GROOT: NODE-SPECIFIC GLOBAL GRAPH PROPAGATION VIA BIDIRECTIONAL FOREST SCANS

Anonymous authors
Paper under double-blind review

ABSTRACT

Long-range graph propagation often relies on deep local message passing or dense global interaction mechanisms. We introduce GROOT (Global Rerooting Operator over Trees), a graph neural layer built around a bidirectional forest-scan operator. It compiles a sparse graph into a cached rooted spanning forest and computes node-specific global states through one bottom-up aggregation followed by one top-down rerooting scan. Learned channel-wise gates induce an implicit path kernel in which every source can influence every target within its forest component, without materializing a dense node-pair matrix. Edges outside the forest contribute through a single sparse source-correction term injected into the shared scan. We prove the exact equivalence between the scan and the normalized path-product kernel, characterize its conditional permutation equivariance, and establish $O(ED + ND)$ structural work and $O(E + ND)$ memory for width D. A custom CUDA implementation enables training on large-scale sparse graphs. Across five random seeds, GROOT consistently improves over strong local and global baselines in power-flow estimation, gas-network state estimation, water-network sensor recovery, and graph classification. On a large-scale London road graph, GROOT improves accuracy over GraphSAGE and IM-MPNN by 4.98% and 0.94%, respectively. Controlled experiments isolate the contributions of bidirectional scanning and learned gates, while molecular and citation tasks further characterize the forest-path bias. Overall, these results establish forest scanning as a practical global operator for sparse graphs with reusable tree-aligned propagation structure. Code is available at https://anonymous.4open.science/r/groot-FCC2.

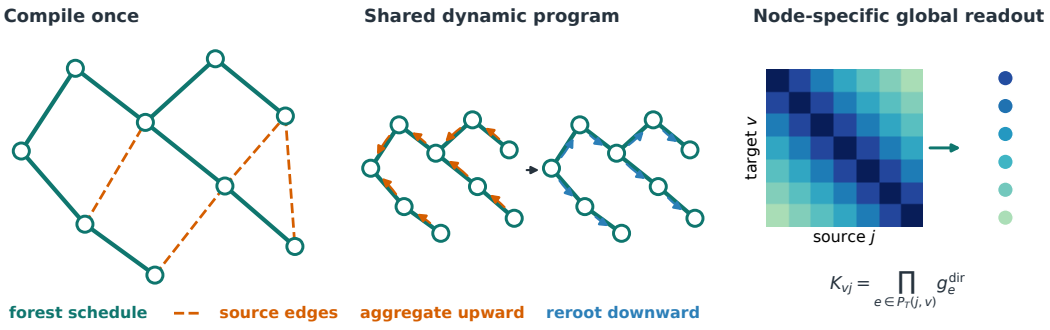
1 INTRODUCTION

Many graph problems are local in representation but global in dependency. A pressure estimate at one junction can depend on distant boundary conditions; a road-network label can encode eccentricity far beyond a shallow receptive field; and a graph-level decision can require combining evidence from several remote regions. A message-passing neural network (MPNN) reaches such information only after enough layers to traverse the relevant distance (Gilmer et al., 2017; Kipf & Welling, 2017), which can expose optimization, smoothing, and information-bottleneck effects (Alon & Yahav, 2021; Topping et al., 2022). Global attention shortens the communication path, but dense variants require quadratic node-pair storage, while sparse approximations introduce a different computational and statistical bias (Rampasek et al., 2022; Shirzad et al., 2023; Wu et al., 2023).

We study a third design point: compile the graph topology into a reusable dynamic-programming schedule. A rooted spanning forest supplies one path between every connected pair. Instead of treating each root-to-node path as a sequence, we share all overlapping computation. A bottom-up pass summarizes every subtree; a top-down rerooting pass subtracts a child's own contribution before transmitting the remainder of the component. With learned channel-wise gates, the two passes exactly evaluate a normalized, target-dependent path-product kernel for all nodes. The kernel is global within each component, but its implementation is linear in the forest size.

This idea has substantial ancestry. Upward/downward recurrences are standard in rerooting dynamic programming and belief propagation (Pearl, 1988); Tree-RNNs and Tree-LSTMs learn on tree-structured inputs (Tai et al., 2015); and recent forest-based graph learning (FGL) uses sampled trees, tree aggregation, and fusion for semi-supervised node classification (Li et al., 2026). Classical

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.



(a) Topology (b) Bidirectional scan (c) Implicit path kernel

Figure 1: Overview of GROOT. (a) Topology compilation caches a spanning-forest schedule and routes complementary edges into the source state. (b) All overlapping paths share one upward aggregation and one downward rerooting pass. (c) Their composition evaluates a target-dependent path-product kernel and returns one global state per node.

rerooting supplies the two-pass primitive. GROOT turns it into a mathematically explicit graph-layer contract: a reusable forest-path operator for general sparse graphs, a correction for omitted edges, a faithful CUDA realization, and a predictable task-dependent bias.

We answer this question with GROOT, a graph neural layer built around a global rerooting operator over a compiled spanning forest. Figure 1 summarizes the layer. Given node features and a fixed forest, GROOT first injects non-tree edges through one sparse source correction. It then performs a learned bottom-up/top-down scan and returns the normalized rerooted state through a residual block. Every contribution enters the same scan trunk. This makes the operator's mathematical action inspectable: for target v and source j, the effective coefficient is the product of learned gates along their unique forest path.

Our empirical study is organized around mechanism, applicability, and cost. On controlled path and cycle tasks, a two-layer GROOT retains near-perfect accuracy for source–query distances up to 64, where a parameter-matched two-layer GCN remains at chance. Across five seeds, GROOT consistently improves the strongest included baselines on SimBench power flow and GasLib-134 state estimation, improves water-network sensor recovery under a fixed training budget, and raises PROTEINS graph-classification accuracy by about 4.5 percentage points over GCN and GIN. On the 568,795-node London graph, direct official-code comparisons show gains of 4.98 accuracy points over GraphSAGE and 0.94 over IM-MPNN. Stable-ChebNet on ECHO-Synth and GCN/GCNII on OGBN-Arxiv provide complementary boundary cases, associating the gains with structured forest-path propagation.

Our contributions are:

- We define a graph layer whose learned node-specific global kernel is exactly factorized by a bottom-up/top-down forest scan, with a sparse non-tree source correction for general graphs.
- We establish exact path-kernel equivalence, component-wide receptive field, conditional permutation equivariance, and $O(ED + ND)$ structural work without an $N \times N$ interaction matrix.
- We provide a custom CUDA implementation with cached schedules, exact backward propagation, workspace reuse, and optional recomputation; it runs all 72 scaling configurations up to one million nodes without out-of-memory failure.
- We evaluate the operator under five-seed frozen-test protocols across controlled long-range, physical-network, graph-classification, and large transductive tasks, and report both favorable regimes and measured boundaries.

2 Related Work

Local, deep, and globally augmented GNNs. GCN, GIN, GatedGCN, and their descendants aggregate along observed edges (Kipf & Welling, 2017; Xu et al., 2019; Bresson & Laurent, 2018). APPNP and GCNII improve deep propagation through personalized diffusion and initial residuals (Klicpera et al., 2019; Chen et al., 2020). Virtual-node models add a shared graph-level state (Gilmer et al., 2017; Hu et al., 2020); this provides constant-hop global communication but broadcasts the same summary to every target. Graph transformers instead model non-local interactions through attention. GraphGPS combines local message passing with global attention, Exphormer sparsifies the attention graph, and SGFormer simplifies the global mechanism for large graphs (Rampasek et al., 2022; Shirzad et al., 2023; Wu et al., 2023). HopFormer restricts each attention head to a selected n-hop mask, with cost proportional to the materialized mask support (Yun et al., 2026). IM-MPNN interleaves message passing across a hierarchy of coarsened graphs, shortening communication distances at coarser scales (Finder et al., 2025). Bundle Neural Networks realize global diffusion through learned geometric structure (Bamberger et al., 2025), while state-space approaches recurrently propagate graph states or serialize nodes (Ceni et al., 2025; Behrouz & Hashemi, 2024). GROOT factorizes a different object: a component-wide, source–target-specific path-product kernel on a cached forest. Its coefficients are evaluated by shared rerooting without materializing an n-hop pair mask or constructing a multiscale graph hierarchy.

Trees as neural computation structures. Belief propagation, inside–outside algorithms, and rerooting dynamic programming reuse overlapping computations on trees (Pearl, 1988). Recursive neural networks and Tree-LSTMs learn bottom-up states, with bidirectional variants adding top-down context (Tai et al., 2015). Spanning-tree preprocessing has also been used to simplify graphs before applying conventional GNN layers. Most directly, FGL samples several homophily-informed spanning trees, applies a general combine–disentangle tree aggregator, and fuses tree outputs for semi-supervised node classification (Li et al., 2026). GROOT builds on this algorithmic lineage with a distinct layer contract: a single fixed forest for reusable sparse-topology execution, an explicit channel-wise path-product kernel, one non-tree source correction, one residual scan trunk, and a custom level-synchronous CUDA operator. The contribution lies in this complete factorization and execution contract, beyond the shared use of two tree passes.

Long-range evaluation. Long-range behavior is often discussed through oversquashing or sensitivity decay (Alon & Yahav, 2021; Topping et al., 2022); recent work cautions that topology, task, and computational bottlenecks should not be conflated (Arnaiz-Rodriguez & Errica, 2026). LRGB provides graph benchmarks with longer-range demands (Dwivedi et al., 2022), while ECHO isolates communication over long hops on controlled and molecular tasks (Miglior et al., 2026). City-Networks introduces road graphs with more than 10^5 nodes and eccentricity-derived targets (Liang et al., 2026). Our evidence map combines controlled and application-driven tasks: distance sweeps, component ablations, forest retraining, kernel-equivalence tests, and exclusive-GPU scaling each isolate a separate question.

3 GROOT: A Forest-Compiled Global Graph Layer

Let $G = (\mathcal{V}, \mathcal{E})$ be a sparse graph with $N = |\mathcal{V}|$ nodes, $E = |\mathcal{E}|$ directed edge records, and raw node features $X^{(0)} \in \mathbb{R}^{N \times D_{\text{in}}}$. An input adapter maps the raw features to width D; below, $X \in \mathbb{R}^{N \times D}$ denotes the hidden state entering a GROOT layer. A topology compiler constructs a rooted spanning forest $T \subseteq \mathcal{E}$ with one tree per connected component. For each non-root node v, let $p(v)$ be its parent and $d(v)$ its depth. The complementary edge set $\bar{\mathcal{E}} = \mathcal{E} \setminus T$ is retained as a sparse adjacency $A_{\bar{\mathcal{E}}}$. The forest is fixed for a run and cached independently of the trainable parameters.

3.1 Source correction and gates

We first project node values,

$$V = XW_V, \qquad S = V + \text{sigmoid}(\alpha)\, A_{\bar{\mathcal{E}}} V, \tag{1}$$

where $W_V \in \mathbb{R}^{D\times D}$ and $\alpha \in \mathbb{R}$ is one learned scalar per layer, shared across nodes, channels, and complementary edges. Equation (1) folds one sparse correction into the values entering the global scan. On a tree, $\bar{\mathcal{E}}$ is empty and $S = V$.

Each forest edge receives two channel-wise gates. Without edge attributes, the implementation uses

$$\ell_v^{\uparrow} = X_v \odot a_s + X_{p(v)} \odot a_r + b, \qquad g_v^{\uparrow} = \mathrm{sigmoid}(\ell_v^{\uparrow}), \tag{2}$$

$$\ell_v^{\downarrow} = X_{p(v)} \odot a_s + X_v \odot a_r + b, \qquad g_v^{\downarrow} = \mathrm{sigmoid}(\ell_v^{\downarrow}), \tag{3}$$

with learned channel scales $a_s, a_r, b \in \mathbb{R}^D$. Edge features, when present, add a learned edge logit. The architecture permits $g^{\uparrow} \neq g^{\downarrow}$, although Section 5.4 shows that direction-specific gate values are task dependent; learned gates and two propagation directions are the robust ingredients.

3.2 Bottom-up aggregation and exact rerooting

The scan carries a value state and a scalar-per-channel mass state. Initialize

$$q_v \leftarrow S_v, \qquad m_v \leftarrow \mathbf{1}. \tag{4}$$

Process depth buckets from leaves to roots. Every non-root node updates its parent by

$$q_{p(v)} \leftarrow q_{p(v)} + g_v^{\uparrow} \odot q_v, \qquad m_{p(v)} \leftarrow m_{p(v)} + g_v^{\uparrow} \odot m_v. \tag{5}$$

After this pass, (q_v, m_v) summarizes the entire subtree rooted at v.

For the top-down pass, initialize $(t_r, z_r) = (q_r, m_r)$ at every root r. At child v of parent $p = p(v)$, remove the child's upward contribution before transmitting the outside state:

$$o_v = t_p - g_v^{\uparrow} \odot q_v, \qquad c_v = z_p - g_v^{\uparrow} \odot m_v, \tag{6}$$

$$t_v = q_v + g_v^{\downarrow} \odot o_v, \qquad z_v = m_v + g_v^{\downarrow} \odot c_v. \tag{7}$$

The two gates have different algebraic roles. The upward gate $g_v^{\uparrow}$ removes the exact child branch previously added to p in Eq. (5); the downward gate $g_v^{\downarrow}$ then weights the remaining parent-side state as it crosses the same edge toward v. Hence Eq. (7) and Proposition 1 both permit $g^{\uparrow} \neq g^{\downarrow}$.

The output of the core scan is

$$U_v = \frac{t_v}{z_v + \varepsilon}. \tag{8}$$

Equations (5)–(8) jointly define the forest rerooting map

$$\mathcal{R}_T\big(S; g^{\uparrow}, g^{\downarrow}\big) = U,$$

which we call the global rerooting operator. It maps the corrected source states to node-specific, component-wide states under the compiled forest and realized gates.

Subtracting the current subtree in Eq. (7) is essential: directly broadcasting a root state would duplicate child-subtree contributions and reduce the operation to a common summary.

The full residual layer is

$$X' = \mathrm{Norm}(X + \mathrm{Dropout}(W_O\,\mathrm{GELU}(U)))\,, \tag{9}$$

where W_O is optional when dimensions already match. Together with the feature-dependent gates, complementary-edge source correction, learned projections, and residual update, the complete GROOT layer defines the nonlinear graph operator

$$\Phi_{T, A_{\bar{\mathcal{E}}}, \theta} : X \longmapsto X',$$

where θ collects the trainable parameters. Multiple layers may be stacked; each layer retains a component-wide receptive field but learns a new feature transformation and gates.

3.3 THE IMPLICIT PATH KERNEL

For source j and target v in the same tree, let $P_T(j, v)$ denote their unique forest path. Each edge on the path is traversed upward before their lowest common ancestor and downward afterward. Define

$$K_{vj} = \prod_{e \in P_T(j,v)} g_e^{\mathrm{dir}(j \to v)}, \tag{10}$$

with an empty-path product of one. Products and sums are channel-wise.

Proposition 1 (Exact factorization)**.** *For every fixed forest, source matrix S, and nonnegative gates, Eqs.* (5)–(8) *compute*

$$U_v = \frac{\sum_{j \in C(v)} K_{vj} \odot S_j}{\sum_{j \in C(v)} K_{vj} + \varepsilon}, \tag{11}$$

where $C(v)$ is the forest component containing v.

Proof sketch. Induction from the leaves shows that Eq. (5) equals the path-product sum over each subtree. For child v, Eq. (7) removes exactly the terms whose path to $p(v)$ first enters through v, leaving all sources outside that subtree. Multiplying the remainder by $g_v^{\downarrow}$ appends the final parent-to-child factor, while q_v supplies the within-subtree terms. The mass recurrence applies the same argument to unit sources, producing the denominator of Eq. (11). A complete induction is given in Appendix A. □

Proposition 2 (One-layer component receptive field)**.** *Holding the gates fixed, if all gates on $P_T(j, v)$ are nonzero, then the channel-wise coefficient*

$$\frac{\partial U_v}{\partial S_j} = \frac{K_{vj}}{\sum_{k \in C(v)} K_{vk} + \varepsilon}$$

is nonzero. Thus one core scan connects every ordered pair in a forest component.

This statement concerns reachability, not equal influence. Products of gates below one can attenuate long paths, and the normalization in Eq. (8) controls scale but does not remove that inductive bias.

4 CUDA EXECUTION

The topology compiler stores `parent`, nodes sorted by depth, level pointers, roots, component identifiers, and the non-tree sparse adjacency. These objects are built once for a fixed topology and reused across epochs and layers. In batched multi-graph datasets, each graph is compiled once and its cached schedule is reused whenever that graph is sampled; packed disjoint forests let nodes from different graphs at the same depth share a launch.

The CUDA backend processes all nodes and channels of one depth bucket in parallel. A forward scan uses one bottom-up and one top-down traversal, plus root initialization, for approximately $2H + 2$ level kernels. The custom backward implements the analytic reverse recurrences rather than exposing hundreds of small indexing operations to general-purpose autograd. Gate evaluation and parameter-gradient reductions are fused where compatible. Reusable workspaces limit allocator traffic; an optional recomputation mode discards saved scan states and reconstructs them during backward. Unit tests compare the reference Torch recurrence, the CUDA recurrence, and the explicit kernel in Eq. (11). Maximum observed discrepancies are 2.38×10^{-7} for explicit-kernel forward values, 3.58×10^{-7} for Torch/CUDA outputs, and 1.12×10^{-8} for gradients.

Preprocessing and execution are reported separately. CPU forest construction can dominate a one-off call on million-node graphs, whereas cached GPU execution is fast. This separation is important for the intended regime: fixed or slowly changing sparse topologies trained for many optimization steps. For rapidly changing graphs, repeated compilation may outweigh the benefit of the scan.

5 EXPERIMENTS

We ask four questions: **Q1** does the scan realize long-range communication; **Q2** where does its forest-path bias improve predictive performance; **Q3** which components are responsible; and **Q4** do time and memory remain practical at scale?

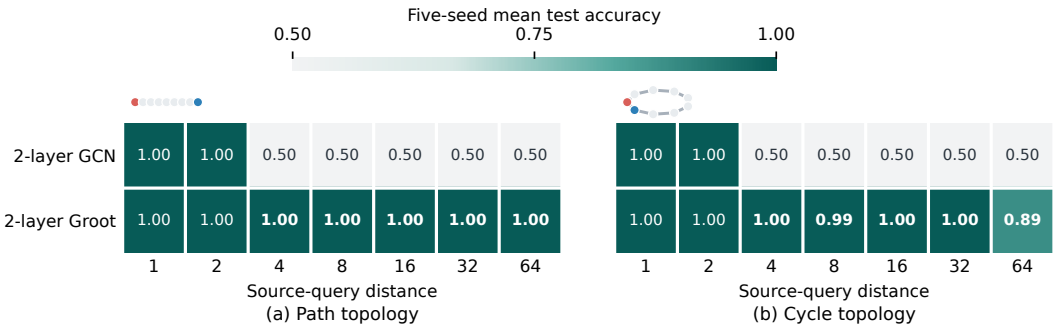


Figure 2: Controlled long-range communication as an accuracy matrix. Each cell is the five-seed mean test accuracy at one source–query distance. Beyond the two-hop receptive field, GCN remains at chance while GROOT preserves the source bit across both topologies; the cycle at distance 64 exposes attenuation along the compiled forest path.

5.1 PROTOCOL

Unless stated otherwise, formal training uses seeds $\{0, 17, 42, 123, 2027\}$ while keeping data splits, forest priorities, and task masks fixed. Hyperparameters are fixed across training seeds. Each run selects a checkpoint using validation data and evaluates the frozen test split once. We report the mean and sample standard deviation over five seeds. PROTEINS uses each outer training seed as one statistical unit and averages ten fixed folds within that seed. Some physical-network models continue improving at the common epoch cap; these are compared only as equal fixed-budget runs and are marked in Appendix D. Timing results use repeated CUDA events on an exclusive NVIDIA L40, not training seeds.

Baselines are chosen to test distinct explanations: GCN/GIN(E) for local propagation, GatedGCN for learned edge weighting, Tree-GINE for tree-restricted local propagation, Virtual-GINE for a shared global state, GCNII/APPNP for deep diffusion, and SGFormer for sparse global mixing. City-Networks includes both the benchmark authors' public five-run results and direct London reruns of official GraphSAGE and IM-MPNN code using our five seeds and fixed 5,000-epoch budget. Dataset roles and evaluation protocols are in Appendix D.

5.2 Q1: CONTROLLED LONG-RANGE COMMUNICATION

We place one random bit at a source node of a 128-node path or cycle and ask a query node to recover it at distances $1, 2, 4, 8, 16, 32, 64$. Both models use two layers and nearly matched parameter counts: GCN has 9,089 parameters and width 64; GROOT has 8,931 parameters and width 62. The complete matrix contains 2 topologies $\times 7$ distances $\times 2$ models $\times 5$ seeds $(140/140$ runs).

For distances greater than two, the average GROOT–GCN accuracy difference is $+0.4873$. Path accuracies at distances 4–64 are $1.000/1.000/1.000/1.000/0.997$; cycle accuracies are $0.995/0.994/1.000/1.000/0.887$, while GCN is 0.5 throughout. This isolates receptive field from natural-dataset correlations. The end-to-end training-time ratio is 1.96–$4.53\times$ on these small graphs, establishing the communication effect together with its small-graph latency cost.

5.3 Q2: TASK-ALIGNED REAL-WORLD PERFORMANCE

Table 1 covers three fixed-topology node regression/recovery tasks and one graph classification benchmark. SimBench Mixed predicts voltage magnitude on several power-grid topologies (Meinecke et al., 2020); BattLeDIM reconstructs held-out pressure sensors across a public water-network split (Vrachimis et al., 2022); GasLib-134 estimates normalized gas-network state (Schmidt et al., 2017); and PROTEINS is a standard graph-classification dataset (Morris et al., 2020).

On SimBench, GROOT wins all five paired seeds against Virtual-GINE; the paired MAE difference has 95% t interval $[-2.354, -1.819] \times 10^{-5}$. On GasLib, it wins all seeds against Virtual-GINE, SGFormer, and GINE, with all paired intervals excluding zero. On PROTEINS, the gain

Table 1: Five-seed frozen-test results. Arrows indicate metric direction. Baselines in each block share the task protocol with GROOT. Bold marks the best listed mean.

Dataset / metric	Model	Test mean ± sample SD
SimBench Mixed, v_m MAE ↓	GINE	$(2.981 \pm 0.039) \times 10^{-4}$
	GatedGCN	$(2.797 \pm 0.015) \times 10^{-4}$
	Tree-GINE	$(3.211 \pm 0.027) \times 10^{-4}$
	Virtual-GINE	$(3.507 \pm 0.201) \times 10^{-5}$
	GROOT	$\mathbf{(1.420 \pm 0.064) \times 10^{-5}}$
BattLeDIM, pressure MAE (m) ↓	GCN	0.5387 ± 0.0113
	GINE	0.5447 ± 0.0125
	GatedGCN	0.5375 ± 0.0213
	Tree-GINE	0.5398 ± 0.0143
	Virtual-GINE	0.1540 ± 0.0272
	GROOT	$\mathbf{0.1183 \pm 0.0133}$
GasLib-134, NRMSE ↓	GINE	1.2312 ± 0.0246
	SGFormer	0.2153 ± 0.0232
	Virtual-GINE	0.1692 ± 0.0108
	GROOT	$\mathbf{0.1427 \pm 0.0101}$
PROTEINS, accuracy ↑	GCN	0.7006 ± 0.0102
	GIN	0.7008 ± 0.0090
	GROOT	$\mathbf{0.7461 \pm 0.0060}$

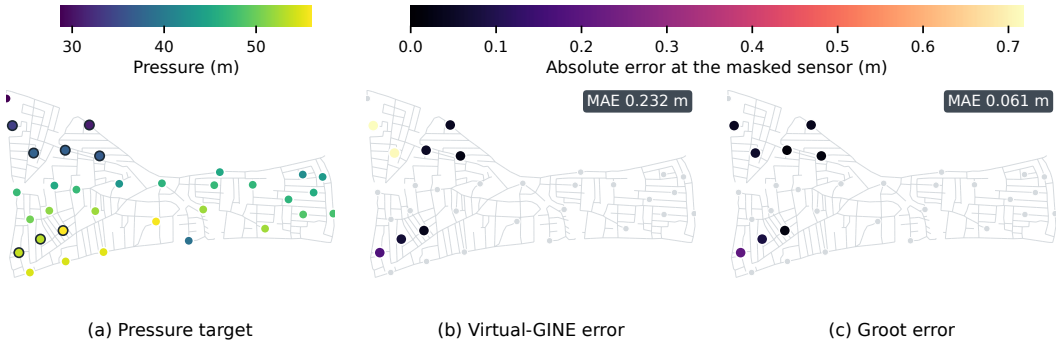
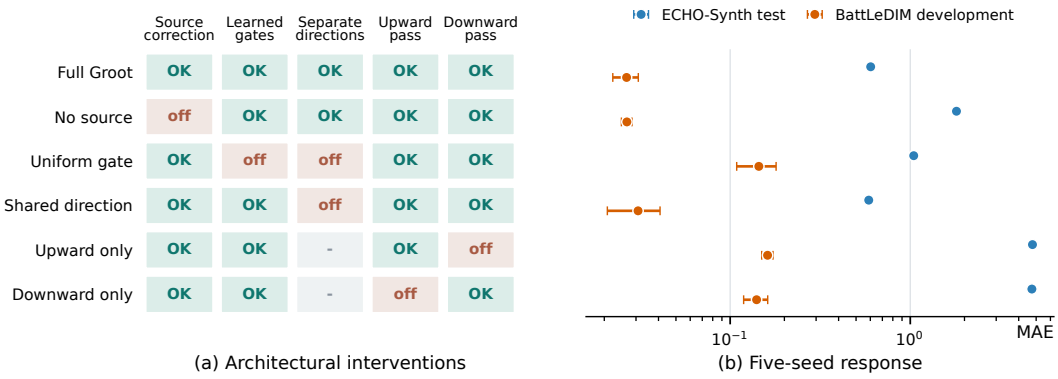


Figure 3: A representative BattLeDIM recovery example from the frozen 2019 test split. The example is selected deterministically at the median per-example GROOT error for seed 0. (a) Pressure targets at all sensors; dark outlines identify the held-out cluster. (b,c) Absolute error at the same held-out sensors for Virtual-GINE and GROOT, with shared color limits. Gray points are observed sensors.

over GCN is 0.0455 with interval $[0.0311, 0.0598]$, and the gain over GIN is 0.0453 with interval $[0.0321, 0.0586]$. BattLeDIM also shows five of five wins against Virtual-GINE, although its paired interval $[-0.0806, 0.0092]$ crosses zero because the improvement varies across seeds. Together, the results support fixed-topology global field estimation and graph-level aggregation, while distinguishing consistent direction from statistical precision.

Large road networks. City-Networks predicts local-eccentricity labels on road graphs designed to require distant information (Liang et al., 2026). Appendix Table 3 uses the public 16-layer protocol. On Paris, GROOT matches GCN and trails GraphSAGE/Exphormer. On London, the largest graph, direct official-code reruns give 52.98 ± 0.75 for GraphSAGE and 57.02 ± 0.31 for IM-MPNN, versus 57.96 ± 0.30 for GROOT. GROOT wins all five paired seeds against IM-MPNN; the mean accuracy difference is $+0.94$ percentage points with 95% t interval $[+0.35, +1.52]$. It also wins all five seeds against GraphSAGE by $+4.98$ points on average. All three methods obtain late validation optima under the common 5,000-epoch cap; we therefore report equal fixed-budget results. GraphGPS is reported out of memory on a 48-GB GPU.



(a) Architectural interventions (b) Five-seed response

Figure 4: Component-level interventions and their five-seed response. (a) The matrix identifies the computation changed by each variant. (b) Points are mean MAE and bars are sample SD on a logarithmic axis. Learned gates and both scan directions are necessary in both tasks; source correction matters on cyclic ECHO-Synth but is neutral on the near-tree water network.

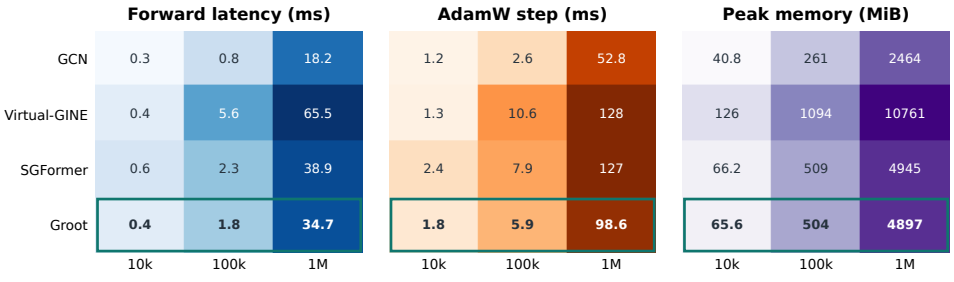
5.4 Q3: Mechanism ablations

Figure 4 changes one component at a time. ECHO-Synth SSSP supplies controlled cyclic and acyclic graphs (Miglior et al., 2026); BattLeDIM uses only the 2018 development split for this mechanism study and never reads the 2019 test set. Every entry averages the same five seeds; exact values appear in Appendix Table 4.

Removing either pass causes the largest degradation, demonstrating that subtree aggregation and rerooting are complementary. Uniform gates are also consistently worse, isolating the contribution of learned path weighting. Source correction is decisive on cyclic ECHO but neutral on the near-tree water network, matching its intended role. Finally, sharing the realized gate across directions is competitive on both tasks and slightly better on ECHO, showing that both shared and direction-specific parameterizations are viable.

Forest choice is part of the model specification. To distinguish retraining sensitivity from checkpoint mismatch, we independently train BFS, DFS, and MST forests on SimBench for all five seeds. Their MAEs are (1.4170 ± 0.0729), (1.3575 ± 0.0744), and $(1.3785 \pm 0.0384) \times 10^{-5}$, respectively. Reasonable forests perform similarly after independent training, even though changing the forest under a frozen checkpoint can strongly perturb predictions.

5.5 Q4: Scaling and synchronization depth



(a) Inference (b) Training (c) Memory
Graph size (average degree 8, width 64)

Figure 5: Exclusive-L40 scaling at average degree 8 and width $D = 64$. Cells report forward latency, full AdamW-step latency, and peak allocated memory with resident structures and optimizer state. The outlined GROOT row reaches one million nodes in every measurement; repetition counts are given in Appendix Table 8.

Across $N \in \{10^4, 10^5, 10^6\}$, average degree $\{4, 8, 16\}$, and width $D \in \{64, 128\}$, all 72 cells for GCN, Virtual-GINE, SGFormer, and GROOT succeed. At one million nodes, GROOT costs about twice GCN, runs faster than Virtual-GINE, and matches or exceeds SGFormer speed with nearly identical memory. Figure 5 shows one slice; Appendix Tables 8 and 9 give the full regime.

At $N = 10^6$, CPU construction takes $86.7/152.9/294.7$ s for degree $4/8/16$, while transfer and GPU structure creation take $0.061/0.078/0.154$ s. These cached costs are excluded from per-step timing. Compiling/loading all PROTEINS forests takes $2.61/0.20$ s and all Peptides-func training forests $62.12/2.25$ s; cached GROOT epochs remain 3.1–3.3$\times$ slower than GCN/GIN. Appendix Figure 9 audits synchronization depth over all 466 formal schedules.

5.6 BOUNDARY RESULTS

Boundary tasks map the single-path bias. Stable-ChebNet obtains 0.2660 ± 0.0196 MAE on ECHO-Synth versus 0.6028 ± 0.0046 for GROOT (Hariri et al., 2025); on OGBN-Arxiv, GROOT reaches 0.6776 accuracy versus $0.7174/0.7191$ for GCN/GCNII. GROOT beats small local and virtual-node implementations on Peptides-func, while larger modern long-range models score higher. It is near DenseGCN/SGFormer on LamaH (Klingler et al., 2021), and its positive PHEME trend has intervals crossing zero (Zubiaga et al., 2016). These outcomes associate gains with propagation geometry captured by a reusable tree skeleton.

6 LIMITATIONS AND CONCLUSION

GROOT maps each connected source–target pair to one forest path. Stacked layers recompute features and gates and reinject the non-tree correction, mixing new intermediate states while each core scan retains its single-path kernel. This bias may be less suitable for repeated circulation or many alternative routes. Path products can attenuate long dependencies, level-synchronous execution requires $O(H)$ launches for forest height H, and CPU compilation adds a one-time cost. The present implementation therefore targets fixed or slowly varying sparse graphs with useful tree skeletons; forest mixtures, height-aware compilation, and reduced synchronization are natural extensions.

Within this regime, two shared scans realize a target-specific, component-wide kernel with linear structural work. The layer is mathematically explicit, CUDA-executable at million-node scale, and effective across fixed-topology field and graph-level tasks.

ETHICS STATEMENT

The experiments use public benchmark data and introduce no new human-subject data. Several tasks concern infrastructure networks, where benchmark accuracy alone is insufficient for operational use. Deployment in water, gas, or power systems would require uncertainty calibration, distribution-shift evaluation, failure-mode analysis, and domain-specific human oversight. The fixed forest can encode structural bias by attenuating alternative routes, so high-stakes applications should document compiled structures and sensitivity analyses.

REPRODUCIBILITY STATEMENT

The supplementary material includes the reference recurrence, CUDA implementation, fixed forest builders, data adapters, training scripts, exact seed policy, raw per-seed results, and plotting source. Appendix D documents data roles, checkpoint selection, fixed-budget flags, and statistical summaries; Sections 3–5 and the remaining appendices specify the algorithm, proofs, timing scopes, and implementation audits. Kernel-equivalence, gradient, permutation, packed-forest, and CUDA tests are included; the audited suite reports 35 passing tests and six environment-gated CUDA-extension tests, with the extension separately validated on the training server.

REFERENCES

Uri Alon and Eran Yahav. On the bottleneck of graph neural networks and its practical implications. In *International Conference on Learning Representations*, 2021.

Adrian Arnaiz-Rodriguez and Federico Errica. Oversmoothing, oversquashing, heterophily, long-range, and more: Demystifying common beliefs in graph machine learning. In *International Conference on Learning Representations*, 2026.

Jacob Bamberger, Federico Barbero, Xiaowen Dong, and Michael M. Bronstein. Bundle neural networks for message diffusion on graphs. In *International Conference on Learning Representations*, 2025.

Ali Behrouz and Farnoosh Hashemi. Graph mamba: Towards learning on graphs with state space models. In *ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, 2024.

Xavier Bresson and Thomas Laurent. Residual gated graph convnets. In *International Conference on Learning Representations Workshop*, 2018.

Andrea Ceni, Alessio Gravina, Claudio Gallicchio, Davide Bacciu, Carola-Bibiane Schoenlieb, and Moshe Eliasof. Message-passing state-space models: Improving graph learning with modern sequence modeling. *arXiv preprint arXiv:2505.18728*, 2025.

Ming Chen, Zhewei Wei, Zengfeng Huang, Bolin Ding, and Yaliang Li. Simple and deep graph convolutional networks. In *International Conference on Machine Learning*, 2020.

Vijay Prakash Dwivedi, Ladislav Rampasek, Mikhail Galkin, Ali Parviz, Guy Wolf, Anh Tuan Luu, and Dominique Beaini. Long range graph benchmark. In *Advances in Neural Information Processing Systems Datasets and Benchmarks Track*, 2022.

Shahaf E. Finder, Ron Shapira Weber, Moshe Eliasof, Oren Freifeld, and Eran Treister. Improving the effective receptive field of message-passing neural networks. In *International Conference on Machine Learning*, 2025.

Justin Gilmer, Samuel S. Schoenholz, Patrick F. Riley, Oriol Vinyals, and George E. Dahl. Neural message passing for quantum chemistry. In *International Conference on Machine Learning*, 2017.

Ali Hariri, Alvaro Arroyo, Alessio Gravina, Moshe Eliasof, Carola-Bibiane Schoenlieb, Davide Bacciu, Kamyar Azizzadenesheli, Xiaowen Dong, and Pierre Vandergheynst. Return of chebnet: Understanding and improving an overlooked gnn on long-range tasks. *arXiv preprint arXiv:2506.07624*, 2025.

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. In *Advances in Neural Information Processing Systems*, 2020.

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

Johannes Klicpera, Aleksandar Bojchevski, and Stephan Guennemann. Predict then propagate: Graph neural networks meet personalized pagerank. In *International Conference on Learning Representations*, 2019.

Christian Klingler, Katrin Schulz, and Mathew Herrnegger. Lamah-ce: Large-sample data for hydrology and environmental sciences for central europe. *Earth System Science Data*, 13:4529–4565, 2021.

Jin Li, Shenghao Gao, Kaichen Zhang, Xinlong Chen, Ying Sun, and Hui Xiong. Forest-based graph learning for semi-supervised node classification. In *International Conference on Learning Representations*, 2026.

Huidong Liang, Haitz Saez de Ocariz Borde, Baskaran Sripathmanathan, Michael Bronstein, and Xiaowen Dong. Towards quantifying long-range interactions in graph machine learning: A large graph dataset and a measurement. In *International Conference on Learning Representations*, 2026.

Stefan Meinecke, Dzenan Sarajlic, Simon R. Drauz, Andreas Klettke, Lars-Peter Lauven, Christian Rehtanz, Albert Moser, and Martin Braun. Simbench–a benchmark dataset of electric power systems to compare innovative solutions based on power flow analysis. *Energies*, 13(12):3290, 2020.

Luca Miglior, Matteo Tolloso, Alessio Gravina, and Davide Bacciu. Can you hear me now? a benchmark for long-range graph propagation. In *International Conference on Learning Representations*, 2026.

Christopher Morris, Nils M. Kriege, Franka Bause, Kristian Kersting, Petra Mutzel, and Marion Neumann. Tudataset: A collection of benchmark datasets for learning with graphs. *ICML Workshop on Graph Representation Learning and Beyond*, 2020.

Judea Pearl. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann, 1988.

Ladislav Rampasek, Mikhail Galkin, Vijay Prakash Dwivedi, Anh Tuan Luu, Guy Wolf, and Dominique Beaini. Recipe for a general, powerful, scalable graph transformer. In *Advances in Neural Information Processing Systems*, 2022.

Martin Schmidt, Dirk Assmann, Robert Burlacu, Jesco Humpola, Imke Joormann, Nikolaos Kanelakis, Thorsten Koch, Daniel Oucherif, Marc E. Pfetsch, Lars Schewe, Robert Schwarz, and Marc Sirvent. Gaslib–a library of gas network instances. *Data*, 2(4):40, 2017.

Hamed Shirzad, Ameya Velingker, Balaji Venkatachalam, Danica J. Sutherland, and Ali Kemal Sinop. Exphormer: Sparse transformers for graphs. In *International Conference on Machine Learning*, 2023.

Kai Sheng Tai, Richard Socher, and Christopher D. Manning. Improved semantic representations from tree-structured long short-term memory networks. In *Annual Meeting of the Association for Computational Linguistics*, 2015.

Jake Topping, Francesco Di Giovanni, Benjamin P. Chamberlain, Xiaowen Dong, and Michael M. Bronstein. Understanding over-squashing and bottlenecks on graphs via curvature. In *International Conference on Learning Representations*, 2022.

Stelios G. Vrachimis, Demetrios G. Eliades, Riccardo Taormina, Avi Ostfeld, Zoran Kapelan, Shuming Liu, Marios Kyriakou, Pavlos Pavlou, Meirong Qiu, and Marios M. Polycarpou. Battle of the leakage detection and isolation methods. *Drinking Water Engineering and Science*, 15:1–17, 2022.

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

Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? In *International Conference on Learning Representations*, 2019.

Sanggeon Yun, Raheeb Hassan, Ryozo Masukawa, Sungheon Jeong, and Mohsen Imani. Hopformer: Sparse graph transformers with explicit receptive field control. *arXiv preprint arXiv:2602.02268*, 2026.

Arkaitz Zubiaga, Maria Liakata, Rob Procter, Geraldine Wong Sak Hoi, and Peter Tolmie. Analysing how people orient to and spread rumours in social media by looking at conversational threads. *PLOS ONE*, 11(3):e0150989, 2016.

A Proof of Exact Path-Kernel Factorization

We prove Proposition 1 channel-wise; fixing a channel turns all products into scalar products. For a node v, let T_v be its rooted subtree. After the bottom-up pass,

$$q_v = \sum_{j \in T_v} \left(\prod_{e \in P_T(j,v)} g_e^{\uparrow} \right) S_j, \quad m_v = \sum_{j \in T_v} \prod_{e \in P_T(j,v)} g_e^{\uparrow}. \tag{12}$$

The claim holds at leaves. If it holds for the children of v, substituting their expressions into Eq. (5) appends the final child-to-parent gate and partitions T_v into v plus its child subtrees, proving Eq. (12).

Assume the top-down state at parent p equals the path-kernel sum over its component. Terms whose source lies in child subtree T_v reach p through v and are exactly $g_v^{\uparrow} q_v$; subtracting this quantity leaves every source outside T_v exactly once. Multiplication by $g_v^{\downarrow}$ appends the parent-to-child factor to those paths. Adding q_v supplies all sources inside T_v, whose paths to v use only upward factors. Hence $t_v = \sum_j K_{vj} S_j$. The identical recurrence with unit sources gives $z_v = \sum_j K_{vj}$. The root initializes the induction, and Eq. (8) completes the proof.

B Forest Compilation and Permutation Audit

Our default compiler uses priority BFS. It finds connected components, selects the minimum-priority node as each root, computes BFS depth, and selects one preceding-level parent by edge priority. Priorities are random but fixed by a forest seed and are transported with nodes and edges under a permutation. DFS and feature-weighted minimum-spanning forests are also implemented. The layer never rebuilds the forest during a forward pass.

The permutation audit applies random node and edge reindexings, transports priorities and sparse structures, and aligns outputs back to the original order. It separately tests output sensitivity to changing the forest seed. On an undirected tree, changing only the root leaves the normalized all-pairs path kernel unchanged up to floating-point error when gate semantics are transported consistently. On cyclic graphs, changing the selected tree edges changes $P_T(j, v)$ and is expected to change the operator.

C Reference Forward Algorithm

Given X, a compiled forest (p, d), and $A_{\bar{E}}$, one GROOT layer executes:

1. Compute V, the corrected sources S, and both tree-edge gates with Eqs. (1)–(3). Initialize $q \leftarrow S$ and $m \leftarrow \mathbf{1}$.
2. For $h = H, H-1, \ldots, 1$, process all v with $d(v) = h$ in parallel and reduce $g_v^{\uparrow} q_v$ and $g_v^{\uparrow} m_v$ into parent states $(q_{p(v)}, m_{p(v)})$.
3. At each root, initialize $(t_r, z_r) \leftarrow (q_r, m_r)$.
4. For $h = 1, 2, \ldots, H$, process all v with $d(v) = h$ in parallel. Read the already-computed parent state, subtract the child's upward branch, and apply Eq. (7).
5. Normalize $U_v = t_v / (z_v + \varepsilon)$ and apply the residual output map in Eq. (9).

Nodes from distinct trees and packed graphs share depth buckets but never exchange state. The reference Torch implementation follows this schedule directly; the CUDA backend preserves the same data dependencies while fusing pointwise work and gradient reductions.

D DATASETS AND PROTOCOL DETAILS

Table 2: Role of each evaluated dataset. A schedule is the packed forest actually received by one scan; its node count can exceed that of an individual graph.

Dataset	Task and metric	Repetitions	Evidence role
Controlled path/cycle	Remote-bit recovery; accuracy	5 seeds	Receptive-field mechanism
ECHO-Synth SSSP	Shortest-path regression; MAE	5 seeds	Ablation and spectral boundary
SimBench Mixed	Bus-voltage field; v_m MAE	5 seeds	Fixed-topology continuous field
BattLeDIM	Missing pressure sensors; MAE (m)	5 seeds	Public cross-year recovery
GasLib-134	Gas-network state; NRMSE	5 seeds	Near-tree field estimation
LamaH	River forecasting/recovery; NSE@24h	5 seeds	Neutral/boundary evidence
PHEME-9	Leave-one-event-out stance; macro-F1	5 seeds	Cross-domain trend
PROTEINS	Graph classification; accuracy	5 outer seeds, 10 folds each	Graph-level aggregation
Peptides-func	Multilabel graph classification; AP	5 seeds	LRGB boundary
OGBN-Arxiv	Transductive node classification; accuracy	Audit run	Citation-network boundary
City-Networks	Eccentricity-derived node labels; accuracy	5 seeds	Large-graph long range

Training and selection. Formal seeds are $[0, 17, 42, 123, 2027]$. Data splits, forest seed, observation masks, and simulation noise are held fixed when training seed changes. Models on BattLeDIM, SimBench, GasLib, ECHO, and PHEME use a common maximum of 1,000 epochs and validation patience where applicable. A run reaching the cap is evaluated from its validation-best checkpoint and tagged `budget_limited=true`; this denotes an equal fixed-budget comparison, not convergence to an asymptotic optimum. City-Networks uses the benchmark's long 5,000-epoch protocol. No checkpoint is selected on test performance.

Statistical summaries. We report sample SD across training seeds. When models share seeds, paired differences and two-sided 95% Student-t intervals use the five paired values. The independent unit for PROTEINS is the outer training seed, not each of the ten folds. City rows marked as author-reported use different repetitions, so no paired interval is computed against them. The direct London GraphSAGE and IM-MPNN reruns use the same five seeds as GROOT; paired differences and intervals are therefore reported for those comparisons.

E Additional Results

E.1 Large road networks

Table 3: City-Networks test accuracy (%). "Reported" rows are five-run means from the dataset authors and are not paired with our runs. "Direct" rows use official code, our five fixed seeds, validation-selected checkpoints, and one frozen-test evaluation under the same 5,000-epoch budget as GROOT.

Model	Paris (114,127 nodes)	London (568,795 nodes)
GCN	53.2 ± 0.3	50.1 ± 0.7
GraphSAGE (reported)	54.6 ± 0.2	55.4 ± 0.2
GCNII	51.3 ± 0.2	48.2 ± 0.3
GraphGPS	52.1 ± 0.6	OOM
Exphormer	$\mathbf{55.1 \pm 0.8}$	49.5 ± 0.4
SGFormer	52.0 ± 0.8	48.3 ± 0.3
GraphSAGE (direct)	–	52.98 ± 0.75
IM-MPNN (direct)	–	57.02 ± 0.31
GROOT	53.24 ± 0.16	$\mathbf{57.96 \pm 0.30}$

For the direct London comparisons, GROOT beats GraphSAGE by 4.9809 percentage points on average (paired 95% t interval $[3.8641, 6.0978]$) and IM-MPNN by 0.9385 points ($[0.3535, 1.5234]$), winning all five seeds in both cases. GraphSAGE, IM-MPNN, and GROOT all reach late validation optima and are marked `budget_limited=true`; the comparison therefore holds at the shared fixed budget. Mean training wall time is 2,786.6 s for GraphSAGE and 7,386.6 s for IM-MPNN, with peak allocated memory of 19,812 and 11,824 MiB, respectively. GROOT uses about 40,480 MiB on London, so its accuracy advantage comes with a substantial memory cost.

E.2 Forest compilation over graph collections

Forest construction is performed once per graph and cached. Compiling the 1,113 PROTEINS graphs takes 2.611 s on CPU and loading the cache takes 0.197 s; compiling the 10,873 Peptides-func training graphs takes 62.118 s and loading takes 2.252 s. With cached structures, median warm epochs are 0.0525/0.0503/0.1678 s for GCN/GIN/GROOT on PROTEINS and 1.2175/3.8122 s for GCN/GROOT on Peptides-func. Thus preparation is amortizable, but cached end-to-end GROOT epochs remain 3.13–3.34$\times$ slower than the local baselines. These measurements exclude first-time CUDA extension compilation and use the same resident structures and batch order within each dataset.

E.3 Exact mechanism-ablation values

Table 4: Five-seed mechanism ablations (MAE ↓) underlying Figure 4. "No source" removes Eq. (1); "uniform" removes learned input-dependent gates; "shared direction" keeps learned node/channel gates but uses the same realized value in both directions.

Variant	ECHO-Synth test	BattLeDIM development
Full GROOT	0.6028 ± 0.0046	$\mathbf{0.0266 \pm 0.0043}$
No source correction	1.8053 ± 0.0247	0.0267 ± 0.0018
Uniform gate	1.0450 ± 0.0273	0.1443 ± 0.0356
Shared-direction gate	$\mathbf{0.5881 \pm 0.0093}$	0.0308 ± 0.0100
Upward only	4.7701 ± 0.0074	0.1613 ± 0.0116
Downward only	4.7340 ± 0.0106	0.1402 ± 0.0214

E.4 NEUTRAL AND BOUNDARY TASKS

Table 5: Additional five-seed scope results spanning neutral, positive-trend, and boundary regimes.

Task / metric	Model	Mean ± sample SD
LamaH standard, NSE ↑	DenseGCN	0.5251 ± 0.0105
	SGFormer	0.5225 ± 0.0220
	GROOT	0.5273 ± 0.0128
LamaH connected outage, NSE ↑	SGFormer	0.1090 ± 0.0552
	GROOT	0.1306 ± 0.0866
LamaH random outage, NSE ↑	SGFormer	0.3692 ± 0.0261
	GROOT	0.3885 ± 0.0193
PHEME-9, macro-F1 ↑	GCN	0.2676 ± 0.0153
	Virtual-GCN	0.2534 ± 0.0127
	SGFormer	0.2765 ± 0.0124
	GROOT	0.2835 ± 0.0190
Peptides-func, AP ↑	GCN	0.5842 ± 0.0113
	Virtual node	0.5791 ± 0.0107
	GROOT	0.6064 ± 0.0091

On PHEME, GROOT wins four of five seeds against each listed baseline, with paired 95% intervals crossing zero. On LamaH connected outage it wins one of five seeds against SGFormer; on random outage it wins four of five and the interval crosses zero. These cases support trends or parity under the present sample size.

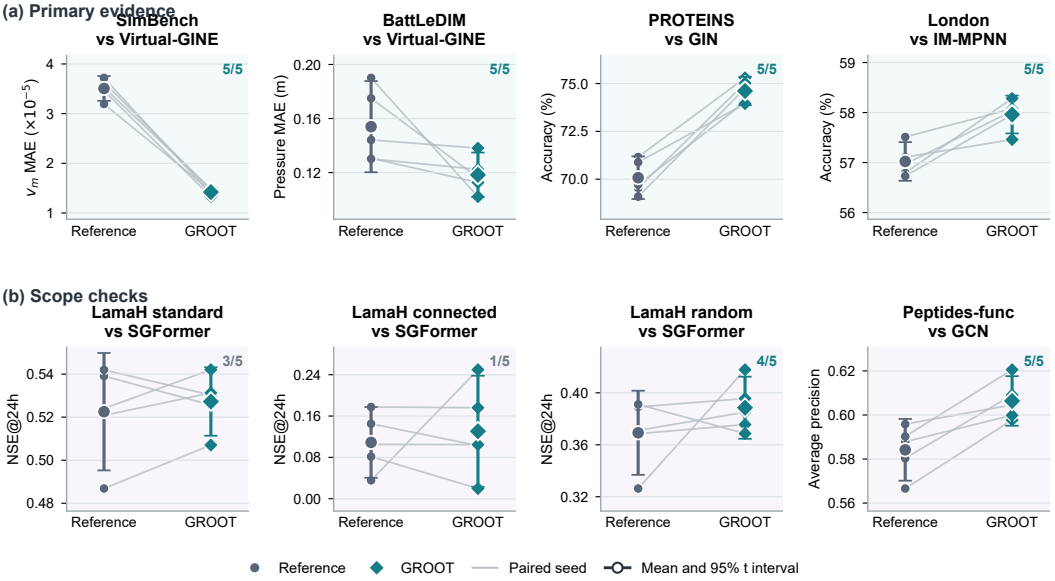


Figure 6: Paired five-seed outcomes across primary and scope tasks. Each thin segment connects the reference model and GROOT trained with the same seed; large markers and whiskers show the mean and two-sided 95% Student-t interval for each model. The upper-right count is the number of paired seeds won by GROOT under the task's native metric direction. For PROTEINS, each point is one outer training seed aggregated over its ten folds.

E.5 DEPTH, EFFECT, AND COST

The depth-response study uses BattLeDIM 2018 development data with the five training seeds in Appendix D; it is a mechanism analysis rather than a frozen-test main result. Figure 7 compares

predictive effect with exclusive-GPU cost for four complementary architectures. GROOT trails the strongest global baseline at one and two layers, then obtains the lowest mean MAE at four and eight layers. At four layers, it wins all five paired seeds against both Virtual-GINE and SGFormer, with 95% intervals for the paired MAE differences of $[-0.0364, -0.0106]$ and $[-0.0331, -0.0101]$. At eight layers, it wins all five seeds against SGFormer and four of five against Virtual-GINE; the latter interval $[-0.0189, 0.0039]$ crosses zero. Four of five GROOT runs reach the 1,000-epoch cap at both four and eight layers; four Virtual-GINE runs do so at eight layers. The comparison therefore describes a common fixed training budget.

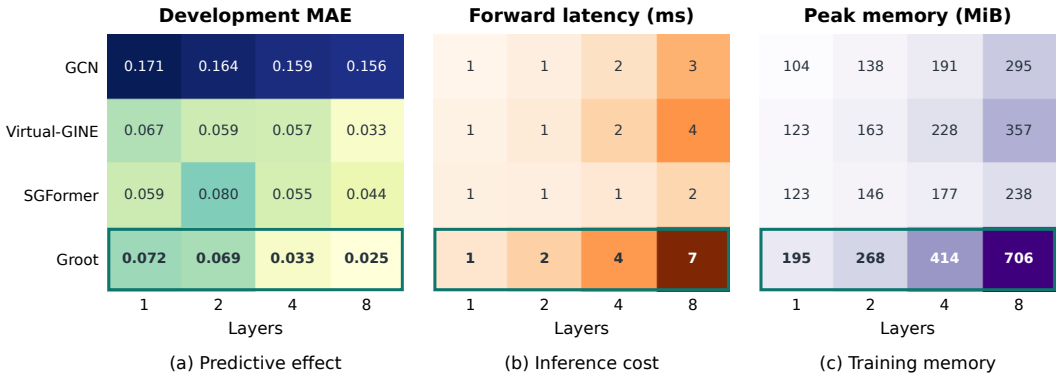


Figure 7: Depth–effect–cost audit on BattLeDIM. Each cell reports the absolute value, and the outlined row identifies GROOT. (a) Development MAE is the mean over five training seeds. (b,c) Forward latency and peak allocated training memory use a fixed 25,120-node packed batch on an exclusive NVIDIA L40, with 10 warmups and 30 CUDA-event repetitions.

Table 6: BattLeDIM development MAE (five-seed mean ± sample SD) by depth. Bracketed values are parameter counts in thousands; lower MAE is better.

Model	1 layer	2 layers	4 layers	8 layers
GCN	.1712 ± .0096 [10.3]	.1643 ± .0080 [18.6]	.1588 ± .0085 [35.4]	.1563 ± .0050 [68.9]
Virtual-GINE	.0671 ± .0054 [10.1]	.0586 ± .0139 [18.6]	.0569 ± .0077 [35.5]	.0325 ± .0059 [69.4]
SGFormer	.0587 ± .0068 [15.0]	.0796 ± .0187 [17.9]	.0549 ± .0099 [23.6]	.0438 ± .0041 [35.0]
GROOT	.0721 ± .0074 [10.3]	.0691 ± .0144 [18.6]	.0333 ± .0056 [35.2]	.0250 ± .0058 [68.5]

Table 7 measures the same four layer stacks on a fixed packed batch of 25,120 nodes. Each cell reports forward latency, full AdamW-step latency, and peak allocated memory. The timing run uses an exclusive NVIDIA L40, 10 warmups, and 30 CUDA-event repetitions. GROOT's forward, training-step, and memory ratios relative to the nearly parameter-matched GCN remain within 2.24–2.48×, 2.22–2.52×, and 1.87–2.39×, respectively, across the depth sweep.

Table 7: Depth-cost audit. Each cell is forward ms / AdamW-step ms / peak MiB; lower is better.

Model	1 layer	2 layers	4 layers	8 layers
GCN	.579 / 1.906 / 104	.931 / 2.819 / 138	1.639 / 4.622 / 191	3.060 / 8.258 / 295
Virtual-GINE	.763 / 2.731 / 123	1.150 / 5.823 / 163	2.170 / 5.610 / 228	3.838 / 11.074 / 357
SGFormer	.860 / 3.464 / 123	.942 / 3.814 / 146	1.297 / 4.982 / 177	2.031 / 7.578 / 238
GROOT	1.433 / 4.225 / 195	2.239 / 7.100 / 268	3.675 / 10.778 / 414	7.490 / 20.715 / 706

E.6 ONE-MILLION-NODE RAW SCALING POINTS

Table 8: Range of GROOT cost ratios at $N = 10^6$ across degree $4/8/16$ and width $64/128$. Lower is better. Each cell uses two warmups and five measured CUDA-event repetitions with resident structures and optimizer state.

Reference	Forward	AdamW step	Peak memory
GCN	1.80–2.07×	1.68–2.25×	1.70–2.16×
SGFormer	0.87–1.00×	0.78–0.84×	0.98–0.99×

Table 9: Representative $N = 10^6$, degree 8 scaling points on an exclusive NVIDIA L40. Values are forward milliseconds / AdamW-step milliseconds / peak MiB.

Model	$D = 64$	$D = 128$
GCN	18.20 / 52.82 / 2,464	35.24 / 91.54 / 4,418
Virtual-GINE	65.51 / 127.55 / 10,761	113.94 / 235.24 / 21,011
SGFormer	38.89 / 127.00 / 4,945	73.68 / 226.25 / 9,340
GROOT	34.69 / 98.57 / 4,897	68.97 / 188.91 / 9,291

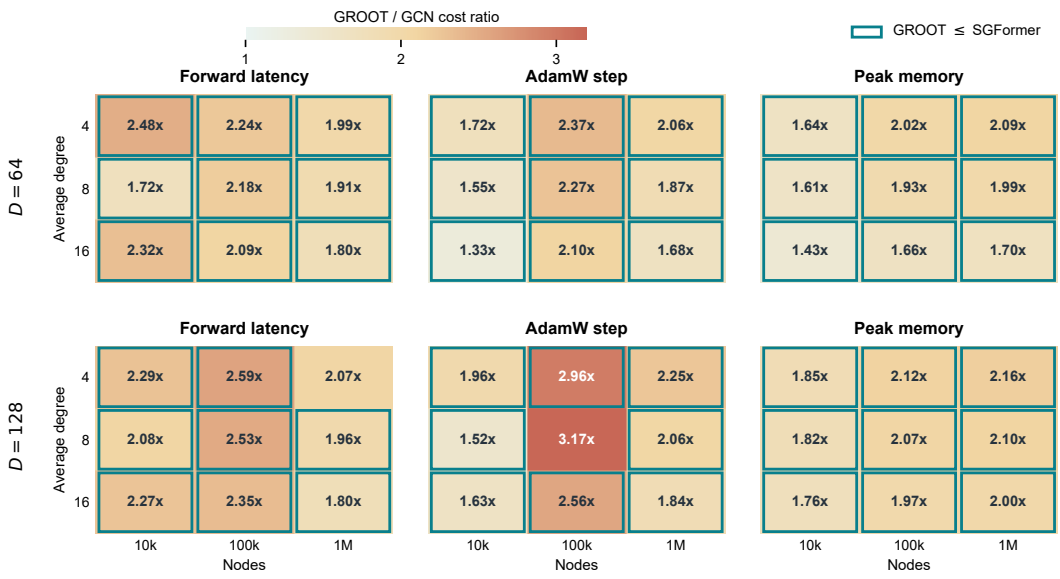


Figure 8: Complete GROOT scaling surface on an exclusive NVIDIA L40. Cell color and text report the GROOT-to-GCN ratio for forward latency, full AdamW-step latency, and peak allocated memory; lower is better. A teal outline marks configurations where GROOT costs no more than SGFormer for the same metric. The 10k/100k/1M-node cases use 5/3/2 warmups and 20/10/5 measured CUDA-event repetitions, respectively, with resident graph structures and optimizer state.

F CUDA Timing Scope and Forest Height

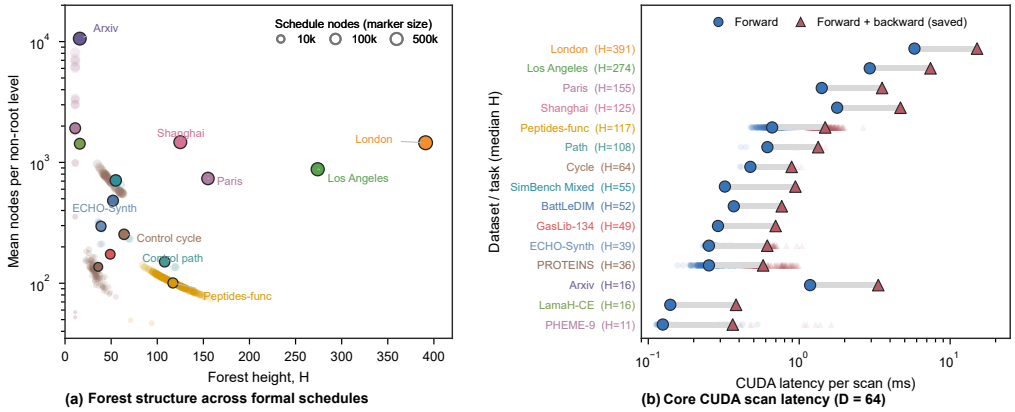


Figure 9: Forest structure and core CUDA scan latency over all formal schedules. (a) Height versus average within-level parallelism; translucent points are schedules and outlined points are dataset medians. (b) Dataset-level median forward and saved-state forward+backward latency at $D = 64$. Inputs and forests are resident on the GPU. The schedule-level Spearman correlation between height and forward latency is 0.938; it is descriptive because schedules share datasets and latency also depends on node count and within-level width.

The forest-height diagnostic covers 11 formal dataset groups, 32,063 instance forests, and 466 packed schedules. Every schedule satisfies $\sum_h |L_h| = N$ and matches its archived forest SHA-256. Core scans use $D = 64$, five warmups and 20 measured repetitions in each of two independent exclusive timing blocks. Inputs and forests are resident on the GPU. The measurement excludes value/output projections, non-tree SpMM, optimizer updates, forest construction, and stacking multiple layers. Figure 9 keeps all schedules; no instance is selected by latency.

Height correlates with latency, but node count and within-level width also matter: a controlled 16,384-node path with median $H = 108$ is 29.7% slower than a cycle with $H = 64$; ECHO and Peptides have similar packed node counts, yet Peptides ($H = 117$) is 2.62× slower than ECHO ($H = 39$). London reaches $H = 391$ and 568,795 nodes, but one core width-64 scan remains 5.80 ms forward and 15.07 ms including exact backward. These are single-scan measurements, not complete 16-layer model latency.

G Implementation Verification

Tests establish explicit-kernel, Torch, and CUDA output/gradient agreement across sizes, widths, dtypes, gates, packed unequal-depth components, and workspace/recomputation modes. On a 16-layer City-Networks Paris run, recomputation preserves epoch-one loss and accuracy while reducing peak allocation from 10,575.9 to 8,167.7 MiB (22.8%).