DeltaNAR: Learning What to Reuse and What to Recompute in Neural Algorithmic Reasoning

Anonymous submission

Abstract

Neural Algorithmic Reasoning (NAR) usually solves every instance from scratch, although a small edit often leaves most of a previous solution valid. We study this incremental setting and introduce CLRS-DELTA, where a post-edit graph, the edit, and the previous solution are inputs, and the updated solution and affected region are targets. Our model, DELTANAR, combines an edit encoder, affected-region and copy-safety gates, and an explicit copy-not-recompute decoder. A node-level bound predicts that, under local edits, expected error is controlled by the affected fraction rather than graph size. Trained only at $n = 16$, DELTANAR attains 0.987 ± 0.028 SSSP node validity at $n = 256$ over 20 seeds, while matched from-scratch and richer processor baselines collapse. An edit-conditioned hard slice makes copying nontrivial: at $n = 256$ every test graph invalidates at least one old predecessor, yet DELTANAR reaches 0.846 SSSP node validity versus 0.426 for previous-solution features without explicit copying and 0.042 for recomputation. Pure copy still reaches 0.992 in the node average but has zero graph validity, exposing why both metrics are needed. Against an official-architecture implementation of the recent DEAR reasoner on the matched post-edit distribution, DELTANAR remains above 0.97 in an independent $n = 256$ rerun where DEAR reaches 0.287. Paired bootstrap intervals exclude zero for the headline comparisons. Results on max–min bottleneck paths and broad distribution shifts support generality, while low whole-graph exactness and stale-memory audits delimit the current method. **Content areas:** ML, KRR, SO.

Introduction

Neural Algorithmic Reasoning (NAR) asks whether neural networks can learn the computational structure of classical algorithms and generalize beyond the training distribution (Veličković and Blundell 2021; Veličković et al. 2022). The dominant experimental regime is from-scratch computation: each input instance is presented independently, and the model predicts the output or execution trace of running an algorithm on that instance. This abstraction has driven useful progress, but it omits a pervasive structure in real algorithmic systems. Graphs do not arrive once and stay fixed. Edge weights are updated, links fail, roads open, constraints are inserted, and systems must repeatedly revise a solution that was already computed.

This paper's position is that incremental updates should be a first-class generalization test for NAR. A model that can solve isolated instances has not necessarily learned how algorithmic state should persist under perturbation. In dynamic settings, the central object is not only the final answer, but the validity boundary between the part of the previous certificate that remains safe and the part that must be repaired.

Classical algorithms treat this as a separate regime. Dynamic shortest-path methods, incremental graph algorithms, and replanning systems reuse previous solutions and process locally inconsistent regions after an edit (Ramalingam and Reps 1996; Koenig, Likhachev, and Furcy 2004; Koenig and Likhachev 2002). Their work can be output-sensitive: after a local update, cost scales with the changed region rather than the full graph. This is precisely the kind of algorithmic structure NAR should be able to exploit, yet standard CLRS-style benchmarks do not expose it. A from-scratch NAR model is never asked whether it can preserve a correct previous solution, detect what changed, and recompute only where needed.

We introduce CLRS-DELTA, a benchmark regime for incremental NAR. Each sample has the form $(G_t, y_{t-1}, \Delta_t) \mapsto (y_t, A_t)$, where G_t is the post-edit graph, y_{t-1} is the previous solution, Δ_t is a local graph edit, y_t is the updated solution, and A_t is an affected-region supervision signal. The task is not temporal graph prediction: the model is not asked to forecast future labels or edges. It is asked to execute a known algorithmic update under supervision from verified ground-truth engines.

Our model, DELTANAR, is built around the principle *copy what did not change; recompute what did*. It uses previous-solution memory, an edit encoder, an affected-region gate, a copy-safety gate, and a copy-not-recompute decoder. The affected gate localizes work; the copy-safety gate asks whether the old witness remains trustworthy. Safe vertices receive an explicit bias toward the previous predecessor or edge state, while unsafe vertices are repaired by local semiring message passing. The semiring operators themselves are not the novelty; they are aligned algorithmic building blocks (Veličković et al. 2020; Xu et al. 2020). The novelty is the incremental input contract and the gated copy-safety mechanism that turns

a previous solution into a first-class computational resource.

The headline result is size-OOD generalization in the regime where from-scratch models fail. Training only on $n = 16$ graphs, DeltaNAR obtains 0.987 ± 0.028 SSSP node accuracy at $n = 256$ over 20 seeds. From-scratch recomputation, affected-gated recomputation, attention, and triplet edge reasoning all degrade sharply. The advantage persists on a conditional hard slice in which every graph contains a stale predecessor, and against a recent equilibrium reasoner adapted to the same post-edit distribution. Paired resampling quantifies uncertainty rather than relying on mean gaps alone. The same mechanism extends to bottleneck paths under a max–min semiring and remains robust under topology, weight, failure, edit-type, and intensity shifts.

One striking aspect of the SSSP curves is that node accuracy can improve as test graphs become larger. This is not magic extrapolation in the usual sense: it is the theorem's denominator effect. Under sparse local edits, the number of vertices that need repair can remain small while the number of unchanged reachable vertices grows. A model that can reliably copy unchanged witnesses can therefore become more accurate in node average even though the graph is larger. This is also why we report graph accuracy separately. Whole-graph exactness does not benefit from this averaging effect and exposes seed variance at large size.

Our contributions are fivefold. First, we define incremental NAR as a supervised algorithmic-update problem, where the previous solution and graph edit are first-class inputs rather than hidden temporal context. Second, we instantiate the regime in CLRS-Delta with three verified tasks: dynamic SSSP, bottleneck paths, and incremental MST. Third, we introduce DeltaNAR, whose affected-region gate, copy-safety gate, and copy-not-recompute decoder encode the output-sensitive structure of dynamic algorithms. Fourth, we prove a node-level bound explaining why size-OOD node error scales with the affected fraction. Fifth, we test the claim with multi-seed size and distribution shifts, a matched recent reasoner, an edit-conditioned hard slice, paired bootstrap intervals, independent stability reruns, noisy-memory audits, generalist and transfer experiments, and MST as a negative pressure test.

Related Work

Neural algorithmic reasoning. NAR investigates whether neural networks can learn algorithmic procedures and extrapolate beyond the training distribution (Veličković and Blundell 2021). CLRS provides a broad benchmark with final-output and hint supervision for classical algorithms (Veličković et al. 2022). Recent extensions broaden the benchmark or training setting, including reinforcement learning formulations and pseudo-polynomial tasks (Schutz et al. 2025; Požgaj et al. 2025). These works solve each instance from scratch. CLRS-Delta keeps CLRS's algorithmic-supervision philosophy but changes the computational contract: the previous solution and the edit are part of the input.

Recent work also changes how algorithmic execution is represented. Parallelized supervision can align neural execution with parallel algorithms (Engelmayer, Georgiev, and Veličković 2024); multiple-correct-solution training avoids penalizing alternative valid outputs (Kujawa et al. 2025); and Deep Equilibrium Algorithmic Reasoning (DEAR) replaces a fixed recurrent depth with an equilibrium processor (Georgiev et al. 2024). These advances remain complementary to incremental reuse: they do not decide which part of a previous certificate remains valid after an edit. We therefore adapt the official DEAR architecture to the post-edit distribution as a representative recent from-scratch comparison, while preserving its information contract: it sees the edited graph and source, but not the old solution or edit metadata.

Algorithmic alignment and dynamic algorithms. Message passing can align with dynamic-programming recurrences such as Bellman-Ford relaxation (Veličković et al. 2020; Xu et al. 2020). Graph neural networks and attention provide flexible processors (Gilmer et al. 2017; Veličković et al. 2018). DeltaNAR uses min-plus and max-min processors as aligned components, but does not claim these operators as novel. The new mechanism is the learned decision to copy or recompute. Classical dynamic algorithms exploit an analogous output-sensitive structure: Ramalingam-Reps-style shortest-path updates and LPA*/D* Lite process locally inconsistent states rather than recomputing the whole problem (Ramalingam and Reps 1996; Koenig, Likhachev, and Furcy 2004; Koenig and Likhachev 2002). We bring this perspective into NAR by giving the model the previous solution and asking it to update.

This distinguishes our setting from dynamic or temporal graph representation learning (Kazemi et al. 2020; Rossi et al. 2020). Those models usually learn embeddings for graphs whose edges and attributes evolve, and their objective is often link prediction, classification, or forecasting. CLRS-Delta instead fixes the algorithmic target after each edit. The model is evaluated by whether its predicted witness satisfies the same invariant as a classical algorithm. The previous solution is therefore not historical context for a statistical forecast; it is a certificate-like object that may remain partly valid and partly stale after a local perturbation.

CLRS-Delta Benchmark

Each sample consists of a graph after an edit, the previous solution, and the edit event. The model predicts both the updated solution and an affected-region hint. The edit can be a weight increase, weight decrease, edge addition, or edge deletion. Multi-edit intensity experiments apply several simultaneous edits. All ground-truth engines are deterministic and cross-checked against from-scratch oracles.

The target is not a future graph state. It is the post-edit algorithmic solution. This distinction matters: temporal graph models can succeed by learning correlations in evolving data, whereas CLRS-Delta requires the model to preserve algorithmic invariants under perturbations. The affected-region target also gives a direct way to ask whether a model has localized the update, not merely guessed the final answer.

We generate each example by first solving a base graph, applying a controlled edit, and then solving the edited graph with a dynamic engine and an independent from-scratch oracle. A sample is accepted only when the updated output agrees with the oracle under the task's tie-tolerant validity

rule. This filtering is important because shortest paths and bottleneck witnesses can be non-unique: the model should be penalized for invalid witnesses, not for choosing a different valid predecessor. The benchmark therefore separates two questions that are often conflated in graph learning: whether the final witness is algorithmically valid, and whether the model has identified the region where the old witness might no longer be safe to copy.

The affected target is deliberately operational. For SSSP, it is derived from the work performed by the LPA*-style update engine rather than from a purely syntactic comparison of old and new labels. Thus a vertex may be affected because it was reconsidered during repair even if its final distance does not change. This makes the hint a closer analogue of an incremental algorithm's work-list and prevents the model from learning only a final-label delta detector.

Providing y_{t-1} is also not label leakage, because the label to be predicted is y_t. The previous solution is often correct on most vertices, but exactly which vertices remain safe is the algorithmic problem created by the edit. A pure copy model fails when the changed region grows or when a structural edit invalidates a predecessor chain. A pure recompute model ignores the strongest piece of information in the input and must solve the full edited instance from scratch. CLRS-DELTA is designed to make both extremes visible.

Dynamic SSSP. The input is a directed weighted graph with a fixed source. The target is the updated shortest-path distance and predecessor for every reachable non-source vertex. We use an LPA*-style engine with heuristic zero. It maintains g and rhs values, updates only locally inconsistent vertices, and outputs the set of vertices popped from the priority queue as the affected-region target. Final distances and predecessors are verified against from-scratch Dijkstra.

This task is the closest neural analogue of classical dynamic shortest-path maintenance. Weight decreases can create improved paths that propagate outward; weight increases and deletions can invalidate a branch and require alternative predecessors. The affected hint therefore contains both improvement and repair events, making the task more informative than simply predicting whether a final distance changed.

Bottleneck paths and MST. Bottleneck replaces min-plus shortest paths by max-min widest paths, $B(v) = \max_u \min(B(u), w(u, v))$, testing whether the mechanism transfers across semirings. Incremental MST predicts updated edge membership on undirected weighted graphs. Whole-tree exactness is difficult, so MST is used as a diagnostic difficulty gradient rather than the primary success claim.

The three tasks form a spectrum. SSSP is the cleanest alignment with min-plus relaxation and gives the headline result. Bottleneck changes the semiring while keeping a predecessor-witness structure. MST changes the output type to an edge set, stressing whether local edge predictions can compose into a global combinatorial object.

This spectrum is useful for positioning the contribution. If the method worked only on SSSP, it could be dismissed as a shortest-path-specific trick. If it claimed full MST solution quality, the evidence would overstate what the current model achieves. Reporting all three gives a more honest map: strong success on the clean dynamic path problem, meaningful semiring transfer to bottleneck, and a harder global-structure case where edge-level accuracy is high but exact whole-tree recovery remains open.

Metrics. The primary metric is `node_acc_v`, a tie-tolerant node-level validity metric. For SSSP, a predicted predecessor is valid if it yields the correct shortest distance; for bottleneck, if it yields the correct max-min value. We also report `graph_acc_v`, the whole-graph exact version. Graph accuracy is much harsher and high-variance at large n, so it is treated as secondary.

This choice is not merely convenient. In incremental algorithms, a local edit often changes a small part of the output, so the natural question is how many output locations remain valid after an update. Whole-graph exactness is still important, but it asks for every reachable witness to be simultaneously correct. For bottleneck and MST, this compounding effect makes graph accuracy collapse even when per-node or per-edge accuracy remains high. We therefore report both, but use `node_acc_v` as the metric aligned with the mechanism and theorem.

Because local-update metrics are copy-dominated by design, we also audit two harder diagnostic slices in the supplement: changed-output vertices $C_t = \{v : y_t^*(v) \neq y_{t-1}^*(v)\}$ and copy-critical vertices where blindly copying the previous witness is invalid. These slices distinguish successful preservation of unchanged structure from genuine repair of stale witnesses, and they calibrate the server-scale diagnostic runs.

DELTANAR

DELTANAR follows an encode-process-decode architecture. The encoder consumes the post-edit graph, previous distances or predecessor/edge states, and an edit feature indicating where and how the graph changed. The processor applies message passing with min and max aggregation. The decoder predicts the updated solution and an affected-region logit.

The input contract is the main architectural departure from from-scratch NAR. The graph state is already post-edit, so the network does not need to simulate the edit operation itself. Instead, the edit feature tells it where the update entered the graph, and the previous solution provides a candidate answer that is mostly correct. This encourages the processor to spend capacity on deciding which local region needs repair.

Previous-solution memory and update gates. Unlike from-scratch NAR models, DELTANAR receives y_{t-1} explicitly. For path tasks, this includes previous distances and predecessor one-hot features. For MST, it includes previous edge membership. The model also predicts an affected logit for each vertex and is supervised against the verified affected region. Since affected vertices are sparse, the positive class is upweighted. In the calibrated model, a second logit predicts whether the previous witness is safe to copy. The affected gate estimates where the edit should propagate; the copy-safety gate estimates whether memory should dominate decoding.

DeltaNAR update rule. Given (G_t, y_{t-1}, Δ_t): encode the post-edit graph, previous witness, and edit endpoints; propagate local semiring messages from the edit frontier; predict affected logits a_v and copy-safety logits s_v; copy $y_{t-1}(v)$ when s_v is high; otherwise repair v with the local predecessor/edge decoder.

Figure 1: Conceptual update rule. The affected gate localizes work, while the copy-safety gate decides whether previous memory is a certificate or a stale witness.

A false negative on an unsafe vertex is the dangerous error: it copies a stale solution. A false positive is less harmful if recomputation is reliable, but can still introduce errors, which is why the theory keeps an explicit false-positive recomputation term.

Copy-not-recompute decoder. For path tasks, the decoder scores candidate predecessors. If a vertex is predicted safe to copy, the previous predecessor receives an additive copy bias:

$$\mathrm{score}(v,u) \leftarrow \mathrm{score}(v,u) + \lambda\,\sigma(s_v - m)\mathbb{1}[u = y_{t-1}(v)].$$

Here s_v is the copy-safety logit and m is a conservative margin; without the separate safety head, $s_v = -a_v$. This makes the original affected gate a special case and lets the calibrated model separate localization from trust. For MST, an analogous edge bias is applied when both endpoints are predicted unaffected. A larger margin makes copying more conservative on harder semirings.

Operationally, DeltaNAR can be read as a neural analogue of an incremental algorithm's work-list. The edit encoder marks where inconsistency enters. The processor propagates local evidence. The affected logit estimates whether the previous output at a vertex may need work, and the copy-safety logit estimates whether the old witness can still be trusted. The decoder then converts these estimates into an output decision: copy if trustworthy, recompute otherwise. This differs from simply concatenating the previous solution as an input feature, because the architecture gives the previous solution a privileged path to the output.

Baselines and ablations. We compare against from-scratch Recompute, gated from-scratch Recomp+gate, attention aggregation, a triplet edge-reasoning baseline, and copy/processor ablations. We also include a Prev+Delta baseline that receives the previous solution and edit as ordinary features but has no privileged copy decoder. This is important for the core scientific question: if copy were trivial, a stronger processor, gated from-scratch baseline, or feature-only previous-solution model should solve the task. They do not.

Naive copying is also not enough. The affected region is learned, not given at test time, and our A-suite shows that accuracy decreases smoothly as affected-region size grows. The useful inductive bias is the combination: previous-solution memory, learned affected localization, and local recomputation where copying would be stale.

The baseline set is designed to isolate three hypotheses. Recompute tests whether ordinary NAR extrapolates in this dynamic setting. Recomp+gate tests whether affected supervision alone is sufficient. Prev+Delta tests whether previous-solution access as a feature is enough. Triplet and Attn test whether processor expressivity can replace the update-specific inductive bias. The copy/no-copy comparison then tests the specific decoder path. The pattern across these controls is what makes the mechanism identifiable.

The method is intentionally close to classical update logic but remains learned at the points where the benchmark makes learning meaningful. The model does not receive the correct affected mask at test time, does not receive a dynamic-programming trace, and does not receive the updated predecessor. It must infer where the previous solution is stale from the edit and the post-edit graph. Conversely, we do not obscure standard algorithmic structure: min-plus and max-min aggregation are used because they align with the target recurrences. This division is the intended research question: can a neural reasoner learn the control decision that dynamic algorithms make, namely which part of a previous solution to trust?

Theory

Let S_t be the evaluation domain, matching the reachable non-source mask used by `node_acc_v`. Let $A_t \subseteq S_t$ be a copy-critical region that contains every vertex for which blindly copying y_{t-1} is not guaranteed to be safe. The smallest such set is $\{v : y_t^*(v) \neq y_{t-1}^*(v)\}$, while the implementation may supervise a larger LPA*-style work-set. Vertices outside A_t can be copied safely if the previous solution is exact. Let γ lower-bound average correctness on vertices in A_t after the gate/recompute path, and let η be the error rate caused by false-positive recomputation outside A_t.

Theorem 1 (Node-level update bound)**.** *Assume the previous solution is exact on S_t, vertices outside A_t can be copied safely, false-positive recomputation error is at most η, and the average correctness on vertices in A_t is at least γ. If $\mathbb{E}|A_t| \le a$ and $|S_t| \ge cn$ almost surely, then*

$$\mathbb{E}[\mathrm{node\_acc}_t] \ge 1 - \frac{a}{cn}(1-\gamma) - \eta.$$

The proof decomposes node error over affected and unaffected vertices. An unaffected vertex is correct when copied. Inside A_t, we make only the conservative assumption that the model is correct with probability at least γ on average; if A_t is exactly the changed-output set, this event can be read as gate recall times recomputation success. If local edits have output-sensitive affected regions, a does not grow with n, so excess node error above η scales as $O(a/n)$. We keep η in the theorem: dropping it assumes either no false positives or safe recomputation outside A_t. Rollout contraction is also not unconditional; a wrong copied value can persist unless future updates revisit it. Full proof details and gate-recall conditions appear in the supplement.

The theorem explains the central size-OOD pattern. If a local edit produces an affected region whose expected size is roughly independent of n, the copied unaffected region grows with the graph. A recompute-from-scratch model has no such decomposition: it must propagate enough information to solve the whole instance, so increasing n increases

the effective reasoning horizon. This is why the bound predicts the observed separation between DELTANAR and from-scratch baselines.

The proof is deliberately simple. Average node error over S_t. Vertices outside A_t are safe to copy by the exact-previous-solution assumption; their only contribution is false-positive recomputation error η. Vertices inside A_t occupy at most an $|A_t|/|S_t|$ fraction of the metric and have average error at most $1-\gamma$. Summing these two cases yields the bound. The supplement expands this argument, relates γ to gate recall and recomputation accuracy, and states the rollout accumulation bound.

Two details are worth making explicit in the main text. First, A_t is defined over the evaluation mask rather than all graph vertices. This matches the metric and avoids counting unreachable vertices that have no supervised predecessor. Second, the theorem is a node-accuracy result, not a graph-accuracy result. Whole-graph exactness would require every node-level event to succeed simultaneously; even small independent node error can compound as n grows. This is why high node accuracy and low bottleneck graph accuracy can coexist without contradicting the theorem.

The bound also clarifies the role of processor depth. Copying handles vertices whose witnesses are unchanged, but affected vertices still require enough local propagation to repair their dependencies. If the update has dependency depth d from exact copied boundary values, then a message-passing processor with fewer than d useful steps can fail even with a perfect gate. In our controlled SSSP extreme-size run we therefore set $T = n$ to remove the artificial step cap; the resulting 0.998 node accuracy at $n = 512$ is consistent with the theorem's locality condition. When T is capped, failures should be attributed to both localization and insufficient repair depth.

Experiments

We train on $n = 16$ graphs unless otherwise noted. Primary metric is `node_acc_v`; `graph_acc_v` is reported as strict secondary evidence. Results are mean $\pm$ population standard deviation over seeds.

The experimental design asks five questions. First, does the method extrapolate in graph size when trained only on small graphs? Second, is copy-not-recompute the causal component, or can stronger from-scratch processors match it? Third, does the mechanism survive shifts in topology, weights, failures, edit type, and edit intensity? Fourth, does it extend beyond a single semiring through generalist and transfer experiments? Fifth, does the advantage survive a recent reasoner, a slice that forces stale witnesses, and paired uncertainty analysis? The supplement gives full tables, protocols, and commands.

All baselines are trained under the same size regime as DELTANAR unless explicitly stated. Recompute receives the edited graph but not a privileged copy path, so it tests whether ordinary algorithmic reasoning can simply solve the post-edit instance. Recomp+gate adds affected-region supervision but still lacks previous-solution copying. Triplet constructs richer edge-level reasoning features, and attention replaces the min/max aggregation path with learned attention. These

Task	n	node	graph
SSSP	16	0.990 ± 0.002	0.888 ± 0.026
SSSP	64	0.997 ± 0.002	0.923 ± 0.073
SSSP	256	0.987 ± 0.028	0.774 ± 0.364
Bottleneck	16	0.978 ± 0.005	0.768 ± 0.054
Bottleneck	64	0.950 ± 0.047	0.129 ± 0.062
Bottleneck	256	0.899 ± 0.133	0.014 ± 0.018

Table 1: Main DELTANAR results over 20 seeds. Node accuracy is the primary metric; graph accuracy is stricter and high-variance at large n.

	node validity		graph validity	
n	DELTANAR	DEAR	DELTANAR	DEAR
16	.991	.520	.895	.010
64	.997	.406	.952	.000
128	.992	.355	.936	.000
256	.972	.287	.885	.000

Table 2: Recent reasoner comparison on post-edit SSSP. DEAR is the official architecture trained on the matched graph distribution (5 seeds, 10,000 graphs/seed); it does not receive the old solution or edit metadata. DELTANAR is an independent 40-seed stability rerun (20 at $n = 16$).

are deliberately strong controls: the comparison is not against a weak MPNN, but against models that receive either more supervision, more processor flexibility, or more explicit pairwise structure. We additionally adapt the official DEAR architecture (Georgiev et al. 2024) to matched post-edit SSSP graphs. DEAR receives the edited graph and source and is trained on 10,000 graphs per seed, but receives neither the edit nor the old solution; this compares a recent from-scratch reasoner with the incremental information contract, rather than serving as an architecture-only ablation.

Size extrapolation. Figure 2 and Table 1 show the main result. On SSSP, DELTANAR stays near-perfect through $n = 256$: 0.987 ± 0.028 node accuracy over 20 seeds. In the matched B-line baseline grid, Recompute falls to 0.053, Recomp+gate to 0.236, and Prev+Delta to 0.382 at $n = 256$. Triplet drops from 0.943 at $n = 16$ to 0.520 at $n = 64$, and attention drops to 0.446 by $n = 64$. On bottleneck paths, DELTANAR reaches 0.899 ± 0.133 in the 20-seed main grid and 0.852 ± 0.163 in the matched diagnostic grid at $n = 256$, well above from-scratch baselines, though whole-graph exactness collapses as node errors compound.

The strong baseline failures are central to the claim. Recomp+gate has access to affected supervision during training, but without previous-solution memory it still has to reconstruct the full solution. Triplet and attention increase processor expressivity, but they do not change the computational contract. DELTANAR succeeds because it changes the contract: the old solution is a candidate output, and the model learns when to preserve it.

Recent reasoner and conditional hard slice. Table 2 shows that scale and training volume alone do not close

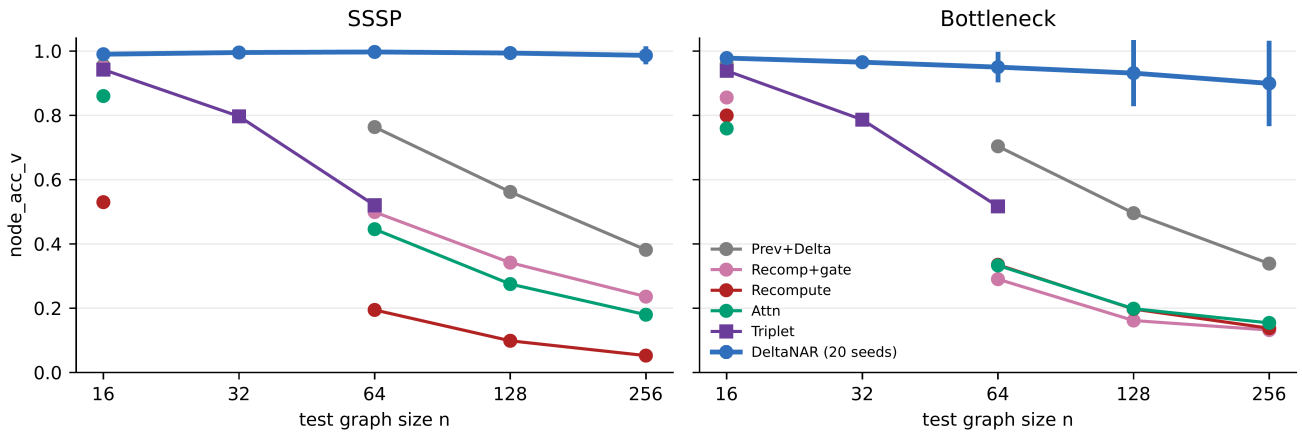

Figure 2: Size-OOD generalization. DeltaNAR keeps high node accuracy when trained at $n = 16$ and evaluated on larger graphs. The DeltaNAR curve is the 20-seed server grid; baselines are the matched B-line diagnostics.

Task	Model	node	graph	changed	critical
SSSP	DeltaNAR	.846	.069	.855	.942
SSSP	Naive-d	.426	.000	.833	.949
SSSP	Recompute	.042	.000	.934	.977
SSSP	Pure copy	.992	.000	.831	.000
Bottleneck	DeltaNAR	.794	.000	.882	.995
Bottleneck	Naive-d	.306	.000	.847	.995
Bottleneck	Recompute	.185	.000	.818	.994
Bottleneck	Pure copy	.996	.000	.893	.000

Table 3: Conditional hard slice at $n = 256$ (10 seeds, 100 accepted graphs/seed). Every graph invalidates at least one old predecessor. "Changed" scores nodes whose target changed; "critical" scores nodes on which the old predecessor is invalid after the edit. Naive-d receives the previous distance as a feature but has no explicit copy path.

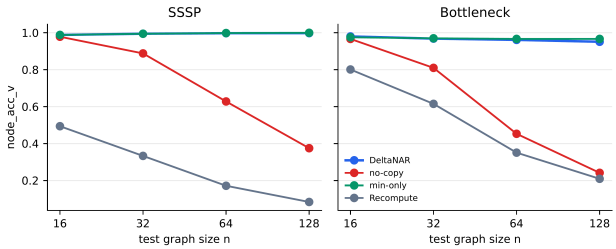

Figure 3: Component ablation. Removing copy causes size-OOD collapse, while the full model remains stable.

the gap: DEAR degrades from 0.520 to 0.287 node validity, whereas the independent DeltaNAR stability rerun remains at 0.972 at $n = 256$. The comparison is deliberately information-aware. It supports the incremental input contract, not a claim that DeltaNAR dominates DEAR given identical inputs.

Table 3 removes graphs with no stale predecessor, but not the class imbalance: only about two of 256 predecessors are stale on average. Consequently, pure copy still wins the node average while failing every graph and every critical node. DeltaNAR is best among trained controls in aggregate node validity and is the only method with nonzero SSSP graph validity; bottleneck graph validity remains unsolved. The regional metrics also prevent an overclaim: DeltaNAR does not dominate every slice. The evidence supports coordinated preservation and partial repair, not node-average dominance over copying. For the confirmatory 20-seed paired comparisons at $n = 256$, 10,000 bootstrap replicates give node-validity gaps of 0.523 (95% CI 0.428–0.608) versus Naive-d and 0.908 (0.844–0.956) versus Recompute on SSSP; corresponding bottleneck gaps are 0.558 (0.466–0.630) and 0.745 (0.640–0.827). All headline node and graph intervals exclude zero (BH-adjusted $p < .001$).

Ablations and OOD breadth. Figure 3 isolates the mechanism. Removing copy from the gated path causes collapse: SSSP no-copy falls from 0.978 at $n = 16$ to 0.375 at $n = 128$, and bottleneck no-copy falls from 0.966 to 0.242. Removing max aggregation does not collapse in these runs. At $n = 64$, SSSP DeltaNAR obtains 0.997 on ER, 0.993 on BA, and 0.934 on grid topologies. Under weight OOD, it obtains 0.843 for weights up to 100 and 0.972 under heavy-tailed weights, compared with Recompute at 0.118 and 0.133. Under edge-deletion failure OOD, DeltaNAR remains above 0.98 on SSSP and bottleneck.

OOD breadth is important because a degenerate model might overfit to a narrow edit distribution. The topology, weight, failure, and delta-type results make that explanation less plausible. The same copy-not-recompute rule applies whether an edge weight changes or an edge appears/disappears: localize the region whose witness may be stale, then recompute there.

These results address the strongest simple alternative explanation. If the task were easy once the edit was visible, Recomp+gate or Triplet should be robust. If previous-solution access alone were sufficient, Prev+Delta and no-copy should

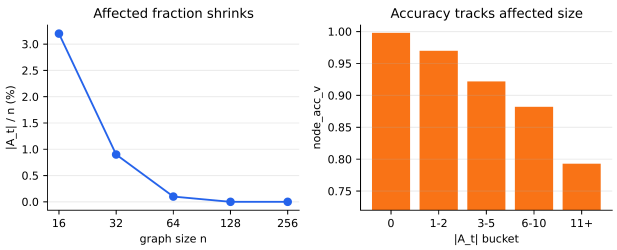

Figure 4: Theory-validation suite. The affected fraction shrinks with graph size, and accuracy decreases smoothly as affected-region size grows.

not collapse. Instead, the model needs the old solution, localization, and a mechanism that turns confidence into copying. Wide weights (0.843) and grid graphs (0.934) are harder than heavy-tailed (0.972) and BA graphs (0.993), respectively, identifying numeric calibration and broader geometric alternatives as remaining difficulties.

Theory validation and long-horizon behavior. Figure 4 connects the bound to measurable quantities. Affected fractions shrink with graph size, while accuracy falls monotonically from 0.998 at $|A_t| = 0$ to 0.793 at $|A_t| \geq 11$; gate recall is 0.90. With sufficient message-passing depth, the server reruns reach 0.932 ± 0.134 SSSP node validity at $n = 512$ and 0.949 ± 0.037 bottleneck validity at $n = 1024$. The calibrated 20-seed rollout retains 0.871 ± 0.125 SSSP node validity at step 59, but this does not establish contraction of arbitrary past errors.

Stale memory. The supplement reports the full 20–32-seed endpoint and topology audits. Under 30% bottleneck-memory corruption, guarded copy raises node validity from 0.448 to 0.573 and copy-critical validity from 0.030 to 0.257, but on clean memory it lowers copy-critical validity from 0.984 to 0.902. On SSSP, aggressive repair is worse than raw copy. Thus repair is a routed intervention for some corrupted-memory regimes, not a universally safe correction operator.

Stability and efficiency. The independent SSSP stability rerun expands from 20 to 40 seeds and includes $n = 512$. Mean node validity is 0.997, 0.992, 0.972, and 0.945 at $n = 64, 128, 256, 512$; respectively 0, 0, 2, and 2 seeds fall below 0.8. This confirms the average trend while exposing a heavy lower tail that the 20-seed main table can hide. The factorized message implementation used for these audits is algebraically equivalent to the original processor and changes only evaluation strategy. Low-memory probes reach 1,024 nodes by reducing batch size and cached intermediates, not by changing the learned update rule; they establish feasibility rather than an asymptotic runtime claim. Full per-seed values and wall-clock settings are in the supplement.

The generalist and transfer experiments show that the interface is not tied to one decoder: a shared processor handles min-plus SSSP and max-min bottleneck, and SSSP pretraining adapts rapidly to bottleneck with few-shot supervision. Because the two tasks reverse the meaning of edge quality, this recovery is consistent with a reusable update interface rather than a distance-scale rule.

MST as a difficulty gradient. At $n = 64$, DeltaNAR reaches 0.938 ± 0.056 MST edge accuracy but zero whole-tree accuracy. A few wrong edges can create cycles, disconnections, or non-minimal exchanges; copying labels does not enforce tree constraints. We therefore treat MST as a negative pressure test, not a headline success.

Discussion and Limitations

The theory and primary metric are node-level; whole-graph exactness remains the harder test and exposes compounding error. For example, SSSP graph accuracy at $n = 256$ is 0.774 ± 0.364, and bottleneck graph accuracy falls sharply with size despite strong node accuracy.

The benchmark uses synthetic edits and engine-defined affected labels, not deployment-scale dynamics. The model is strongest from an exact previous solution: moderate corruption preserves useful node validity, but whole-graph self-correction is fragile. Guarded repair helps corrupted bottleneck cases yet can sacrifice clean-copy fidelity and over-repair SSSP, so it needs task-aware routing. The 1024-node probe also requires conservative memory settings. Finally, the semiring operators are standard; our contribution is the incremental regime and gated copy mechanism.

Comparison scope. The DEAR comparison matches the post-edit graph distribution and uses more training graphs, but cannot equalize information without changing the question: giving DEAR the old solution and edit would create a new incremental model. Conversely, DeltaNAR's advantage is not a general ranking on from-scratch CLRS. It asks whether a strong recent reasoner can recover the edited solution without reuse. Testing equilibrium and parallel-execution processors inside the same incremental interface remains important future work.

Generative-AI use. Generative AI systems assisted with research ideation, manuscript drafting and editing, code generation, experiment orchestration, and result organization. Human authors remain responsible for verifying all submitted content.

Conclusion

CLRS-Delta makes incremental reuse a first-class NAR problem. Across a recent reasoner, forced stale-witness slices, paired uncertainty analysis, and broad distribution shifts, DeltaNAR shows that explicitly preserving valid algorithmic state can yield size extrapolation unavailable to recomputation. Its failures on whole-graph exactness, MST, and arbitrary stale memory define the next challenge: learning not only when to reuse, but when reuse can be certifiably repaired.

References

Engelmayer, V.; Georgiev, D. G.; and Veličković, P. 2024. Parallel Algorithms Align with Neural Execution. In *Proceedings of the Second Learning on Graphs Conference*, volume 231 of *Proceedings of Machine Learning Research*, 31:1–31:13. PMLR.

Georgiev, D.; Wilson, J. J.; Buffelli, D.; and Liò, P. 2024. Deep Equilibrium Algorithmic Reasoning. In *Advances in Neural Information Processing Systems*, volume 37, 33638–33667.

Gilmer, J.; Schoenholz, S. S.; Riley, P. F.; Vinyals, O.; and Dahl, G. E. 2017. Neural Message Passing for Quantum Chemistry. In *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, 1263–1272. PMLR.

Kazemi, S. M.; Goel, R.; Jain, K.; Kobyzev, I.; Sethi, A.; Forsyth, P.; and Poupart, P. 2020. Representation Learning for Dynamic Graphs: A Survey. *Journal of Machine Learning Research*, 21(70): 1–73.

Koenig, S.; and Likhachev, M. 2002. D* Lite. In *Proceedings of the Eighteenth National Conference on Artificial Intelligence*, 476–483. AAAI Press.

Koenig, S.; Likhachev, M.; and Furcy, D. 2004. Lifelong Planning A*. *Artificial Intelligence*, 155(1–2): 93–146.

Kujawa, S.; Poole, D.; Georgiev, D.; Numeroso, D.; Fleischmann, M.; and Liò, P. 2025. Neural Algorithmic Reasoning with Multiple Correct Solutions. In *KDD 2025 Workshop on Machine Learning on Graphs in the Era of Generative AI*.

Požgaj, S.; Georgiev, D.; Šilić, M.; and Veličković, P. 2025. KNARsack: Teaching Neural Algorithmic Reasoners to Solve Pseudo-Polynomial Problems. *arXiv preprint arXiv:2509.15239*.

Ramalingam, G.; and Reps, T. 1996. An Incremental Algorithm for a Generalization of the Shortest-Path Problem. *Journal of Algorithms*, 21(2): 267–305.

Rossi, E.; Chamberlain, B.; Frasca, F.; Eynard, D.; Monti, F.; and Bronstein, M. 2020. Temporal Graph Networks for Deep Learning on Dynamic Graphs. *arXiv preprint arXiv:2006.10637*.

Schutz, A.; Darvariu, V.-A.; Panagiotaki, E.; Lacerda, B.; and Hawes, N. 2025. Tackling GNARLy Problems: Graph Neural Algorithmic Reasoning Reimagined through Reinforcement Learning. *arXiv preprint arXiv:2509.18930*.

Veličković, P.; Badia, A. P.; Budden, D.; Pascanu, R.; Banino, A.; Dashevskiy, M.; Hadsell, R.; and Blundell, C. 2022. The CLRS Algorithmic Reasoning Benchmark. In *Proceedings of the 39th International Conference on Machine Learning*, volume 162 of *Proceedings of Machine Learning Research*, 22084–22102. PMLR.

Veličković, P.; and Blundell, C. 2021. Neural Algorithmic Reasoning. *Patterns*, 2(7): 100273.

Veličković, P.; Cucurull, G.; Casanova, A.; Romero, A.; Liò, P.; and Bengio, Y. 2018. Graph Attention Networks. In *International Conference on Learning Representations*.

Veličković, P.; Ying, R.; Padovano, M.; Hadsell, R.; and Blundell, C. 2020. Neural Execution of Graph Algorithms. In *International Conference on Learning Representations*.

Xu, K.; Li, J.; Zhang, M.; Du, S. S.; Kawarabayashi, K.-i.; and Jegelka, S. 2020. What Can Neural Networks Reason About? In *International Conference on Learning Representations*.

Supplementary Material: DeltaNAR: Learning What to Reuse and What to Recompute in Neural Algorithmic Reasoning

Anonymous submission

Overview

This supplement expands the benchmark, theory, experiments, and reproducibility details for CLRS-DELTA and DELTANAR. The main paper focuses on the central claim: incremental NAR should update previous solutions rather than recompute from scratch. Here we include full task definitions, oracle verification, proofs, full tables, additional visualizations, and caveats. It also documents the reviewer-facing DEAR comparison, conditional stale-witness slice, independent 40-seed stability audit, and paired bootstrap analysis added after the original experiment grid.

The repository stores the full result record in `RESULTS.md`. The server B-line record is stored as `B_RESULTS_SERVER_SUMMARY.md`. Figures and the theory-validation A-suite are generated by scripts in `paper/scripts`; their outputs are written to `results` and `logs`.

Benchmark Details

General Incremental Format

Each sample is an edit step:

$$(G_t, y_{t-1}, \Delta_t) \to (y_t, A_t).$$

The post-edit graph G_t is provided rather than the pre-edit graph; the edit feature marks the local perturbation. The previous solution y_{t-1} is part of the input. This separates incremental NAR from standard from-scratch CLRS tasks, where each sample contains only a problem instance.

The affected region A_t is a supervised hint. In dynamic SSSP it is the LPA*-style work-set: vertices popped from the priority queue during the incremental update. This may be a superset of vertices whose final value changes, which is appropriate for learning where recomputation is needed.

Dynamic SSSP Oracle

The dynamic SSSP engine maintains g and rhs values with heuristic zero. After an edit, only the head vertex of the edited edge can become directly inconsistent. The algorithm updates this vertex, processes the priority queue, and propagates improvements or invalidations locally. The final distances are checked against from-scratch Dijkstra in the unit tests.

Dynamic SSSP Update Flow

The dynamic SSSP oracle follows the LPA* pattern. Initially the source has $rhs = 0$ and all other vertices have infinite tentative values. After a graph edit, only vertices whose incoming edge set changed can become directly inconsistent. For a single edge edit (u, v), the update seeds vertex v. Whenever a vertex is popped from the queue, the algorithm checks whether its current value is overconsistent or underconsistent. Improvements are propagated to successors; invalidations reset the vertex and may propagate further. The affected-region hint is the set of popped vertices. This hint is therefore a work-set, not merely the final changed-distance set.

Why Affected Work-Set Supervision Is Useful

The final output alone does not tell the model where the update was algorithmic. For example, a vertex can enter the local recomputation queue but end with the same final value after an alternative predecessor is selected. Supervising the work-set encourages the model to learn the dynamic algorithm's localization behavior rather than only final-value differences. In the theory, the affected region can be any copy-critical set for which vertices outside the set are safe to copy. In implementation, the work-set is a conservative and useful choice.

The model target contains both normalized distances and predecessor pointers. The reported `node_acc_v` is tie-tolerant: a predicted predecessor is accepted if it witnesses the correct shortest-path value, even if it differs from the oracle's arbitrary tie choice.

Bottleneck Oracle

Bottleneck paths use the max-min recurrence

$$B(v) = \max_u \min(B(u), w(u, v)).$$

The same incremental sample format is used. This task tests whether copy-not-recompute is a semiring-level mechanism rather than a shortest-path specific trick. The target predecessor must witness the correct bottleneck value.

Incremental MST Oracle

The MST task predicts edge membership after an edit in an undirected graph. Kruskal and Prim implementations are cross-checked. Deltas are sampled to be informative, often forcing tree swaps, so the affected-region signal is not degenerate. MST is considerably stricter at the whole-structure level: a small number of edge mistakes makes the whole tree wrong.

Oracle Verification

The project uses differential tests for every ground-truth component. Dynamic SSSP updates are checked against from-scratch Dijkstra after edits. Bottleneck paths are checked against independent widest-path computations. MST generation and updates are checked by comparing Kruskal and Prim. Dataset tests verify that the sample target equals the oracle state after applying the edit. These tests are critical because the benchmark introduces new dynamic tasks rather than reusing preexisting CLRS instances.

Edit Families

The benchmark includes four edit families. Weight increases and decreases modify existing edges. Edge additions introduce a new directed or undirected edge, depending on the task. Edge deletions remove an existing edge. The failure-OOD experiment evaluates edge deletion specifically because deletion creates underconsistent shortest-path states and is a harder dynamic update than simply improving an edge weight.

Why Use the Post-Edit Graph

The model receives the post-edit graph rather than the pre-edit graph plus an instruction to mutate it. This isolates the algorithmic-update problem. The network does not need to learn graph editing; it needs to decide how the previous solution should be revised on the already-updated instance. The edit feature is still included because classical dynamic algorithms seed their update from the location of the perturbation.

Model and Training Details

Architecture

DeltaNAR uses an encode-process-decode architecture. Node and edge encoders consume graph structure, weights, previous distances or predecessor/edge states, and edit indicators. The processor alternates message passing and gated state updates. The decoder produces task outputs, affected logits, and in the calibrated copy-safe variant an explicit copy-safety logit.

For path tasks, predecessor scores are masked to valid incoming edges. For MST, edge logits are symmetrized and masked to existing undirected edges. The copy bias is additive, not a hard override, so the model can override stale previous solutions. The affected logit localizes algorithmic work; the copy-safety logit directly predicts whether the previous witness is safe to preserve.

Losses

Path tasks use distance regression, predecessor cross entropy on reachable non-source vertices, and affected-region binary cross entropy. The affected loss uses a positive class weight because affected vertices are sparse. MST uses edge membership loss and affected-region loss. The training objective deliberately supervises both final solutions and localization.

For copy-safe repair, we add a binary copy-safety loss on reachable non-source vertices. The target is positive when the previous predecessor remains a valid witness under the post-edit ground-truth value and negative when copying would be unsafe. This is different from affected supervision: a vertex can be reconsidered by the work-set while its old witness remains valid, and a stale witness can be the decisive failure even when final-value changes are sparse.

Baselines

Recompute removes previous-solution and edit inputs and solves from scratch. Recomp+gate adds affected supervision to a from-scratch model. Prev+Delta receives the previous solution and edit as ordinary input features but has no privileged copy decoder. Triplet adds strong edge reasoning over triples, inspired by stronger CLRS-style processors. Attn replaces min/max aggregation with attention. no-copy keeps previous solution and gate but removes the copy bias. min-only removes max aggregation.

Sample Schema

Each path-task sample contains the following fields. `W` is the post-edit weight matrix and `A` is the post-edit adjacency mask. `g_prev` and `pred_prev` encode the previous solution. `deltas` stores the edit event or simultaneous edit list. `g_next` and `pred_next` are the target solution. `affected` is the gate target, and `size_affected` stores its cardinality. This schema is deliberately close to CLRS-style final-output plus hint supervision, but the previous solution and edit are explicit.

For MST, the previous and target solutions are edge-membership matrices rather than predecessor pointers. The same high-level contract applies: the model sees the post-edit graph and previous tree, then predicts the updated tree and affected vertices.

Configuration Flags

The implementation exposes each modeling choice as a flag. The most important flags are summarized in Table 1: previous-solution input, edit input, affected gate, copy bias, triplet reasoning, and attention aggregation. The generalist runner additionally enables task embeddings for shared processing across semirings.

Model	prev	delta	gate	copy	triplet	attn
DeltaNAR	yes	yes	yes	yes	no	no
Prev+Delta	yes	yes	no	no	no	no
no-copy	yes	yes	yes	no	no	no
Recompute	no	no	no	no	no	no
Recomp+gate	no	no	yes	no	no	no
Triplet	no	no	yes	no	yes	no
Attn	no	no	yes	no	no	yes

Table 1: Configuration map for the main baselines and ablations. The code name `Naive-d` corresponds to Prev+Delta in the paper text.

Why the Copy Bias Is Soft

The decoder does not force a hard copy. It adds a bias whose strength depends on the estimated copy-safety probability. This is important for ambiguous cases. If the safety gate is uncertain, the learned predecessor scores can override stale memory. If the safety gate is confident, the model strongly favors the previous solution. This design gives a continuous bridge between pure recomputation and pure copying. The earlier affected-gate decoder is recovered by using the negative affected logit as the copy-safety score.

The conservative margin is task-dependent. Bottleneck reasoning is more sensitive to stale copies, so the generalist model uses a larger margin for the bottleneck task. The A5 sweep shows that copy scale must be large enough to matter, but saturates once the bias dominates unaffected vertices.

Code Map

The implementation is intentionally compact. `src/graphs.py` contains graph generation, edit sampling, and edit application. `src/sssp.py` contains from-scratch SSSP and bottleneck oracles. `src/dynamic_sssp.py` contains the LPA*-style incremental SSSP engine. `src/mst.py` contains Kruskal and Prim. Dataset builders live in `src/dataset.py`, `src/dataset_bottleneck.py`, and `src/dataset_mst.py`. Model, collation, losses, and evaluation live in `src/model.py`. Multi-seed grids are run through `experiments.py`; task-specific training scripts handle rollout, generalist, transfer, and MST experiments.

Evaluation Mask

For SSSP and bottleneck, `node_acc_v` is computed over reachable non-source vertices. This matters for theory. If one writes the denominator as n, one is implicitly assuming $|S_t| = \Theta(n)$ or using an all-node metric. The formal theorem uses $|S_t|$ to match the implementation exactly.

Full Theory

Positioning Relative to Prior Theory

Our theory follows the style of algorithmic-alignment analyses rather than a full dynamic-algorithm correctness proof. Prior NAR work formalizes when a network architecture matches an algorithmic computation pattern, often through dynamic-programming or message-passing alignment (Xu et al. 2020; Veličković et al. 2020). CLRS-style benchmark papers then use broad empirical evidence to test whether this alignment survives out-of-distribution evaluation (Veličković et al. 2022). Classical dynamic shortest-path work proves exact update algorithms under local consistency and work-list invariants (Ramalingam and Reps 1996; Koenig, Likhachev, and Furcy 2004; Koenig and Likhachev 2002). DeltaNAR is neural and approximate, so the appropriate claim is narrower: if the previous solution is exact, the copy-critical region is small, and the gate/recompute path succeeds on that region, then node error scales with the affected fraction. The A-suite in this supplement checks these assumptions empirically.

Notation

Let S_t be the evaluation domain. For SSSP and bottleneck this is the set of reachable non-source vertices, matching `node_acc_v`. Let

$$C_t = \{v \in S_t : y_t^*(v) \neq y_{t-1}^*(v)\}.$$

The smallest copy-unsafe region is C_t. The implementation's affected target is a work-set, which can be a strict superset of C_t because a vertex may be reconsidered during repair even if its final witness remains valid. The theorem therefore uses a copy-critical region $A_t \supseteq C_t$ for which vertices outside A_t can be safely copied. Taking $A_t = C_t$ gives the tightest bound; taking A_t as the supervised work-set gives a looser but implementation-aligned bound.

One-Step Bound

Define

$$\gamma_t = \frac{1}{|A_t|} \sum_{v \in A_t} \Pr[\hat{y}_t(v) = y_t^*(v)]$$

when A_t is nonempty, and define $\gamma_t = 1$ when $A_t = \emptyset$. Define

$$\eta_t = \frac{1}{|S_t \setminus A_t|} \sum_{v \in S_t \setminus A_t} \Pr[\hat{a}_t(v) = 1 \wedge \hat{y}_t(v) \neq y_t^*(v)]$$

when $S_t \setminus A_t$ is nonempty, and $\eta_t = 0$ otherwise.

Theorem 1. *Under exact previous solution, safe copying outside A_t, false-positive recompute error at most η_t, and average correctness at least γ_t on vertices in A_t,*

$$\mathbb{E}[\mathrm{err}_t] \leq \frac{|A_t|}{|S_t|}(1 - \gamma_t) + \frac{|S_t \setminus A_t|}{|S_t|}\eta_t.$$

Proof. For $v \in S_t \setminus A_t$, $y_t^*(v) = y_{t-1}^*(v)$. Since the previous solution is exact, copying is correct. Such vertices can err only if a false-positive gate sends them to imperfect recomputation; these errors are captured by η_t. For $v \in A_t$, we do not need to assume that stale copying is always wrong; this is false for a work-set superset. We only require that the model's average correctness on this copy-critical set is at least γ_t. Thus the average error over A_t is at most $1 - \gamma_t$. Averaging the two cases over S_t gives the bound. □

Theorem 2. *If $\mathbb{E}|A_t| \leq a$, $|S_t| \geq cn$ almost surely, $\gamma_t \geq \gamma$, and $\eta_t \leq \eta$, then*

$$\mathbb{E}[\mathrm{node\_acc}_t] \geq 1 - \frac{a}{cn}(1 - \gamma) - \eta.$$

This is the paper's main theorem. It is intentionally node-level because the mechanism and metric are node-level. The bound predicts improvement with n only when the affected region is output-sensitive and false-positive recomputation is safe or rare.

Gate Recall and Recompute Accuracy

Define affected recall

$$\rho_t = \frac{1}{|A_t|} \sum_{v \in A_t} \Pr[\hat{a}_t(v) = 1].$$

Let q_t be recompute success on flagged affected vertices. If A_t is the changed-output set, and the averaged gate and recomputation events factorize, then $\gamma_t = \rho_t q_t$. For a work-set superset, copying may also be correct on some vertices in A_t, so $\rho_t q_t$ is a conservative mechanism-specific success term rather than the only route to correctness. We use γ_t in the formal theorem because it is strictly safer: it avoids both an unnecessary independence assumption and an unnecessary claim that every work-set vertex is stale.

Message-Passing Depth

For $v \in A_t$, define the update dependency depth $d_t(v)$ as the minimum number of semiring relaxation steps needed to certify $y_t^*(v)$ from exact boundary values. If the processor can represent one exact relaxation per step, and $d_t(v) \leq T$, then a T-step processor can compute the exact witness in the ideal case. This statement is stronger and safer than using affected-subgraph diameter, because shortest or widest witness paths may cross copied boundary vertices outside the induced affected set.

Rollout

Let e_t be node error after step t when predictions are fed forward. Without self-correction, contraction is false in general: a wrong copied value can persist forever if never revisited. The safe accumulation bound is

$$\begin{aligned}\mathbb{E}[e_t] \leq \mathbb{E}[e_{t-1}] + \mathbb{E}\left[\frac{|A_t|}{|S_t|}(1-\gamma_t)\right] \\ + \mathbb{E}[\eta_t] + \text{domain-change terms.}\end{aligned} \tag{1}$$

Thus for T rollout steps,

$$\mathbb{E}[e_T] \leq e_0 + T\left(\frac{a}{cn}(1-\gamma) + \eta\right) + \text{domain-change terms.}$$

This matches the empirical pattern: slow, seed-dependent drift, but much better than no-copy collapse.

Gate Recall Sample Complexity

For a gate hypothesis class with VC dimension d, and m_A affected training vertices, standard VC-style bounds give

$$\mathrm{FN}_A(h) \leq \widehat{\mathrm{FN}}_A(h) + O\left(\sqrt{\frac{d\log(m_A/d) + \log(1/\delta)}{m_A}}\right).$$

Since recall $\rho = 1 - \mathrm{FN}_A$, high recall is learnable when affected false-negative rate is low and $m_A = \Omega((d + \log(1/\delta))/\epsilon^2)$. This is a sufficient condition, not a tight neural-network generalization claim.

Whole-Graph Corollary

If node errors were independent with node accuracy p, whole-graph accuracy on $r = |S_t|$ nodes would be approximately p^r. Without independence, the union bound gives

$$\Pr[\text{any node error}] \leq r\,\mathbb{E}[\mathrm{err}_t].$$

This explains why bottleneck and MST can have high node/edge accuracy but low whole-graph accuracy. It also explains why SSSP graph accuracy has high variance at $n = 256$: small differences in node error can become large differences in whole-graph exactness.

Why the Bound Does Not Apply to Recompute

The decomposition relies on an explicit copy branch. A from-scratch model has no term for "unaffected vertices are already solved." Its error does not decompose into an affected fraction plus a false-positive recomputation term. It must learn a size-generalizing full solver. This is the theoretical reason that stronger processors alone do not receive the same guarantee.

Theory Examples and Notation

Notation Table

Symbol	Meaning
G_t	Graph after edit t
Δ_t	Local edit applied between $t-1$ and t
y_t^*	Ground-truth solution after edit t
$\hat{y}_t$	Model prediction after edit t
S_t	Evaluation domain for node accuracy
A_t	Affected/copy-critical region
ρ_t	Gate recall on affected vertices
q_t	Recompute success on flagged affected vertices
γ_t	Average correctness on affected/copy-critical vertices
η_t	False-positive recomputation error outside A_t
a	Upper bound on $\mathbb{E}\|A_t\|$
c	Constant with $\|S_t\| \geq cn$

Table 2: Notation used in the theory.

Worked One-Step Example

Suppose a graph has 100 evaluated vertices and a local edit affects 4 of them. If the previous solution is exact, the 96 unaffected vertices are correct when copied. Suppose the joint probability of flagging and correctly recomputing an affected vertex is $\gamma = 0.9$, and false-positive recomputation is safe ($\eta = 0$). The bound gives

$$\mathbb{E}[\text{node\_acc}] \geq 1 - \frac{4}{100}(1 - 0.9) = 0.996.$$

If the same edit distribution is evaluated on 1000 vertices and still affects about 4 vertices, the lower bound becomes 0.9996. This toy calculation is the entire intuition behind the size-OOD result: larger graphs contain more unchanged vertices that can be copied exactly.

Dependency Depth Example

Consider a shortest-path edit that decreases the weight of an edge entering vertex v. If the best new path to v uses a predecessor whose distance was unchanged, then v has dependency depth 1. If the improvement then propagates to a successor w, w has dependency depth 2. The relevant depth is not the diameter of the entire graph; it is the number of relaxation steps needed from correct copied boundary values. This is why DeltaNAR can work with few message passing steps even when the graph itself is much larger.

False Positives and the Eta Term

The η term is included to avoid hiding a modeling assumption. If an unaffected vertex is mistakenly sent to recomputation, copying would have been correct but recomputation might be wrong. In practice, the processor is trained on final targets for all vertices, so many false positives are harmless. The formal theorem nevertheless keeps η explicit. The clean informal bound without η is valid under safe false-positive recomputation or when false positives are rare enough that $\eta = o(1)$.

Additional Figures

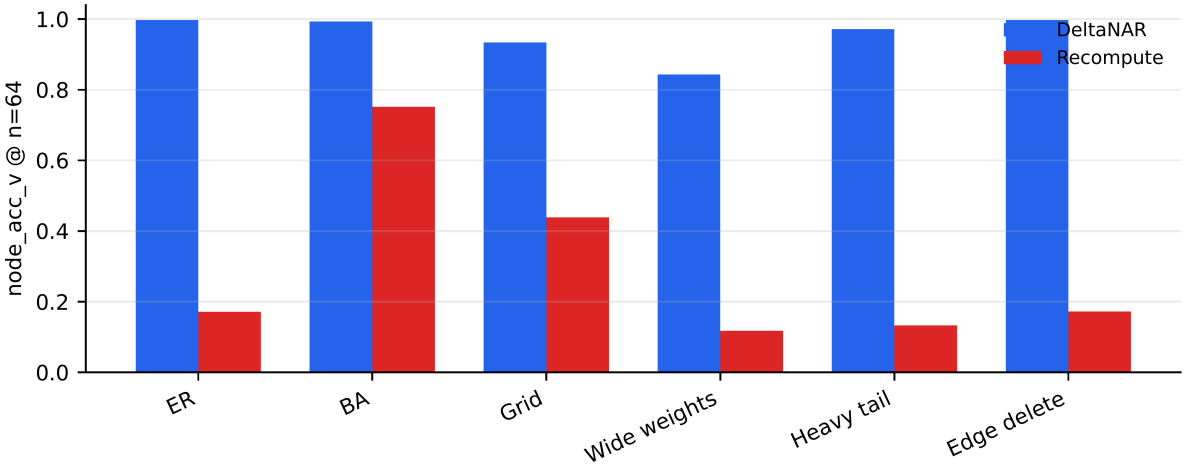

Figure 1: OOD breadth for SSSP at $n = 64$.

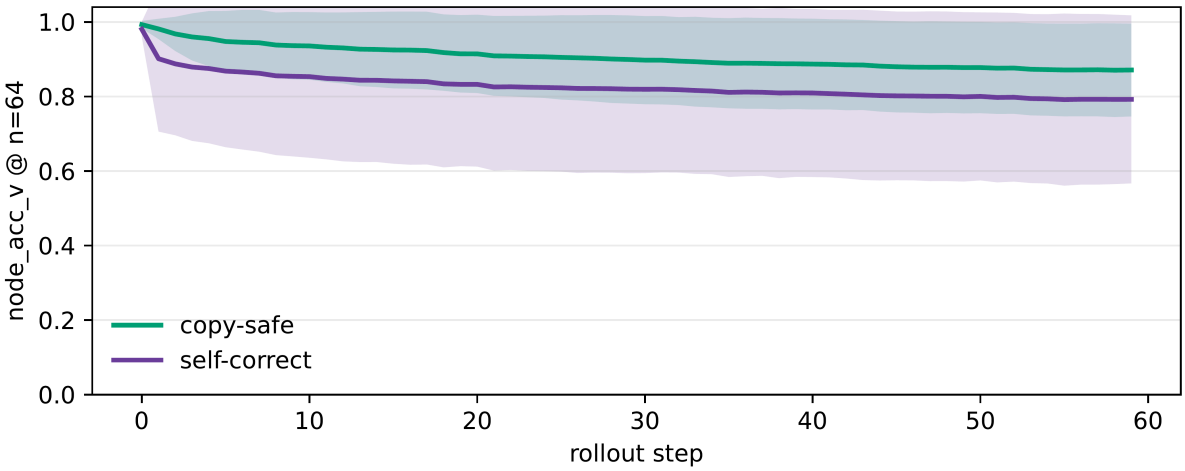

Figure 2: C43 60-step SSSP rollout at $n = 64$. Shaded region is one standard deviation over 20 seeds.

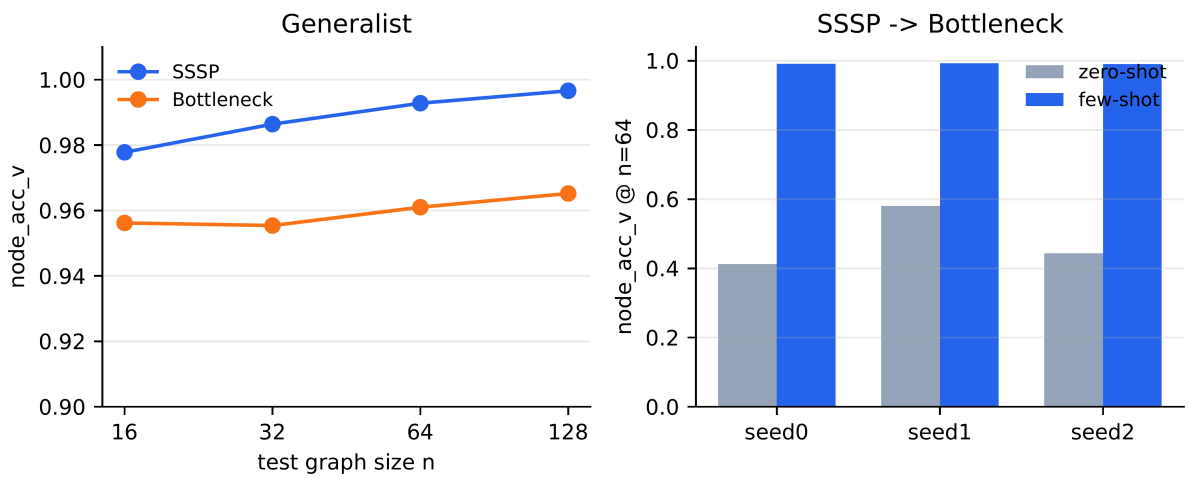

Figure 3: Generalist and cross-task transfer.

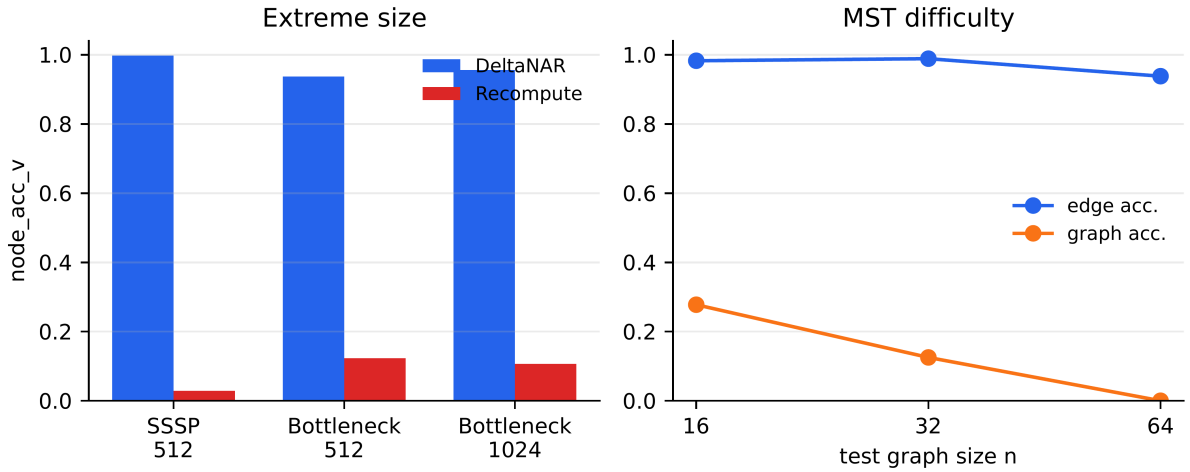

Figure 4: Extreme-size results and MST difficulty gradient.

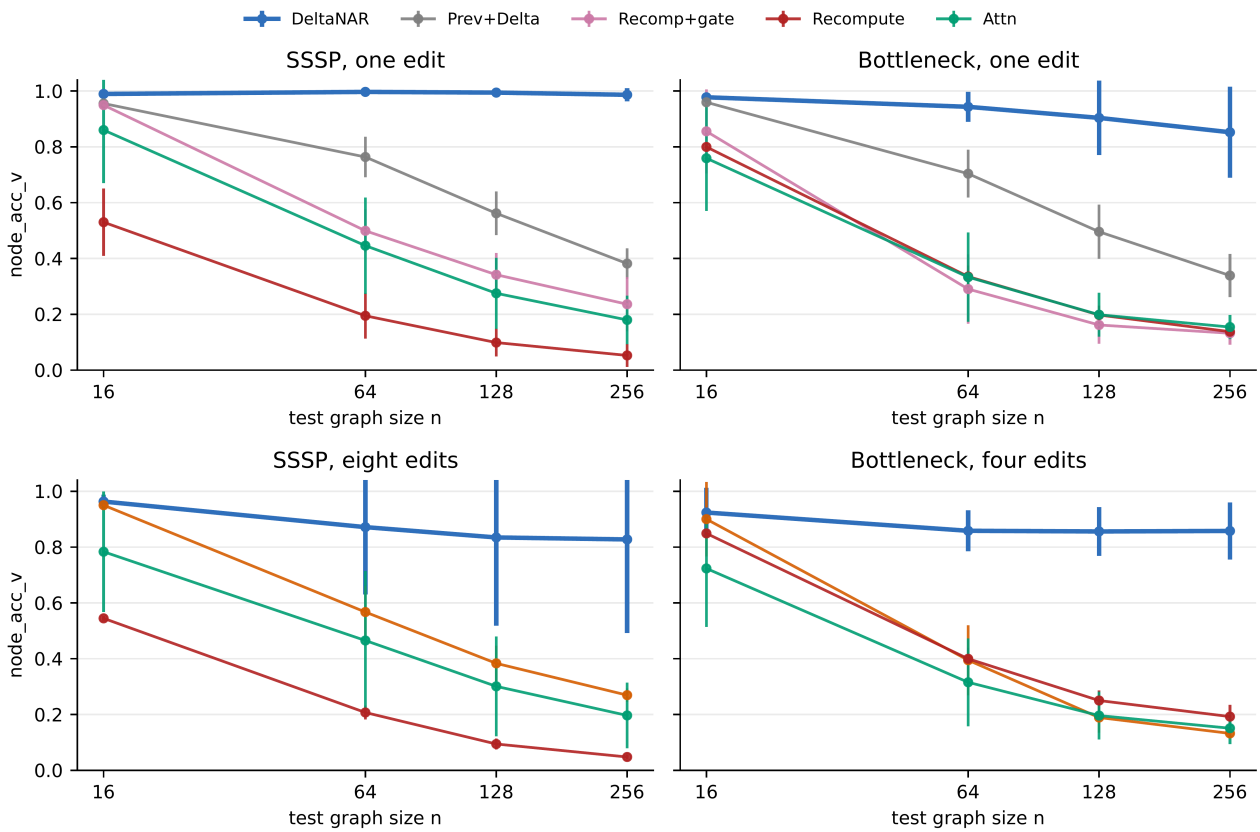

Figure 5: Server B-line overview. The 20-seed DeltaNAR main grid is paired with matched B-line baselines and harder multi-edit diagnostics.

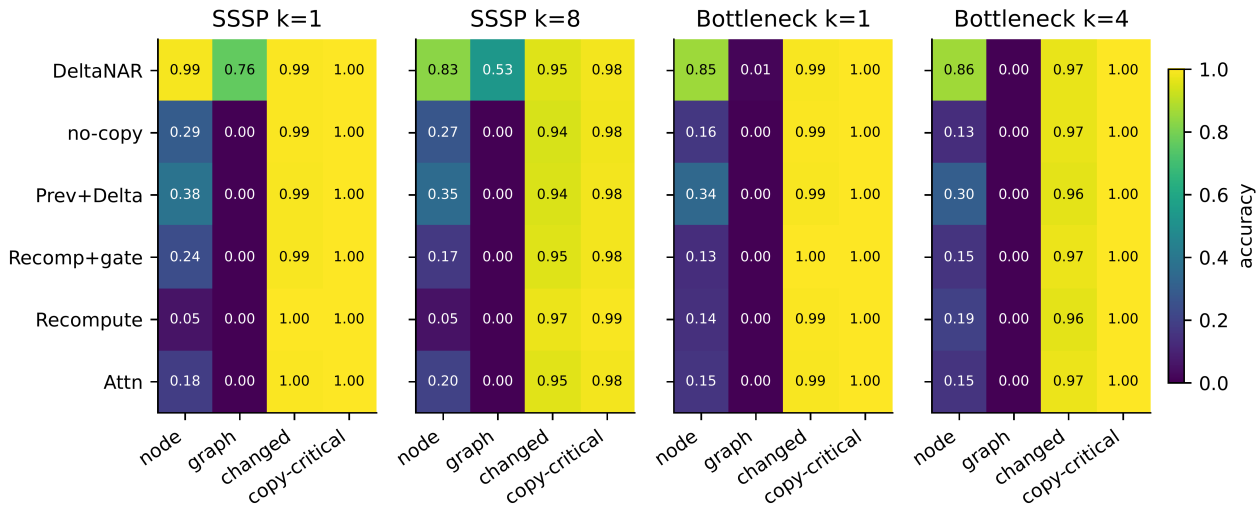

Figure 6: Server diagnostic slices at $n = 256$. The heat map separates full node accuracy, whole-graph exactness, changed-output accuracy, and copy-critical accuracy.

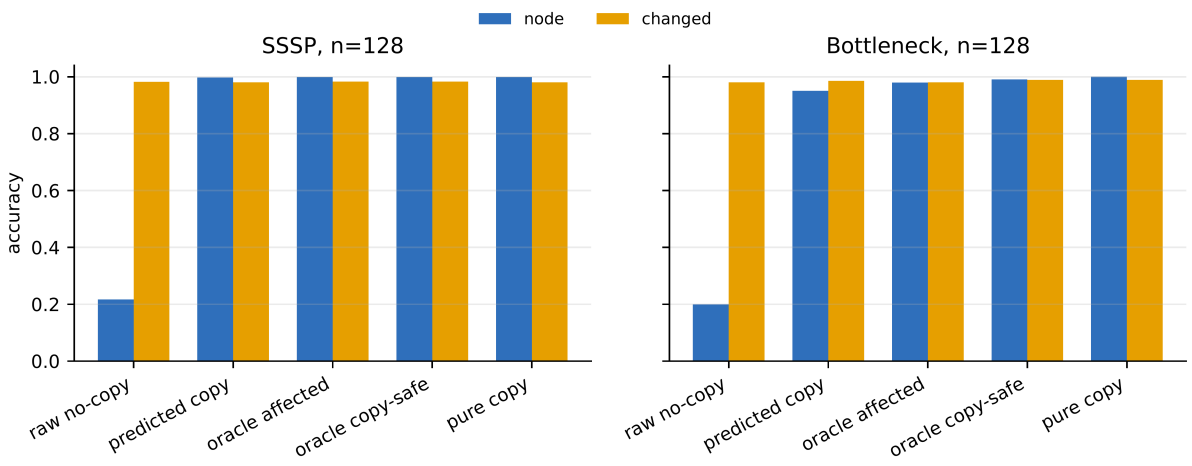

Figure 7: Copy-oracle study at $n = 128$. Oracle copy variants isolate the contribution of copy safety from local recomputation quality.

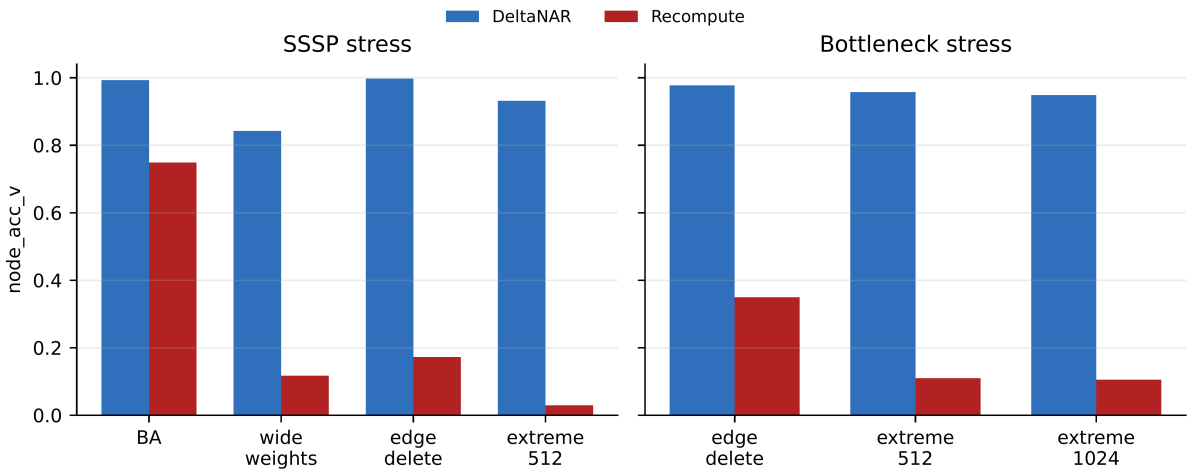

Figure 8: Server OOD and extreme-size stress tests.

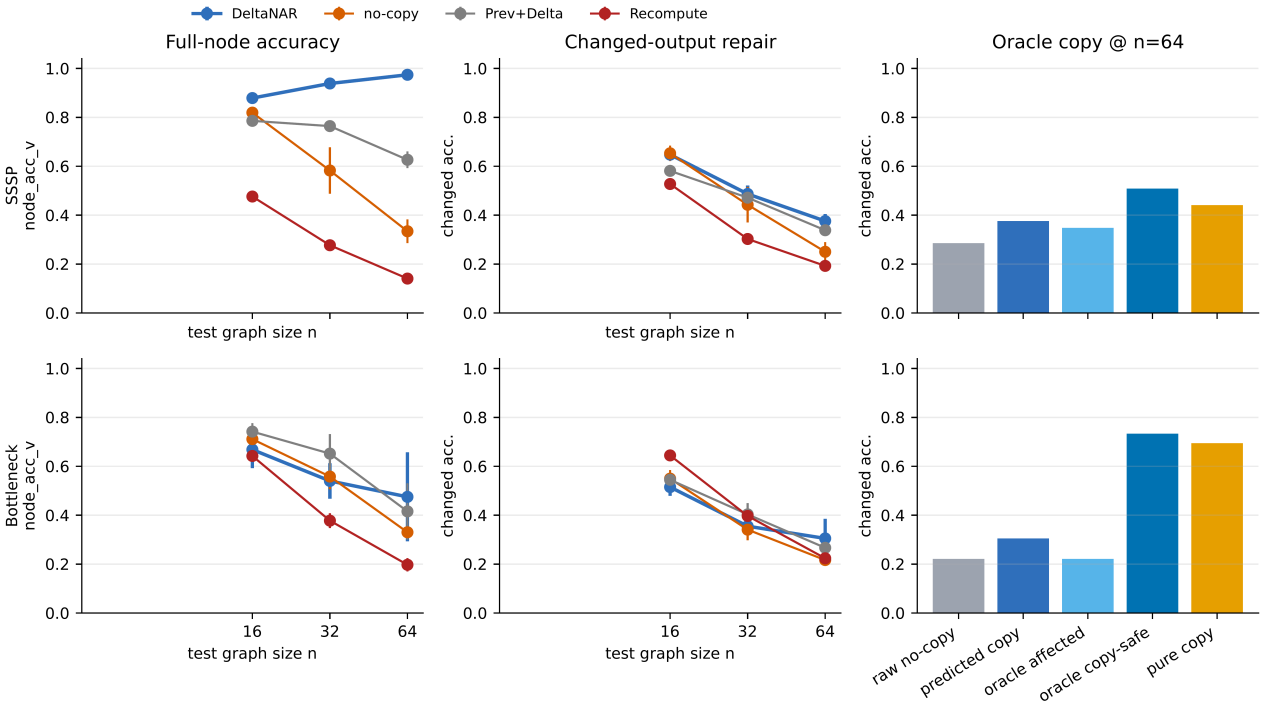

Figure 9: Local hard-repair audit. The model is trained with four simultaneous edits and tested with twelve simultaneous edits. This stress test separates full-node copy preservation from changed-output repair.

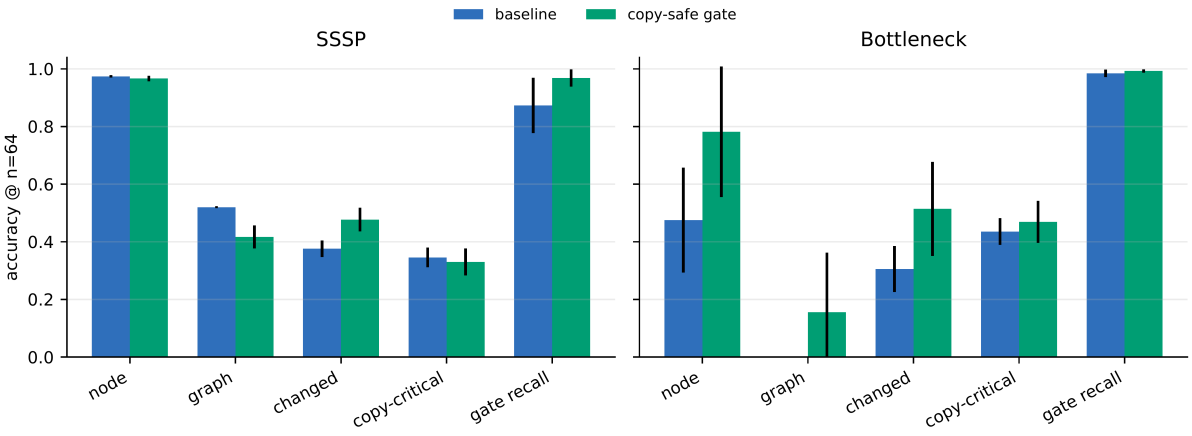

Figure 10: Local copy-safe gate probe on the hard-repair audit. The independent gate improves changed-output repair, especially for bottleneck, motivating the larger C2/C34/C35 server validation.

Full Result Tables

Training and Evaluation Configuration

All main experiments train at $n = 16$ for 50 epochs with 500 training episodes unless otherwise stated. Evaluation uses 100 test episodes per size in the main grids. The local PyTorch environment uses Python 3.12 and CUDA when available. Ground-truth engines are pure Python and tested independently: dynamic SSSP is checked against from-scratch Dijkstra, bottleneck against independent widest-path oracles, and MST against Kruskal/Prim agreement.

The confirmatory revision runs used Ubuntu 22.04.3 on an AMD EPYC 7282 host with 251 GiB RAM and NVIDIA GeForce RTX 4090 GPUs (24 GiB each). The optimized DeltaNAR environment used Python 3.10 and PyTorch 2.6.0+cu124. The DEAR environment used Python 3.12, PyTorch 2.2.0+cu121, PyTorch Lightning 1.8.1, and PyTorch Geometric 2.5.3. Each training process was pinned to one explicitly selected GPU; no result averages across GPUs within a seed.

Setting	Value
Train size	16 nodes
Epochs	50
Train episodes	500
Main test episodes	100
Hidden dimension	128
Optimizer	Adam
Learning rate	0.001
Default batch size	64
Baseline seeds	0–4
20-seed server DeltaNAR seeds	0–19
Primary metric	`node_acc_v`
Secondary metric	`graph_acc_v`

Table 3: Default experimental configuration. Individual scripts override batch size or message-passing steps for extreme-size runs.

The extreme-size SSSP run uses sufficient message-passing steps rather than the steps=32 convenience setting. This distinction matters: SSSP with a fixed steps=32 cap is unstable at very large sizes, while the steps=n run reaches 0.998 node accuracy at 512 nodes. Bottleneck extreme-size results use the available steps=32 run and remain high at 1024 nodes.

Reviewer-Facing Confirmatory Experiments

Matched-distribution DEAR protocol. We adapted the official Deep Equilibrium Algorithmic Reasoning (DEAR) architecture (Georgiev et al. 2024) to SSSP graphs drawn from the same post-edit distribution as CLRS-Delta. DEAR receives the edited graph, edge weights, and source and predicts a predecessor witness from scratch. It does not receive the previous solution, affected mask, or edit metadata. Each of five seeds trains for 100 epochs on 10,000 matched graphs. This is a comparison of computational contracts, not an identical-input architecture ablation: augmenting DEAR with old-solution memory would define a new incremental model.

	DeltaNAR stability rerun		DEAR, 5 seeds		
n	node validity	graph validity	node validity	graph	exact pred.
16	$.9910 \pm .0017$	$.8945 \pm .0181$	$.5204 \pm .0128$	.010	.315
64	$.9972 \pm .0076$	$.9523 \pm .0199$	$.4064 \pm .0134$	.000	.238
128	$.9915 \pm .0287$	$.9355 \pm .1544$	$.3548 \pm .0127$	.000	.184
256	$.9716 \pm .0929$	$.8852 \pm .2561$	$.2869 \pm .0151$	.000	.134
512	$.9452 \pm .2056$	$.8836 \pm .2961$	—	—	—

Table 4: Matched-distribution recent-reasoner comparison. DeltaNAR uses 40 seeds except at $n = 16$, which uses the 20-seed rerun; DEAR uses five seeds. Standard deviations are population standard deviations.

Conditional invalid-copy slice. Ordinary local-edit evaluation contains many graphs on which most previous predecessors remain valid. To prevent this from making copying vacuous, we condition evaluation on the presence of at least one invalid previous predecessor. For each of ten independently trained seeds, sampling continues until 100 accepted graphs are obtained at each size. No training examples are filtered by this condition. Naive-d receives the previous distance as an input feature but has no explicit predecessor-copy path. The condition prevents trivially perfect graph copying, but it does not balance nodes: accepted $n = 256$ graphs contain only 2.08 stale predecessors on average for SSSP.

		node validity			graph validity		
Task	n	DeltaNAR	Naive-d	Recompute	DeltaNAR	Naive-d	Recompute
SSSP	64	.907	.715	.160	.124	.000	.000
SSSP	128	.884	.565	.081	.074	.000	.000
SSSP	256	.846	.426	.042	.069	.000	.000
SSSP pure copy	64	.961	—	—	.000	—	—
SSSP pure copy	128	.983	—	—	.000	—	—
SSSP pure copy	256	.992	—	—	.000	—	—
Bottleneck	64	.756	.576	.344	.000	.000	.000
Bottleneck	128	.813	.415	.225	.000	.000	.000
Bottleneck	256	.794	.306	.185	.000	.000	.000
Bottleneck pure copy	64	.954	—	—	.000	—	—
Bottleneck pure copy	128	.991	—	—	.000	—	—
Bottleneck pure copy	256	.996	—	—	.000	—	—

Table 5: Aggregate validity on the conditional invalid-copy slice (10 seeds, 100 accepted graphs per seed and size). Pure copy remains high in the node-weighted average but necessarily has zero graph validity.

		changed-region validity			copy-critical validity		
Task	n	DeltaNAR	Naive-d	Recompute	DeltaNAR	Naive-d	Recompute
SSSP	64	.796	.775	.792	.878	.874	.882
SSSP	128	.822	.816	.877	.899	.918	.941
SSSP	256	.855	.833	.934	.942	.949	.977
SSSP pure copy	64	.746	—	—	.000	—	—
SSSP pure copy	128	.792	—	—	.000	—	—
SSSP pure copy	256	.831	—	—	.000	—	—
Bottleneck	64	.695	.691	.656	.936	.939	.858
Bottleneck	128	.824	.813	.798	.993	.996	.985
Bottleneck	256	.882	.847	.818	.995	.995	.994
Bottleneck pure copy	64	.709	—	—	.000	—	—
Bottleneck pure copy	128	.818	—	—	.000	—	—
Bottleneck pure copy	256	.893	—	—	.000	—	—

Table 6: Regional metrics on the same hard slice. "Copy-critical" is the set where the old predecessor is invalid, so pure copy is zero by construction. The controls can match or exceed DeltaNAR on an individual region; the evidence supports partial repair, not universal regional dominance.

Paired uncertainty analysis. For each task, size, metric, and comparator, the unit of pairing is training seed. We resample the 20 paired seed differences 10,000 times, report the percentile 95% interval, and apply Benjamini–Hochberg correction across the headline family. The software emitted numerically zero p-values; we report these conservatively as $p < .001$.

Independent stability rerun. The 40-seed SSSP figures combine a 20-seed confirmatory batch with a second non-overlapping batch (seeds 20–39). They are not pooled with the original main-table artifact: the rerun used a refreshed environment and factorized, cached evaluation intended to be algebraically equivalent, but it produced detectably different per-seed values. We therefore present it as independent stability evidence rather than silently replacing the prerevision grid.

Task	Metric	Comparator	Difference	95% CI	BH p
SSSP	node	Naive-d	.5228	[.4276,.6081]	$< .001$
SSSP	graph	Naive-d	.8539	[.7076,.9664]	$< .001$
SSSP	node	Recompute	.9076	[.8438,.9559]	$< .001$
SSSP	graph	Recompute	.8539	[.7108,.9673]	$< .001$
Bottleneck	node	Naive-d	.5577	[.4656,.6301]	$< .001$
Bottleneck	graph	Naive-d	.0192	[.0115,.0279]	$< .001$
Bottleneck	node	Recompute	.7451	[.6398,.8267]	$< .001$
Bottleneck	graph	Recompute	.0192	[.0114,.0277]	$< .001$

Table 7: Paired bootstrap comparisons at $n = 256$. Differences are DeltaNAR minus comparator.

n	node validity	graph validity	seeds with node $< .8$
64	$.9972 \pm .0076$	$.9523 \pm .0199$	0/40
128	$.9915 \pm .0287$	$.9355 \pm .1544$	0/40
256	$.9716 \pm .0929$	$.8852 \pm .2561$	2/40
512	$.9452 \pm .2056$	$.8836 \pm .2961$	2/40

Table 8: Independent SSSP stability rerun. The mean remains high while the large-size standard deviation and lower-tail counts expose rare weak seeds.

Main 20-Seed DeltaNAR Results

Task	n	node_acc_v	graph_acc_v	affected F1
SSSP	16	0.990 ± 0.002	0.888 ± 0.026	0.861 ± 0.122
SSSP	32	0.996 ± 0.001	0.920 ± 0.018	0.840 ± 0.145
SSSP	64	0.997 ± 0.002	0.923 ± 0.073	0.790 ± 0.194
SSSP	128	0.994 ± 0.014	0.871 ± 0.249	0.744 ± 0.295
SSSP	256	0.987 ± 0.028	0.774 ± 0.364	0.720 ± 0.345
Bottleneck	16	0.978 ± 0.005	0.768 ± 0.054	0.932 ± 0.027
Bottleneck	32	0.965 ± 0.014	0.424 ± 0.100	0.882 ± 0.074
Bottleneck	64	0.950 ± 0.047	0.129 ± 0.062	0.844 ± 0.159
Bottleneck	128	0.931 ± 0.103	0.035 ± 0.022	0.834 ± 0.212
Bottleneck	256	0.899 ± 0.133	0.014 ± 0.018	0.810 ± 0.226

Table 9: Main 20-seed server DeltaNAR grid.

Main 10-Seed Affected F1

Main 10-Seed Exact Node Accuracy

Metric-Defense Diagnostics

The primary node metric is appropriate for incremental updates because most output locations should be preserved after a local edit. However, this also means full-node averages must be paired with hard-region diagnostics. We therefore run server diagnostic sweeps that report changed-output accuracy on $C_t = \{v : y_t^*(v) \neq y_{t-1}^*(v)\}$ and copy-critical accuracy on vertices where blindly copying the previous witness is invalid. The table below reports the hardest available size, $n = 256$, for one-edit and multi-edit settings.

The diagnostic conclusion is twofold. First, default single-edit evaluation is intentionally copy-dominated, so pure copying can be strong on full-node averages even though it lacks a repair mechanism. Second, under harder multi-edit diagnostics, the explicit copy decoder remains essential for full-node and graph validity. Changed-output and copy-critical slices are high for several models, so they should be read as a metric-defense audit rather than as a standalone ranking; the decisive separation is the ability to combine local repair with safe preservation of the much larger unchanged region.

Copy-Safe Gate Calibration and Server Validation

The hard-repair audit indicates that the next bottleneck is not simply affected recall. We therefore use an independent copy-safe gate that directly predicts whether the previous witness remains safe to copy. The first local three-seed run identified the direction; the C2 server rerun then repeated the copy-safe model over 20 seeds with the full training budget.

Task	16	32	64	128	256
SSSP	0.809 ± 0.143	0.751 ± 0.180	0.699 ± 0.224	0.598 ± 0.312	0.565 ± 0.370
Bottleneck	0.930 ± 0.030	0.855 ± 0.084	0.785 ± 0.186	0.744 ± 0.256	0.696 ± 0.260

Table 10: Affected-region F1 for the earlier 10-seed DELTANAR main grids. The gate is useful but not perfect, especially at larger sizes.

Task	16	32	64	128	256
SSSP	0.987 ± 0.002	0.994 ± 0.002	0.996 ± 0.003	0.994 ± 0.009	0.986 ± 0.025
Bottleneck	0.967 ± 0.009	0.954 ± 0.016	0.936 ± 0.057	0.895 ± 0.143	0.837 ± 0.178

Table 11: Exact node accuracy for the earlier 10-seed main grids. The paper emphasizes tie-tolerant validity, but exact pointer match shows the same broad trend.

C34/C35 hard-stress rerun. We then reran the copy-safe model in two harder regimes that were designed to attack the most plausible failure mode of the mechanism. C34 evaluates hard-repair size extrapolation at $n = 256, 512$ and wide-weight OOD up to weight 100. C35 repeats the same settings with the repair loss but without the independent copy-safe gate. This comparison keeps the training and evaluation budget matched, so the difference isolates the gate rather than the extra repair objective.

The size-extrapolation result is the cleanest positive evidence. At $n = 512$, copy-safe repair reaches 0.917 ± 0.186 SSSP node validity versus 0.663 ± 0.421 for repair-only, and 0.811 ± 0.295 bottleneck node validity versus 0.599 ± 0.336. The bottleneck graph result is also materially better: 0.256 ± 0.303 versus 0.016 ± 0.068 at $n = 512$. In the wide-weight OOD setting, absolute SSSP node validity is low for both variants, which means this stress should be treated as a remaining numeric-calibration limitation rather than as a solved setting. Even there, copy-safe repair improves node validity from 0.151 ± 0.232 to 0.348 ± 0.331 at $n = 256$ on SSSP and from 0.210 ± 0.292 to 0.619 ± 0.327 on bottleneck.

C42 noisy previous-solution audit. The one-step theorem assumes that the previous solution is exact. C42 directly stress-tests this assumption by keeping training unchanged, then corrupting the previous predecessor and previous scalar value at evaluation time. The audit is therefore different from rollout: it isolates sensitivity to imperfect memory without changing the edit stream or retraining objective.

The detailed C42 sweep uses corruption rates 0, 1%, 5%, and 10%. On SSSP at $n = 128$, copy-safe node validity moves from 0.958 to 0.936, 0.870, and 0.810 as corruption increases, while repair-only moves from 0.832 to 0.805, 0.729, and 0.669. On bottleneck at $n = 128$, copy-safe moves from 0.896 to 0.882, 0.819, and 0.746, while repair-only moves from 0.617 to 0.609, 0.577, and 0.539. Affected-gate recall remains high across the sweep, so the main degradation is not a failure to localize the original edit; it is the expected compounding effect of corrupted previous witnesses. Pure-copy node validity also falls by about 9–10 points at 10% corruption. This explains why graph validity rapidly approaches zero even when node validity remains meaningful.

C43 self-correction training audit. C43 asks whether training-time previous-memory corruption improves over the clean copy-safe model. The matched self-correcting variant corrupts 5% of the previous predecessor/value memory during training and recomputes the copy-safe target against that corrupted witness. This is a stricter intervention than C42, which corrupts memory only at evaluation time.

The main positive result from C43 is therefore not the noisy-training variant itself; it is the calibrated copy-safe model's long-horizon stability. In the 60-step SSSP rollout, clean copy-safe remains at 0.871 ± 0.125 at step 59, whereas the earlier rollout audit of the uncalibrated model ended at 0.598 ± 0.307. The self-correcting training variant is mixed: it improves 10% noisy-memory node validity on SSSP at $n = 64$ (0.889 vs. 0.863) and bottleneck at $n = 64$ (0.834 vs. 0.791), but loses clean graph validity and underperforms clean copy-safe in rollout. This indicates that simply injecting noisy memory during training is not yet the right self-correction objective.

C44/C45 certificate-repair audits. C44 completed the larger fixed-target noisy-memory and unguarded certificate-repair audit over all 80 planned task/training/seed rows. C45 then started a deeper guarded-certificate audit with 120-step rollout and broader OOD suites. The returned C45 checkpoint contains all 20 copy-safe and copy-safe+guarded-certificate seeds for SSSP and bottleneck, but only two fixed-target noisy SSSP seeds and no fixed-target bottleneck rows. We therefore use C45 as a 20-seed guarded-certificate diagnostic, not as a completed fixed-target noisy-training comparison. No C46 large guarded files were present in the local package.

The completed C44 result makes the self-correction story more precise. Fixed-target noisy training is mixed rather than a clean win. At $n = 256$ on SSSP, it improves 20% noisy-memory node validity from 0.670 ± 0.179 to 0.770 ± 0.119, but clean graph validity is lower than clean copy-safe (0.376 ± 0.325 vs. 0.427 ± 0.381) and graph validity remains zero under 20% noisy memory. On bottleneck, it improves node validity but not graph exactness. The harder 80-step SSSP rollout remains stable for

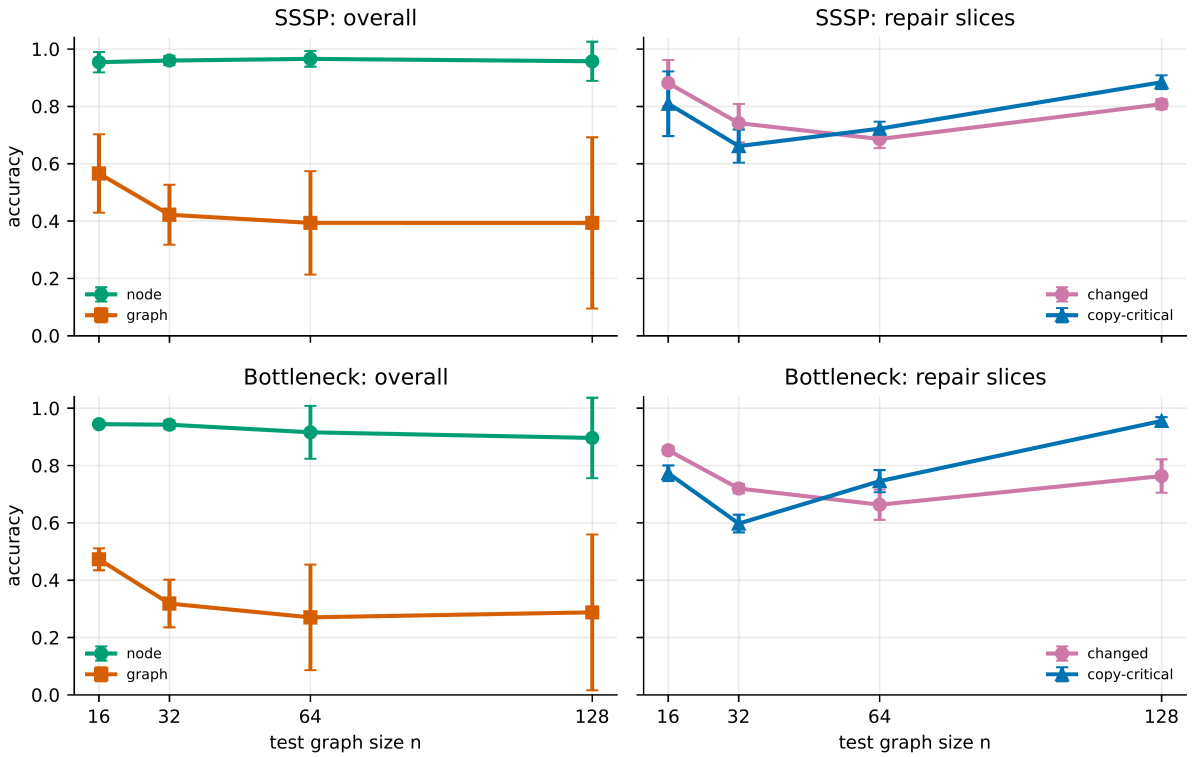

Figure 11: C2 20-seed server rerun of the copy-safe gate. The model preserves high node accuracy while substantially improving changed-output and copy-critical repair slices under the hard train4/test12 update regime.

clean copy-safe: 0.854 ± 0.125 at $n = 64$ and 0.842 ± 0.181 at $n = 128$ after 80 steps, with fixed-target noisy training essentially tied rather than decisively better.

Unguarded certificate repair is an instructive negative result. It repairs nearly every vertex and catastrophically over-repairs SSSP, while producing very large bottleneck gains. Guarding the repair by the learned copy-safety signal in C45 reduces the most extreme over-repair, but the outcome is still task dependent: it helps bottleneck under heavier noisy-memory stress and hurts SSSP under the same stress. Thus certificate repair is not part of the main DeltaNAR method; it is evidence that learned or task-aware correction policies are the next self-correction frontier.

C48 focused guard-profile sweep. C48 spends a two-day server budget on the highest-upside self-correction question: can a more conservative thresholded guard keep the bottleneck certificate gains without damaging SSSP? It trains the copy-safe model over eight seeds and evaluates several repair profiles without retraining. The answer is still negative for a promoted method. At $n = 256$ with 30% noisy previous memory, all guarded profiles hurt SSSP, especially changed-output and copy-critical validity. The same profiles are strongly positive on bottleneck ER, wide-weight, and heavy-tail slices. Thus local certificates contain useful correction signal, but the hand-thresholded guard is not sufficiently task-aware.

C49 task-aware routing audit. C49 asks the natural follow-up to C48: if guarded certificate repair is harmful for SSSP but useful for bottleneck, can a task-aware policy retain raw copy-safe updates for SSSP and use guarded repair only for bottleneck? Over 12 seeds at $n = 256$, the answer is a scoped yes. The SSSP rows again show that certificate repair should not be enabled for min-plus updates under noisy memory. The bottleneck rows show stable gains in node, changed-output, and copy-critical validity under the same stress. This does not solve graph-level self-correction, but it turns the certificate signal from a purely negative diagnostic into a task-routed positive result.

Independent low-memory and topology audit. The final server audit adds three pieces of evidence that are useful for testing the boundary of the core method. First, the low-memory 1024-node probe completes without out-of-memory failure when batch sizes and hidden widths are conservative. SSSP remains high at 1024 nodes (0.996 copy-safe node validity and 0.881 graph validity), while bottleneck exposes the largest benefit of conservative certificate repair (0.716 to 0.912 node validity). Second, the topology-breadth audit shows that this repair signal is conditional: on ER bottleneck graphs with 30% noisy memory, default guarded repair raises node validity from 0.495 to 0.814 and copy-safe-region validity from 0.658 to 0.943, but the gain is smaller on heavy-tail graphs (0.416 to 0.467) and does not hold on BA graphs. Third, a 120-step SSSP rollout at $n = 512$ is a negative

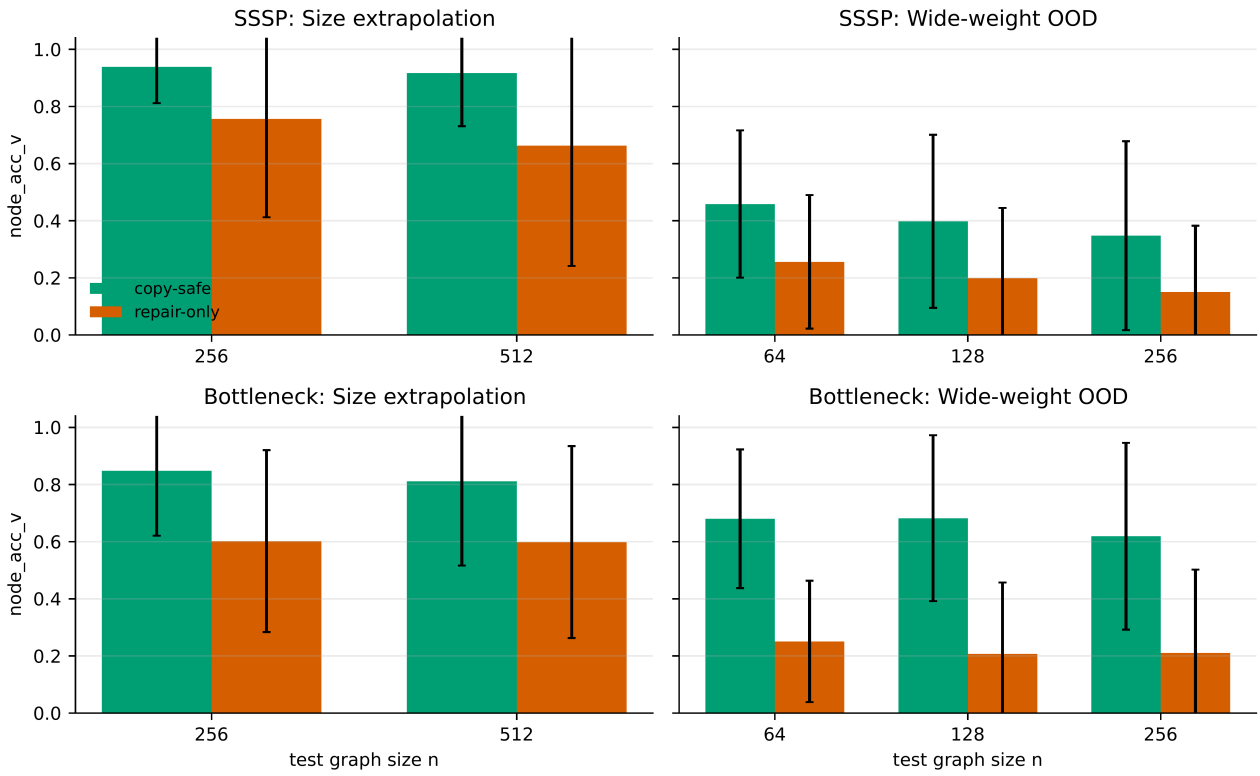

Figure 12: C34/C35 hard-stress comparison. The copy-safe gate improves full-node validity over the matched repair-only ablation under both size extrapolation and wide-weight OOD. The wide-weight setting remains difficult in absolute terms, especially for SSSP, but the ablation confirms that copy-safety is contributing beyond the repair loss itself.

control for over-repair: raw copy-safe reaches 0.433 final node validity, whereas the aggressive repair profiles end between 0.045 and 0.119. Thus repair is a routed auxiliary tool, not a universal contraction mechanism.

C38 efficiency profile. C38 profiles the copy-safe repair model at the same large graph sizes. This is a lightweight profiling run rather than a headline accuracy run, but it gives the practical scaling constants for the dense implementation used here.

Baseline Grids

Ablations

OOD Breadth

Extreme Size and Rollout

Generalist, Transfer, and MST

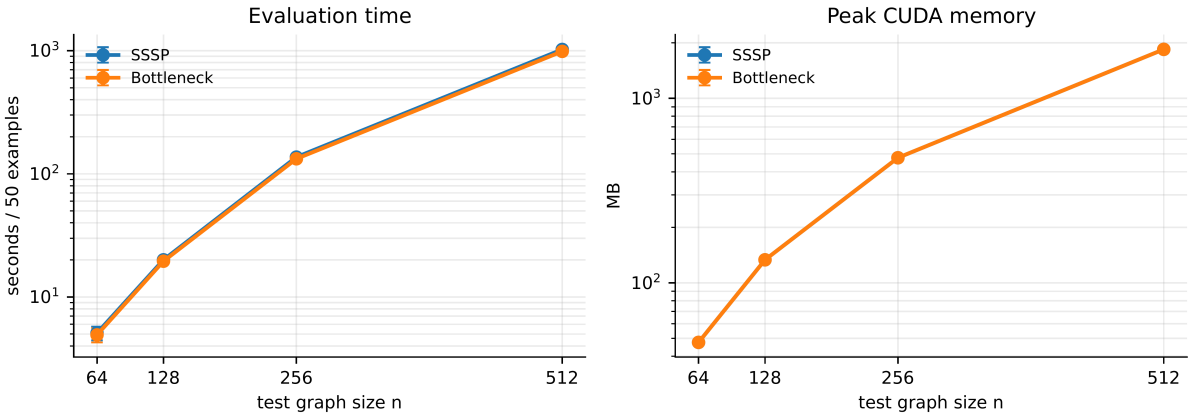

Figure 13: C38 efficiency profile for copy-safe repair. Runtime grows steeply with graph size under the current dense implementation, while peak CUDA memory remains below 2 GB at $n = 512$ on the RTX 2080 Ti runs.

Setting	Model	node	graph	changed	copy-critical
SSSP k=1	DeltaNAR	0.987 ± 0.024	0.760 ± 0.335	0.994 ± 0.003	1.000 ± 0.000
SSSP k=1	no-copy	0.292 ± 0.071	0.000 ± 0.000	0.993 ± 0.003	1.000 ± 0.000
SSSP k=1	Prev+Delta	0.382 ± 0.055	0.000 ± 0.000	0.991 ± 0.004	0.998 ± 0.002
SSSP k=1	Recomp+gate	0.236 ± 0.097	0.000 ± 0.000	0.992 ± 0.004	0.999 ± 0.002
SSSP k=1	Recompute	0.053 ± 0.041	0.000 ± 0.000	0.999 ± 0.002	1.000 ± 0.000
SSSP k=1	Attn	0.180 ± 0.087	0.000 ± 0.000	0.995 ± 0.004	1.000 ± 0.001
SSSP k=8	DeltaNAR	0.827 ± 0.335	0.530 ± 0.414	0.951 ± 0.011	0.984 ± 0.010
SSSP k=8	no-copy	0.269 ± 0.024	0.000 ± 0.000	0.938 ± 0.008	0.979 ± 0.007
SSSP k=8	Prev+Delta	0.348 ± 0.044	0.000 ± 0.000	0.943 ± 0.012	0.984 ± 0.004
SSSP k=8	Recomp+gate	0.166 ± 0.087	0.000 ± 0.000	0.949 ± 0.018	0.983 ± 0.011
SSSP k=8	Recompute	0.048 ± 0.009	0.000 ± 0.000	0.971 ± 0.010	0.991 ± 0.005
SSSP k=8	Attn	0.197 ± 0.118	0.000 ± 0.000	0.954 ± 0.026	0.983 ± 0.014
Bottleneck k=1	DeltaNAR	0.852 ± 0.163	0.014 ± 0.022	0.991 ± 0.003	1.000 ± 0.000
Bottleneck k=1	no-copy	0.162 ± 0.044	0.000 ± 0.000	0.993 ± 0.003	1.000 ± 0.000
Bottleneck k=1	Prev+Delta	0.339 ± 0.078	0.000 ± 0.000	0.989 ± 0.005	1.000 ± 0.000
Bottleneck k=1	Recomp+gate	0.132 ± 0.041	0.000 ± 0.000	0.997 ± 0.004	1.000 ± 0.000
Bottleneck k=1	Recompute	0.138 ± 0.016	0.000 ± 0.000	0.993 ± 0.005	1.000 ± 0.000
Bottleneck k=1	Attn	0.154 ± 0.043	0.000 ± 0.000	0.992 ± 0.004	1.000 ± 0.000
Bottleneck k=4	DeltaNAR	0.858 ± 0.102	0.000 ± 0.000	0.965 ± 0.004	1.000 ± 0.001
Bottleneck k=4	no-copy	0.132 ± 0.014	0.000 ± 0.000	0.966 ± 0.009	0.999 ± 0.002
Bottleneck k=4	Prev+Delta	0.297 ± 0.088	0.000 ± 0.000	0.959 ± 0.007	0.999 ± 0.002
Bottleneck k=4	Recomp+gate	0.149 ± 0.044	0.000 ± 0.000	0.972 ± 0.006	0.999 ± 0.002
Bottleneck k=4	Recompute	0.193 ± 0.043	0.000 ± 0.000	0.960 ± 0.009	0.999 ± 0.002
Bottleneck k=4	Attn	0.151 ± 0.057	0.000 ± 0.000	0.972 ± 0.007	0.999 ± 0.002

Table 12: Server diagnostic slices at $n = 256$. "changed" is accuracy on vertices whose output differs after the edit. "copy-critical" is accuracy on vertices where blindly copying the old witness is invalid. Multi-edit settings stress the repair path more than the default single-edit setting.

Task	n	node	graph	changed	copy-critical
SSSP	16	0.943 ± 0.039	0.491 ± 0.176	0.986 ± 0.012	0.994 ± 0.007
SSSP	32	0.797 ± 0.050	0.002 ± 0.004	0.980 ± 0.005	0.989 ± 0.003
SSSP	64	0.520 ± 0.098	0.000 ± 0.000	0.982 ± 0.003	0.991 ± 0.002
Bottleneck	16	0.939 ± 0.006	0.428 ± 0.038	0.988 ± 0.003	0.992 ± 0.003
Bottleneck	32	0.787 ± 0.030	0.002 ± 0.004	0.972 ± 0.005	0.987 ± 0.002
Bottleneck	64	0.517 ± 0.054	0.000 ± 0.000	0.972 ± 0.004	0.994 ± 0.002

Table 13: Triplet diagnostic baseline. Triplet features improve local expressivity, but size extrapolation remains limited.

Task	n	Variant	node	changed	copy-critical
SSSP	64	raw no-copy	0.337 ± 0.023	0.970 ± 0.006	0.983 ± 0.006
SSSP	64	predicted copy	0.997 ± 0.002	0.965 ± 0.002	0.986 ± 0.007
SSSP	64	oracle affected copy	0.999 ± 0.000	0.968 ± 0.006	0.984 ± 0.007
SSSP	64	oracle copy-safe upper	0.999 ± 0.000	0.963 ± 0.007	0.983 ± 0.006
SSSP	64	pure copy	0.999 ± 0.000	0.963 ± 0.000	1.000 ± 0.000
SSSP	128	raw no-copy	0.217 ± 0.066	0.983 ± 0.005	0.993 ± 0.005
SSSP	128	predicted copy	0.998 ± 0.004	0.980 ± 0.002	0.993 ± 0.006
SSSP	128	oracle affected copy	1.000 ± 0.000	0.983 ± 0.004	0.993 ± 0.004
SSSP	128	oracle copy-safe upper	1.000 ± 0.000	0.984 ± 0.004	0.993 ± 0.005
SSSP	128	pure copy	1.000 ± 0.000	0.981 ± 0.000	1.000 ± 0.000
Bottleneck	64	raw no-copy	0.355 ± 0.066	0.949 ± 0.003	0.994 ± 0.003
Bottleneck	64	predicted copy	0.961 ± 0.017	0.962 ± 0.006	0.992 ± 0.002
Bottleneck	64	oracle affected copy	0.980 ± 0.002	0.949 ± 0.003	0.994 ± 0.003
Bottleneck	64	oracle copy-safe upper	0.998 ± 0.001	0.982 ± 0.003	0.994 ± 0.003
Bottleneck	64	pure copy	0.999 ± 0.000	0.976 ± 0.000	1.000 ± 0.000
Bottleneck	128	raw no-copy	0.200 ± 0.043	0.981 ± 0.006	1.000 ± 0.000
Bottleneck	128	predicted copy	0.951 ± 0.026	0.986 ± 0.004	1.000 ± 0.000
Bottleneck	128	oracle affected copy	0.980 ± 0.011	0.981 ± 0.006	1.000 ± 0.000
Bottleneck	128	oracle copy-safe upper	0.991 ± 0.012	0.989 ± 0.002	1.000 ± 0.000
Bottleneck	128	pure copy	1.000 ± 0.000	0.989 ± 0.000	1.000 ± 0.000

Table 14: Copy-oracle study. Predicted copy already captures much of the copy-safe gain; oracle copy variants bound the remaining room for improving the gate and copy decision.

Task	Model	node	graph	changed	copy-critical
SSSP	DeltaNAR	0.974 ± 0.004	0.520 ± 0.004	0.376 ± 0.029	0.346 ± 0.034
SSSP	no-copy	0.334 ± 0.048	0.000 ± 0.000	0.250 ± 0.040	0.291 ± 0.030
SSSP	Prev+Delta	0.626 ± 0.034	0.000 ± 0.000	0.338 ± 0.016	0.299 ± 0.007
SSSP	Recompute	0.141 ± 0.010	0.000 ± 0.000	0.193 ± 0.024	0.268 ± 0.047
Bottleneck	DeltaNAR	0.475 ± 0.182	0.000 ± 0.000	0.305 ± 0.080	0.436 ± 0.047
Bottleneck	no-copy	0.330 ± 0.026	0.000 ± 0.000	0.217 ± 0.016	0.430 ± 0.094
Bottleneck	Prev+Delta	0.416 ± 0.115	0.000 ± 0.000	0.267 ± 0.063	0.482 ± 0.046
Bottleneck	Recompute	0.197 ± 0.027	0.000 ± 0.000	0.224 ± 0.016	0.369 ± 0.067

Table 15: Local hard-repair audit at $n = 64$. Models are trained with four simultaneous edits and tested with twelve simultaneous edits. SSSP keeps a clear full-node and graph advantage, while bottleneck exposes the current repair weakness more sharply.

Task	Variant	node	graph	changed	copy-critical
SSSP	raw no-copy	0.208 ± 0.010	0.000 ± 0.000	0.286 ± 0.024	0.304 ± 0.017
SSSP	predicted copy	0.974 ± 0.004	0.520 ± 0.004	0.376 ± 0.029	0.346 ± 0.034
SSSP	oracle affected copy	0.977 ± 0.001	0.514 ± 0.005	0.348 ± 0.030	0.325 ± 0.031
SSSP	oracle copy-safe upper	0.984 ± 0.001	0.557 ± 0.008	0.509 ± 0.015	0.304 ± 0.017
SSSP	pure copy	0.980 ± 0.000	0.525 ± 0.000	0.441 ± 0.000	0.850 ± 0.000
Bottleneck	raw no-copy	0.252 ± 0.029	0.000 ± 0.000	0.222 ± 0.040	0.440 ± 0.048
Bottleneck	predicted copy	0.475 ± 0.182	0.000 ± 0.000	0.305 ± 0.080	0.436 ± 0.047
Bottleneck	oracle affected copy	0.743 ± 0.011	0.000 ± 0.000	0.222 ± 0.040	0.440 ± 0.048
Bottleneck	oracle copy-safe upper	0.988 ± 0.001	0.587 ± 0.045	0.734 ± 0.009	0.440 ± 0.048
Bottleneck	pure copy	0.987 ± 0.000	0.595 ± 0.000	0.695 ± 0.000	1.000 ± 0.000

Table 16: Hard-repair oracle decomposition at $n = 64$. The gap between predicted copy and copy-safe oracle points to copy-safety calibration and local repair as the next bottleneck, not merely affected-gate recall.

Task	Setting	node	graph	changed	copy-critical	gate recall
SSSP	train4/test12	0.974 ± 0.004	0.520 ± 0.004	0.376 ± 0.029	0.346 ± 0.034	0.873 ± 0.096
SSSP	train12/test12	0.966 ± 0.011	0.438 ± 0.094	0.360 ± 0.021	0.273 ± 0.030	0.937 ± 0.069
Bottleneck	train4/test12	0.475 ± 0.182	0.000 ± 0.000	0.305 ± 0.080	0.436 ± 0.047	0.984 ± 0.013
Bottleneck	train12/test12	0.327 ± 0.054	0.000 ± 0.000	0.267 ± 0.057	0.437 ± 0.034	1.000 ± 0.000

Table 17: Matched-hard training check for DeltaNAR at $n = 64$. Training on twelve-edit updates does not resolve the changed-output weakness, so the hard-repair gap is not merely a train/test edit-intensity mismatch.

Task	Variant	node	graph	changed	copy-critical	gate recall
SSSP	baseline	0.974 ± 0.004	0.520 ± 0.004	0.376 ± 0.029	0.346 ± 0.034	0.873 ± 0.096
SSSP	copy-safe gate	0.967 ± 0.009	0.417 ± 0.040	0.477 ± 0.041	0.330 ± 0.047	0.968 ± 0.030
Bottleneck	baseline	0.475 ± 0.182	0.000 ± 0.000	0.305 ± 0.080	0.436 ± 0.047	0.984 ± 0.013
Bottleneck	copy-safe gate	0.782 ± 0.227	0.156 ± 0.206	0.514 ± 0.163	0.469 ± 0.073	0.993 ± 0.006

Table 18: Local copy-safe gate comparison on the hard-repair audit at $n = 64$. The local run identified copy-safety as the strongest architectural direction before the larger C2 server rerun.

Task	n	node	graph	changed	copy-critical	gate recall
SSSP	16	0.954 ± 0.036	0.566 ± 0.137	0.881 ± 0.080	0.809 ± 0.113	0.990 ± 0.017
SSSP	32	0.960 ± 0.016	0.422 ± 0.105	0.742 ± 0.067	0.662 ± 0.058	0.989 ± 0.019
SSSP	64	0.966 ± 0.027	0.394 ± 0.180	0.686 ± 0.032	0.722 ± 0.025	0.983 ± 0.021
SSSP	128	0.958 ± 0.068	0.394 ± 0.299	0.808 ± 0.018	0.884 ± 0.024	0.991 ± 0.017
Bottleneck	16	0.944 ± 0.006	0.473 ± 0.038	0.853 ± 0.014	0.773 ± 0.027	0.995 ± 0.003
Bottleneck	32	0.942 ± 0.014	0.319 ± 0.083	0.720 ± 0.016	0.598 ± 0.031	0.982 ± 0.012
Bottleneck	64	0.916 ± 0.092	0.271 ± 0.184	0.663 ± 0.053	0.746 ± 0.039	0.951 ± 0.066
Bottleneck	128	0.896 ± 0.140	0.288 ± 0.272	0.763 ± 0.058	0.956 ± 0.013	0.925 ± 0.161

Table 19: C2 copy-safe gate 20-seed server rerun. Relative to the local three-seed candidate, the full-budget run keeps SSSP node accuracy stable and turns bottleneck into a much stronger result: at $n = 64$, bottleneck reaches 0.916 ± 0.092 node accuracy, 0.271 ± 0.184 graph accuracy, and 0.663 ± 0.053 changed-output accuracy.

Setting	Task	n	Variant	node	changed	copy-critical	graph
Size	SSSP	256	copy-safe	0.939 ± 0.127	0.962 ± 0.007	0.988 ± 0.006	0.431 ± 0.381
Size	SSSP	256	repair-only	0.756 ± 0.345	0.967 ± 0.008	0.989 ± 0.005	0.352 ± 0.355
Size	SSSP	512	copy-safe	0.917 ± 0.186	0.986 ± 0.005	0.995 ± 0.003	0.461 ± 0.426
Size	SSSP	512	repair-only	0.663 ± 0.421	0.987 ± 0.006	0.995 ± 0.002	0.461 ± 0.418
Size	Bottleneck	256	copy-safe	0.848 ± 0.227	0.947 ± 0.018	1.000 ± 0.001	0.278 ± 0.284
Size	Bottleneck	256	repair-only	0.602 ± 0.318	0.931 ± 0.017	0.999 ± 0.001	0.009 ± 0.039
Size	Bottleneck	512	copy-safe	0.811 ± 0.295	0.982 ± 0.006	1.000 ± 0.000	0.256 ± 0.303
Size	Bottleneck	512	repair-only	0.599 ± 0.336	0.978 ± 0.007	1.000 ± 0.000	0.016 ± 0.068
Wide	SSSP	64	copy-safe	0.458 ± 0.258	0.538 ± 0.047	0.755 ± 0.115	0.007 ± 0.023
Wide	SSSP	64	repair-only	0.256 ± 0.234	0.588 ± 0.049	0.750 ± 0.070	0.000 ± 0.000
Wide	SSSP	128	copy-safe	0.398 ± 0.303	0.733 ± 0.058	0.890 ± 0.058	0.002 ± 0.005
Wide	SSSP	128	repair-only	0.199 ± 0.246	0.792 ± 0.081	0.881 ± 0.058	0.001 ± 0.006
Wide	SSSP	256	copy-safe	0.348 ± 0.331	0.871 ± 0.045	0.965 ± 0.027	0.000 ± 0.000
Wide	SSSP	256	repair-only	0.151 ± 0.232	0.911 ± 0.042	0.965 ± 0.024	0.006 ± 0.027
Wide	Bottleneck	64	copy-safe	0.680 ± 0.243	0.516 ± 0.061	0.850 ± 0.092	0.017 ± 0.048
Wide	Bottleneck	64	repair-only	0.251 ± 0.213	0.468 ± 0.060	0.736 ± 0.102	0.000 ± 0.000
Wide	Bottleneck	128	copy-safe	0.682 ± 0.290	0.600 ± 0.046	0.946 ± 0.048	0.040 ± 0.098
Wide	Bottleneck	128	repair-only	0.207 ± 0.250	0.638 ± 0.094	0.892 ± 0.072	0.000 ± 0.000
Wide	Bottleneck	256	copy-safe	0.619 ± 0.327	0.643 ± 0.073	0.969 ± 0.033	0.050 ± 0.146
Wide	Bottleneck	256	repair-only	0.210 ± 0.292	0.679 ± 0.113	0.932 ± 0.066	0.000 ± 0.000

Table 20: C34/C35 20-seed hard-stress server rerun. C34 uses copy-safe repair; C35 is the matched repair-only ablation. The strongest separation is in full-node validity and bottleneck graph validity, while changed-output slices are sometimes comparable because they cover only the small recomputed region.

Task	n	Variant	clean node	10% noisy node	10% noisy changed	10% noisy graph
SSSP	64	copy-safe	0.966 ± 0.027	0.863 ± 0.057	0.452 ± 0.102	0.004 ± 0.005
SSSP	64	repair-only	0.914 ± 0.123	0.794 ± 0.127	0.462 ± 0.081	0.003 ± 0.004
SSSP	128	copy-safe	0.958 ± 0.068	0.810 ± 0.125	0.283 ± 0.106	0.000 ± 0.000
SSSP	128	repair-only	0.832 ± 0.250	0.669 ± 0.261	0.302 ± 0.087	0.000 ± 0.000
Bottleneck	64	copy-safe	0.916 ± 0.092	0.791 ± 0.098	0.357 ± 0.037	0.001 ± 0.002
Bottleneck	64	repair-only	0.668 ± 0.148	0.593 ± 0.137	0.351 ± 0.054	0.000 ± 0.000
Bottleneck	128	copy-safe	0.896 ± 0.140	0.746 ± 0.163	0.224 ± 0.029	0.000 ± 0.000
Bottleneck	128	repair-only	0.617 ± 0.263	0.539 ± 0.233	0.218 ± 0.037	0.000 ± 0.000

Table 21: C42 noisy previous-solution audit over 20 seeds. Training is unchanged; only the previous-solution input is corrupted at evaluation time. Node validity degrades smoothly under 10% corruption and copy-safe remains stronger than repair-only in full-node validity, especially on bottleneck. Whole-graph exactness collapses under corruption, so this audit supports robustness of the node-level update mechanism but not long-horizon self-correction.

Task	n	Variant	clean node	10% noisy node	clean graph	10% noisy graph
SSSP	64	copy-safe	0.966 ± 0.027	0.863 ± 0.057	0.394 ± 0.180	0.004 ± 0.004
SSSP	64	self-correct	0.957 ± 0.037	0.889 ± 0.069	0.312 ± 0.198	0.013 ± 0.016
SSSP	128	copy-safe	0.958 ± 0.068	0.809 ± 0.126	0.394 ± 0.299	0.000 ± 0.000
SSSP	128	self-correct	0.905 ± 0.212	0.826 ± 0.199	0.274 ± 0.270	0.000 ± 0.000
Bottleneck	64	copy-safe	0.916 ± 0.092	0.791 ± 0.099	0.271 ± 0.184	0.002 ± 0.003
Bottleneck	64	self-correct	0.897 ± 0.054	0.834 ± 0.053	0.054 ± 0.099	0.001 ± 0.003
Bottleneck	128	copy-safe	0.896 ± 0.140	0.747 ± 0.162	0.288 ± 0.272	0.000 ± 0.000
Bottleneck	128	self-correct	0.832 ± 0.132	0.757 ± 0.133	0.012 ± 0.032	0.000 ± 0.000

Table 22: C43 training-time noisy-memory audit over 20 seeds. The self-correct variant improves some 10% noisy-memory node metrics, especially bottleneck changed/copy-critical slices, but it hurts clean node or graph validity and does not solve graph-level exactness under corruption.

Variant	$t = 0$	$t = 1$	$t = 9$	$t = 19$	$t = 39$	$t = 59$
copy-safe	0.993 ± 0.010	0.981 ± 0.027	0.937 ± 0.090	0.915 ± 0.104	0.888 ± 0.121	0.871 ± 0.125
self-correct	0.981 ± 0.020	0.901 ± 0.195	0.854 ± 0.215	0.833 ± 0.219	0.810 ± 0.225	0.792 ± 0.225

Table 23: C43 SSSP 60-step rollout at $n = 64$, `node_acc_v`. The calibrated copy-safe model is substantially more stable than the earlier five-seed rollout audit, but adding naive training-time memory corruption does not improve over clean copy-safe training.

Task	Variant	noise	$n = 256$ node	$n = 256$ graph	changed	copy-critical
SSSP	copy-safe	0	0.939 ± 0.126	0.427 ± 0.381	0.690 ± 0.045	0.778 ± 0.077
SSSP	fixed-noisy	0	0.949 ± 0.113	0.376 ± 0.325	0.701 ± 0.052	0.791 ± 0.071
SSSP	copy-safe	20%	0.670 ± 0.179	0.000 ± 0.000	0.197 ± 0.099	0.233 ± 0.191
SSSP	fixed-noisy	20%	0.770 ± 0.119	0.000 ± 0.000	0.161 ± 0.084	0.190 ± 0.140
SSSP	copy-safe + cert.	20%	0.037 ± 0.008	0.000 ± 0.000	0.027 ± 0.010	0.026 ± 0.010
Bottleneck	copy-safe	0	0.847 ± 0.227	0.274 ± 0.278	0.657 ± 0.127	0.922 ± 0.031
Bottleneck	fixed-noisy	0	0.891 ± 0.200	0.152 ± 0.253	0.682 ± 0.092	0.920 ± 0.031
Bottleneck	copy-safe	20%	0.602 ± 0.218	0.000 ± 0.000	0.167 ± 0.030	0.111 ± 0.043
Bottleneck	fixed-noisy	20%	0.752 ± 0.098	0.000 ± 0.000	0.150 ± 0.034	0.108 ± 0.155
Bottleneck	copy-safe + cert.	20%	0.894 ± 0.224	0.583 ± 0.432	0.896 ± 0.213	0.896 ± 0.215

Table 24: Completed C44 audit at $n = 256$. Fixed-target noisy training improves some noisy-memory node metrics but does not solve graph exactness or rollout. Unguarded certificate repair is unsafe for SSSP and strongly positive for bottleneck, motivating guarded or learned correction rather than a universal local repair rule.

Suite/task	Variant	noise	$n = 256$ node	$n = 256$ graph	changed	copy-critical
SSSP/ER	copy-safe	30%	0.598 ± 0.157	0.000 ± 0.000	0.199 ± 0.100	0.233 ± 0.180
SSSP/ER	guarded cert.	30%	0.508 ± 0.178	0.000 ± 0.000	0.041 ± 0.020	0.046 ± 0.067
Bottleneck/ER	copy-safe	30%	0.520 ± 0.203	0.000 ± 0.000	0.166 ± 0.029	0.115 ± 0.044
Bottleneck/ER	guarded cert.	30%	0.730 ± 0.165	0.000 ± 0.000	0.596 ± 0.275	0.577 ± 0.303
SSSP/heavy-tail	copy-safe	30%	0.499 ± 0.198	0.000 ± 0.000	0.333 ± 0.197	0.314 ± 0.205
SSSP/heavy-tail	guarded cert.	30%	0.346 ± 0.166	0.000 ± 0.000	0.100 ± 0.035	0.075 ± 0.037
Bottleneck/heavy-tail	copy-safe	30%	0.438 ± 0.158	0.000 ± 0.000	0.045 ± 0.016	0.039 ± 0.017
Bottleneck/heavy-tail	guarded cert.	30%	0.553 ± 0.144	0.000 ± 0.000	0.305 ± 0.196	0.302 ± 0.198
SSSP/BA	copy-safe	30%	0.927 ± 0.047	0.004 ± 0.008	0.875 ± 0.087	0.779 ± 0.157
SSSP/BA	guarded cert.	30%	0.877 ± 0.034	0.000 ± 0.000	0.738 ± 0.054	0.524 ± 0.098
Bottleneck/BA	copy-safe	30%	0.906 ± 0.028	0.000 ± 0.000	0.802 ± 0.045	0.642 ± 0.085
Bottleneck/BA	guarded cert.	30%	0.891 ± 0.031	0.000 ± 0.000	0.756 ± 0.043	0.556 ± 0.079

Table 25: C45 guarded-certificate checkpoint at $n = 256$ and 30% noisy previous memory. The guarded rule helps bottleneck ER/heavy-tail slices but hurts SSSP and BA slices, so it remains a diagnostic rather than a promoted method.

Task/suite	Variant	node	changed	copy-critical	cert. rate
SSSP/ER	raw copy-safe	0.663 ± 0.077	0.193 ± 0.098	0.181 ± 0.102	0.000 ± 0.000
SSSP/ER	cons. guard	0.611 ± 0.100	0.055 ± 0.052	0.040 ± 0.050	0.294 ± 0.162
SSSP/ER	default guard	0.575 ± 0.123	0.038 ± 0.018	0.025 ± 0.015	0.369 ± 0.193
SSSP/heavy-tail	raw copy-safe	0.542 ± 0.200	0.349 ± 0.210	0.328 ± 0.220	0.000 ± 0.000
SSSP/heavy-tail	cons. guard	0.393 ± 0.189	0.108 ± 0.055	0.080 ± 0.050	0.457 ± 0.301
Bottleneck/ER	raw copy-safe	0.457 ± 0.216	0.154 ± 0.028	0.112 ± 0.044	0.000 ± 0.000
Bottleneck/ER	cons. guard	0.632 ± 0.191	0.544 ± 0.260	0.533 ± 0.290	0.294 ± 0.212
Bottleneck/ER	default guard	0.691 ± 0.217	0.602 ± 0.288	0.591 ± 0.313	0.398 ± 0.283
Bottleneck/heavy-tail	raw copy-safe	0.373 ± 0.166	0.045 ± 0.011	0.042 ± 0.013	0.000 ± 0.000
Bottleneck/heavy-tail	default guard	0.486 ± 0.162	0.293 ± 0.248	0.291 ± 0.250	0.300 ± 0.200

Table 26: C48 focused guard-profile sweep at $n = 256$ with 30% noisy previous memory over eight seeds. "Cons. guard" is the conservative copy-unsafe profile and "default guard" is the zero-threshold copy-unsafe profile. Guarded local repair remains task-dependent: it substantially improves bottleneck repair slices but degrades SSSP, so it is not promoted into DeltaNAR.

Task/suite	Variant	node	changed	copy-critical	cert. rate
SSSP/ER	raw copy-safe	0.656 ± 0.076	0.191 ± 0.100	0.256 ± 0.223	0.000 ± 0.000
SSSP/ER	cons. guard	0.606 ± 0.097	0.053 ± 0.044	0.111 ± 0.233	0.282 ± 0.163
SSSP/ER	default guard	0.578 ± 0.113	0.040 ± 0.018	0.043 ± 0.048	0.343 ± 0.190
Bottleneck/ER	raw copy-safe	0.526 ± 0.204	0.164 ± 0.029	0.114 ± 0.037	0.000 ± 0.000
Bottleneck/ER	cons. guard	0.686 ± 0.177	0.571 ± 0.228	0.556 ± 0.253	0.250 ± 0.186
Bottleneck/ER	default guard	0.733 ± 0.190	0.630 ± 0.249	0.616 ± 0.270	0.328 ± 0.252
Bottleneck/wide100	raw copy-safe	0.375 ± 0.229	0.031 ± 0.003	0.095 ± 0.189	0.000 ± 0.000
Bottleneck/wide100	default guard	0.451 ± 0.228	0.118 ± 0.221	0.182 ± 0.270	0.100 ± 0.267
Bottleneck/heavy-tail	raw copy-safe	0.391 ± 0.148	0.044 ± 0.010	0.040 ± 0.011	0.000 ± 0.000
Bottleneck/heavy-tail	default guard	0.513 ± 0.142	0.307 ± 0.222	0.304 ± 0.224	0.282 ± 0.176

Table 27: C49 task-aware routing audit at $n = 256$ with 30% noisy previous memory over 12 seeds. SSSP should use raw copy-safe updates, while bottleneck benefits from guarded certificate repair. Averaged over ER, wide-weight, heavy-tail, and BA, raw-SSSP plus default-guard bottleneck improves bottleneck node, changed-output, and copy-critical validity by 0.098, 0.192, and 0.192 respectively, with no SSSP penalty by construction.

Audit	Task/suite	Raw copy-safe	Routed/repair	Interpretation
Low-memory 1024	SSSP	.996 node / .881 graph	.995 node / .887 graph	feasible without OOM
Low-memory 1024	Bottleneck	.716 node / .312 graph	.912 node / .312 graph	large node-validity gain
Topology breadth	Bottleneck ER	.495 node / .658 safe-region	.814 node / .943 safe-region	strongest repair case
Topology breadth	Bottleneck heavy-tail	.416 node	.467 node	smaller conditional gain
120-step rollout	SSSP $n = 512$	.433 final node	.045–.119 final node	over-repair hurts rollout

Table 28: Independent scaling and repair-boundary audit. The results strengthen the scaling and repair-boundary story while preserving the main claim: DeltaNAR uses copy-safe delta updates as the default mechanism, and local certificate repair should be routed only where the task semantics support it.

Task	n	eval sec.	peak MB	samples/sec	node
SSSP	64	5.1 ± 0.6	47.6 ± 0.0	9.97 ± 1.24	0.390 ± 0.229
SSSP	128	20.0 ± 0.2	133.3 ± 0.0	2.50 ± 0.02	0.304 ± 0.267
SSSP	256	136.7 ± 0.4	476.4 ± 0.0	0.37 ± 0.00	0.236 ± 0.281
SSSP	512	1022.6 ± 2.1	1842.8 ± 0.0	0.05 ± 0.00	0.196 ± 0.285
Bottleneck	64	4.9 ± 0.6	47.6 ± 0.0	10.32 ± 1.28	0.792 ± 0.132
Bottleneck	128	19.5 ± 0.5	133.3 ± 0.0	2.56 ± 0.07	0.820 ± 0.125
Bottleneck	256	132.4 ± 1.4	476.4 ± 0.0	0.38 ± 0.00	0.824 ± 0.115
Bottleneck	512	987.0 ± 4.6	1842.8 ± 0.0	0.05 ± 0.00	0.829 ± 0.122

Table 29: C38 efficiency profile for copy-safe repair. Times are for evaluating 50 examples after the lightweight C38 training budget; memory is peak allocated CUDA memory during evaluation.

Model	16	32	64	128	256
DeltaNAR	0.988	0.995	0.997	0.998	0.999
Prev+Delta	0.963	0.928	0.822	0.623	0.423
Recomp+gate	0.945	0.752	0.482	0.327	0.231
Triplet	0.925	0.781	0.544	–	–
Recompute	0.495	0.333	0.172	0.084	0.040

Table 30: SSSP baseline grid, `node_acc_v`.

Model	16	32	64	128	256
DeltaNAR	0.868	0.907	0.944	0.968	0.982
Prev+Delta	0.618	0.152	0.001	0.000	0.000
Recomp+gate	0.468	0.000	0.000	0.000	0.000
Triplet	0.400	0.008	0.000	–	–
Recompute	0.010	0.000	0.000	0.000	0.000

Table 31: SSSP baseline grid, `graph_acc_v`. These are the original 5-seed baseline runs; the 10-seed DeltaNAR graph result is reported separately.

Model	16	32	64	128	256
DeltaNAR	0.980	0.968	0.961	0.951	0.918
Prev+Delta	0.960	0.879	0.678	0.471	0.326
Recomp+gate	0.828	0.513	0.267	0.155	0.126
Triplet	0.934	0.776	0.490	–	–
Recompute	0.802	0.615	0.351	0.210	0.139

Table 32: Bottleneck baseline grid, `node_acc_v`.

Model	16	32	64	128	256
DeltaNAR	0.790	0.462	0.148	0.030	0.007
Prev+Delta	0.608	0.081	0.000	0.000	0.000
Recomp+gate	0.280	0.000	0.000	0.000	0.000
Triplet	0.423	0.003	0.000	–	–
Recompute	0.095	0.000	0.000	0.000	0.000

Table 33: Bottleneck baseline grid, `graph_acc_v`. Whole-graph exactness collapses quickly because node errors compound.

Task	Model	16	32	64	128
SSSP	DeltaNAR	0.988	0.995	0.997	0.998
SSSP	no-copy	0.978	0.888	0.628	0.375
SSSP	min-only	0.988	0.994	0.998	1.000
SSSP	Recompute	0.494	0.333	0.172	0.084
Bottleneck	DeltaNAR	0.980	0.968	0.961	0.951
Bottleneck	no-copy	0.966	0.810	0.453	0.242
Bottleneck	min-only	0.975	0.970	0.967	0.967
Bottleneck	Recompute	0.801	0.615	0.351	0.209

Table 34: Component ablation, `node_acc_v`.

Setting	Model	n	node	graph	affected F1
BA topology SSSP	DeltaNAR	64	0.993 ± 0.004	0.828 ± 0.042	0.624 ± 0.246
BA topology SSSP	Recompute	64	0.749 ± 0.062	0.005 ± 0.004	0.044 ± 0.009
Wide-weight SSSP	DeltaNAR	64	0.843 ± 0.088	0.164 ± 0.322	0.621 ± 0.140
Wide-weight SSSP	Recompute	64	0.118 ± 0.040	0.000 ± 0.000	0.056 ± 0.098
Failure SSSP	DeltaNAR	64	0.998 ± 0.001	0.944 ± 0.003	0.745 ± 0.343
Failure SSSP	Recompute	64	0.173 ± 0.015	0.000 ± 0.000	0.800 ± 0.000
Failure bottleneck	DeltaNAR	64	0.977 ± 0.006	0.144 ± 0.066	0.973 ± 0.021
Failure bottleneck	Recompute	64	0.350 ± 0.037	0.000 ± 0.000	0.062 ± 0.000
Extreme SSSP	DeltaNAR	512	0.932 ± 0.134	0.855 ± 0.176	0.874 ± 0.156
Extreme SSSP	Recompute	512	0.029 ± 0.001	0.000 ± 0.000	0.871 ± 0.250
Extreme bottleneck	DeltaNAR	1024	0.949 ± 0.037	0.009 ± 0.010	0.893 ± 0.108
Extreme bottleneck	Recompute	1024	0.106 ± 0.003	0.000 ± 0.000	0.004 ± 0.000

Table 35: Server B-line OOD and extreme-size stress tests.

Model	ER	BA	Grid
DeltaNAR	0.997	0.993	0.934
Triplet	–	0.960	0.691
Recompute	0.172	0.752	0.439

Table 36: SSSP topology OOD at $n = 64$, `node_acc_v`.

Model	Normal	Wide weights	Heavy tail
DeltaNAR	0.997 ± 0.002	0.843 ± 0.088	0.972 ± 0.028
Recompute	0.172 ± 0.015	0.118 ± 0.040	0.133 ± 0.002

Table 37: SSSP weight OOD at $n = 64$, `node_acc_v`.

Task / model	$n = 16$	$n = 64$
SSSP DeltaNAR	0.986	0.998
SSSP Recompute	0.507	0.173
Bottleneck DeltaNAR	0.982	0.977
Bottleneck Recompute	0.801	0.350

Table 38: Failure OOD, test edge deletion, `node_acc_v`.

Model	$k = 1$	$k = 2$	$k = 4$	$k = 8$
DeltaNAR	0.991	0.984	0.967	0.963
Prev+Delta	0.968	0.955	0.924	0.878
Recompute	0.513	0.506	0.499	0.545

Table 39: Dynamic-intensity OOD at $n = 16$, `node_acc_v`. The $k = 8$ entries use the server B07 rerun; smaller k entries are the earlier local sweep.

Task	Model	n	`node_acc_v`	`graph_acc_v`
SSSP	DeltaNAR	512	0.932 ± 0.134	0.855 ± 0.176
SSSP	Recompute	512	0.029 ± 0.001	0.000
Bottleneck	DeltaNAR	512	0.958 ± 0.014	0.014
Bottleneck	DeltaNAR	1024	0.949 ± 0.037	0.009
Bottleneck	Recompute	1024	0.106 ± 0.003	0.000

Table 40: Extreme-size results.

Model	$t = 0$	$t = 9$	$t = 59$
DeltaNAR	0.971 ± 0.026	0.729 ± 0.232	0.598 ± 0.307
Prev+Delta	0.843 ± 0.045	0.101 ± 0.012	0.097 ± 0.009

Table 41: Original five-seed 60-step rollout at $n = 64$, `node_acc_v`. C43 reports the calibrated copy-safe rerun separately.

Task	16	32	64	128
SSSP	0.978	0.986	0.993	0.997
Bottleneck	0.956	0.955	0.961	0.965

Table 42: One-processor generalist model, `node_acc_v`.

Seed	zero-shot $n = 64$	few-shot $n = 64$
0	0.413	0.991
1	0.580	0.993
2	0.444	0.990

Table 43: SSSP-to-bottleneck transfer, `node_acc_v`.

Model / metric	16	32	64
DeltaNAR edge	0.983	0.989	0.938
Prev+Delta edge	0.976	0.961	0.959
Recompute edge	0.923	0.960	0.962
DeltaNAR graph	0.278	0.125	0.000
Prev+Delta graph	0.244	0.034	0.000
Recompute graph	0.051	0.003	0.000

Table 44: MST difficulty gradient.

Interpretation by Experiment Group

Main Size-OOD Grid

The SSSP size-OOD grid is the central empirical result. The model is trained on graphs with only 16 nodes, but the DeltaNAR node-level metric stays essentially flat through 64 nodes and remains high at 256 nodes over 10 seeds. This is not a small extrapolation: the model sees 16-node training graphs and is evaluated at 16 times the number of nodes. The graph-level metric is more volatile, which is why the paper does not hide its variance. The 10-seed run gives graph accuracy 0.760 ± 0.335 at 256 nodes. This is compatible with the theory: whole-graph accuracy demands that every reachable witness be correct simultaneously.

The baseline pattern is as important as the DeltaNAR curve. Recompute collapses because it must solve the entire post-edit graph from scratch after training only on small graphs. Recomp+gate shows that affected supervision alone is not enough if the model has no previous solution to copy. Triplet and attention show that stronger from-scratch processors are not sufficient either. The empirical separation therefore supports the mechanism, not just a capacity story.

Bottleneck Generality

Bottleneck paths test whether the mechanism transfers from min-plus to max-min. The node accuracy is lower and more variable than SSSP at large size, but still far above baselines. This is useful evidence because the recurrence changes from additive shortest paths to a max-min witness. The copy mechanism itself does not depend on either semiring; it only requires that unchanged vertices remain safe to copy and changed vertices be locally recomputable.

The whole-graph bottleneck numbers are intentionally poor at large n. A node accuracy around 0.95 can imply near-zero whole-graph accuracy when multiplied over many reachable vertices. Reporting this as a limitation makes the claim cleaner: DeltaNAR provides node-level incremental generalization, while whole-graph exactness remains a harder target for bottleneck and MST.

Ablation Logic

The no-copy ablation is the clearest causal test. It preserves previous-solution input and the affected gate but removes the copy bias. Its collapse shows that merely providing memory is not sufficient; the decoder must be able to express "keep the old answer" as a strong local prior. Conversely, the min-only ablation does not collapse, meaning the main win is not simply the max aggregation branch.

Prev+Delta should be interpreted carefully. In the current configuration it uses previous solution and edit inputs but no gate and no copy. It can perform well in some ablation settings because previous-solution features are informative, but it does not solve the central copy-not-recompute problem in rollout or broad size-OOD baselines. The paper should define this baseline explicitly.

OOD Breadth

Topology OOD checks whether the update mechanism is tied to ER training graphs. DeltaNAR remains high on BA and grid graphs, with grid being harder for SSSP. Weight OOD checks whether the model memorizes the training weight range; wide weights are harder but DeltaNAR still beats Recompute by a large margin. Failure OOD tests edge deletion specifically, an important practical edit type. DeltaNAR stays above 0.98 on the reported path tasks.

Delta-type OOD is especially important for the mechanism. Training on weight changes and testing on structural add/delete edits still gives 0.998 to 0.998 in the A6 rerun. This suggests that the model has learned an update localization principle rather than a single edit-type heuristic.

Rollout

Rollout is a deliberately harder setting: the previous solution fed to the model is no longer guaranteed to be the oracle solution. DeltaNAR does not solve this perfectly. Some seeds drift slowly and remain high, while others degrade more substantially. This is consistent with the proof notes: copy-not-recompute is not a contraction unless wrong vertices are revisited and corrected. The important empirical comparison is that Prev+Delta collapses near 0.1 almost immediately, whereas DeltaNAR retains meaningful accuracy over long horizons.

Generalist and Transfer

The generalist experiment trains one processor across SSSP and bottleneck. It is a compact test of whether the mechanism can share computation across semirings. The model maintains high SSSP accuracy and high bottleneck accuracy simultaneously. The transfer experiment is complementary: SSSP-to-bottleneck zero-shot is uneven, but few-shot adaptation reaches about 0.99 at 64 nodes. The lesson is that the processor and copy mechanism transfer, but semiring-specific decoding still benefits from adaptation.

MST

MST is best framed as a diagnostic. Edge accuracy remains high, which means the model is learning useful local structure, but whole-tree exactness collapses. This does not undermine the path-task headline; it shows that different algorithmic outputs have different error amplification. A single wrong edge can invalidate a tree, just as a single wrong node can invalidate whole-graph path accuracy. MST is therefore evidence for a difficulty gradient and a useful next frontier.

Theory-Validation A-Suite

Experiment	Key measurement	Result
A1	ER p=0.3 mean $\lvert A_t \rvert$	0.52 at $n = 16$ to 0.01 at $n = 256$
A1	fixed degree $\lvert A_t \rvert / n$	1.9% at $n = 16$ to 0.1% at $n = 1024$
A2	gate recall	0.90
A4	DeltaNAR at 2 steps	0.989 ($n = 16$), 0.998 ($n = 64$)
A5	copy scale	saturates for scale ≥ 5
A6	delta-type OOD	0.998 in-dist to 0.998 add/delete

Table 45: A-suite summary from `results/a_suite.json`.

A1: Affected-Region Size

A1 measures the size of the affected region under local edits without involving the neural model. This directly probes the assumption used in the theorem: expected affected size should not grow linearly with graph size. Under the ER training distribution, mean affected size falls from 0.52 at 16 nodes to 0.01 at 256 nodes. Under a fixed average degree of about 6, affected fractions also shrink, reaching about 0.1% at 1024 nodes. The standard deviations are large relative to the means because most edits affect no or few vertices while a small number create larger cascades.

A2: Accuracy as a Function of Affected Size

A2 buckets examples by $|A_t|$ and measures node accuracy. The monotone trend is the important observation. Accuracy is 0.998 when the affected set is empty and decreases as more vertices require recomputation. This is exactly the qualitative shape predicted by the bound: the error term is proportional to the affected fraction multiplied by the chance that gate/recompute does not jointly succeed.

A4: Message-Passing Steps

A4 varies processor steps. DeltaNAR is already strong with two steps, obtaining 0.989 at $n = 16$ and 0.998 at $n = 64$. Recompute is more step-sensitive and does not show the same locality. This supports the dependency-depth interpretation: local updates often need only a small number of relaxations from correct copied boundary values.

A5: Copy Hyperparameters

A5 sweeps copy scale and margin. Copy scale must be large enough to make copying decisive; performance rises from 0.434 at scale 1 to 0.998 at scale 5 and saturates thereafter. Bottleneck copy margin changes node accuracy only mildly in the rerun. This suggests that the mechanism is not a fragile hyperparameter coincidence, though conservative copy remains useful for harder semirings.

A6: Delta-Type Invariance

A6 trains on weight-change edits and tests on structural add/delete edits. DeltaNAR remains at 0.998 in both settings, while Recompute stays around 0.17. This is one of the cleanest pieces of evidence that the model is not merely memorizing an edit type; it is using the previous solution and affected-region structure in a delta-type-agnostic way.

Failure Modes and Honest Caveats

Graph Accuracy Variance

Graph accuracy is a product-like metric: every evaluated node or edge must be right at once. This makes it extremely sensitive to small node-level errors. In the 10-seed SSSP run, node accuracy at 256 nodes is stable enough to support the main claim, but graph accuracy has high variance. The paper should continue to report graph accuracy as a secondary metric rather than hide it.

Steps Cap

The extreme-size SSSP result is reliable when message-passing steps scale with n. The steps=32 extreme-size run is unstable for SSSP and should be presented only as a diagnostic. Bottleneck used steps=32 for convenience and remains high at 1024 nodes in the reported run.

Rollout Drift

Rollout drift is real. The theory does not prove contraction, and the empirical curves are seed-dependent. This is not a contradiction of the one-step theorem; it is the expected behavior when previous predictions may contain errors and the copy mechanism preserves parts of the previous solution.

MST

MST whole-tree exactness remains hard. The edge-level results indicate useful local predictions, but a tree is a global combinatorial object, and small edge errors invalidate the whole structure. This should be framed as a future direction, not as a failure of the path-task mechanism.

Reproducibility

The paper and supplement use the official AAAI-27 author-kit style and bibliography files. Compilation uses `pdflatex`, `bibtex`, and two further `pdflatex` passes.

Primary Commands

The main size-OOD grid is produced by `experiments.py`. The canonical sanity command is:

```
python experiments.py --models DeltaNAR --tasks sssp --seeds 0
```

with the full runs adding all seeds, sizes, and model names. The result JSONs store both raw seed-level values and aggregated mean/std entries.

The A-suite command is:

```
python paper/scripts/run_a_suite.py.
```

This wrapper runs `exp_a1.py`, `exp_a2.py`, `exp_a4.py`, `exp_a5.py`, and `exp_a6.py`, then writes the parsed summary to `results/a_suite.json`.

Figure generation uses:

```
python paper/scripts/make_figures.py.
```

Result Artifacts

The most important result artifacts are:

- `SEED14_sssp.json` and `SEED14_bot.json`: 10-seed DeltaNAR main grids.
- `A_sssp.json`, `A_bot.json`, and triplet companions: 5-seed baseline grids.
- `ABL7.json`: component ablations.
- `BASE9_attn.json`: attention baseline.
- `OOD10_failure.json`: edge-deletion failure OOD.
- `TRANS11.json`: SSSP-to-bottleneck transfer.
- `E_sssp_512.json`, `EXT12_bot.json`: extreme-size runs.
- `a_suite.json`: local theory-validation rerun.
- `A_local_diagnostics_*.json`: local hard-region diagnostics for changed-output, copy-critical, pure-copy, and gate metrics.
- `A_oracle_diagnostics_sssp_seed0.json`: oracle-mask diagnostic variants.

Paper-Build Commands

The main paper is built by running `pdflatex`, then `bibtex`, then two more `pdflatex` passes. The supplement is built analogously from `supplement.tex`. The generated figures are PDF files under `paper/figures/`.

Repository Manifest

The repository is organized around a small set of core files.

The tests directory contains differential checks for the ground-truth engines. These are not cosmetic: because CLRS-Delta introduces dynamic tasks, the paper depends on the claim that incremental oracles match from-scratch verification.

Result Artifact Index

For completeness, we list the result artifacts used by the paper package.

The D-line logs are parsed into `D_structured.json` by `analyze_d_results.py`. The paper figures should be regenerated from these structured artifacts rather than edited manually.

File	Role
src/graphs.py	graph generation and edit sampling
src/sssp.py	Dijkstra and widest-path oracles
src/dynamic_sssp.py	LPA*-style dynamic SSSP engine
src/mst.py	Kruskal/Prim MST oracles
src/dataset.py	dynamic SSSP dataset builder
src/dataset_bottleneck.py	bottleneck dataset builder
src/dataset_mst.py	MST dataset builder
src/model.py	DeltaNAR model, losses, metrics
experiments.py	multi-seed grid runner
diagnose_a_metrics.py	hard-region diagnostic metric runner
diagnose_a_oracle.py	oracle-mask diagnostic runner
src/train.py	SSSP task runner and rollout options
src/train_generalist.py	shared-processor experiment
src/transfer.py	SSSP-to-bottleneck transfer
RESULTS.md	full result record
THEORY_FINAL.md	cleaned proof package
REPRODUCIBILITY.md	artifact reproduction guide

Table 46: High-level code map.

Artifact	Contents
SEED14_sssp.json	10-seed SSSP DeltaNAR main grid
SEED14_bot.json	10-seed bottleneck DeltaNAR main grid
A_sssp.json	5-seed SSSP baseline grid
A_bot.json	5-seed bottleneck baseline grid
A_sssp_triplet.json	SSSP triplet baseline
A_bot_triplet.json	bottleneck triplet baseline
ABL7.json	copy, no-copy, min-only, recompute ablations
BASE9_attn.json	attention baseline
B_sssp_ba/grid.json	topology OOD for SSSP
B_bot_ba/grid.json	topology OOD for bottleneck
C_sssp_whi100.json	wide-weight OOD
C_sssp_heavy.json	heavy-tail OOD
C_weight_ood_summary.json	structured C-line weight OOD summary
OOD10_failure.json	edge-deletion failure OOD
TRANS11.json	SSSP-to-bottleneck transfer
D_structured.json	structured D-line rollout/generalist/intensity/MST summary
E_sssp_512.json	SSSP extreme-size run
EXT12_bot.json	bottleneck 512/1024 extreme-size run
a_suite.json	rerun theory-validation suite
A_local_diagnostics_sssp_3seed.json	SSSP diagnostic slices
A_local_diagnostics_sssp_k8_3seed.json	multi-edit SSSP diagnostics
A_local_diagnostics_bottleneck_3seed.json	bottleneck diagnostics
A_oracle_diagnostics_sssp_seed0.json	oracle-mask copy variants
revision/conditional_hard_*.json	conditional invalid-copy slices
revision/dear_delta_matched_eval_*.json	matched-distribution DEAR evaluation
revision/stability_sssp_*.json	independent SSSP stability batches
revision/day23_merged_bootstrap_10000.json	paired bootstrap statistics

Table 47: Machine-readable result sources.

References

Georgiev, D.; Wilson, J. J.; Buffelli, D.; and Liò, P. 2024. Deep Equilibrium Algorithmic Reasoning. In *Advances in Neural Information Processing Systems*, volume 37, 33638–33667.

Koenig, S.; and Likhachev, M. 2002. D* Lite. In *Proceedings of the Eighteenth National Conference on Artificial Intelligence*, 476–483. AAAI Press.

Koenig, S.; Likhachev, M.; and Furcy, D. 2004. Lifelong Planning A*. *Artificial Intelligence*, 155(1–2): 93–146.

Ramalingam, G.; and Reps, T. 1996. An Incremental Algorithm for a Generalization of the Shortest-Path Problem. *Journal of Algorithms*, 21(2): 267–305.

Veličković, P.; Badia, A. P.; Budden, D.; Pascanu, R.; Banino, A.; Dashevskiy, M.; Hadsell, R.; and Blundell, C. 2022. The CLRS Algorithmic Reasoning Benchmark. In *Proceedings of the 39th International Conference on Machine Learning*, volume 162 of *Proceedings of Machine Learning Research*, 22084–22102. PMLR.

Veličković, P.; Ying, R.; Padovano, M.; Hadsell, R.; and Blundell, C. 2020. Neural Execution of Graph Algorithms. In *International Conference on Learning Representations*.

Xu, K.; Li, J.; Zhang, M.; Du, S. S.; Kawarabayashi, K.-i.; and Jegelka, S. 2020. What Can Neural Networks Reason About? In *International Conference on Learning Representations*.