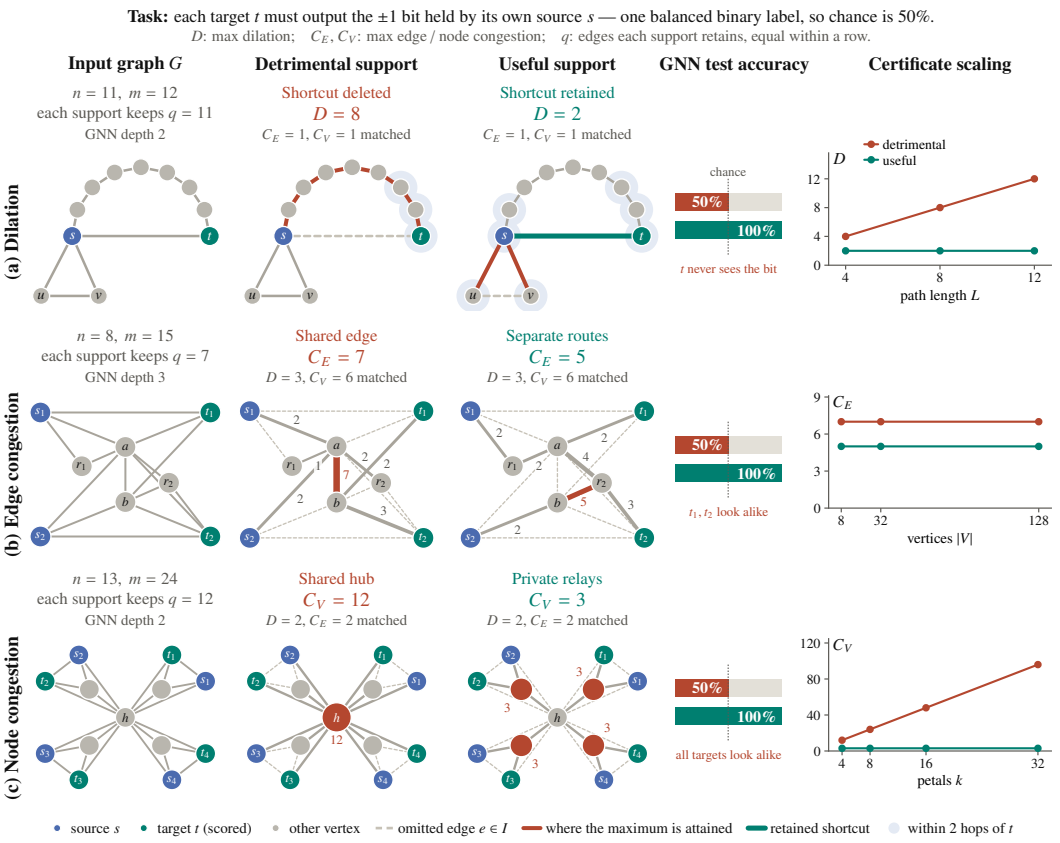

Figure 12: Controlled roles of dilation and congestion in GNN message passing. Each row varies one support quantity while matching the other two at the same edge budget. High dilation causes finite-hop under-reaching, while high edge or node congestion creates shared communication bottlenecks.

B Proofs and Structural Guarantees

The guarantees below are stated for the unweighted setting used in our experiments and in the main-text objective. We assume that $\mathcal{H}$ preserves the connected components of $\mathcal{G}$, so every omitted edge has a supporting path. The weighted formulation is given in Appendix A.3; a weighted extension of the Laplacian bound is provided in the following sections.

B.1 Optimization of the Global Objective is NP-Hard

Theorem B.1 (Hardness of budgeted support sparsification) *For any $\alpha > 0$, optimizing Equation 1 is NP-hard even for connected unweighted graphs with $\beta_E = \beta_V = 0$. The hardness already holds for $q = n - 1$.*

Proof. Set $\beta_E = \beta_V = 0$ and $q = n - 1$. Since $\mathcal{G}$ is connected, every feasible support is a spanning tree $\mathcal{T}$, and minimizing the objective is equivalent to minimizing $D(\mathcal{T})$. For a spanning tree, the maximum stretch satisfies

$$\max_{u \neq v} \frac{\mathrm{dist}_{\mathcal{T}}(u, v)}{\mathrm{dist}_{\mathcal{G}}(u, v)} = \max_{(u,v) \in \mathcal{E}} \mathrm{dist}_{\mathcal{T}}(u, v) = \max\{1, D(\mathcal{T})\}.$$

The first equality follows by taking a shortest u–v path in $\mathcal{G}$ and replacing each of its edges by the corresponding path in $\mathcal{T}$; hence the stretch of any vertex pair is bounded by the maximum stretch of an original edge. The second equality follows because retained tree edges have stretch 1, while $D(\mathcal{T})$ is the maximum stretch over omitted edges.

Thus, minimizing $D(\mathcal{T})$ is equivalent to the Minimum Maximum-Stretch Spanning Tree problem, which is NP-hard (Cai & Corneil, 1995; Emek & Peleg, 2009). ∎

B.2 Dilation Reduction is Non-Submodular

Since directly optimizing Equation 1 is NP-hard, `Scaffold` uses the surrogate rule in Equation 7. A natural question is whether this surrogate admits a general approximation guarantee for the original objective. This is difficult because adding an edge can change other supporting paths and gains of other candidates. Even the dilation-only case is non-submodular, as shown below. However, it does not rule out other forms of approximation analysis, and we leave a general approximation ratio for the full dilation–congestion surrogate to future work.

Proposition B.2 (Dilation reduction is not submodular) *For a fixed backbone $\mathcal{F}$, let*

$$f(S) = D(\mathcal{F}) - D(\mathcal{F} \cup S),$$

where dilation is measured over the edges that remain omitted. Then f is not submodular in general.

Proof. Consider three branches

$$r - a_1 - a_2 - a_3, \qquad r - b_1 - b_2 - b_3, \qquad r - c_1 - c_2,$$

with candidate edges $e_a = (r, a_3)$, $e_b = (r, b_3)$, and $e_c = (r, c_2)$. Their initial dilations are $3, 3, 2$.

Let $A = \emptyset$, $B = \{e_a\}$, and $e = e_b$. Adding e_b to A leaves e_a with dilation 3, so

$$f(A \cup \{e_b\}) - f(A) = 0.$$

After e_a is already retained, adding e_b reduces the maximum remaining dilation from 3 to 2, giving

$$f(B \cup \{e_b\}) - f(B) = 1.$$

Hence the marginal gain increases as the selected set grows, violating the diminishing-returns condition. ∎

B.3 Guarantee for Dilation-Only Selection

The following result applies only when candidate selection is based on dilation; it does not directly apply when congestion changes the ordering.

Theorem B.3 (Dilation-only returned-support bound) *Let $\mathcal{F}$ be a spanning-forest backbone of an unweighted graph, with $|\mathcal{E}_{\mathcal{F}}| \le q < m$, and let*

$$T = q - |\mathcal{E}_{\mathcal{F}}|.$$

Order the initial candidate dilations as $D_{(1)} \ge D_{(2)} \ge \cdots$ and define

$$S_{\mathcal{F}} = \sum_{e \in \mathcal{E} \setminus \mathcal{E}_{\mathcal{F}}} D_{\mathcal{F}}^{\mathrm{path}}(e).$$

If either (i) each insertion chooses a candidate of maximum current dilation, or (ii) the T candidates with largest initial dilation are selected, then

$$D(\mathcal{H}_T) \le D_{(T+1)} \le \frac{S_{\mathcal{F}}}{T+1}.$$

Proof. Adding retained edges cannot increase shortest-path dilation. Let $\theta = D_{(T+1)}$.

For static selection, every remaining candidate initially has dilation at most θ, and therefore also has final dilation at most θ.

For dynamic selection, suppose some edge remaining after T insertions has dilation greater than θ. Its dilation was then greater than θ at every earlier step. Since each step chooses a maximum-dilation candidate, all T selected edges also had current, and hence initial, dilation greater than θ. Together with the remaining edge, this gives $T + 1$ initial candidates with dilation greater than $D_{(T+1)}$, a contradiction. Thus,

$$S_{\mathcal{F}} \ge (T+1) D_{(T+1)},$$

which gives the second inequality. ∎

This result controls the largest remaining detour, but not maximum congestion, which motivates including congestion in the full `Scaffold` score.

B.4 Dilation and Edge Congestion Bound Structural Distortion

Theorem B.4 (Dilation–congestion support bound) *Let $\mathcal{H} \subseteq \mathcal{G}$ be an unweighted support preserving the connected components of $\mathcal{G}$. Then*

$$\boldsymbol{L}_{\mathcal{H}} \preceq \boldsymbol{L}_{\mathcal{G}} \preceq \tau_{\mathcal{H}} \boldsymbol{L}_{\mathcal{H}}, \qquad \tau_{\mathcal{H}} = 1 + D(\mathcal{H}) C_E(\mathcal{H}).$$

Proof. Since $\mathcal{H} \subseteq \mathcal{G}$, $\boldsymbol{L}_{\mathcal{H}} \preceq \boldsymbol{L}_{\mathcal{G}}$.

For an omitted edge $e = (u, v)$ with supporting path P_e, Cauchy–Schwarz gives

$$(x_u - x_v)^2 = \left(\sum_{a=(i,j)\in P_e} (x_i - x_j) \right)^2 \leq |P_e| \sum_{a=(i,j)\in P_e} (x_i - x_j)^2.$$

Because $|P_e| \leq D(\mathcal{H})$,

$$(x_u - x_v)^2 \leq D(\mathcal{H}) \sum_{a=(i,j)\in P_e} (x_i - x_j)^2.$$

Summing over all omitted edges, each retained edge occurs in at most $C_E(\mathcal{H})$ supporting paths. Therefore

$$\boldsymbol{x}^\top (\boldsymbol{L}_{\mathcal{G}} - \boldsymbol{L}_{\mathcal{H}}) \boldsymbol{x} \leq D(\mathcal{H}) C_E(\mathcal{H})\, \boldsymbol{x}^\top \boldsymbol{L}_{\mathcal{H}} \boldsymbol{x}.$$

Adding $\boldsymbol{x}^\top \boldsymbol{L}_{\mathcal{H}} \boldsymbol{x}$ to both sides proves the result. ∎

Weighted extension. For positive edge conductances w_e, let $r_e = 1/w_e$ and define

$$D_R(\mathcal{H}) = \max_{e=(u,v)\in\mathcal{I}} w_e \sum_{a\in P_e} \frac{1}{w_a}.$$

If $\mathcal{H}$ retains the original edge weights, the same argument with weighted Cauchy–Schwarz gives

$$\boldsymbol{L}_{\mathcal{H}} \preceq \boldsymbol{L}_{\mathcal{G}} \preceq \big(1 + D_R(\mathcal{H}) C_E(\mathcal{H})\big) \boldsymbol{L}_{\mathcal{H}}.$$

For unit weights, $D_R(\mathcal{H}) = D(\mathcal{H})$.

B.5 Effective-Resistance, Distance, and Congestion Consequences

Effective resistance. For connected $\mathcal{G}$ and $\mathcal{H}$, Theorem B.4 implies

$$R_{\mathcal{G}}(u, v) \leq R_{\mathcal{H}}(u, v) \leq \tau_{\mathcal{H}} R_{\mathcal{G}}(u, v) \qquad \forall\, u, v \in \mathcal{V}.$$

On $\mathbf{1}^\perp$ the Laplacians are positive definite, so inverting their ordering gives

$$\boldsymbol{L}_{\mathcal{G}}^+ \preceq \boldsymbol{L}_{\mathcal{H}}^+ \preceq \tau_{\mathcal{H}} \boldsymbol{L}_{\mathcal{G}}^+,$$

and the result follows from $R_{\mathcal{X}}(u, v) = (\boldsymbol{e}_u - \boldsymbol{e}_v)^\top \boldsymbol{L}_{\mathcal{X}}^+ (\boldsymbol{e}_u - \boldsymbol{e}_v)$.

Communication distance. For unweighted graphs,

$$\mathrm{dist}_{\mathcal{H}}(u, v) \leq \max\{1, D(\mathcal{H})\}\, \mathrm{dist}_{\mathcal{G}}(u, v).$$

Intuitively, take a shortest u–v path in $\mathcal{G}$ and replace every omitted edge by its supporting path in $\mathcal{H}$. Each replacement has length at most $D(\mathcal{H})$.

Edge-cut concentration. For any $S \subseteq \mathcal{V}$,

$$|\partial_{\mathcal{G}} S| \leq \big(1 + C_E(\mathcal{H})\big) |\partial_{\mathcal{H}} S|.$$

Every omitted edge crossing the cut has a supporting path that also crosses the cut. Assign it to one retained crossing edge on that path. Since each retained edge lies on at most $C_E(\mathcal{H})$ supporting paths, each retained cut edge receives at most $C_E(\mathcal{H})$ such assignments.

Node-neighborhood concentration. Let $N_{\mathcal{X}}(S) \setminus S$ denote the vertices outside S adjacent to a vertex in S in graph $\mathcal{X}$. Then

$$|N_{\mathcal{G}}(S) \setminus S| \leq \big(1 + C_V(\mathcal{H})\big) |N_{\mathcal{H}}(S) \setminus S|.$$

For each $\mathcal{G}$-neighbor of S that is not already an $\mathcal{H}$-neighbor, follow its omitted edge's supporting path and assign the vertex to the first vertex of that path outside S. This vertex is internal to the supporting path, and each such vertex receives at most $C_V(\mathcal{H})$ assignments.

Scope of the guarantees. These results concern the structure of the sparse support. The dilation-only bound does not apply to the full dilation–congestion selection rule, and we do not claim a general approximation ratio or monotonic improvement in downstream GNN accuracy.

B.6 Support Quality Under Edge Unions

Proposition B.5 (Monotonicity under edge addition) *Let $\mathcal{H} \subseteq \mathcal{H}' \subseteq \mathcal{G}$, with both supports preserving the connected components of $\mathcal{G}$ and retaining the original edge weights. Then*

$$\sigma(\boldsymbol{L}_{\mathcal{G}}, \boldsymbol{L}_{\mathcal{H}'}) \leq \sigma(\boldsymbol{L}_{\mathcal{G}}, \boldsymbol{L}_{\mathcal{H}}).$$

Proof. Since adding retained edges adds positive-semidefinite edge Laplacians,

$$\boldsymbol{L}_{\mathcal{H}} \preceq \boldsymbol{L}_{\mathcal{H}'}.$$

By definition of the support number,

$$\boldsymbol{L}_{\mathcal{G}} \preceq \sigma(\boldsymbol{L}_{\mathcal{G}}, \boldsymbol{L}_{\mathcal{H}})\, \boldsymbol{L}_{\mathcal{H}} \preceq \sigma(\boldsymbol{L}_{\mathcal{G}}, \boldsymbol{L}_{\mathcal{H}})\, \boldsymbol{L}_{\mathcal{H}'}.$$

Hence

$$\sigma(\boldsymbol{L}_{\mathcal{G}}, \boldsymbol{L}_{\mathcal{H}'}) \leq \sigma(\boldsymbol{L}_{\mathcal{G}}, \boldsymbol{L}_{\mathcal{H}}).$$

■

For the `Scaffold`-K inference union, $\mathcal{H}^{(k)} \subseteq \mathcal{H}_{\text{union}}$ for every retained support, so

$$\sigma(\boldsymbol{L}_{\mathcal{G}}, \boldsymbol{L}_{\mathcal{H}_{\text{union}}}) \leq \min_k \sigma(\boldsymbol{L}_{\mathcal{G}}, \boldsymbol{L}_{\mathcal{H}^{(k)}}).$$

Thus, taking the union cannot worsen the support-number approximation of any constituent support. This is a structural statement and does not imply monotonic improvement in GNN predictive performance.

C Method Details

This appendix provides additional details on the support backbones, scalable `Scaffold` variants, runtime, and GNN training/inference procedures described in Section 3.

C.1 Scaffold Algorithms, Backbones, and Variants

Let $q = \lceil \delta m \rceil$. For feasible budgets, each `Scaffold` variant starts from a spanning-forest backbone $\mathcal{F}$, with one spanning tree per connected component of $\mathcal{G}$. If the target lies below the spanning-forest floor, construction stops at q edges and guarantees requiring a spanning backbone are not invoked. For a current support $\mathcal{H}$ and candidate set $\mathcal{B}$, SupportScores$(\mathcal{G}, \mathcal{H}, \mathcal{B})$ computes supporting paths and evaluates Eq. 6, with congestion and normalization computed over $\mathcal{B}$.

C.1.1 Support Backbones

The backbone $\mathcal{F}$ may be deterministic, randomized, weight-aware, or low-stretch (Elkin et al., 2005; Abraham et al., 2008; Abraham & Neiman, 2012). Different backbone constructions induce different supporting paths and, consequently, different dilation and congestion profiles. All benchmark graphs in this work are unweighted; weighted extensions are described in Appendix A.3. The subsequent `Scaffold` scoring principle is otherwise unchanged.

We use spanning-forest terminology throughout: `SF` denotes the deterministic construction, `RandSF` a randomized forest, `SPF` a shortest-path forest, and `GLSF`/`LLSF` low-stretch constructions. `MaxSF` and `MinSF` denote maximum- and minimum-weight spanning forests, respectively. Under the unit weights used in our experiments, the two objectives are identical and, with the same deterministic tie-breaking, yield the same forest.

C.1.2 Variant Mechanisms

`Scaffold-Greedy` is the full dynamic reference and recomputes all candidate paths and scores after every insertion; its pseudocode is given in Algorithm 1 in the main paper. The remaining variants progressively reduce this recomputation, as summarized in Figure 14.

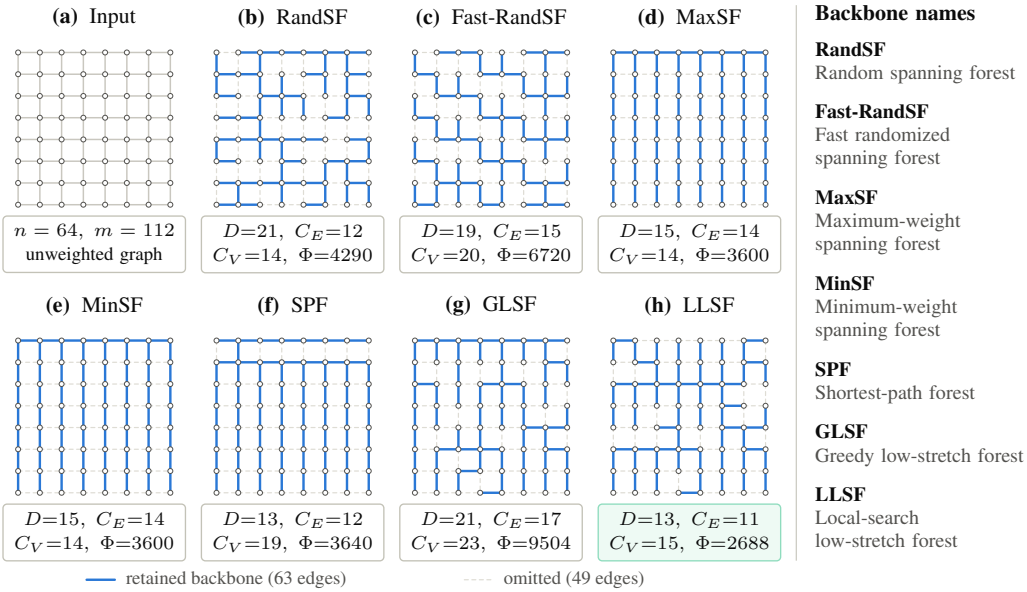

Figure 13: Support-forest backbones on a connected 8×8 grid. (a) Input; (b)–(h) spanning-forest constructions with 63 retained edges (blue) and 49 omitted edges (dashes). Since the input is connected, each forest contains one tree. Panels report dilation D, edge and node congestion C_E, C_V, and $\Phi = (1 + D)(1 + C_E)(1 + C_V)$ before edge additions. Green marks the lowest Φ. MaxSF and MinSF coincide under unit weights; LLSF refines GLSF through local swaps.

`Scaffold-Heap`. `Scaffold-Heap` caches candidate paths and priorities in a max-heap. At each round, a small active set is refreshed, the highest-scoring candidate is inserted, and only candidates affected by this insertion are reconsidered. Cached priorities outside this set may therefore be stale, making `Scaffold-Heap` a lazy approximation to Greedy.

Algorithm 2 `Scaffold-Heap`: selective dynamic refresh

Require: $\mathcal{G}, \delta$, active size κ, refresh cap c
Ensure: Sparse support $\mathcal{H}$
$q \leftarrow \lceil \delta m \rceil$; choose $\mathcal{F}$; $\mathcal{H} \leftarrow \mathcal{F}$
initialize cached paths/scores and max-heap Q
while $|\mathcal{E}_{\mathcal{H}}| < q$ **do**
  $\mathcal{A} \leftarrow$ PopCandidates(Q, κ)
  recompute paths and scores for $\mathcal{A}$ on $\mathcal{H}$
  $e^\star \leftarrow \arg\max_{e \in \mathcal{A}} s(e)$
  $\mathcal{H} \leftarrow \mathcal{H} \cup \{e^\star\}$
  refresh up to c affected candidates and update Q
end while
return $\mathcal{H}$

`Scaffold-Batch`. `Scaffold-Batch` evaluates only a random subset of the remaining candidates at each round. The sampled batch is rescored on the current support and its top-r edges are inserted before the next refresh. Depending on batch settings and selection number, we can make `Scaffold-Batch` scalable for large graphs.

`Scaffold-Fast`. `Scaffold-Fast` removes dynamic rescoring. Since $\mathcal{F}$ is a forest, each candidate whose endpoints lie in the same component has a unique path in $\mathcal{F}$. LCA queries, forest-difference accumulation, and root-prefix sums allow all initial path quantities to be computed together. The resulting ranking is then kept fixed.

ForestScores roots each tree in the forest, constructs the corresponding LCA structures, accumulates candidate-path edge/node loads, and uses prefix sums to recover the quantities in Eq. 6. These are the same initial scores computed by `Scaffold-Greedy` at $t = 0$; `Scaffold-Fast` differs by keeping them fixed during selection.

`Scaffold-Sample`. `Scaffold-Sample` converts support information from multiple randomized spanning forests into reusable sampling weights. Let f_e summarize how frequently edge e

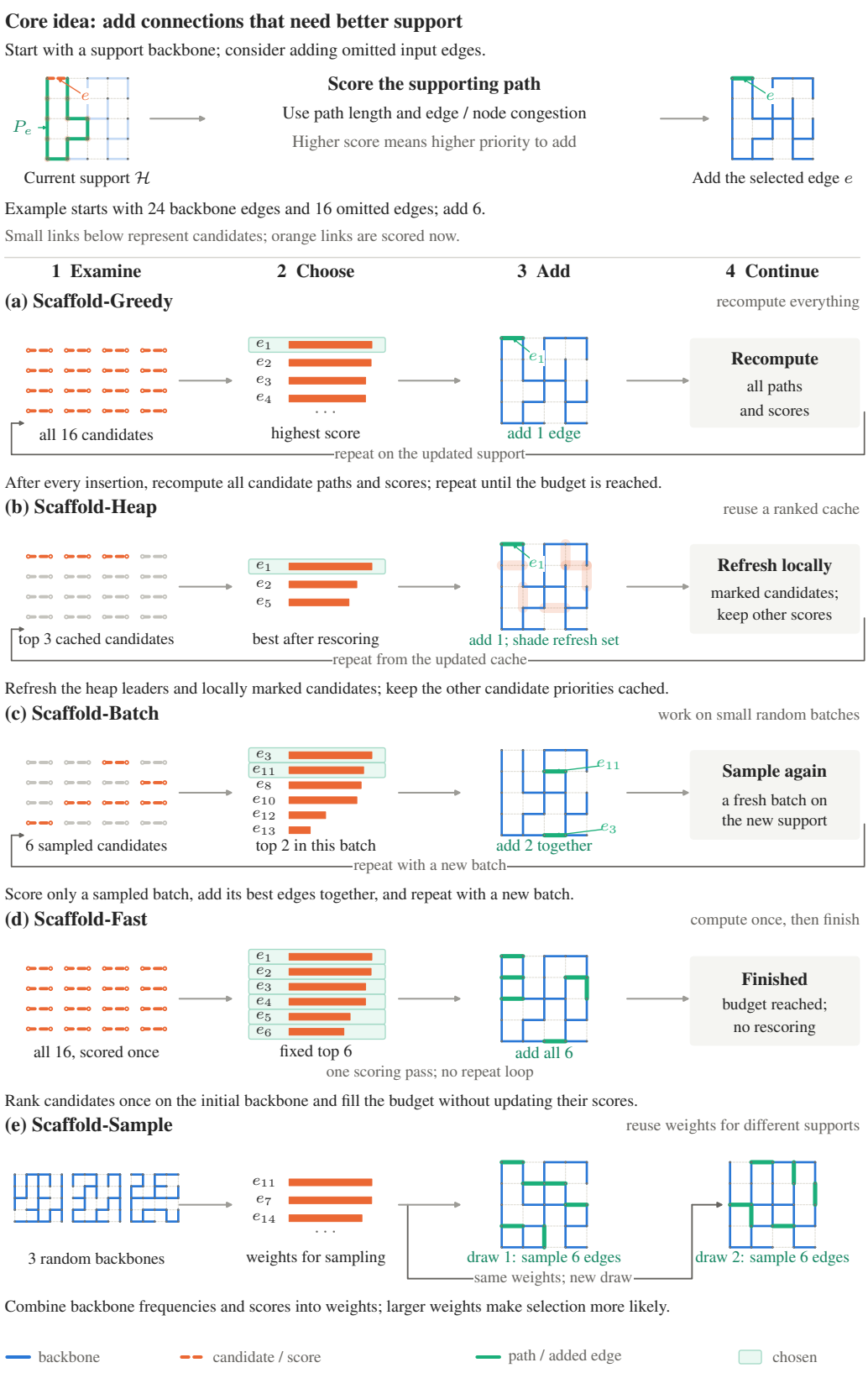

Figure 14: (a) `Greedy` recomputes all paths and scores after each insertion. (b) `Heap` caches priorities and selectively refreshes affected candidates. (c) `Batch` scores sampled batches and inserts their top-r edges. (d) `Fast` scores candidates once on the initial backbone. (e) `Sample` aggregates randomized-backbone signals for repeated sparse draws.

Algorithm 3 SCAFFOLD−BATCH: batched dynamic scoring

Require: $\mathcal{G}, \delta$, batch size b, insertion size r
Ensure: Sparse support $\mathcal{H}$
$q \leftarrow \lceil \delta m \rceil$; choose $\mathcal{F}$; $\mathcal{H} \leftarrow \mathcal{F}$
while $|\mathcal{E}_{\mathcal{H}}| < q$ **do**
  $\mathcal{I} \leftarrow \mathcal{E} \setminus \mathcal{E}_{\mathcal{H}}$
  sample $\mathcal{B} \subseteq \mathcal{I}$, $|\mathcal{B}| = \min\{b, |\mathcal{I}|\}$
  $\widehat{s} \leftarrow \textsc{SupportScores}(\mathcal{G}, \mathcal{H}, \mathcal{B})$
  $\mathcal{S} \leftarrow \textsc{TopK}(\mathcal{B}, \min\{r, q - |\mathcal{E}_{\mathcal{H}}|\}; \widehat{s})$
  $\mathcal{H} \leftarrow \mathcal{H} \cup \mathcal{S}$
end while
return $\mathcal{H}$

Algorithm 4 SCAFFOLD−FAST: static forest scoring

Require: $\mathcal{G}, \delta$
Ensure: Sparse support $\mathcal{H}$
$q \leftarrow \lceil \delta m \rceil$; choose backbone $\mathcal{F}$
$\mathcal{I} \leftarrow \mathcal{E} \setminus \mathcal{E}_{\mathcal{F}}$
$s_0 \leftarrow \textsc{ForestScores}(\mathcal{G}, \mathcal{F})$ {score once}
$\mathcal{S} \leftarrow \textsc{TopK}(\mathcal{I}, q - |\mathcal{E}_{\mathcal{F}}|; s_0)$
return $\mathcal{F} \cup \mathcal{S}$

occurs in the scoring backbones and a_e its normalized candidate-score signal. These quantities are precomputed once and combined into sampling weights π_e according to the policy specified in the experimental settings.

Algorithm 5 SCAFFOLD−SAMPLE: reusable stochastic supports

Require: $\mathcal{G}, \delta$, number of scoring backbones R
Ensure: Sparse support $\mathcal{H}$
$(f, a) \leftarrow \textsc{CachedForestSignals}(\mathcal{G}, R)$ {precompute once}
$q \leftarrow \lceil \delta m \rceil$; choose $\mathcal{F}$
$\mathcal{I} \leftarrow \mathcal{E} \setminus \mathcal{E}_{\mathcal{F}}$; $k \leftarrow q - |\mathcal{E}_{\mathcal{F}}|$
if $k = 0$ **then**
  return $\mathcal{F}$
end if
$\pi \leftarrow \textsc{SamplingWeights}(f, a; \mathcal{I})$
sample exactly k distinct edges $\mathcal{S} \subseteq \mathcal{I}$ using π
return $\mathcal{F} \cup \mathcal{S}$

C.1.3 Example Support Quality Across Scaffold Variants

Figure 15 compares the five `Scaffold` variants using the same `RandSF` backbone and edge budget. Greedy achieves the lowest dilation–congestion objective in this example, with `Heap` remaining close. `Batch`, `Fast`, and `Sample` use progressively cheaper selection policies and produce higher objective values here, illustrating how candidate selection affects support quality at fixed sparsity.

C.2 Scaffold-1 and Scaffold-K Training and Inference

Section 3.3 introduces the two usage modes, and Algorithm 6 summarizes their procedures. `Scaffold`-1 constructs a single q-edge support and uses it throughout training and inference. In contrast, `Scaffold`-K periodically generates new q-edge supports and maintains an online bank $\mathcal{B}$ containing at most the K strongest supports according to validation performance.

At inference, the retained supports are combined as $H_{\text{union}} = \bigcup_{H \in \mathcal{B}} H$ and evaluated in a single forward pass. Since every retained support is a subgraph of H_{union}, the union cannot worsen its support-number approximation relative to any constituent support (Proposition B.5). Realized union sizes are reported in Appendix F.5, while sensitivity to K, refresh frequency, asynchronous generation, and inference topology is examined in Appendix H.3–H.6.

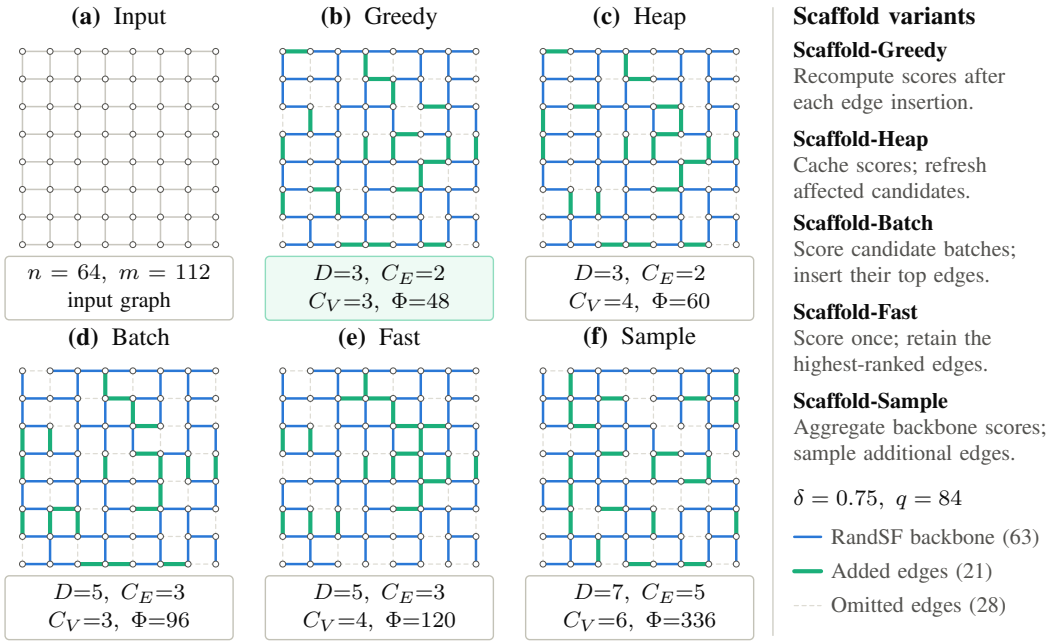

Figure 15: `Scaffold` variants at a fixed edge budget. (a) Input grid. (b)–(f) Each variant retains 84 edges ($\delta = 0.75$): a shared RandSF backbone (blue) and 21 added edges (green); dashed edges are omitted. Panels report dilation D, edge and node congestion C_E, C_V, and $\Phi = (1 + D)(1 + C_E)(1 + C_V)$. Green shading marks the lowest Φ in this example.

Algorithm 6 `Scaffold`-1 and `Scaffold`-K training and inference

(a) `Scaffold`-1

Require: $\mathcal{G}, \boldsymbol{X}, \boldsymbol{y}, \delta$, GNN f_θ
Ensure: Prediction $\widehat{\boldsymbol{y}}$
$\mathcal{H} \leftarrow \textsc{Scaffold}(\mathcal{G}, \delta)$ {q edges}
train f_θ on $\mathcal{H}$
select checkpoint $\theta^\star$
$\widehat{\boldsymbol{y}} \leftarrow f_{\theta^\star}(\boldsymbol{X}, \mathcal{H})$
return $\widehat{\boldsymbol{y}}$

(b) `Scaffold`-K

Require: $\mathcal{G}, \boldsymbol{X}, \boldsymbol{y}, \delta, \rho, K, T_{\text{train}}, f_\theta$
Ensure: Prediction $\widehat{\boldsymbol{y}}$
$\mathcal{B} \leftarrow \emptyset$ {at most K supports}
$J \leftarrow \lceil T_{\text{train}}/\rho \rceil$
for $j = 1, \ldots, J$ **do**
  $\mathcal{H}_j \leftarrow \textsc{Scaffold}(\mathcal{G}, \delta;$ random seed) {q edges}
  train on $\mathcal{H}_j$ for the next ρ epochs
  evaluate $\mathcal{H}_j$ and supports in $\mathcal{B}$ on validation data
  $\mathcal{B} \leftarrow \textsc{TopK}(\mathcal{B} \cup \{\mathcal{H}_j\}, K)$
end for
select checkpoint $\theta^\star$
re-rank supports in $\mathcal{B}$ using $\theta^\star$
$\mathcal{H}_{\text{union}} \leftarrow \bigcup_{\mathcal{H} \in \mathcal{B}} \mathcal{H}$
$\widehat{\boldsymbol{y}} \leftarrow f_{\theta^\star}(\boldsymbol{X}, \mathcal{H}_{\text{union}})$
return $\widehat{\boldsymbol{y}}$

C.3 Runtime and Scalability

For $q = \lceil \delta m \rceil$, `Scaffold` construction consists of a spanning-forest backbone $\mathcal{F}$ followed by selection of $T = q - |\mathcal{E}_{\mathcal{F}}|$ additional edges. The complexities in Table 1 describe the selection stage; total construction cost additionally includes backbone construction.

Measurement protocol. We benchmark seeded unweighted synthetic graphs on a dual AMD EPYC 7282 machine using up to eight CPU workers. Times are medians of three warm complete calls and include input normalization, fresh backbone construction, clustering where applicable, candidate scoring/selection, and sparse-output assembly; graph loading, JIT warm-up, and GNN training are excluded. `Sample` includes preprocessing over $R = 8$ randomized forests and its first draw; subsequent draws reuse the precomputed sampling weights. `Batch` uses ten fixed BFS-based candidate clusters formed by expanding from high-degree seed nodes. Each candidate edge is assigned to one cluster, while supporting paths are evaluated on the shared current support and may cross cluster

boundaries. `Batch` uses $b = 512$ and $r = 64$, while `Heap` uses $\kappa = 16$ and $c = 64$. To stress construction, these benchmarks enable all three score terms, $(\alpha, \beta_E, \beta_V) = (1, 1, 1)$. All budgets are above the spanning-forest floor, and outputs are checked for exact budget and connectivity.

Parallel scaling. Candidate clusters are fixed across worker counts. Workers parallelize supporting-path searches and forest-scoring operations, while dynamic support updates remain sequential. Table 4 reports complete construction time for $P \in \{1, 4, 8\}$ using the same graphs and algorithm settings as Table 1. Note that once `Sample` pre-processing is cached, subsequent exactly budgeted draws take 0.055 s on the 200K-node benchmark.

Table 4: Complete `Scaffold` construction time at $\delta = 0.2$.

Variant	Graph (n, m)	Time (s) P=1	P=4	P=8	Speedup
`Scaffold-Greedy`	(400, 3.1K)	1.63	1.01	0.78	2.08×
`Scaffold-Heap`	(4K, 39.9K)	25.09	25.02	25.16	1.00×
`Scaffold-Batch`	(10K, 250K)	206.12	57.54	31.01	6.65×
`Scaffold-Fast`	(200K, 2.4M)	3.53	2.78	2.30	1.54×
`Scaffold-Sample`	(200K, 2.4M)	33.77	11.81	9.02	3.74×

Times include fresh backbone construction; `Sample` includes preprocessing and its first draw. Speedup is T_1/T_8. Fixed randomness produces the same selected support across worker counts.

Backbone construction. Table 5 isolates backbone construction. `MaxSF`/`MinSF` use Kruskal-style union–find, `RandSF` variants randomize edge order, and `SPF` uses a BFS forest. `GLSF` and `LLSF` provide lower-stretch alternatives, with `LLSF` refining a `GLSF` initialization through local swaps. Under the unit weights used here, `MaxSF` and `MinSF` coincide given the same tie-breaking.

Table 5: Support-backbone construction time on unweighted graphs.

Backbone	Complexity	$n = 200$K, $m = 3.12$M (ms)	$n = 400$, $m = 1.54$K (ms)
`Fast-MaxSF/MinSF`	$\mathcal{O}(m\alpha_{\mathrm{UF}}(n) + B)$	22.72–23.74	0.035–0.036
`MaxSF/MinSF`	$\mathcal{O}(m \log m)$	22.25–22.41	0.035
`Fast-RandSF`	$\mathcal{O}(m\alpha_{\mathrm{UF}}(n))$	98.76	0.113
`RandSF`	$\mathcal{O}(m\alpha_{\mathrm{UF}}(n))$	162	0.234
`SPF`	$\mathcal{O}(m + n \log n)$	817	1.13
`GLSF`	$\mathcal{O}(nm\bar{c})$	—	12,313
`LLSF`	$C_{\mathrm{init}} + \mathcal{O}(Lm\bar{\ell}(n + m) \log n)$	—	392,810†

$B = 256$ and α_{UF} is the inverse Ackermann function; $\bar{c}$ and $\bar{\ell}$ denote the corresponding average construction factors. `LLSF` uses `GLSF` initialization with $L = 10$. †One completed `LLSF` run; other entries are three-run medians. Dashes denote unmeasured cases.

Selection complexity. Candidates sharing a source can reuse a shortest-path search. Let

$$C_{\mathrm{sp}} = \begin{cases} \mathcal{O}(n + q), & \text{BFS}, \\ \mathcal{O}((n + q) \log n), & \text{Dijkstra}. \end{cases}$$

The selection costs are

$$\begin{aligned} \texttt{Scaffold-Greedy}&: \mathcal{O}(TnC_{\mathrm{sp}}), \\ \texttt{Scaffold-Heap}&: \mathcal{O}((n + T(\kappa + c))C_{\mathrm{sp}} + m \log m), \\ \texttt{Scaffold-Batch}&: \mathcal{O}(\lceil T/r \rceil\, bC_{\mathrm{sp}}), \\ \texttt{Scaffold-Fast}&: \mathcal{O}((m + n) \log n), \\ \texttt{Scaffold-Sample}&: \mathcal{O}(R(m + n) \log n) \text{ preprocessing}, \quad \mathcal{O}(m) \text{ per cached draw}. \end{aligned}$$

For `Sample`, forest-derived weights are computed once and budget-specific sampling plans are cached for reuse. Subsequent draws require only fresh sampling and sparse-output assembly, without rebuilding forests or recomputing scores. Changing the budget or backbone rebuilds the sampling plan, while the precomputed forest signals remain reusable. The $\mathcal{O}(m)$ term corresponds to the linear sampling pass. On the 200K-node, 2.4M-edge benchmark with eight CPU workers, cached draws take 0.055 s, compared with 9.02 s for preprocessing and the first draw.

`Heap` and `Batch` reduce dynamic rescoring, while `Fast` and `Sample` avoid repeated full rescoring. Backbone construction is additional to these selection bounds and is included in the measured runtimes above. Detailed GNN training-time and GPU-memory measurements are reported in Appendix G.

C.4 Possible Extension: Localized or Partitioned Supports

The current `Scaffold` formulation uses a spanning-forest backbone so that every omitted edge has a finite supporting path whenever the requested budget is feasible. A possible extension is to apply the same scoring principle locally within graph partitions or clusters, constructing a connected support for each subgraph rather than one global spanning forest.

The main difficulty is the treatment of cross-partition edges. If the endpoints of an omitted edge are disconnected in the current support, its supporting-path dilation is infinite and the standard `Scaffold` score is not directly applicable. A localized variant would therefore require an additional mechanism for cross-partition connectivity, such as retaining a sparse global skeleton, explicitly preserving selected boundary edges, or scoring cross-partition edges separately.

We leave the design and analysis of such localized `Scaffold` constructions to future work.

D Additional Related Work

Graph sparsification methods differ in the information they use and the properties they aim to preserve. Topology methods range from simple degree and similarity heuristics (Hamann et al., 2016; Voudigari et al., 2016; Xu et al., 2007; Satuluri et al., 2011; Liu et al., 2023) to classical constructions with explicit structural objectives. Cut sparsification (Benczúr & Karger, 2015) preserves cuts, spectral sparsification (Spielman & Teng, 2011; Spielman & Srivastava, 2008; Batson et al., 2009) preserves Laplacian structure, and graph spanners (Althöfer et al., 1990) control distance distortion. Semantic methods instead use additional information to identify useful sparse structures. NeuralSparse (Zheng et al., 2020), SparseGAT (Ye & Ji, 2021), PTDNet (Luo et al., 2021), SGCN (Li et al., 2020), Unified-LTH (Chen et al., 2021), AdaGLT (Zhang et al., 2024), and SGS-GNN (Das et al., 2026) learn graph structure together with downstream training signals, while MoG (Zhang et al., 2025) learns task-dependent routing over multiple sparsifier experts.

Sampling methods address a related but different scalability problem. GraphSAGE (Hamilton et al., 2017) samples neighborhoods, FastGCN (Chen et al., 2018) and LADIES (Zou et al., 2019) sample layer-wise computation, GraphSAINT (Zeng et al., 2019) samples training subgraphs, and Cluster-GCN (Chiang et al., 2019) trains on graph partitions. These approaches primarily limit the computation performed in each training iteration rather than construct a single global sparse support. Other related graph-modification approaches include stochastic edge removal with DropEdge (Rong et al., 2019), diffusion-based preprocessing with GDC (Gasteiger et al., 2019), and learned graph structures such as ProGNN (Jin et al., 2020) and NodeFormer (Wu et al., 2022).

Recent work continues to explore GNN sparsification for efficient training and inference (Akkas & Azad, 2025; Liguori et al., 2025; Liao et al., 2024; Ding et al., 2025), alongside broader methods for analyzing and estimating graph sparsification error (Wang & Lopes, 2025). Graph/model co-sparsification and lottery-ticket methods (Chen et al., 2021; Wang et al., 2022; Hui et al., 2023; Zhang et al., 2024), learned graph structure and neural sparsification (Zheng et al., 2020; Li et al., 2020; Luo et al., 2021; Franceschi et al., 2019; Jin et al., 2020; Liu et al., 2022; Wu et al., 2022; Li et al., 2023), and classical spectral, cut, and distance-preserving constructions (Benczúr & Karger, 2015; Spielman & Teng, 2011; Spielman & Srivastava, 2008; Batson et al., 2009; Althöfer et al., 1990) further illustrate the breadth of graph-reduction approaches. Our experiments use representative topology, semantic, and sampling methods rather than attempting an exhaustive comparison across all graph-reduction paradigms.

`Scaffold` is a topology method, but differs in how it evaluates an omitted edge: its importance is determined by the path that must support that edge in the retained graph. With dilation alone, this is closely related to distance-preserving sparsification such as graph spanners. Jointly considering dilation and congestion instead follows the support-theoretic view that even short replacement paths can create severe bottlenecks when many omitted connections share the same retained edges. To our knowledge, prior GNN sparsifiers have not explicitly used this joint support-theoretic dilation–congestion criterion to construct the graph used directly for message passing. The different `Scaffold` variants realize this criterion with different quality-cost trade-offs.

E EXPERIMENTAL SETTINGS

E.1 DATASETS AND SHARED TRAINING SETTINGS

We evaluate on 19 node-classification datasets spanning homophilic, heterophilic, and large-scale regimes. All graphs are converted to undirected, self-loop-free graphs before sparsification. Table 6 reports dataset statistics and evaluation metrics.

Table 6: Dataset statistics for the 19 node-classification benchmarks.

	Graph scale			Structure	Task		
Dataset	$\lvert\mathcal{V}\rvert$	$\lvert\mathcal{E}\rvert$	d	$\mathcal{H}_{\mathrm{adj}}$	C	F	**Metric**
Heterophilic small/medium							
Roman-empire (Platonov et al., 2023b)	22.66K	32.93K	2.91	-0.05	18	300	Acc
Squirrel (Platonov et al., 2023b)	2.22K	47.00K	42.28	0.01	5	2.09K	Acc
Minesweeper (Platonov et al., 2023b)	10.00K	39.40K	7.88	0.01	2	7	AUC
Questions (Platonov et al., 2023b)	48.92K	153.54K	6.28	0.02	2	301	AUC
Chameleon (Platonov et al., 2023b)	890	8.85K	19.90	0.03	5	2.33K	Acc
Amazon-ratings (Platonov et al., 2023b)	24.49K	93.05K	7.60	0.14	5	300	Acc
Homophilic small/medium							
WikiCS (Mernyei & Cangea, 2020)	11.70K	215.60K	36.85	0.58	10	300	Acc
CiteSeer (Sen et al., 2008)	3.33K	4.55K	2.74	0.67	6	3.70K	Acc
Amazon-Computer (Shchur et al., 2018)	13.75K	245.86K	35.76	0.68	10	767	Acc
PubMed (Sen et al., 2008)	19.72K	44.32K	4.50	0.69	3	500	Acc
Cora (Sen et al., 2008)	2.71K	5.28K	3.90	0.77	7	1.43K	Acc
Coauthor-CS (Shchur et al., 2018)	18.33K	81.89K	8.93	0.78	15	6.81K	Acc
Amazon-Photo (Shchur et al., 2018)	7.65K	119.08K	31.13	0.79	8	745	Acc
Coauthor-Physics (Shchur et al., 2018)	34.49K	247.96K	14.38	0.87	5	8.42K	Acc
Large-scale							
ogbn-proteins (Hu et al., 2020)	132.53K	39.56M	597.00	0.05	112[†]	8	AUC
Pokec (Lim et al., 2021)	1.63M	22.30M	27.32	0.42	2	65	Acc
ogbn-arxiv (Hu et al., 2020)	169.34K	1.17M	13.77	0.42	40	128	Acc
ogbn-products (Hu et al., 2020)	2.45M	61.86M	50.52	0.46	47	100	Acc
Reddit (Hamilton et al., 2017)	232.97K	57.31M	492.00	0.74	41	602	Acc

Note. $\lvert\mathcal{V}\rvert$ and $\lvert\mathcal{E}\rvert$ are the numbers of nodes and undirected edges; d is average degree; $\mathcal{H}_{\mathrm{adj}}$ is adjusted homophily (Platonov et al., 2023a); C and F are the number of classes (or tasks) and feature dimension. Small/medium graphs use $\delta \in \{0.3, 0.5, 0.7\}$ and large-scale graphs use $\delta \in \{0.1, 0.3, 0.5\}$. Graphs are made undirected and self-loop-free before sparsification. [†]OGBN-PROTEINS has 112 binary prediction tasks.

E.2 BASELINE TRAINING SETTINGS

Table 7 gives the per-dataset TunedGNN (Luo et al., 2024) configurations shared across methods.

Budget matching and evaluation protocol. All ranked fixed-budget sparsifiers use the same target $q = \lceil \delta m \rceil$ and, whenever applicable, the same data split, GNN architecture, optimization settings, epoch budget, and model-selection rule from Table 7; only the graph support changes. Semantic methods retain their native optimization procedures but are matched to δ. `Scaffold`-1 uses one q-edge support throughout training and inference and is included in the matched-budget ranking. `Scaffold`-K trains on q edges at a time but refreshes supports and infers on their validation-selected union, which may exceed q; it is therefore reported separately. Test labels are never used for support/checkpoint selection.

Large-graph adaptations. Unified-LTH and AdaGLT use sparse-neighborhood minibatching on large graphs. MoG uses random-node partitions on large graphs, with train/evaluation partition counts of $1/1$ for OGBN-ARXIV, $10/5$ for OGBN-PROTEINS, $3/3$ for POKEC, and $10/10$ for REDDIT and OGBN-PRODUCTS.

Sampling settings. We use GraphSAINT random-walk sampling with a TunedGNN backbone and settings. Small/medium graphs use batch size 500, walk length 2, 3 steps per epoch, and coverage 100 with normalization. Large graphs use walk length 4 and 30 steps per epoch, with batch size 6000 (20,000 for OGBN-PRODUCTS and POKEC). For Tuned-GraphSAGE, we follow the same settings specified by the authors in Luo et al. (2024).

Table 7: Per-dataset GCN training configurations shared by all methods.

		Protocol		Architecture			Optimization			Evaluation
	Dataset	**Epochs**	**Runs**	L	H	**Norm./Res.**	**Dropout**	**LR**	**WD**	**Metric**
Heterophilic	Roman-empire	2500	3	9	512	bn, res*	0.50	1e-3	0	Acc
	Squirrel	500	10	4	256	bn, res	0.70	1e-2	5e-4	Acc
	Minesweeper	2000	3	12	64	bn, res	0.20	1e-2	0	AUC
	Questions	1500	3	10	512	res*	0.30	3e-5	0	AUC
	Chameleon	200	10	5	512	–	0.20	5e-3	1e-3	Acc
	Amazon-ratings	2500	3	4	512	bn, res	0.50	1e-3	0	Acc
Homophilic	WikiCS	1000	3	3	256	ln	0.50	1e-3	0	Acc
	CiteSeer	500	5	2	512	–	0.50	1e-3	1e-2	Acc
	Amazon-Computer	1000	3	3	512	ln	0.50	1e-3	5e-5	Acc
	PubMed	500	5	2	256	–	0.70	5e-3	5e-4	Acc
	Cora	500	5	3	512	–	0.70	1e-3	5e-4	Acc
	Coauthor-CS	1500	3	2	512	ln, res	0.30	1e-3	5e-4	Acc
	Amazon-Photo	1000	3	6	256	ln, res	0.50	1e-3	5e-5	Acc
	Coauthor-Physics	1500	3	2	64	ln, res	0.30	1e-3	5e-4	Acc
Large-scale	ogbn-proteins	100	2	3	80	–	0.25	1e-2	0	AUC
	Pokec	2000	2	7	256	bn, res	0.20	5e-4	0	Acc
	ogbn-arxiv	2000	2	5	512	bn, res	0.50	5e-4	5e-4	Acc
	ogbn-products	300	2	5	200	ln	0.50	3e-3	0	Acc
	Reddit	300	2	5	200	ln	0.50	3e-3	0	Acc

Note. * A linear input projection precedes the backbone.

Table 8: Training configurations of evaluated baselines. T is the dataset-specific epoch budget in Table 7; δ is the retained-edge ratio.

	Method(s)	**GNN / budget**	**Training and evaluation**
Anchors	No Graph	MLP; T	Feature-only prediction.
	Spanning Forest	Shared GNN; T	One spanning-forest support for training and inference.
	Full Graph	TunedGNN; T	Original graph without sparsification.
Topology	Random, Local Degree, Rank Degree, Forest Fire, SCAN	Shared GNN; δ; T	One support per run and budget, reused throughout training.
	L-Spar, G-Spar, L-Sim, Spectral (ER), t-Spanner, DSpar	Shared GNN; δ; T	Each sparsifier is evaluated independently using the shared training preset.
Semantic	Unified-LTH	Shared GNN preset	Two 200-epoch mask-search rounds followed by T ticket-training epochs; graph retention δ, no weight pruning.
	AdaGLT	Shared GNN preset	400 mask-search epochs followed by T retraining epochs; adjacency sparsity $1-\delta$, no weight pruning.
	MoG	Shared GNN; δ; T	Three sparsifier experts at retention $[\delta, \delta, \delta]$, trained jointly with the downstream GNN.
Sampling	Tuned-GraphSAGE	Tuned-GraphSAGE; T	Dataset-specific neighborhood sampling on the original graph.
	Tuned-GraphSAINT	Shared GNN; T	Random-walk subgraph sampling with dataset-specific settings.
Scaffold	Scaffold-Greedy	Shared GNN; δ; T	Full dynamic rescoring after every insertion.
	Scaffold-Heap	Shared GNN; δ; T	Cached priorities with selective refresh.
	Scaffold-Batch	Shared GNN; δ; T	Scores sampled batches and inserts top-r edges.
	Scaffold-Fast	Shared GNN; δ; T	Scores once on the initial spanning-forest backbone.
	Scaffold-Sample	Shared GNN; δ; T	Reuses precomputed weights to draw exactly budgeted supports.
	Scaffold settings	$(\alpha, \beta_E, \beta_V) = (1, 1, 0)$	Refreshed supports may be generated asynchronously.
	Scaffold-1	Shared GNN; δ; T	One q-edge support for training and inference.
	Scaffold-K	Shared GNN; δ per support; T	Refreshes every ρ epochs, retains top-K validation supports ($K \in \{3, 5, 10\}$), and infers once on their union.

F Detailed Experimental Results

F.1 Results on Homophilic Graphs

Table 9 shows detailed numerical comparison of `Scaffold` with baselines $\delta \in \{0.3, 0.5, 0.7\}$.

F.2 Results on Heterophilic Graphs

Table 10 shows detailed numerical comparison of `Scaffold` with baselines $\delta \in \{0.3, 0.5, 0.7\}$.

F.3 Results on Large-scale Graphs

Table 11 shows detailed numerical comparison of `Scaffold` with baselines $\delta \in \{0.1, 0.3, 0.5\}$.

Table 9: Homophilic graphs: complete mean ± standard deviation results (%) over eight datasets.

	Method	Cora	CiteSeer	PubMed	Amazon Computer	Amazon Photo	Coauthor CS	Coauthor Physics	WikiCS	GM
Retention ratio $\delta = 0.3$	† below the spanning-forest floor here: Cora, CiteSeer, PubMed									
Semantic	Unified-LTH	72.12 ±3.47	63.76 ±1.19	75.26 ±0.67	86.88 ±0.74	91.94 ±0.36	94.99 ±0.37	96.49 ±0.00	75.49 ±0.17	81.32
Semantic	AdaGLT	72.74 ±0.63	65.26 ±1.00	75.84 ±0.56	91.53 ±0.42	94.16 ±0.19	**95.13** ±0.08	96.65 ±0.00	76.48 ±0.13	82.67
Semantic	MoG	73.66 ±0.77	65.28 ±1.50	71.56 ±0.54	92.55 ±0.05	95.45 ±0.29	94.94 ±0.20	96.93 ±0.10	77.48 ±0.66	82.60
Topology	Random	73.40 ±0.51	65.86 ±0.95	74.26 ±0.62	91.05 ±0.15	94.38 ±0.16	94.71 ±0.04	96.66 ±0.05	77.04 ±0.63	82.64
Topology	Local Degree	73.52 ±0.47	64.76 ±0.84	77.32 ±1.08	89.28 ±0.33	94.49 ±0.03	94.93 ±0.22	96.30 ±0.18	75.69 ±0.48	82.51
Topology	Rank Degree	**76.34** ±0.78	65.10 ±0.24	76.38 ±0.46	88.04 ±0.28	93.79 ±0.05	94.68 ±0.06	96.40 ±0.17	76.13 ±0.53	82.65
Topology	Forest Fire	71.98 ±0.39	63.82 ±1.05	73.46 ±0.85	92.57 ±0.12	95.40 ±0.22	94.67 ±0.33	96.91 ±0.03	79.20 ±0.58	82.59
Topology	SCAN	71.38 ±0.67	64.54 ±0.37	74.32 ±0.50	91.60 ±0.18	94.01 ±0.22	95.09 ±0.02	96.84 ±0.03	75.08 ±0.44	81.97
Topology	L-Spar	71.38 ±1.48	64.32 ±0.57	73.48 ±0.90	90.95 ±0.22	94.07 ±0.20	95.03 ±0.22	96.74 ±0.05	75.10 ±0.25	81.74
Topology	G-Spar	72.02 ±1.05	64.28 ±0.57	73.48 ±0.90	91.02 ±0.21	93.42 ±0.37	94.78 ±0.08	96.73 ±0.06	74.91 ±0.39	81.71
Topology	L-Sim	75.18 ±0.72	66.06 ±0.42	76.98 ±0.74	93.04 ±0.07	94.88 ±0.13	95.02 ±0.29	96.83 ±0.06	78.71 ±0.96	**83.85**
Topology	Spectral	72.86 ±0.85	**66.82** ±0.44	75.04 ±1.43	92.13 ±0.11	95.24 ±0.29	94.68 ±0.13	96.74 ±0.08	78.80 ±0.47	83.28
Topology	t-Spanner	72.60 ±0.90	65.70 ±0.63	74.74 ±0.59	91.51 ±0.11	94.95 ±0.24	94.64 ±0.27	96.75 ±0.10	78.07 ±0.23	82.82
Topology	DSpar	73.90 ±0.36	64.70 ±1.22	77.86 ±0.46	**93.17** ±0.08	**95.62** ±0.33	94.52 ±0.12	96.84 ±0.05	79.47 ±0.68	83.72
Scaf.-1	**Scaffold-Batch-1**	75.62 ±0.61	65.76 ±0.89	78.02 ±0.79	91.43 ±0.42	95.10 ±0.17	94.08 ±0.30	96.67 ±0.07	78.14 ±0.53	83.65
Scaf.-1	**Scaffold-Fast-1**	75.62 ±0.61	65.76 ±0.89	**78.02** ±0.79	91.69 ±0.11	94.92 ±0.17	94.37 ±0.23	96.82 ±0.09	78.26 ±0.50	83.72
Scaf.-1	**Scaffold-Sample-1**	72.08 ±1.26	65.68 ±0.63	76.30 ±1.23	93.06 ±0.11	95.14 ±0.23	94.72 ±0.20	**97.09** ±0.06	**79.48** ±0.32	83.39
Scaf.-K	**Scaffold-Batch-K**	**83.36** ±1.37	**71.12** ±0.79	76.30 ±0.64	92.64 ±0.14	**95.97** ±0.24	95.65 ±0.23	96.97 ±0.16	78.54 ±0.36	85.77
Scaf.-K	**Scaffold-Fast-K**	82.46 ±1.25	71.06 ±0.77	**76.70** ±1.17	**93.66** ±0.09	95.82 ±0.38	**96.00** ±0.15	96.90 ±0.04	**80.16** ±0.45	**86.06**
Scaf.-K	**Scaffold-Sample-K**	74.82 ±0.76	64.76 ±1.72	75.38 ±1.65	93.57 ±0.18	95.57 ±0.27	95.78 ±0.22	**97.22** ±0.14	79.97 ±0.63	83.80
Retention ratio $\delta = 0.5$	† below the spanning-forest floor here: CiteSeer									
Semantic	Unified-LTH	78.76 ±1.94	68.22 ±1.29	76.70 ±1.05	87.60 ±0.53	93.33 ±0.30	94.96 ±0.21	96.74 ±0.00	76.90 ±0.67	83.58
Semantic	AdaGLT	76.98 ±0.55	68.54 ±0.98	77.22 ±1.10	91.87 ±0.48	94.71 ±0.05	95.05 ±0.18	96.81 ±0.00	77.45 ±0.14	84.20
Semantic	MoG	77.60 ±1.20	67.16 ±0.73	74.12 ±1.51	93.15 ±0.12	95.47 ±0.08	95.46 ±0.16	**97.29** ±0.04	78.60 ±1.11	84.12
Topology	Random	78.44 ±0.91	69.04 ±0.45	75.60 ±0.88	92.67 ±0.25	95.27 ±0.29	95.32 ±0.15	97.04 ±0.08	77.94 ±0.72	84.53
Topology	Local Degree	81.26 ±0.54	68.34 ±0.86	80.44 ±0.30	90.31 ±0.35	94.62 ±0.34	95.01 ±0.04	96.79 ±0.04	77.25 ±0.26	84.95
Topology	Rank Degree	**81.34** ±1.13	69.84 ±0.81	79.66 ±0.70	88.96 ±0.16	94.23 ±0.11	94.98 ±0.21	96.73 ±0.15	76.44 ±0.11	84.76
Topology	Forest Fire	76.84 ±0.42	69.50 ±0.75	75.50 ±0.97	93.23 ±0.03	95.40 ±0.32	95.22 ±0.24	97.07 ±0.10	79.31 ±0.53	84.62
Topology	SCAN	78.44 ±0.25	68.82 ±0.50	77.34 ±0.51	92.33 ±0.17	94.38 ±0.42	95.24 ±0.12	97.10 ±0.05	75.13 ±0.43	84.21
Topology	L-Spar	74.92 ±0.65	66.42 ±0.78	75.62 ±1.07	92.08 ±0.08	94.27 ±0.30	95.15 ±0.12	96.87 ±0.06	75.00 ±0.29	83.03
Topology	G-Spar	74.88 ±0.79	66.28 ±0.86	75.62 ±1.07	92.35 ±0.07	94.53 ±0.24	95.21 ±0.17	96.82 ±0.07	75.23 ±0.12	83.09
Topology	L-Sim	79.98 ±0.42	**70.10** ±0.64	79.36 ±1.39	93.29 ±0.10	95.29 ±0.18	95.05 ±0.13	97.02 ±0.08	78.93 ±0.54	85.59
Topology	Spectral	79.90 ±0.73	69.32 ±0.55	78.86 ±0.99	92.90 ±0.12	**95.90** ±0.13	95.33 ±0.15	96.97 ±0.07	79.42 ±0.31	85.50
Topology	t-Spanner	78.82 ±0.87	67.24 ±0.45	77.48 ±0.17	**93.55** ±0.09	95.36 ±0.37	94.87 ±0.06	97.14 ±0.04	79.33 ±0.44	84.82
Topology	DSpar	77.94 ±0.42	68.18 ±0.82	79.10 ±1.09	93.32 ±0.17	95.53 ±0.03	95.49 ±0.24	97.20 ±0.09	**79.97** ±0.36	85.22
Scaf.-1	**Scaffold-Batch-1**	81.06 ±0.66	68.36 ±0.60	**80.72** ±1.09	91.72 ±0.33	95.19 ±0.27	95.28 ±0.31	96.82 ±0.06	78.59 ±0.70	85.41
Scaf.-1	**Scaffold-Fast-1**	81.22 ±0.86	68.36 ±0.60	79.34 ±0.99	91.86 ±0.31	95.21 ±0.36	95.39 ±0.24	96.77 ±0.03	78.68 ±0.56	85.28
Scaf.-1	**Scaffold-Sample-1**	80.72 ±0.44	68.62 ±0.82	79.36 ±0.86	93.32 ±0.24	95.62 ±0.26	**95.57** ±0.16	97.16 ±0.08	79.94 ±0.74	**85.71**
Scaf.-K	**Scaffold-Batch-K**	82.42 ±1.57	71.70 ±0.39	79.10 ±1.02	93.24 ±0.29	95.82 ±0.23	95.94 ±0.14	97.21 ±0.07	78.94 ±0.42	86.29
Scaf.-K	**Scaffold-Fast-K**	**82.72** ±0.86	**71.72** ±0.50	**80.72** ±0.70	93.24 ±0.06	95.58 ±0.14	**96.04** ±0.06	97.28 ±0.08	79.19 ±0.42	**86.58**
Scaf.-K	**Scaffold-Sample-K**	82.04 ±1.51	68.76 ±1.08	79.98 ±0.99	**93.68** ±0.10	**95.86** ±0.27	95.86 ±0.19	**97.41** ±0.06	**80.14** ±0.82	86.14
Retention ratio $\delta = 0.7$										
Semantic	Unified-LTH	79.72 ±2.26	69.18 ±0.95	77.00 ±1.77	91.08 ±0.07	93.97 ±0.32	95.26 ±0.14	96.67 ±0.00	77.74 ±0.60	84.51
Semantic	AdaGLT	80.20 ±0.95	69.46 ±0.43	77.94 ±1.72	92.32 ±0.21	94.95 ±0.29	95.53 ±0.10	97.01 ±0.00	78.40 ±0.32	85.16
Semantic	MoG	81.50 ±0.81	70.14 ±0.99	76.72 ±1.62	93.55 ±0.14	95.64 ±0.08	95.56 ±0.10	97.29 ±0.09	79.53 ±0.62	85.67
Topology	Random	81.10 ±0.72	70.16 ±0.53	76.82 ±0.55	93.30 ±0.23	95.84 ±0.25	95.57 ±0.27	97.21 ±0.05	78.57 ±0.54	85.49
Topology	Local Degree	82.74 ±0.68	70.20 ±0.55	80.56 ±0.83	91.48 ±0.13	94.75 ±0.29	95.48 ±0.01	97.04 ±0.07	78.38 ±0.12	85.83
Topology	Rank Degree	82.46 ±0.57	70.40 ±0.45	80.18 ±0.48	90.63 ±0.25	94.92 ±0.20	95.63 ±0.08	96.77 ±0.04	77.26 ±0.32	85.53
Topology	Forest Fire	80.60 ±0.40	**71.50** ±0.48	78.52 ±0.99	93.29 ±0.36	95.64 ±0.06	95.55 ±0.11	97.29 ±0.08	79.54 ±0.66	85.98
Topology	SCAN	81.82 ±0.63	70.08 ±0.24	79.02 ±0.63	93.19 ±0.11	95.16 ±0.11	95.60 ±0.19	97.21 ±0.08	77.50 ±0.39	85.64
Topology	L-Spar	78.02 ±0.67	69.64 ±0.79	77.90 ±1.08	92.58 ±0.22	95.10 ±0.21	95.70 ±0.10	97.10 ±0.04	77.15 ±0.35	84.79
Topology	G-Spar	77.92 ±0.74	69.66 ±0.77	77.90 ±1.08	92.34 ±0.05	94.86 ±0.43	95.47 ±0.04	97.01 ±0.09	77.21 ±0.24	84.70
Topology	L-Sim	80.42 ±0.35	70.56 ±1.18	80.42 ±0.26	93.58 ±0.22	95.51 ±0.11	95.59 ±0.12	97.24 ±0.04	79.45 ±0.24	86.07
Topology	Spectral	82.78 ±0.72	70.34 ±0.82	79.04 ±0.41	93.41 ±0.17	**96.33** ±0.32	95.46 ±0.13	97.11 ±0.05	79.51 ±0.34	86.21
Topology	t-Spanner	80.56 ±0.27	70.72 ±0.61	78.20 ±0.58	93.51 ±0.18	95.84 ±0.32	95.31 ±0.09	97.27 ±0.04	79.75 ±0.55	85.85
Topology	DSpar	81.20 ±0.86	70.44 ±0.37	**81.00** ±0.59	**94.07** ±0.08	95.73 ±0.13	95.76 ±0.14	97.37 ±0.11	**80.31** ±0.49	86.47
Scaf.-1	**Scaffold-Batch-1**	**83.80** ±0.92	70.94 ±0.91	79.68 ±0.48	92.27 ±0.06	95.40 ±0.26	**95.82** ±0.16	97.04 ±0.08	79.00 ±0.39	86.25
Scaf.-1	**Scaffold-Fast-1**	83.66 ±0.68	70.18 ±0.83	79.82 ±1.37	92.10 ±0.17	95.69 ±0.36	95.59 ±0.24	96.98 ±0.03	79.02 ±0.38	86.12
Scaf.-1	**Scaffold-Sample-1**	83.72 ±0.63	70.50 ±0.63	80.34 ±0.92	93.49 ±0.41	95.49 ±0.41	95.67 ±0.14	**97.37** ±0.07	79.85 ±0.36	**86.55**
Scaf.-K	**Scaffold-Batch-K**	83.78 ±0.79	72.12 ±0.37	**80.60** ±0.60	93.19 ±0.31	95.88 ±0.24	95.62 ±0.27	97.34 ±0.15	79.57 ±0.53	86.80
Scaf.-K	**Scaffold-Fast-K**	**84.24** ±0.91	**72.22** ±0.64	78.98 ±1.05	93.43 ±0.30	**95.99** ±0.31	**95.92** ±0.25	97.41 ±0.08	79.69 ±0.51	86.75
Scaf.-K	**Scaffold-Sample-K**	84.20 ±0.79	71.70 ±0.47	80.20 ±1.21	**93.53** ±0.19	95.80 ±0.49	95.89 ±0.16	**97.46** ±0.06	**80.28** ±0.58	**86.91**
Ratio-independent comparisons										
	No graph (MLP)	59.64 ±1.31	57.12 ±0.55	70.86 ±1.44	87.04 ±0.36	92.09 ±0.51	95.10 ±0.11	96.31 ±0.02	73.48 ±0.25	77.51
	Spanning Forest	80.10 ±0.26	69.42 ±0.56	79.48 ±0.46	90.45 ±0.25	93.97 ±0.08	95.47 ±0.19	96.90 ±0.02	76.65 ±0.22	84.76
	Full graph (T-GCN)	84.56 ±0.63	72.30 ±0.47	81.22 ±1.11	93.80 ±0.20	95.95 ±0.33	95.72 ±0.04	**97.50** ±0.12	80.27 ±0.44	87.22
	SCAFFOLD-GCN*	**84.60** ±0.52 (0.9)	**72.46** ±0.48 (0.8)	**81.54** ±0.40 (0.9)	**93.95** ±0.10 (0.7)	**96.14** ±0.14 (0.7)	**96.02** ±0.04 (0.9)	97.43 ±0.02 (0.5)	**80.38** ±0.41 (0.4)	**87.37**

Green, blue, and amber mark the top three *matched-budget* methods; Scaf.-1 is included (one fixed support at δ), while Scaf.-K is excluded because its union exceeds the budget and is shown below the rule with its best variant bolded. The ratio-independent block is ranked separately. GM is the geometric mean. †Ratio below the dataset's spanning-forest floor ($(|\mathcal{V}| - c)/|\mathcal{E}|$: no connected support exists. ‡Support rebuilt asynchronously. *Per-dataset max over $\delta \in [0.1, 0.9]$; parentheses give the lowest δ attaining it.

Table 10: Heterophilic graphs: complete mean ± standard deviation results (%) over six datasets.

	Method	Squirrel	Chameleon	Roman empire	Amazon ratings	Minesweeper	Questions	GM
Retention ratio $\delta = 0.3$	†below the spanning-forest floor here: Roman empire, Questions							
Semantic	Unified-LTH	38.17 ±2.09	37.61 ±3.67	65.43 ±0.58	40.49 ±0.52	69.70 ±1.12	72.42 ±0.64	51.75
Semantic	AdaGLT	42.30 ±3.91	38.03 ±4.12	**81.43** ±0.00	48.05 ±0.35	62.60 ±1.27	72.34 ±0.33	55.27
Semantic	MoG	43.25 ±1.12	43.40 ±3.13	70.44 ±0.60	51.24 ±0.33	74.02 ±0.32	68.90 ±1.68	57.07
Topology	Random	40.90 ±1.70	37.85 ±3.35	67.07 ±0.24	48.40 ±0.43	72.39 ±1.04	71.89 ±0.90	54.48
Topology	Local Degree	41.80 ±1.66	41.37 ±2.59	70.79 ±0.31	51.47 ±0.71	72.53 ±1.15	74.39 ±0.83	56.92
Topology	Rank Degree	40.17 ±1.89	40.35 ±3.82	70.53 ±0.33	51.23 ±0.52	71.09 ±0.21	71.49 ±0.48	55.67
Topology	Forest Fire	43.15 ±1.90	41.28 ±1.96	67.43 ±0.62	48.37 ±0.22	70.76 ±0.29	71.34 ±0.46	55.53
Topology	SCAN	39.50 ±1.10	36.05 ±3.41	73.37 ±0.23	50.25 ±0.61	78.53 ±0.35	72.72 ±0.45	55.74
Topology	L-Spar	39.17 ±1.31	36.11 ±3.06	72.24 ±0.66	49.84 ±0.19	78.45 ±0.86	70.29 ±1.82	55.13
Topology	G-Spar	39.77 ±1.57	38.62 ±4.13	72.20 ±0.42	49.64 ±0.61	**78.97** ±0.51	70.35 ±1.77	55.92
Topology	L-Sim	39.23 ±2.38	39.71 ±3.97	71.33 ±0.21	**51.66** ±0.09	77.17 ±0.41	**75.39** ±0.60	56.75
Topology	Spectral	43.94 ±1.18	42.98 ±3.60	66.72 ±0.42	46.86 ±0.41	70.48 ±1.94	74.00 ±0.88	55.99
Topology	t-Spanner	42.65 ±2.01	38.71 ±3.72	67.09 ±0.45	47.96 ±1.01	71.13 ±1.03	72.42 ±0.61	54.89
Topology	DSpar	43.48 ±1.56	39.73 ±3.87	67.07 ±0.09	47.71 ±0.61	72.56 ±2.00	72.77 ±0.47	55.49
Scaf.-1	**Scaffold-Batch-1**	42.74 ±1.42	**43.88** ±5.13	68.89 ±0.68	51.49 ±0.70	72.39 ±0.74	75.06 ±0.32	57.50
Scaf.-1	**Scaffold-Fast-1**	43.35 ±2.22	43.63 ±3.65	68.89 ±0.68	51.24 ±0.62	72.54 ±0.90	75.06 ±0.36	**57.56**
Scaf.-1	**Scaffold-Sample-1**	**44.62** ±1.67	42.50 ±4.38	68.97 ±0.43	50.74 ±0.35	71.67 ±0.30	74.10 ±0.54	57.26
Scaf.-K	**Scaffold-Batch-K**	42.13 ±1.55	43.50 ±3.94	**74.74** ±0.40	53.50 ±0.19	86.07 ±1.30	77.74 ±0.50	60.50
Scaf.-K	**Scaffold-Fast-K**	**43.99** ±2.12	43.11 ±3.01	74.70 ±0.23	**53.61** ±0.47	**88.54** ±0.75	**77.88** ±0.42	**61.17**
Scaf.-K	**Scaffold-Sample-K**	43.99 ±1.59	**43.89** ±2.40	66.87 ±0.82	51.82 ±0.29	77.02 ±2.03	74.25 ±0.34	58.05
Retention ratio $\delta = 0.5$	†below the spanning-forest floor here: Roman empire							
Semantic	Unified-LTH	37.07 ±2.23	36.39 ±3.69	66.85 ±0.81	43.86 ±0.57	74.75 ±0.67	74.89 ±0.84	52.99
Semantic	AdaGLT	43.52 ±2.31	36.42 ±3.77	**88.95** ±0.00	50.25 ±0.15	70.97 ±1.71	73.87 ±0.73	57.76
Semantic	MoG	44.30 ±1.80	44.20 ±3.97	74.14 ±0.20	**52.98** ±0.22	82.17 ±0.45	73.13 ±0.14	59.91
Topology	Random	42.50 ±2.43	41.20 ±3.59	70.93 ±0.25	51.22 ±0.29	79.00 ±0.31	73.41 ±0.40	57.70
Topology	Local Degree	42.91 ±1.67	43.30 ±4.34	76.59 ±0.06	52.31 ±0.63	80.89 ±0.71	77.70 ±0.49	60.03
Topology	Rank Degree	41.30 ±1.80	42.00 ±3.52	76.39 ±0.44	51.41 ±0.85	79.88 ±0.67	76.28 ±0.31	58.84
Topology	Forest Fire	44.13 ±1.58	43.48 ±2.84	72.19 ±0.43	50.70 ±0.23	79.61 ±1.37	73.45 ±0.96	58.74
Topology	SCAN	39.15 ±2.05	38.53 ±2.70	76.70 ±0.22	51.94 ±0.13	91.28 ±0.23	75.36 ±0.29	58.80
Topology	L-Spar	39.09 ±1.88	38.70 ±3.38	76.59 ±0.11	51.60 ±0.40	**91.38** ±0.40	70.87 ±1.22	58.16
Topology	G-Spar	39.25 ±1.98	38.00 ±4.30	76.37 ±0.27	51.73 ±0.75	91.22 ±0.36	70.84 ±1.26	58.00
Topology	L-Sim	40.40 ±1.84	41.68 ±2.50	78.20 ±0.58	51.92 ±0.49	91.25 ±0.43	**78.46** ±0.28	**60.48**
Topology	Spectral	44.92 ±1.03	43.55 ±2.55	70.67 ±0.62	49.78 ±0.45	78.50 ±1.19	77.01 ±0.61	58.86
Topology	t-Spanner	42.88 ±1.12	43.37 ±3.95	70.91 ±0.33	51.02 ±0.12	78.44 ±1.25	73.26 ±1.35	58.15
Topology	DSpar	**44.93** ±1.80	**45.61** ±3.89	70.51 ±0.33	50.19 ±0.42	79.69 ±0.53	77.07 ±0.59	59.54
Scaf.-1	**Scaffold-Batch-1**	44.06 ±1.72	45.60 ±4.46	75.63 ±0.45	51.85 ±0.93	79.75 ±0.59	77.69 ±0.22	60.45
Scaf.-1	**Scaffold-Fast-1**	42.93 ±1.70	41.83 ±3.23	75.63 ±0.45	51.91 ±0.66	79.19 ±0.73	77.01 ±0.36	59.19
Scaf.-1	**Scaffold-Sample-1**	44.87 ±1.84	44.30 ±4.62	75.76 ±0.54	51.30 ±0.90	78.82 ±0.36	76.82 ±0.11	60.03
Scaf.-K	**Scaffold-Batch-K**	42.04 ±2.07	43.08 ±3.12	**79.73** ±0.22	54.36 ±0.46	90.49 ±0.29	**78.18** ±0.42	61.77
Scaf.-K	**Scaffold-Fast-K**	42.69 ±2.84	43.34 ±3.19	79.38 ±0.71	**54.43** ±0.23	**92.01** ±0.42	78.00 ±0.12	**62.10**
Scaf.-K	**Scaffold-Sample-K**	**44.86** ±2.17	**44.14** ±3.12	71.67 ±0.56	53.73 ±0.39	85.94 ±0.46	77.78 ±0.49	60.89
Retention ratio $\delta = 0.7$								
Semantic	Unified-LTH	36.66 ±2.89	36.55 ±4.18	70.81 ±0.37	47.15 ±0.27	82.40 ±0.81	75.97 ±0.29	55.11
Semantic	AdaGLT	43.25 ±1.78	36.72 ±3.10	**90.08** ±0.00	51.64 ±0.33	81.28 ±1.45	73.36 ±0.59	59.43
Semantic	MoG	44.19 ±2.22	44.64 ±2.60	79.65 ±0.37	53.28 ±0.34	87.53 ±0.14	76.00 ±0.24	61.80
Topology	Random	43.06 ±1.63	44.08 ±3.81	77.92 ±0.41	52.26 ±0.09	85.18 ±0.50	75.65 ±0.28	60.66
Topology	Local Degree	43.45 ±1.79	44.37 ±4.33	83.35 ±0.22	52.14 ±0.69	86.30 ±0.37	77.98 ±0.38	61.92
Topology	Rank Degree	42.02 ±2.03	43.10 ±2.08	82.73 ±0.19	51.59 ±0.35	86.26 ±0.81	77.15 ±0.34	60.98
Topology	Forest Fire	43.15 ±2.40	42.93 ±4.54	78.76 ±0.17	52.08 ±0.32	85.26 ±0.54	75.37 ±0.65	60.46
Topology	SCAN	38.80 ±1.70	35.87 ±2.91	80.49 ±0.74	52.34 ±0.12	90.20 ±0.33	77.17 ±0.68	58.68
Topology	L-Spar	38.22 ±1.83	36.92 ±3.22	79.97 ±0.27	52.60 ±0.20	90.43 ±0.22	73.56 ±0.69	58.35
Topology	G-Spar	38.60 ±1.71	36.09 ±3.61	80.46 ±0.18	52.49 ±0.11	**90.46** ±0.10	73.57 ±0.70	58.27
Topology	L-Sim	43.31 ±1.07	41.21 ±2.69	82.33 ±0.25	52.45 ±0.26	89.28 ±0.07	78.37 ±0.44	61.47
Topology	Spectral	**45.58** ±1.32	45.95 ±1.83	76.26 ±0.63	51.40 ±0.40	83.95 ±0.53	78.10 ±0.54	61.45
Topology	t-Spanner	43.11 ±2.30	45.47 ±3.95	78.17 ±0.19	52.65 ±0.61	85.22 ±0.72	76.61 ±0.69	61.22
Topology	DSpar	44.55 ±1.55	**45.95** ±4.48	77.26 ±0.66	52.38 ±0.20	85.51 ±0.53	**78.47** ±0.16	61.78
Scaf.-1	**Scaffold-Batch-1**	44.36 ±2.52	45.89 ±3.57	87.67 ±0.49	52.60 ±0.19	85.52 ±0.32	78.06 ±0.15	**63.02**
Scaf.-1	**Scaffold-Fast-1**	43.37 ±2.32	40.81 ±4.68	88.01 ±0.59	**53.36** ±0.81	86.98 ±0.76	78.29 ±0.29	61.96
Scaf.-1	**Scaffold-Sample-1**	44.94 ±1.85	45.31 ±4.04	87.96 ±0.78	52.61 ±0.27	85.08 ±0.62	77.81 ±0.14	62.98
Scaf.-K	**Scaffold-Batch-K**	44.60 ±2.20	**44.29** ±2.15	86.32 ±0.56	**54.73** ±0.20	93.16 ±0.40	78.46 ±0.22	**63.92**
Scaf.-K	**Scaffold-Fast-K**	43.56 ±2.08	42.78 ±3.34	**88.26** ±0.88	54.52 ±0.55	**93.35** ±0.34	**78.67** ±0.28	63.55
Scaf.-K	**Scaffold-Sample-K**	**44.65** ±2.12	44.17 ±5.28	84.25 ±0.50	54.37 ±0.35	88.93 ±0.33	78.38 ±0.13	63.07
Ratio-independent comparisons								
	No graph (MLP)	40.07 ±1.40	36.48 ±3.09	66.40 ±0.33	50.01 ±0.06	51.49 ±0.86	70.29 ±0.28	50.99
	Spanning Forest	37.59 ±1.74	36.73 ±4.52	82.45 ±0.56	49.95 ±0.77	67.03 ±0.58	73.89 ±0.61	55.16
	Full graph (T-GCN)	44.77 ±2.36	**45.46** ±4.42	**91.05** ±0.13	53.61 ±0.63	**97.18** ±0.45	78.47 ±0.29	**65.05**
	SCAFFOLD-GCN⋆	**45.42** ±1.60 (0.6)	45.32 ±3.86 (0.3)	89.79 ±0.06 (0.9)	**54.49** ±0.56 (0.6)	94.84 ±0.30 (0.9)	**78.67** ±0.23 (0.6)	64.96

Green, blue, and amber mark the top three *matched-budget* methods; Scaf.-1 is included (one fixed support at δ), while Scaf.-K is excluded because its union exceeds the budget and is shown below the rule with its best variant bolded. The ratio-independent block is ranked separately. GM: geometric mean. †Below the spanning-forest floor, no connected support exists. ‡Support rebuilt asynchronously. ⋆Per-dataset max over $\delta \in [0.1, 0.9]$; parentheses give the lowest δ attaining it.

Table 11: Large-scale graphs: complete mean ± standard deviation results (%) over five datasets.

	Method	Reddit	ogbn-products	ogbn-arxiv	ogbn-proteins	Pokec	GM
Retention ratio $\delta = 0.1$	†below the spanning-forest floor here: ogbn-arxiv						
Semantic	Unified-LTH	91.90 ±0.00	71.72 ±0.00	55.53 ±0.00	66.46 ±0.00	65.50 ±0.00	69.26
Semantic	AdaGLT	71.45 ±0.00	61.17 ±0.00	55.94 ±0.00	63.58 ±0.00	62.99 ±0.00	62.83
Semantic	MoG	85.15 ±0.07	63.45 ±0.09	56.25 ±0.15	71.12 ±0.08	64.14 ±0.01	67.36
Topology	Random	93.13 ±0.08	71.93 ±0.03	61.64 ±0.04	76.50 ±0.72	68.86 ±0.01	73.71
Topology	Local Degree	81.85 ±0.16	73.40 ±0.08	61.87 ±0.05	69.09 ±0.16	**75.48** ±0.05	72.03
Topology	Rank Degree	71.85 ±0.45	61.09 ±0.01	58.47 ±0.26	55.64 ±0.91	64.09 ±0.05	61.99
Topology	Forest Fire	94.97 ±0.04	73.72 ±0.13	62.41 ±0.18	**78.20** ±0.14	72.07 ±0.04	75.56
Topology	SCAN	83.96 ±0.02	65.85 ±0.04	61.59 ±0.07	64.55 ±0.28	65.08 ±0.00	67.78
Topology	L-Spar	83.61 ±0.03	65.85 ±0.08	60.90 ±0.02	64.55 ±0.21	64.82 ±0.01	67.52
Topology	G-Spar	83.54 ±0.04	65.76 ±0.10	60.93 ±0.08	64.43 ±0.20	64.81 ±0.00	67.47
Topology	L-Sim	93.39 ±0.01	73.66 ±0.05	64.56 ±0.08	76.38 ±0.48	70.74 ±0.01	75.17
Topology	Spectral	95.06 ±0.01	73.86 ±0.14	62.13 ±0.19	77.23 ±0.35	70.95 ±0.03	75.11
Topology	t-Spanner	93.47 ±0.00	71.84 ±0.17	61.80 ±0.02	78.01 ±0.50	68.89 ±0.02	74.07
Topology	DSpar	**95.09** ±0.02	**77.18** ±0.26	54.85 ±0.08	53.43 ±0.01	60.46 ±0.04	66.50
Scaf.-1	**Scaffold-Batch-1**	93.56 ±0.01	73.02 ±0.21	**64.56** ±0.19	75.15 ±0.27	71.65 ±0.06	75.01
Scaf.-1	**Scaffold-Fast-1**	93.56 ±0.01	73.30 ±0.39	63.36 ±0.43	77.20 ±0.12	71.65 ±0.06	75.19
Scaf.-1	**Scaffold-Sample-1**	94.94 ±0.10	74.80 ±0.16	62.10 ±0.01	76.80 ±0.72	73.05 ±0.14	**75.63**
Scaf.-K	**Scaffold-Batch-K**‡	93.49 ±0.53	76.47 ±0.43	**69.71** ±0.81	77.48 ±0.52	72.97 ±0.06	77.62
Scaf.-K	**Scaffold-Fast-K**‡	93.50 ±0.36	**76.57** ±0.41	69.23 ±1.60	**78.88** ±0.34	73.02 ±0.04	77.82
Scaf.-K	**Scaffold-Sample-K**‡	**95.11** ±0.01	75.62 ±0.29	69.65 ±0.05	77.22 ±1.11	**76.94** ±0.17	**78.48**
Retention ratio $\delta = 0.3$							
Semantic	Unified-LTH	92.68 ±0.00	75.26 ±0.00	56.37 ±0.00	67.15 ±0.00	68.23 ±0.00	70.98
Semantic	AdaGLT	85.09 ±0.00	66.74 ±0.00	55.93 ±0.00	52.31 ±0.00	64.42 ±0.00	63.96
Semantic	MoG	90.13 ±0.08	68.27 ±0.05	56.42 ±0.16	72.72 ±1.75	66.94 ±0.07	70.08
Topology	Random	94.69 ±0.02	79.08 ±0.06	67.77 ±0.18	76.83 ±0.08	76.33 ±0.05	78.47
Topology	Local Degree	90.82 ±0.15	77.62 ±0.16	68.34 ±0.17	71.10 ±0.00	**80.07** ±0.09	77.20
Topology	Rank Degree	77.28 ±0.04	63.13 ±0.02	62.59 ±0.07	66.01 ±0.78	67.29 ±0.10	67.06
Topology	Forest Fire	95.41 ±0.00	80.65 ±0.05	68.75 ±0.21	**78.25** ±0.32	79.43 ±0.10	80.05
Topology	SCAN	90.28 ±0.01	72.56 ±0.15	67.36 ±0.37	70.62 ±0.42	68.56 ±0.02	73.44
Topology	L-Spar	90.06 ±0.01	72.64 ±0.02	65.82 ±0.05	70.80 ±0.39	67.42 ±0.02	72.87
Topology	G-Spar	90.05 ±0.01	72.55 ±0.05	65.86 ±0.06	70.39 ±0.56	67.35 ±0.01	72.76
Topology	L-Sim	94.35 ±0.05	79.20 ±0.09	**70.69** ±0.08	77.95 ±0.11	76.56 ±0.06	79.38
Topology	Spectral	95.41 ±0.03	80.00 ±0.15	69.88 ±0.15	77.09 ±0.66	79.65 ±0.07	79.99
Topology	t-Spanner	94.72 ±0.01	79.30 ±0.01	67.51 ±0.09	77.43 ±0.12	76.38 ±0.07	78.59
Topology	DSpar	95.22 ±0.01	77.43 ±0.08	55.19 ±0.13	51.94 ±1.16	69.39 ±0.03	68.12
Scaf.-1	**Scaffold-Batch-1**	95.12 ±0.21	79.94 ±0.36	70.12 ±0.40	75.72 ±0.04	79.10 ±0.10	79.59
Scaf.-1	**Scaffold-Fast-1**	94.92 ±0.09	77.56 ±0.18	70.46 ±0.30	75.55 ±0.10	78.61 ±0.05	79.02
Scaf.-1	**Scaffold-Sample-1**	**95.43** ±0.18	**81.07** ±0.25	70.65 ±0.42	77.80 ±0.63	80.00 ±0.01	**80.60**
Scaf.-K	**Scaffold-Batch-K**‡	94.78 ±0.09	79.82 ±0.78	71.42 ±0.15	76.00 ±0.19	79.42 ±0.08	79.92
Scaf.-K	**Scaffold-Fast-K**‡	94.93 ±0.11	79.27 ±0.56	71.43 ±0.40	74.52 ±0.05	**81.53** ±0.08	79.95
Scaf.-K	**Scaffold-Sample-K**‡	**95.36** ±0.19	**80.46** ±1.12	**72.58** ±0.01	**76.91** ±0.56	80.30 ±0.49	**80.78**
Retention ratio $\delta = 0.5$							
Semantic	Unified-LTH	92.96 ±0.00	77.05 ±0.00	57.01 ±0.00	64.73 ±0.00	69.87 ±0.00	71.33
Semantic	AdaGLT	89.44 ±0.00	72.57 ±0.00	56.62 ±0.00	59.33 ±0.00	66.62 ±0.00	67.99
Semantic	MoG	91.61 ±0.09	70.97 ±0.17	56.65 ±0.15	70.79 ±0.05	69.47 ±0.03	71.06
Topology	Random	95.04 ±0.01	80.66 ±0.09	70.55 ±0.42	77.17 ±0.92	80.61 ±0.02	80.42
Topology	Local Degree	93.32 ±0.05	80.12 ±0.03	69.74 ±0.37	75.21 ±0.63	82.26 ±0.08	79.75
Topology	Rank Degree	79.04 ±0.10	65.47 ±0.17	63.14 ±0.12	68.20 ±1.01	70.64 ±0.08	69.09
Topology	Forest Fire	**95.50** ±0.04	**82.10** ±0.22	71.10 ±0.10	77.54 ±0.58	82.31 ±0.03	81.33
Topology	SCAN	92.13 ±0.01	77.19 ±0.15	70.64 ±0.14	74.62 ±0.05	74.64 ±0.03	77.51
Topology	L-Spar	91.84 ±0.00	76.55 ±0.08	67.68 ±0.02	74.87 ±0.18	69.71 ±0.05	75.68
Topology	G-Spar	91.81 ±0.03	76.49 ±0.10	67.64 ±0.05	74.30 ±0.38	69.70 ±0.00	75.54
Topology	L-Sim	94.82 ±0.02	80.68 ±0.07	71.67 ±0.18	76.75 ±0.24	79.97 ±0.01	80.43
Topology	Spectral	95.44 ±0.05	81.89 ±0.02	71.93 ±0.02	77.08 ±0.02	82.46 ±0.01	81.40
Topology	t-Spanner	95.27 ±0.02	80.70 ±0.11	70.25 ±0.20	77.22 ±0.13	80.70 ±0.09	80.43
Topology	DSpar	95.26 ±0.01	76.90 ±0.23	55.95 ±0.11	54.99 ±1.48	73.69 ±0.04	69.83
Scaf.-1	**Scaffold-Batch-1**	95.32 ±0.12	79.27 ±0.30	70.99 ±0.53	74.99 ±0.27	81.82 ±0.06	80.07
Scaf.-1	**Scaffold-Fast-1**	95.32 ±0.12	79.30 ±0.14	70.91 ±0.80	74.26 ±0.50	81.82 ±0.06	79.90
Scaf.-1	**Scaffold-Sample-1**	95.40 ±0.16	81.99 ±0.06	**72.19** ±0.07	**77.73** ±1.44	**83.03** ±0.12	**81.72**
Scaf.-K	**Scaffold-Batch-K**‡	95.25 ±0.12	80.99 ±0.32	72.71 ±0.04	75.68 ±1.04	82.21 ±0.07	81.01
Scaf.-K	**Scaffold-Fast-K**‡	95.19 ±0.15	80.61 ±0.11	72.67 ±0.19	75.03 ±0.21	82.29 ±0.15	80.79
Scaf.-K	**Scaffold-Sample-K**‡	**95.42** ±0.11	**81.79** ±0.28	**73.08** ±0.45	**76.55** ±1.79	**83.15** ±0.22	**81.66**
Ratio-independent comparisons							
	No graph (MLP)	69.66 ±0.09	45.46 ±0.19	55.60 ±0.16	72.91 ±0.17	62.31 ±0.02	60.34
	Spanning Forest	86.07 ±0.14	67.79 ±0.09	67.27 ±0.12	74.13 ±0.16	69.36 ±0.01	72.61
	Full graph (T-GCN)	95.44 ±0.05	80.67 ±0.10	72.83 ±0.20	77.48 ±0.06	**86.24** ±0.04	82.17
	SCAFFOLD-GCN⋆	**95.60** ±0.09 (0.5)	**82.25** ±0.14 (0.6)	**73.47** ±0.21 (0.7)	**77.71** ±0.21 (0.5)	85.93 ±0.02 (0.9)	**82.65**

Green, blue, and amber mark the top three *matched-budget* methods; Scaf.-1 is included (one fixed support at δ), while Scaf.-K is excluded because its union exceeds the budget and is shown below the rule with its best variant bolded. The ratio-independent block is ranked separately. GM: geometric mean. †Below the spanning-forest floor, no connected support exists. ‡Support rebuilt asynchronously. ⋆Per-dataset max over $\delta \in [0.1, 0.9]$; parentheses give the lowest δ attaining it.

F.4 CASE STUDY OF PERFORMANCE OF DIFFERENT DATASETS

The results in Tables 9,10,11 show cases where edge budget alone does not explain performance.

Budgets below the spanning-forest floor. For sparse graphs such as ROMAN-EMPIRE, CITESEER, and CORA, some evaluated retention ratios fall below the spanning-forest floor. At these budgets, no method can simultaneously satisfy the edge budget and preserve all connected components, so the resulting degradation should primarily be interpreted as a consequence of an infeasible connectivity budget rather than edge-selection quality.

Local neighborhood preservation on MINESWEEPER. MINESWEEPER is a notable exception to the aggregate trend: local similarity-based sparsifiers outperform `Scaffold`-1 at $\delta = 0.7$. This suggests that, for some tasks, preserving the distribution of retained edges within individual neighborhoods can be more important than optimizing global support paths.

Backbone choice on CHAMELEON and SQUIRREL. These small heterophilic graphs are the two cases where `Scaffold`-1 uses the low-stretch SLSF backbone. Here `Scaffold`-1, Spectral, and MoG obtain similar performance, while several degree- or similarity-based sparsifiers are noticeably weaker. This is consistent with the importance of preserving short, well-distributed communication paths on these graphs.

`Scaffold`-K is not uniformly better than `Scaffold`-1. Refreshing supports generally improves performance, but the gain is dataset dependent. In particular, `Scaffold`-K slightly underperforms `Scaffold`-1 on OGBN-PROTEINS, REDDIT, and OGBN-PRODUCTS at the matched target ratios, while improving substantially on datasets such as OGBN-ARXIV and POKEC. Because `Scaffold`-K trains on individual q-edge supports but evaluates on their union, its benefit depends on whether the refreshed supports provide complementary useful structure. Also, irregular swap of support graph can be harmful. The effects of refresh frequency and training versus inference topology are examined further in Appendix H.4.

F.5 REALIZED INFERENCE UNION SIZE OF SCAFFOLD-K

Table 12 shows that the realized `Scaffold`-K union varies substantially across datasets and variants, ranging from nearly a single q-edge support to almost the full graph. This variability motivates reporting `Scaffold`-K separately from the matched-budget `Scaffold`-1 results.

Table 12: Realized union size for Scaffold-K at the targets δ_d.

Dataset	δ_d	Scaffold-Batch			Scaffold-Fast			Scaffold-Sample		
		K	$\delta_\cup$	$\lvert E_\cup\rvert/q$	K	$\delta_\cup$	$\lvert E_\cup\rvert/q$	K	$\delta_\cup$	$\lvert E_\cup\rvert/q$
Cora	0.7	10	0.998	1.43	10	0.870	1.24	10	0.844	1.21
CiteSeer	0.7	10	0.981	1.40	10	0.910	1.30	10	0.752	1.07
Amazon Photo	0.3	10	0.905	3.02	10	0.808	2.69	10	0.695	2.32
WikiCS	0.3	10	0.329	1.10	10	0.300	1.00	10	0.821	2.74
Amazon Computers	0.3	10	0.442	1.47	10	0.436	1.45	5	0.570	1.90
Coauthor CS	0.5	10	0.989	1.98	10	0.651	1.30	10	0.720	1.44
PubMed	0.5	10	0.889	1.78	10	0.500	1.00	10	0.500	1.00
Coauthor Physics	0.3	10	0.533	1.78	10	0.498	1.66	5	0.576	1.92
Chameleon	0.3	5	0.725	2.42	10	0.300	1.00	5	0.453	1.51
Squirrel	0.3	5	0.300	1.00	10	0.300	1.00	5	0.549	1.83
Minesweeper	0.7	10	1.000	1.43	10	0.700	1.00	10	0.760	1.09
Roman Empire	0.7	10	0.858	1.23	10	0.992	1.42	10	0.761	1.09
Amazon Ratings	0.7	10	0.999	1.43	10	0.975	1.39	10	0.898	1.28
Questions	0.7	10	0.963	1.38	10	0.973	1.39	10	0.744	1.06
ogbn-proteins	0.3	10	0.305	1.02	10	0.324	1.08	10	0.987	3.29
ogbn-arxiv	0.3	10	0.583	1.94	10	0.541	1.80	10	0.802	2.67
Reddit	0.3	10	0.325	1.08	10	0.325	1.08	10	0.994	3.31
Pokec	0.3	10	0.479	1.60	10	0.523	1.74	10	0.840	2.80
ogbn-products	0.3	10	0.626	2.09	10	0.585	1.95	10	0.754	2.51

$q = \lceil \delta_d |E| \rceil$ and $\delta_\cup = |E_\cup|/|E|$.

G RUNTIME AND MEMORY DETAILS WITH GNN BENCHMARKS

The runtime of Scaffold variants on synthetic graphs is provided in Appendix C.3. In this section, we compare them with other sparsifiers and methods when used with GNNs.

G.1 SPARSIFICATION TIME AND SCALING OF SCAFFOLD

CPU scaling of `Scaffold` support generation. Table 13 shows how the scalable Scaffold variants benefit from additional CPU workers on REDDIT and OGBN-PRODUCTS. Fast and Batch remain relatively expensive on these large graphs, motivating asynchronous construction when supports are refreshed during training. Sample has a larger one-time startup cost, but subsequent supports are generated much faster by reusing its precomputed sampling state.

Table 13: CPU scaling of Scaffold support generation at $\delta = 0.3$.

Variant / stage	CPU workers					
	1	2	4	8	16	32
REDDIT						
Scaffold-Fast	498.38	284.13	213.08	196.54	201.54	182.08
Scaffold-Batch	630.43	355.01	265.93	242.33	232.81	216.50
Sample: first support	504.96	358.56	232.61	139.49	124.69	87.25
Sample: subsequent support	1.40	1.27	1.22	1.20	1.18	1.16
OGBN-PRODUCTS						
Scaffold-Fast	379.32	230.34	175.69	156.10	115.04	109.39
Scaffold-Batch	539.84	333.42	247.68	202.80	159.35	148.05
Sample: first support	1109.74	615.16	345.92	198.35	125.22	90.23
Sample: subsequent support	1.51	1.35	1.29	1.25	1.21	1.21

Medians over 3 completed sessions. Sample subsequent-support time reuses cached weights and sampling state. Dataset loading and warm-up are excluded.

Startup and reuse cost of `Scaffold-Sample`. Table 14 further separates the one-time initialization of Scaffold-Sample from repeated support generation. Most startup cost comes from constructing the forests and sampling weights, whereas loading saved weights and drawing subsequent supports is substantially cheaper. This reuse makes Scaffold-Sample particularly suitable for Scaffold-K, where sparse supports are refreshed repeatedly during training.

Table 14: Scaffold-Sample startup and reuse costs at $\delta = 0.3$.

Stage	CPU workers					
	1	2	4	8	16	32
REDDIT						
First support (total)	504.96	358.56	232.61	139.49	124.69	87.25
Forests + weights	472.57	320.16	195.16	107.40	87.36	54.80
Save weights	25.70	30.71	30.91	25.53	30.79	25.71
Other first-support work	6.73	6.60	6.57	6.53	6.54	6.53
Startup from saved weights	37.02	47.40	36.21	35.74	41.83	36.11
Subsequent support	1.40	1.27	1.22	1.20	1.18	1.16
OGBN-PRODUCTS						
First support (total)	1109.74	615.16	345.92	198.35	125.22	90.23
Forests + weights	1097.54	600.69	332.41	185.97	113.01	75.17
Save weights	2.09	4.35	2.49	2.35	2.13	4.72
Other first-support work	10.29	10.14	10.09	10.04	10.01	10.02
Startup from saved weights	14.90	15.04	14.82	15.31	14.40	15.07
Subsequent support	1.51	1.35	1.29	1.25	1.21	1.21

Medians over 3 sessions. Indented rows decompose the first-support cost; subsequent supports reuse the existing sampler. Independently computed medians may not sum exactly.

G.2 TRAINING PROFILE BREAKDOWN OF SCAFFOLD-FAST-1 AND FULL-GCN.

Table 15 (data behind Figure 7) shows how much time Scaffold-Fast-1 and Full-Graph (TunedGNN (Luo et al., 2024)) spend on sparsification and training. We profile two large datasets, REDDIT and OGBN-PRODUCTS, each run for 300 epochs on an H100 80GB GPU with 8 CPU workers. We further break down training time to show the cost of Gather-Scatter aggregation with the COO format and SpMM with the CSR/CSC format. The benefits of Scaffold are greatest on large graphs, at low retention ratios, and over long runs, where per-epoch savings amortize the sparsification overhead.

Table 15: `Gather-Scatter/SpMM` values for Figure 7. Times are seconds over 300 epochs; GPU is peak allocated memory (GiB).

Variant	δ	S	Gather-Scatter				SpMM			
			Agg.	Rest	Total	GPU	Agg.	Rest	Total	GPU
REDDIT										
Scaffold-Fast-1	0.1	133.9	37.5	70.8	242.3	9.98	12.6	64.6	211.2	3.97
Scaffold-Fast-1	0.3	174.8	138.4	132.1	445.3	24.52	27.2	106.9	308.9	5.54
Scaffold-Fast-1	0.5	223.5	236.7	190.3	650.5	39.09	41.5	146.8	411.7	7.35
Full-GCN	1.0	0.0	553.0	336.2	889.3	75.43	77.0	249.8	326.8	11.84
OGBN-PRODUCTS										
Scaffold-Fast-1	0.1	122.2	57.2	326.9	506.2	25.18	74.7	313.3	510.2	25.17
Scaffold-Fast-1	0.3	179.7	133.4	402.3	715.4	35.21	94.2	369.0	642.8	26.17
Scaffold-Fast-1	0.5	224.4	274.2	475.6	974.1	45.23	116.4	424.5	765.3	27.18
Full-GCN	1.0	0.0	591.4	655.7	1247.0	70.33	180.1	560.4	740.5	29.69

S: support generation; Agg./Rest: aggregation/other training work; Total $= S +$ Agg. $+$ Rest. Fast is the Scaffold-Fast variant. Each entry is one profiling run.

G.3 Sparsification and end-to-end runtime

Sparsification time. Table 16 compares standalone sparsification time at the same retention ratio, $\delta = 0.3$, across five large-scale graphs. Among the Scaffold variants, `Scaffold-Sample` is the fastest on all five graphs, while `Scaffold-Fast` and `Scaffold-Batch` incur higher cost from additional support-specific computation. The benefit of `Scaffold-Sample` really comes when we need to resparsify at subsequent epochs during training.

Table 16: Sparsification time (s) at $\delta = 0.3$ on five large-scale graphs.

Method	Reddit	ogbn-products	ogbn-arxiv	ogbn-proteins	Pokec
Random	1.23	1.37	0.015	0.819	0.582
Local degree	41.20	30.39	0.543	24.01	10.02
Rank degree	249.83	69.20	4.05	169.86	27.50
Forest Fire	2501.61	—	10.33	1414.05	692.14
SCAN	78.66	47.57	0.649	40.52	13.32
L-Spar	80.46	49.19	0.594	40.83	11.48
G-Spar	78.30	47.63	0.654	40.20	13.43
L-Sim	82.70	54.08	0.664	42.61	13.04
Spectral‡	387.50	379.95	5.80	286.24	108.09
Spanning tree†	36.21	17.25	0.225	2.10	3.32
Scaffold-Fast	72.00	153.05	6.30	96.12	80.09
Scaffold-Batch	208.85	250.71	9.27	115.42	78.77
Scaffold-Sample	48.83	133.89	6.06	24.09	11.68

†: spanning-tree reference with unmatched retention; —: unavailable. `Scaffold-Sample` includes one-time weight preparation.
‡: approximate effective resistance; using the Spielman–Srivastava JL method (Spielman & Srivastava, 2008), avoiding explicit Laplacian pseudoinversion. We use $d = \lceil 4 \ln n \rceil$ Gaussian projections with approximate-Cholesky solves (Gao et al., 2023) (10^{-2} tolerance) on the smaller graph, 64 Rademacher projections with PCG on OGBN-PRODUCTS, OGBN-PROTEINS, and POKEC, and TGT+ (Zhang et al., 2023) with 128 eigenpairs on REDDIT

End-to-end runtime. Table 17 compares end-to-end runtime at the same $\delta = 0.3$ retention ratio over 300 training epochs, including one-time sparsification cost. Despite this initial overhead, all three Scaffold variants reduce total runtime relative to Full-GCN on REDDIT, OGBN-PRODUCTS, and OGBN-ARXIV. For example, `Scaffold-Sample` requires 287.1 s versus 713.4 s on REDDIT, 642.0 s versus 1154.0 s on OGBN-PRODUCTS, and 76.7 s versus 96.7 s on OGBN-ARXIV. `Scaffold-Fast` and `Scaffold-Batch` show the same trend, indicating that the lower sparse-graph training cost amortizes support construction over the 300-epoch horizon. The marked AdaGLT and Unified-LTH configurations exceed the 3600 s run limit and are projected.

Table 17: End-to-end runtime (s) at $\delta = 0.3$ over 300 epochs. S: sparsification; T_{300}: training; $E_{300} = S + T_{300}$.

Method	Reddit			ogbn-products			ogbn-arxiv		
	S	T_{300}	E_{300}	S	T_{300}	E_{300}	S	T_{300}	E_{300}
Full-GCN	0.0	713.4	713.4	0.0	1154.0	1154.0	0.0	96.7	96.7
Scaffold-Fast-1	72.0	237.0	309.0	153.1	549.0	702.0	6.3	76.3	82.6
Scaffold-Batch-1	208.8	235.3	444.2	250.7	522.4	773.1	9.3	72.8	82.1
Scaffold-Sample-1	48.8	238.3	287.1	133.9	508.1	642.0	6.1	70.7	76.7
DSpar	4.0	224.4	228.4	4.1	491.8	496.0	0.1	70.4	70.5
MoG	incl.	1432.3	1432.3	incl.	2536.4	2536.4	incl.	154.1	154.1
AdaGLT	incl.	8819.6	8819.6†	incl.	15213.3	15213.3†	incl.	916.3	916.3
Unified-LTH	5.4	7144.1	7149.5†	6.7	13161.1	13167.8†	0.1	395.9	396.0

† marks AdaGLT/Unified-LTH totals exceeding the 3600 s run limit and are projected from the epochs they ran; "incl." means graph-selection cost is included in training.

H Additional Ablation and Structural Analysis

H.1 Scaffold Variants and Support Backbones with Greedy

Since `Scaffold-Greedy` is expensive to use in large graphs, Figure 16 compares the performance of variants and backbones on two small graphs Cora and Chameleon. Across support backbones, `Greedy`, `Heap`, and `Batch` generally provide the strongest reductions in dilation and congestion as expected, while `Fast` and `Sample` trade structural refinement for substantially lower construction cost. As in the main results, predictive performance does not strictly follow the structural score and also depends on the dataset and backbone.

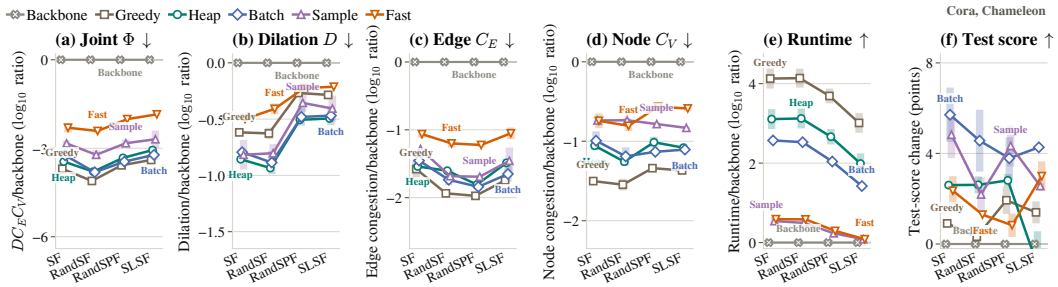

Figure 16: `Scaffold` variants across support backbones. Extension of Figure 11 including the full Greedy reference on Cora and Chameleon, where its computational cost remains tractable.

H.2 Effect of the Path Node and Edge Congestion Parameter p_E, p_V

We study the effect of the path aggregation parameters p_E and p_V of Equation 4 and 5. We compare average ($p = 1$), quadratic ($p = 2$), and maximum ($p = \infty$) aggregation using `Scaffold-Batch` with $(\alpha, \beta_E, \beta_V) = (1, 1, 1)$, while keeping the support backbone and edge budget fixed.

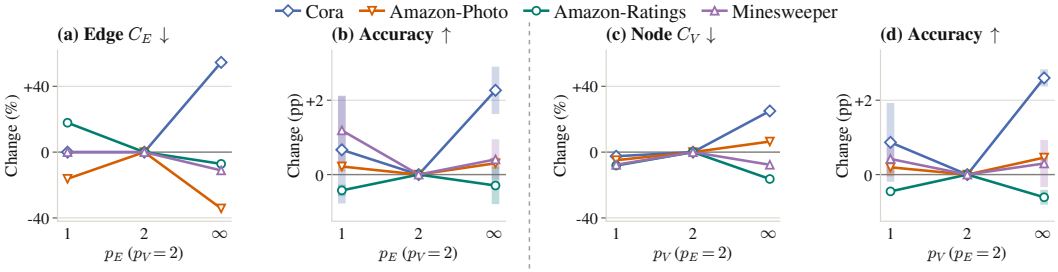

Figure 17: Effect of path congestion aggregation. (a–b) vary p_E with $p_V = 2$; (c–d) vary p_V with $p_E = 2$. Changes are relative to $p_E = p_V = 2$.

Figure 17 shows that changing the aggregation order affects both the resulting edge/node congestion and downstream accuracy. Larger p_E places more emphasis on highly loaded edges, while larger p_V emphasizes highly loaded nodes. For these GNN benchmarks, the effect is dataset dependent,

and reductions in maximum congestion do not always translate directly into higher accuracy. We therefore use $p_E = p_V = 2$ as the default.

H.3 Sensitivity to the number of supports K

We vary K while keeping training settings and per-support budgets fixed. Evaluation unions K validation-selected supports; training refreshes supports even at $K = 1$. On Cora, increasing K from 1 to 20 raises peak coverage from 70% to nearly 100%, while accuracy increases by 0.58 (`Fast`) and 1.20 (`Sample`) percentage points. On Amazon-Computers, both variants gain 0.85 points by $K = 4$, with little further improvement at $K = 20$. On Coauthor-CS, both variants have lower accuracy at $K = 20$ than at $K = 16$ despite higher peak coverage. The experiment shows that retaining more support for evaluation is useful; however, it comes at a computational cost.

Table 18: Sensitivity to K. Accuracy is mean ± standard deviation (%). Cov. is the maximum logged union edge coverage (%), excluding self-loops. Green boxes pair each column's best accuracy with its coverage at the same K.

K	**Cora** $\delta = 0.7$, 5 runs				**Amazon-Computers** $\delta = 0.3$, 3 runs				**Coauthor-CS** $\delta = 0.3$, 3 runs			
	Fast		Sample		Fast		Sample		Fast		Sample	
	Acc.	Cov.	Acc.	Cov.	Acc.	Cov.	Acc.	Cov.	Acc.	Cov.	Acc.	Cov.
1	82.96 ±0.86	70.0	82.70 ±0.85	70.0	91.88 ±0.28	30.0	92.58 ±0.13	30.0	95.27 ±0.14	30.0	94.81 ±0.36	30.0
2	82.70 ±0.86	87.4	82.62 ±0.58	80.3	92.35 ±0.08	33.8	93.04 ±0.42	49.7	95.35 ±0.06	46.7	95.35 ±0.16	37.6
3	82.78 ±0.37	94.3	82.72 ±0.99	85.4	92.52 ±0.42	36.4	93.29 ±0.36	59.6	95.81 ±0.04	58.8	95.36 ±0.22	45.2
4	82.82 ±0.70	97.0	82.96 ±0.73	90.8	92.73 ±0.27	38.3	93.43 ±0.32	68.6	95.72 ±0.05	67.3	95.45 ±0.21	52.8
5	83.28 ±1.14	94.1	82.94 ±0.62	95.1	92.65 ±0.18	39.9	93.26 ±0.08	77.2	95.98 ±0.15	74.0	95.50 ±0.05	58.3
6	83.00 ±0.76	99.0	82.86 ±0.96	91.3	92.75 ±0.33	41.2	93.34 ±0.24	83.2	95.91 ±0.16	79.0	95.71 ±0.04	64.5
8	83.14 ±0.44	99.6	82.96 ±0.34	93.3	92.79 ±0.30	42.1	93.30 ±0.37	89.5	95.86 ±0.23	86.0	95.84 ±0.11	75.0
10	82.84 ±1.06	100.0	83.32 ±0.81	88.7	**92.84** ±0.13	44.2	93.25 ±0.09	92.4	95.92 ±0.16	90.1	95.62 ±0.34	76.8
12	83.22 ±0.70	99.0	83.22 ±0.67	95.1	92.58 ±0.46	44.7	93.32 ±0.29	94.0	95.77 ±0.00	93.0	95.77 ±0.12	85.1
16	**83.80** ±1.07	100.0	83.44 ±0.86	99.1	92.61 ±0.19	46.6	**93.61** ±0.16	95.4	**95.99** ±0.20	96.4	**95.97** ±0.08	85.7
20	83.54 ±0.70	100.0	**83.90** ±1.04	99.3	92.72 ±0.22	47.6	93.58 ±0.22	99.6	95.56 ±0.39	97.9	95.90 ±0.11	91.7

H.3.1 Evaluation on the full graph

If memory permits, evaluating on the full graph can be preferable while training on a q-edge budget, since evaluation requires less memory. However, the following results show that full-graph evaluation does not consistently yield higher scores. One possible reason is that the best K supports may exclude noisy edges and provide better support for evaluation. For small graphs with larger K, the support union can cover most or all of the graph.

Table 19: Scaffold evaluation at each dataset's target retention δ. `Scaffold`-1 uses one fixed support for training and evaluation; `Scaffold`-K refreshes supports and evaluates their validation-selected union ($K = 10$). `Scaffold`-Full trains on sparse supports and evaluates on the full graph. Full-GCN trains and evaluates on the full graph. Reported mean ± SD (%).

Dataset	δ	**Variant**	**Scaffold-1**	**Scaffold-K**	**Scaffold-Full**	**Full-GCN**
Cora	0.7	Batch	83.80 ±0.92	↓ 83.54 ±0.71	↑ **84.58** ±0.56	84.56 ±0.63
CiteSeer	0.7	Fast	70.18 ±0.83	↑ 71.52 ±0.97	↑ **71.54** ±0.29	72.30 ±0.47
PubMed	0.5	Fast	79.34 ±0.99	↓ 79.08 ±0.78	↑ **80.46** ±0.63	81.22 ±1.11
Amazon-Photo	0.3	Batch	95.10 ±0.17	↑ **95.92** ±0.15	↑ 95.82 ±0.28*	95.95 ±0.33
Amazon-Computers	0.3	Sample	93.06 ±0.11	↑ **93.57** ±0.18	↑ 93.30 ±0.38	93.80 ±0.20
Coauthor-CS	0.5	Fast	95.39 ±0.24	↑ **96.04** ±0.06	↑ 95.99 ±0.25	95.72 ±0.04
Coauthor-Physics	0.3	Sample	97.09 ±0.06	↑ 97.22 ±0.14	↑ **97.43** ±0.06	97.50 ±0.12
WikiCS	0.3	Sample	79.48 ±0.32	↑ 79.97 ±0.63	↑ **80.21** ±0.50	80.27 ±0.44
Amazon-Ratings	0.7	Batch	52.60 ±0.19	↑ **54.73** ±0.20	↓ 52.36 ±0.36*	53.61 ±0.63
Minesweeper†	0.7	Fast	86.98 ±0.76	↑ **93.35** ±0.34	↑ 93.22 ±0.23	97.18 ±0.45
Questions†	0.7	Sample	77.81 ±0.14	↑ 78.38 ±0.13	↑ **78.45** ±0.40*	78.47 ±0.29

Blue bold: best Scaffold mean; ↑/↓: vs. Scaffold-1. †AUC; otherwise accuracy. *Time-limited training. Variants chosen by Scaffold-K validation. Training used separate runs and settings.

H.4 Support-Refresh Frequency

Figure 18 studies support-refresh frequency on Cora using a shared fast approximation of random spanning forest (RandSF) sequence. Refreshing every epoch reduces the gap to the full graph to 0.89 percentage points, compared with 2.09 and 3.79 points when refreshing every 10 and 50 epochs, respectively. Matched-support controls show a similar effect: with identical support sets, every-epoch refresh improves accuracy by 2.65 points over every 50 epochs and by 1.06 points over every 10 epochs. This suggests that stale supports can hinder optimization beyond the effect of support diversity. We treat this single-setting experiment as a mechanism ablation.

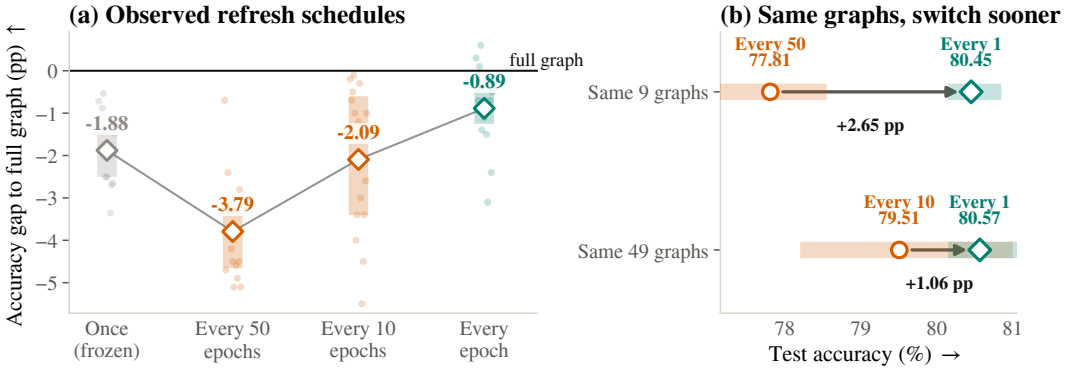

Figure 18: Support-refresh ablation on Cora with fast approximation of random forest (RandSF) at a 70% edge budget. **(a)** Accuracy across refresh schedules. **(b)** Matched-support comparisons with identical support sets.

H.5 Impact of Synchronous vs. Asynchronous Support Generation

For large graphs, constructing a new sparse support can be expensive and may stall GPU training. Scaffold-Sample mitigates this by precomputing edge-sampling probabilities, making subsequent support sampling inexpensive. Other variants, such as Fast and Batch, can require substantially more time to construct each support. We therefore generate the next support asynchronously on CPU workers while the GPU trains on the current one, and swap supports once the new support is ready. As shown in Figure 18, frequent support refresh is generally preferable to long refresh intervals.

On Cora, synchronization has little effect on runtime. On the relatively larger Amazon-Computers graph, asynchronous generation reduces runtime by 7.9× for Fast and 1.4× for Sample, with negligible accuracy changes. The smaller gap for Sample reflects its cheap sampling step, whereas variants with more expensive support construction benefit substantially from asynchronous generation. This makes asynchronous refresh particularly important for scaling Scaffold-K and Scaffold-Full to large graphs.

Table 20: Synchronous versus asynchronous support generation. Accuracy is mean ± SD; runtime is mean seconds per run.

Dataset	δ	Method	Accuracy (%)		Time (s)	
			Sync	Async	Sync	Async
Cora	0.7	**Scaffold-Fast**	82.92 ±0.47	82.94 ±0.90	12.6	12.3
		Scaffold-Sample	83.28 ±0.26	83.00 ±0.29	11.8	11.4
Amazon-Computers	0.3	**Scaffold-Fast**	92.09 ±0.09	92.18 ±0.15	280.6	35.7
		Scaffold-Sample	92.52 ±0.28	92.55 ±0.36	50.0	36.8

H.6 Training Support Refresh and Inference Topology

Figure 19 compares fixed-support and every-epoch support-refresh training under different inference topologies. On Cora, full-graph inference improves both fixed-support and refreshed training, whereas on Amazon-Photo the differences among inference choices are small. On Squirrel, the best-5 support ensemble performs best after support refresh, while full-graph inference provides little benefit. For Chameleon, full-graph inference substantially improves fixed-support training (38.90% to 43.93%), but provides almost no additional gain after every-epoch refresh (43.22%

versus 43.17%). Overall, training-time support refresh and inference topology interact differently across datasets, and full-graph inference is most beneficial when training on a single fixed support.

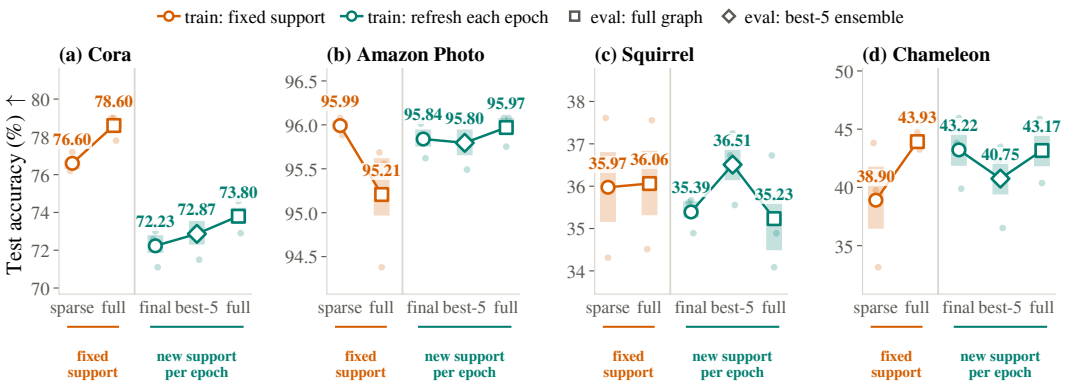

Figure 19: Training support refresh and inference topology at a 70% edge budget. Models are trained on either a fixed support or a new support each epoch and evaluated on the corresponding sparse support, the best-5 supports, or the full graph.

H.7 Scaffold−K vs Random sparse sample at every epoch

To test whether the choice of edges matters beyond repeated sampling, we compare Scaffold-K with Random-K. Random-K draws a uniform random edge sample each epoch and evaluates on the union of its $K = 5$ samples with the highest validation accuracy. We match the main-table GCN settings, splits, and training budgets. `Random-1` is the fixed Random baseline from the main table. Each training sample retains 30% of the edges; a union may exceed this budget.

Table 21: Sampling comparison at $\delta = 0.3$. Test accuracy (mean ± SD, %), with 10 runs per setting. Scaffold uses the Sample variant.

Method	Chameleon	Squirrel
Random-1	37.85 ±3.35	40.90 ±1.70
Scaffold-1	**42.50** ±4.38	**44.62** ±1.67
Random-K	43.09 ±3.30	43.24 ±2.03
Scaffold-K	**43.89** ±2.40	**43.99** ±1.59
Full-GCN	45.46 ±4.42	44.77 ±2.36

H.8 Structural implications of Scaffold

Figure 20 highlights how `Scaffold` preserves communication under sparsification. The spanning backbone preserves connected components above the spanning-forest floor, while low dilation keeps more omitted relations within the finite-hop receptive field of a GNN. Randomized backbones further expose complementary routes across repeated supports: on Cora at $\delta = 0.7$, refreshing `RandSF` improves accuracy by 2.13 points and approaches full-graph performance while increasing four-hop coverage.

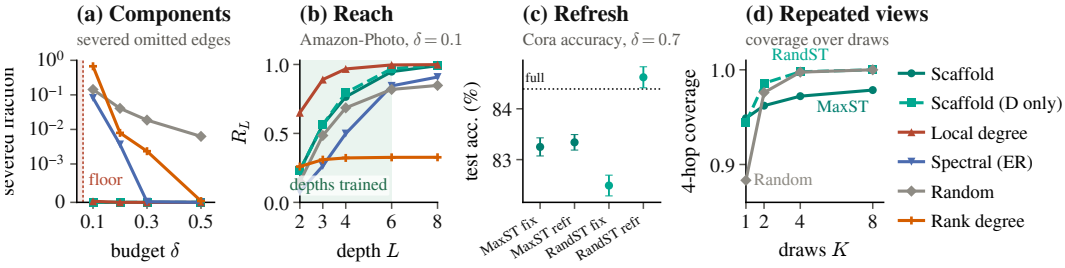

Figure 20: The spanning backbone preserves connectivity; low-dilation maintains finite-hop access, and randomized refresh exposes complementary communication routes across repeated sparse views.

H.9 COMPATIBILITY OF SCAFFOLD ACROSS GNN ARCHITECTURES

SCAFFOLD is backbone-agnostic, and it sparsifies the graph without modifying the GNN architecture. Table 22 evaluates `Scaffold` with GCN, GAT (Veličković et al., 2017), GIN (Xu et al., 2018), and GraphSAGE on AMAZON-COMPUTERS and OGBN-ARXIV at $\delta = 0.3$. `Scaffold`-1 uses one fixed support, whereas `Scaffold`-K asynchronously refreshes supports during training and evaluates on the union of $K = 5$ validation-best supports. All methods within each dataset use matched training settings and hardware, so runtime comparisons are made within each dataset and architecture. Across architectures, `Scaffold` generally reduces runtime while remaining competitive in accuracy. Compared with `Scaffold-1`, `Scaffold-K` can recover some accuracy through support refresh, potentially because different sparse views expose complementary information to the GNN.

The lower panel reports the number of asynchronous support swaps applied during training.

Table 22: `Scaffold` across GNN architectures at $\delta = 0.3$. Accuracy is mean $\pm$ SD; time is mean wall-clock runtime. Raised annotations show accuracy gap (pp) and speedup relative to the corresponding full-graph model.

Method	**Amazon Computers** $\delta_d = 0.3$ $\lvert V\rvert$ 13.8K $\lvert E\rvert$ 245.9K		**ogbn-arxiv** $\delta_d = 0.3$ $\lvert V\rvert$ 169.3K $\lvert E\rvert$ 1.2M	
	Acc.	Time	Acc.	Time
GCN	93.77 ±0.03	39.4 s	72.58 ±1.57	1.9 min
Scaf.-GCN^{1}	92.86 ±0.44 -0.91	34.9 s 1.13×	69.83 ±0.36 -2.75	1.8 min 1.06×
Scaf.-GCN$^{K,\text{async}}$	93.51 ±0.08 -0.25	36.3 s 1.08×	72.15 ±0.46 -0.43	1.7 min 1.16×
GAT	93.49 ±0.26	59.0 s	72.13 ±0.35	4.8 min
Scaf.-GAT^{1}	92.78 ±0.08 -0.71	35.1 s 1.68×	69.24 ±0.00 -2.88	2.5 min 1.92×
Scaf.-GAT$^{K,\text{async}}$	93.38 ±0.41 -0.11	41.0 s 1.44×	70.76 ±1.47 -1.36	2.4 min 1.97×
GIN	92.58 ±0.10	26.0 s	70.30 ±0.25	1.7 min
Scaf.-GIN^{1}	92.13 ±0.54 -0.45	21.5 s 1.21×	56.56 ±1.51 -13.74	1.7 min 1.06×
Scaf.-GIN$^{K,\text{async}}$	91.22 ±0.03 -1.36	26.0 s 1.00×	68.91 ±0.54 -1.39	83.5 s 1.26×
GraphSAGE	93.18 ±0.03	27.8 s	71.71 ±0.48	46.4 s
Scaf.-SAGE^{1}	92.89 ±0.33 -0.29	19.5 s 1.43×	68.88 ±0.28 -2.82	45.3 s 1.02×
Scaf.-SAGE$^{K,\text{async}}$	92.53 ±0.49 -0.65	19.2 s 1.45×	70.75 ±0.18 -0.96	41.4 s 1.12×

Applied async swaps (seeds 42 / 43)

Architecture	**Amazon Computers**	**ogbn-arxiv**
GCN	774 / 494	923 / 827
GAT	832 / 834	956 / 950
GIN	631 / 619	875 / 873
GraphSAGE	264 / 276	269 / 273