Cost Accounting for Reactive Computational Graphs: Exhaustive Sweeps, Sequential Mutation, and the Backward-Locality Gap

Abdallah Khemais
ISITCOM, University of Sousse

July 2026

Abstract

Exhaustive site-by-site interventions on a neural network's computational graph — activation-patching sweeps, circuit-discovery searches, systematic ablation studies — mutate the graph at every candidate site in turn, and their cost is dominated by recomputation after each mutation. On a reactive graph engine whose invalidation provably touches exactly the downstream cone of a mutated node, we give a complete cost accounting for such workloads. First, the aggregate speedup of an exhaustive sweep over independent full recomputations is not a universal constant: if per-layer computational weight varies regularly with depth with Karamata index q, the ratio converges to $(q+2)/(q+1)$ when weight is concentrated near the output and to $q+2$ when concentrated near the input, recovering 2 only in the depth-uniform case; a wall-clock corollary computed from measured interpreter constants predicts a ceiling of ≈ 1.79, strictly below 2, until interpreter overhead is compiled away. Second, we prove the exact cost of a *sequence* of persistent mutations, never undone between insertions: the interleaved cost exceeds the sum of isolated costs by an exact overcount $\Delta(\pi) \geq 0$ summed over comparable site pairs, with closed-form extremes over insertion orders, while *batched* application is order-independent and sub-additive, costing exactly the union of the sites' downstream cones plus the fresh nodes. Third, we prove the exact mirror of forward locality for the backward pass and show it forces the aggregate sweep speedup to collapse to 1 under backpropagation on architectures without long skip connections — delimiting precisely the regime (inference-time sweeps) where the speedup applies undiminished. Every identity is validated on the reference implementation in *NeuroDSL* [3], a reactive define-and-run graph engine in Julia: measured sweep ratios on real graphs converge to the predicted limits under four non-uniform cost profiles (E4); the training-mode ratio collapses to 1 at the predicted rate (E5); and all 18 per-graft sequential costs and the batched total match the closed forms at zero tolerance across three insertion orders (E7).

1 Introduction

The workhorse of mechanistic interpretability is the *sweep*: patch or ablate each candidate site of a trained network in turn, measure the effect on a metric, restore the baseline, and move to the next site. Circuit-discovery searches, causal-tracing protocols, and systematic robustness checks all share this shape, and their cost is dominated not by the mutation itself — overwriting one activation or one rule is cheap — but by the *recomputation* each mutation forces before the metric can be read, multiplied by the number of candidate sites, which grows with model size.

How much of that recomputation is actually necessary depends on the execution model. Eager frameworks such as PyTorch [1] re-run the full forward pass per intervention; compiled pipelines (`torch.compile`, `jax.jit` [2]) may additionally re-trace when the intervention changes the traced program. Neither maintains a notion of *partial validity*: the framework cannot distinguish the subgraph whose cached values are still correct from the subgraph invalidated by the intervention.

A *reactive* graph engine can. *NeuroDSL* [3] keeps the computational graph as a persistent DAG in which nodes own their cached values and a mutation triggers an invalidation wave provably confined to the mutated node's downstream cone — a single-mutation locality theorem restated as Theorem 1 below, and proved in a companion study of exact network surgery on the same engine [4].

This paper asks the question that theorem leaves open: what does an entire *workload* of mutations cost? We answer it exactly, in three regimes.

Contributions.

1. **Aggregate sweep cost.** For an exhaustive sweep that patches every site and restores each before the next, the speedup over independent full recomputations converges to $(q+2)/(q+1)$ or $q+2$ in the Karamata index q of the network's cost-by-depth profile — 2 only in the depth-uniform case (Theorem 2). A wall-clock corollary, computed from interpreter constants measured on the reference engine, predicts a ceiling of ≈ 1.79, strictly below the combinatorial limit, until interpreter overhead is compiled away (Corollary 1).

2. **Sequential and batched mutation cost.** For a sequence of *persistent* grafts (never undone, as a growth schedule or a cumulative multi-site intervention performs), the interleaved cost is exactly the isolated sum plus an overcount $\Delta(\pi) \geq 0$ summed over comparable site pairs, with closed-form extremes over insertion orders (Theorem 3, Corollary 2); batched application is order-independent and sub-additive, costing exactly the union of the cones plus the fresh nodes (Proposition 1).

3. **The backward-locality gap.** The exact mirror of the locality theorem holds for the backward pass (Theorem 4), and it forces the aggregate sweep speedup to collapse to 1 under backpropagation on architectures without long skip connections (Corollary 3) — a boundary of the aggregate result, not a retraction of it: inference-time sweeps, the case that motivates this paper, are unaffected.

4. **Empirical validation at zero tolerance where the claims are exact** (Section 7): measured sweep ratios on real reactive graphs converge to the predicted limits under four cost profiles and both orientations (E4); the training-mode ratio collapses to 1 on a graph with genuine intra-layer width (E5); and all 18 per-graft sequential costs, both order extremes, and the order-independent batched total match the closed forms exactly (E7), alongside a measured negative result on bookkeeping drift with mutation count.

2 Related Work

Dynamic graph engines. PyTorch [1] is define-by-run; JAX [2] traces pure functions. Neither maintains a persistent reactive graph across steps: validity of cached computation is not a first-class notion, so an intervention pays a full forward pass (or a re-trace) regardless of how little of the graph it actually affects. NeuroDSL's design [3] is closer to incremental computation systems, applied to differentiable programs; this paper quantifies exactly what that buys, and does not buy, for sweep-shaped workloads.

Interventional sweeps. Activation patching, causal tracing, and automated circuit discovery all iterate a patch–measure–restore loop over candidate sites; their published cost analyses count forward passes, implicitly assuming each intervention costs a full recomputation. The accounting here replaces that assumption with exact combinatorial identities for engines that recompute only what a mutation invalidates.

Exact surgery. The companion study of exact network surgery on the same engine [4] proves the single-mutation locality theorem (restated as Theorem 1) together with functional-exactness guarantees for grafted residual blocks; the present paper takes the single-mutation statement as its starting point and derives the workload-level asymptotics.

3 Preliminaries

Definition 1 (Computational graph)**.** *A computational graph is a tuple* $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathrm{op}, \theta)$ *where* $(\mathcal{V}, \mathcal{E})$ *is a finite DAG,* $\mathrm{op}(v)$ *assigns to each non-input node an operator, and* $\theta(v)$ *its (possibly empty) parameter set. Distinguished subsets* $\mathcal{V}_{\mathrm{in}} \subset \mathcal{V}$ *(sources) and a node* v_{out} *(output) induce, by composition along the topological order, a function* $F_{\mathcal{G}} : \mathcal{X} \to \mathcal{Y}$*.*

Definition 2 (Grafting)**.** *Let* $e = (u, w) \in \mathcal{E}$ *be an edge of* $\mathcal{G}$ *and* $\mathcal{H}$ *a computational graph with a single input and a single output, realizing a function* $F_{\mathcal{H}}$*. The grafting* $\mathcal{S}(\mathcal{G}, e, \mathcal{H})$ *produces* $\mathcal{G}'$ *by removing* e *and adding edges* $(u, \mathrm{in}(\mathcal{H}))$ *and* $(\mathrm{out}(\mathcal{H}), w)$*, i.e., the value flowing along* e *now passes through* $\mathcal{H}$*.*

Defining insertion on an *edge* (rather than "at a node") removes any ambiguity about which consumers of u are rerouted: exactly those along e. Two mutation primitives cover every workload in this paper: a *graft* adds $h = |\mathcal{V}(\mathcal{H})|$ fresh computable nodes at a site, and a *patch* redefines the rule of an existing node in place (e.g., replacing an activation by a stored or corrupted value) and is later *restored* by redefining the rule back — two mutations at the same site, each triggering the same invalidation.

In a reactive engine, every node carries a validity flag; a mutation at a node s invalidates its dependents, and the next demand-driven evaluation (`demand!`) recomputes exactly the invalid region. We formalize the invalidation procedure as Algorithm 1 and restate the locality theorem this paper's accounting is built on; the NeuroDSL routine `_invalidate_downstream!` implements this algorithm (implementation conformance is an engineering claim, verified by the test suite, not part of the proof).

Algorithm 1 INVALIDATE$(\mathcal{G}, s)$ — reactive invalidation from a seed node s

```
Q ← [s];   seen ← {s}
while Q not empty do
    v ← pop(Q)
    for each w with (v, w) ∈ E do
        if w ∉ seen then
            valid(w) ← false;   seen ← seen ∪ {w};   push(Q, w)
        end if
    end for
end while
```

Definition 3 (Downstream cone)**.** *For* $s \in \mathcal{V}$*, let*

$$\mathcal{V}_s^+ = \{\, v \in \mathcal{V} \mid \textit{there is a path of length} \geq 1 \textit{ from } s \textit{ to } v \,\}.$$

By convention $s \notin \mathcal{V}_s^+$ *unless* s *lies on a cycle, which the DAG property excludes; when the mutation is a graft, the seed* $s = \mathrm{out}(\mathcal{H})$ *is a freshly created node, which is born unvalued and needs no invalidation.*

Theorem 1 (Structural locality)**.** *On a DAG, Algorithm 1 terminates and the set of nodes it marks invalid is exactly* $\mathcal{V}_s^+$*. Its complexity is* $O(|\mathcal{V}_s^+| + |\mathcal{E}_s^+|)$*, where* $\mathcal{E}_s^+$ *is the set of edges internal to the cone.*

Proof. Termination. Each node enters seen at most once and is pushed at most once; $\mathcal{V}$ is finite.

Soundness (marked $\Rightarrow$ in cone). We show by induction on the order of marking that every node marked invalid lies in $\mathcal{V}_s^+$. The first nodes marked are the immediate successors of s, which are in $\mathcal{V}_s^+$ via paths of length 1. Inductively, a node w is marked only when popped from the queue along an edge (v, w) with v previously marked or $v = s$; by the induction hypothesis there is a path $s \rightsquigarrow v$ (possibly empty if $v = s$), which extended by (v, w) gives a path of length ≥ 1 from s to w. Hence $w \in \mathcal{V}_s^+$.

Completeness (in cone $\Rightarrow$ marked). Let $w \in \mathcal{V}_s^+$ and let $s = v_0 \to v_1 \to \cdots \to v_k = w$, $k \geq 1$, be a witnessing path. By induction on i: $v_0 = s \in$ seen and is processed; if v_i is processed, then when it is popped, its successor v_{i+1} is either already in seen (hence was marked and pushed earlier) or is marked and pushed now. Either way v_{i+1} is marked and eventually processed. Thus $v_k = w$ is marked.

Preservation. A node $v \notin \mathcal{V}_s^+$ is never marked by soundness, so valid(v) is untouched: its cached value survives the mutation.

Complexity. Each node in the cone is popped once and each internal edge scanned once. □

Theorem 1 bounds the cost of a *single* mutation by its downstream cone. Everything that follows is the accounting for workloads built out of many such mutations.

4 Aggregate Cost of an Exhaustive Sweep

A caller that tests every candidate site exhaustively – an ablation sweep, a circuit-discovery search, or a systematic robustness check – pays the sum of the per-site cone costs across every site, and it is natural to ask how this aggregate compares to the cost of testing each site by an independent full recomputation. The answer depends on exactly one number: how computational weight is distributed across depth.

Definition 4 (Layered cost model)**.** *A* layered DAG *of depth L assigns to each layer $j \in \{1, \ldots, L\}$ a computational weight $w_j > 0$ (e.g. its node count) and hosts S candidate sites per layer, each site's downstream cone coinciding, up to an $O(1)$ per-site remainder that vanishes in the ratios below, with the suffix $\sum_{j>i} w_j$ of the layer i it belongs to. Write $N(L) = \sum_{j=1}^{L} w_j$ for the total weight and*

$$\rho(L) \;=\; \frac{SL \cdot N(L)}{S \sum_{i=1}^{L} \sum_{j>i} w_j}$$

for the ratio of SL independent full recomputations to the aggregate cost of exhaustively patching every site, each restored via Theorem 1's exact cone.

Theorem 2 (Aggregate sweep ratio)**.** *Suppose $w_j \sim \ell(j)\, j^q$ as $j \to \infty$ for some $q \geq 0$ and some ℓ slowly varying at infinity (Karamata: $\ell(cx)/\ell(x) \to 1$ for every $c > 0$). Then*

$$\rho(L) \;\longrightarrow\; \frac{q+2}{q+1} \qquad (L \to \infty).$$

If instead the weight profile is reversed, $w_j \sim \ell(L{+}1{-}j)\,(L{+}1{-}j)^q$ (layers near the input *carry the heavier weight), the limit is*

$$\rho(L) \;\longrightarrow\; q + 2.$$

In particular $q = 0$ (uniform per-layer weight, $w_j \equiv M$) gives $\rho(L) \to 2$ under either profile.

Proof. Write $S(n) = \sum_{j=1}^{n} w_j$; by Karamata's theorem for regularly varying sequences, $S(n) \sim n\, w_n/(q+1)$. For the output-heavy profile, at $i = tL$ for fixed $t \in (0, 1)$, $w_i \sim \ell(L)(tL)^q$ and

$$\sum_{j>i} w_j \;=\; N(L) - S(i) \;\sim\; \frac{L\, w_L}{q+1}\big(1 - t^{q+1}\big).$$

Riemann-summing over $i = 1, \ldots, L$ (i.e., over $t \in (0, 1)$),

$$\sum_{i=1}^{L} \sum_{j>i} w_j \;\sim\; L \cdot \frac{L\, w_L}{q+1} \int_0^1 (1 - t^{q+1})\, dt \;=\; \frac{L^2 w_L}{q+1} \cdot \frac{q+1}{q+2} \;=\; \frac{L^2 w_L}{q+2},$$

and since $N(L) = S(L) \sim L w_L/(q+1)$, substitution gives $\rho(L) \sim [L \cdot L w_L/(q+1)]/[L^2 w_L/(q+2)] = (q+2)/(q+1)$. The reversed profile follows by the index substitution $j \mapsto L+1-j$, $i \mapsto L-i$, which turns the same computation into $\sum_{j>i} w_j \mapsto S(L-i) \sim (L-i) w_{L-i}/(q+1)$ against the same $N(L)$, giving the mirror-image Riemann integral $\int_0^1 t^{q+1} dt \cdot (q+1) = (q+1)/(q+2)$ of the denominator's leading coefficient and hence $\rho(L) \to q+2$. □

Remark 1 (Why 2 is not universal)**.** *The ratio* 2 *is the mean of a linear ramp: a site drawn uniformly over depth has, in expectation, half the total weight downstream of it, and in general* $\rho = 1/\mathbb{E}[U]$ *for* U *the (normalized) downstream-weight fraction of a uniformly drawn site. Theorem 2 shows* 2 *is the* $q = 0$ *instance of a one-parameter family: back-loading cost toward the output (*$q > 0$*, output-heavy)* lowers *the limit toward* 1*, while front-loading it toward the input* raises *the limit without bound as* q *grows. The aggregate ratio is a direct, computable readout of* where *computation is concentrated in the network, not a universal constant of reactive invalidation.*

Corollary 1 (Wall-clock refinement)**.** *Under the uniform profile (*$q = 0$*), suppose each recomputation additionally pays a fixed overhead proportional to total graph size,* $\beta N(L)$ *– the* $O(|\mathcal{V}|)$ *topological-order walk of the demand-driven evaluator, measured on the reference implementation at* $a = 0.042$ *ms per cone node with an intercept of* 2.24 *ms on a* 412*-node model, i.e.* $\beta \approx 0.0054$ *ms/node (Section 7) – on top of the* $a\,|\mathcal{V}_i^+|$ *cost Theorem 1 guarantees. Then the wall-clock aggregate ratio converges to*

$$\rho_{\text{wallclock}}(L) \;\longrightarrow\; \frac{a+\beta}{a/2+\beta} \;\approx\; 1.79,$$

strictly below the unit-cost limit of 2*, since* β *is paid once per site regardless of cone size and contributes equally to numerator and denominator, while the* a*-term alone retains the* $1/2$ *averaging of Theorem 2.*

Proof. With $\text{cost}(i) = a\,M(L-i) + \beta ML$ per site and $K = SL$ sites, the numerator (naive) is $K(a+\beta)ML$ and the denominator is $a \cdot SM\frac{L(L-1)}{2} + K\beta ML \sim SML^2(a/2+\beta)$ to leading order; the ratio of leading terms is $(a+\beta)/(a/2+\beta)$. □

Remark 2. *This is a testable refinement, not a restatement: it predicts that any exhaustive, depth-uniform sweep on this engine converges to an aggregate speedup* strictly below 2 *in wall-clock time –* ≈ 1.79 *at the constants above – until the interpreter overhead* β *is eliminated by a compilation layer (Section 8), at which point the pure combinatorial limit of Theorem 2 is recovered.*

5 Sequential Mutation Cost: Interleaved and Batched Grafting

Theorem 1 bounds a *single* graft; Section 4 bounds an exhaustive sweep of patches that are each undone before the next is tried. A growth schedule – and equally a *cumulative* multi-site intervention that leaves each patch in place – does neither: it applies a *sequence* of grafts to the same graph, none of them undone, each possibly landing inside a cone already disturbed by an earlier one. This section asks what that sequence costs, in two regimes – *interleaved* (a full `demand!` between grafts, as a loop that logs a metric after each mutation would do) and *batched* (all grafts applied before the next `demand!`, exactly the cost profile of a multi-site patch set applied at once) – and shows both are governed exactly by Theorem 1 applied recursively, with no new axiom required.

Lemma 1 (Cone trace invariance under grafting)**.** *Let $\mathcal{G}' = \mathcal{S}(\mathcal{G}, e, \mathcal{H})$ with $e = (u, w)$ and $\mathcal{H}$ built from fresh nodes, connected from* $\mathrm{in}(\mathcal{H})$ *to* $\mathrm{out}(\mathcal{H})$ *and wired into $\mathcal{G}'$ only via $(u, \mathrm{in}(\mathcal{H}))$ and $(\mathrm{out}(\mathcal{H}), w)$. Then for any node $f \in \mathcal{V}(\mathcal{G}) \setminus \{u\}$: (i) $\mathcal{V}_f^+(\mathcal{G}') \cap \mathcal{V}(\mathcal{G}) = \mathcal{V}_f^+(\mathcal{G})$, i.e. reachability among original nodes is unchanged; (ii) the h computable nodes of $\mathcal{H}$ belong to $\mathcal{V}_f^+(\mathcal{G}')$ iff $u \in \mathcal{V}_f^+(\mathcal{G}) \cup \{f\}$, i.e. iff f's downstream cone in $\mathcal{G}$ already contained u (equivalently: e lies downstream of f).*

Proof. Any path between original nodes that used e in $\mathcal{G}$ rewrites in $\mathcal{G}'$ as $u \to \mathrm{in}(\mathcal{H}) \rightsquigarrow \mathrm{out}(\mathcal{H}) \to w$, and conversely any path in $\mathcal{G}'$ between original nodes that enters $\mathcal{H}$ must do so at $\mathrm{in}(\mathcal{H})$ (its only in-edge from an original node) and leave at $\mathrm{out}(\mathcal{H})$ (its only out-edge to one), since $\mathcal{H}$'s internal wiring never touches original nodes; replacing that segment with e recovers a path in $\mathcal{G}$. This proves (i). For (ii), a node of $\mathcal{H}$ is reachable from f in $\mathcal{G}'$ iff some path from f reaches u (then continues $u \to \mathrm{in}(\mathcal{H}) \rightsquigarrow \cdot$), which happens iff $u \in \mathcal{V}_f^+(\mathcal{G}) \cup \{f\}$. □

Theorem 3 (Exact interleaved cost)**.** *Let $u_1, \ldots, u_K$ be distinct nodes of $\mathcal{G}_0$ (no two related by ancestry through each other's own graft, i.e. each u_k survives as a node through every graft not performed at u_k itself), grafted in the order π (a permutation of $1, \ldots, K$) with block sizes $h_1, \ldots, h_K$, each graft followed by a full* `demand!` *before the next is applied. The recomputation cost of the k-th graft (π-order) is exactly*

$$\mathrm{cost}_{\pi(k)} \;=\; \left|\mathcal{V}^+_{u_{\pi(k)}}(\mathcal{G}_0)\right| \;+\; h_{\pi(k)} \;+\; \sum_{\substack{j<k \\ u_{\pi(j)} \in \mathcal{V}^+_{u_{\pi(k)}}(\mathcal{G}_0)}} h_{\pi(j)},$$

and therefore $C_{\mathrm{seq}}(\pi) = C_{\mathrm{iso}} + \Delta(\pi)$ where $C_{\mathrm{iso}} = \sum_k \left(|\mathcal{V}^+_{u_k}(\mathcal{G}_0)| + h_k\right)$ is the cost of K grafts on independent copies of $\mathcal{G}_0$, and

$$\Delta(\pi) \;=\; \sum_{\substack{(j,k)\,:\,\pi \text{ applies } j \text{ before } k \\ u_j \in \mathcal{V}^+_{u_k}(\mathcal{G}_0)}} h_j \;\;\geq\; 0,$$

with equality iff π never grafts a site while a strictly upstream site's graft is still outstanding.

Proof. By induction on k. At the first graft, $\mathcal{G}_0$'s own cone $\mathcal{V}^+_{u_{\pi(1)}}(\mathcal{G}_0)$ is recomputed by Theorem 1, plus the $h_{\pi(1)}$ fresh nodes of $\mathcal{H}_{\pi(1)}$ (unvalued, hence "recomputed" trivially): this is the $k = 1$ case with an empty sum. Assume the formula holds through graft $k-1$, giving a fully valid $\mathcal{G}_{k-1}$. Applying Lemma 1 $k-1$ times (once per prior graft, each preserving reachability among nodes not part of that graft's own block) shows $\mathcal{V}^+_{u_{\pi(k)}}(\mathcal{G}_{k-1})$ consists of exactly the original nodes of $\mathcal{V}^+_{u_{\pi(k)}}(\mathcal{G}_0)$, together with the fresh block of every prior graft $j < k$ whose site $u_{\pi(j)}$ lies in $\mathcal{V}^+_{u_{\pi(k)}}(\mathcal{G}_0)$ (Lemma, part (ii), applied with $f = u_{\pi(k)}$). Theorem 1 recomputes exactly this cone, plus the $h_{\pi(k)}$ fresh nodes of the current graft. Summing over k and separating the $j = k$ diagonal terms from the cross terms gives $C_{\mathrm{seq}}(\pi) = C_{\mathrm{iso}} + \Delta(\pi)$; each term of Δ is a size $h_j \geq 0$, so $\Delta(\pi) \geq 0$, with equality iff no pair (j, k) with j before k has u_j downstream of u_k – i.e. π is upstream-first. □

Corollary 2 (Order extremes, uniform blocks)**.** *If the K sites are totally ordered by depth (a sequential architecture) and $h_k \equiv h$, then over all $K!$ orders: $\Delta_{\min} = 0$ (shallowest-first), $\Delta_{\max} = h\binom{K}{2}$ (deepest-first), and $\mathbb{E}_\pi[\Delta(\pi)] = h\binom{K}{2}/2$ under a uniformly random order (each of the $\binom{K}{2}$ depth-ordered pairs contributes h independently with probability $1/2$, by linearity of expectation over the event "the deeper site of the pair is applied first").*

Proposition 1 (Batched grafting: sub-additive and order-independent)**.** *If all K grafts are applied to $\mathcal{G}_0$ before any* `demand!`*, the total number of nodes requiring recomputation at the next*

`demand!` *is exactly*

$$C_{\text{batch}} \;=\; \Big|\bigcup_{k=1}^{K} \mathcal{V}^{+}_{u_k}(\mathcal{G}_0)\Big| \;+\; \sum_{k=1}^{K} h_k,$$

independent of the order in which the K grafts were applied, and $C_{\text{batch}} \le C_{\text{iso}}$, strictly whenever some pair of sites is comparable ($u_j \in \mathcal{V}^{+}_{u_k}(\mathcal{G}_0)$ for some $j \neq k$).

Proof. Order-independence: redefining an existing node's rule re-invalidates its cone regardless of what was invalid before (`addrule!`'s contract, exercised by every graft in this paper), and re-invalidating an already-invalid node is idempotent, so the final valid/invalid partition after K unread grafts depends only on the *set* of rules redefined, not the order of redefinition. A node needs recomputation iff it is a fresh node of some block ($\sum_k h_k$ of these, always invalid), or an original node lying in $\mathcal{V}^{+}_{u_j}(\mathcal{G}_0)$ for some j – exactly the union $\bigcup_k \mathcal{V}^{+}_{u_k}(\mathcal{G}_0)$, by definition of the downstream cone (Section 3). No correction term for the sites u_k themselves is needed: by the same definition, $u_k \notin \mathcal{V}^{+}_{u_k}(\mathcal{G}_0)$ for every k, and $u_k \in \mathcal{V}^{+}_{u_j}(\mathcal{G}_0)$ for some $j \neq k$ exactly when u_k genuinely is an ordinary downstream node of another graft's site and is therefore correctly recomputed; a site that is not downstream of any other selected site simply never appears in the union at all, with no subtraction required to remove it. Sub-additivity: $\big|\bigcup_k \mathcal{V}^{+}_{u_k}(\mathcal{G}_0)\big| \le \sum_k |\mathcal{V}^{+}_{u_k}(\mathcal{G}_0)|$ with equality iff the cones are pairwise disjoint, i.e. no two sites are comparable; hence $C_{\text{batch}} \le C_{\text{iso}}$. □

Remark 3 (Composing exactness across a schedule)**.** *The companion surgery study [4] proves an identity-morphism theorem for a single function-preserving graft on an arbitrary graph; nothing in its hypotheses refers to the graph's history. It therefore applies verbatim to every intermediate graph $\mathcal{G}_{k-1}$ in a sequence, and functional exactness composes by induction across an entire growth schedule – the guarantee a multi-graft schedule implicitly relies on at every step, made explicit here rather than re-derived per instance.*

Remark 4 (What is proved and what is only measured)**.** *Theorem 3 and Proposition 1 are exact combinatorial identities – verified below at zero tolerance, not by convergence. They govern* recomputation *cost only. A separate, non-combinatorial question – whether the* bookkeeping *cost of a graft (rewiring consumers, rebuilding the consumers/topological-order caches) drifts with the number of prior mutations at fixed graph size, beyond the size-dependence already captured by the wall-clock Corollary 1's β term – has no theorem attached and is reported as a measurement only (E7 below).*

6 Locality Under Backpropagation: Where the Speedup Evaporates

Theorem 1 and its aggregate consequence (Theorem 2) concern *forward* recomputation: after a mutation at s, only $\mathcal{V}^{+}_{s}$ must be re-evaluated. Training also propagates a backward pass, and the chain rule ties the gradient of every ancestor of s to the local computation performed at s. This section states the exact mirror of Theorem 1 for this upstream dependency and draws an honest corollary: on the sequential architectures this paper's cost model targets, the mirror set is almost the whole graph, and the aggregate speedup of Theorem 2 does not transfer to an exhaustive sweep performed under backpropagation.

Definition 5 (Upstream cone)**.** *For $s \in \mathcal{V}$, let $\mathcal{V}^{-}_{s} = \{v \in \mathcal{V} \mid$ there is a path of length ≥ 1 from v to $s\}$ – the mirror image of the downstream cone $\mathcal{V}^{+}_{s}$ of Section 3, obtained by reversing every edge.*

Theorem 4 (Backward locality)**.** *Let ℓ be a distinguished loss node with $s \in \mathcal{V}^{-}_{\ell}$ (i.e., s contributes to the loss). Mirroring Algorithm 1 on the graph with every edge reversed, seeded at s,*

terminates and marks exactly $\mathcal{V}_s^-$ (termination, soundness, and completeness are the proof of Theorem 1 verbatim, with every edge (u, w) read as (w, u)). This set is exactly the collection of nodes whose backward pass depends, via the chain rule, on the (possibly mutated) local Jacobian at s: for $v \notin \mathcal{V}_s^- \cup \{s\}$, $\partial\ell/\partial v$ is computed entirely from paths that never traverse s and is therefore unaffected by a mutation there.

Proof. Termination, soundness, and completeness of the reversed traversal follow Theorem 1's proof with the edge direction flipped. For the gradient claim, write $\partial\ell/\partial v = \sum_{\pi:\, v \rightsquigarrow \ell} \prod_{(a,b)\in\pi} \partial b/\partial a$, a sum over directed paths π from v to ℓ; a term is affected by s's local computation iff s lies on π. If $v \in \mathcal{V}_s^-$, some witnessing path $v \rightsquigarrow s$ extends through s to ℓ (since $s \in \mathcal{V}_\ell^-$ gives $s \rightsquigarrow \ell$), so at least one term of the sum passes through s. If $v \notin \mathcal{V}_s^- \cup \{s\}$, no path from v reaches s, so no term can. □

Corollary 3 (Training-mode locality collapse)**.** *In the layered cost model of Definition 4, suppose every node of layer $j < i$ is an ancestor of every node of layer i (resp. $j > i$ a descendant) – true whenever the architecture has no skip connection bypassing an entire layer, as in a stack of Transformer blocks composed purely sequentially. Then a mutation at a site s of local cost c_k in layer i touches, under backpropagation,*

$$\left|\mathcal{V}_s^- \cup \{s\} \cup \mathcal{V}_s^+\right| \;=\; N(L) - w_i + c_k,$$

the whole graph minus the untouched remainder of s's own layer, and the aggregate ratio of an exhaustive sweep performed under backpropagation *satisfies*

$$\rho_{\text{train}}(L) \;=\; \frac{L\,N(L)}{\sum_{i=1}^{L}\big(N(L) - w_i + \bar{c}\big)} \;\longrightarrow\; 1 \qquad (L \to \infty),$$

regardless of the cost profile (w_j) or its Karamata index – in sharp contrast to Theorem 2.

Proof. $\sum_{i=1}^{L}(N(L) - w_i + \bar{c}) = LN(L) - N(L) + L\bar{c}$, so $\rho_{\text{train}}(L) = LN(L)/(LN(L) - N(L) + L\bar{c}) = L/(L - 1 + L\bar{c}/N(L)) \to 1$ as $L \to \infty$, since $\bar{c}/N(L) \to 0$ for any cost profile with $N(L) \to \infty$. □

Remark 5. *This is not a retraction of Theorems 1 and 2: every measurement in this paper (E4, E5, E7) and the exhaustive patching sweeps that motivate it are performed at inference, with no backward pass through the swept sites – exactly where Theorem 2 applies undiminished. Corollary 3 instead delimits the claim precisely: reactive locality is a property of* value *propagation, not of* gradient *propagation, on architectures without long skip connections. Growing a network mid-training – the subject of the companion surgery study [4] – is unaffected in practice, since a growth schedule grafts one site at a time rather than exhaustively sweeping candidates under backpropagation.*

7 Experimental Results

All measurements use the reference implementation (NeuroDSL, Julia, Float32, CPU). Where a claim is an exact combinatorial identity (E7), the comparison is at zero tolerance on integer invalid counts; where it is an asymptotic limit (E4, E5), convergence across increasing L is reported. The interpreter constants used by Corollary 1 ($a = 0.042$ ms per cone node, intercept 2.24 ms on a 412-node model, hence $\beta \approx 0.0054$ ms/node) were measured in the companion surgery study's cost-versus-depth experiment [4] on this same engine and are reused here unchanged.

7.1 E4: The Aggregate Sweep Ratio Under Non-Uniform Cost Profiles (Theorem 2)

Theorem 2 is a statement about an abstract layered-cost model; we verify it on a real graph in the reference implementation, not a combinatorial simulation. For each $(L, q, \text{profile})$, a sequential graph of L layers is built where layer j contributes w_j nodes (a chain of `:relu` unary ops) with w_j set, up to rounding, proportional to j^q (output-heavy) or $(L-j+1)^q$ (input-heavy) and normalized so total graph size is ≈ 4000 nodes regardless of q; the graph is evaluated once (so every node is `valid`), and $\rho(L)$ is computed from the *true* downstream cone of each layer's last node, measured by `_downstream_nodes` – the same routine `sweep_patch_sites!` uses in the patching-sweep tooling built on this engine, not a re-derivation for this test.

Table 1: E4 — measured $\rho(L)$ at $L = 320$ nodes-per-cone-profile, vs. Theorem 2's prediction.

q	Predicted (output-heavy)	Measured, L=320	Measured (input-heavy) vs. predicted q+2
0.0	2.0000	2.0057	2.0057 vs. 2.0
0.5	1.6667	1.6722	2.5057 vs. 2.5
1.0	1.5000	1.5064	3.0004 vs. 3.0
2.0	1.3333	1.3471	3.9247 vs. 4.0

Every entry converges toward its predicted limit as L grows across the tested range $L \in \{20, 40, 80, 160, 320\}$ (not shown in full: the q=1, input-heavy case alone tightens from 3.1585 at L=20 to 3.0004 at L=320), confirming both the asymptotic derivation and that the abstract layered-cost model of Definition 4 is not merely a convenient fiction: a real reactive graph's measured cones obey it. The uniform case ($q = 0$) recovers 2 from either profile, as it must, and is the special case already exploited for exhaustive activation-patching sweeps built on this same engine.

7.2 E5: Training-Mode Collapse (Corollary 3)

A layered graph with genuine intra-layer width is built exactly as in E4, except each layer's w_j branches are computed in *parallel* from the previous layer (no branch depends on another) and then folded pairwise into a single aggregate before the next layer – so a candidate site's siblings within its own layer are neither its ancestors nor its descendants, the general case Corollary 3 addresses (unlike a purely sequential single-path graph, where $c_k = w_i$ and the corollary degenerates to the exact identity $\rho_{\text{train}} \equiv 1$ at every finite L). Measuring $\rho_{\text{train}}(L) = \big|\text{sites}\big| \cdot N(L) / \sum_s |\mathcal{V}_s^- \cup \{s\} \cup \mathcal{V}_s^+|$ directly (ancestors by a plain reverse walk over each node's recorded inputs, descendants by `_downstream_nodes`) gives, independently of q and of the output-/input-heavy profile: 1.0526 (L=10), 1.0256 (20), 1.0126 (40), 1.0062 (80), 1.0031 (160) – converging to 1 exactly as Corollary 3 predicts, at a rate insensitive to the cost profile that gave Theorem 2 its rich q-dependence under forward-only recomputation.

7.3 E7: Sequential Mutation Cost (Theorem 3, Proposition 1)

On a 400-node synthetic chain (built and `demand!`-ed exactly as in E4), $K = 6$ sites are selected at roughly even depth and $H = 20$ fresh nodes are grafted at each. **Interleaved regime**: for three orders – shallowest-first, deepest-first, and a fixed random permutation – a full `demand!` is issued before each graft, and the number of nodes marked invalid immediately after each graft is compared to Theorem 3's closed-form prediction, computed once from $\mathcal{G}_0$'s cones before any graft. **Batched regime**: the same K grafts are applied back-to-back with no intermediate `demand!`, for the same three orders, and the total invalid count is compared to Proposition 1.

Every one of the 18 per-graft measurements across the three orders matches Theorem 3's formula exactly, and $\Delta(\pi)$ realizes both extremes of Corollary 2 precisely (0 and $H\binom{K}{2} = 300$).

Table 2: E7 — per-graft recomputation cost, interleaved regime, three orders (K=6, H=20). Every entry matches Theorem 3 exactly (zero tolerance).

Order	**Measured (per graft)**	$\Delta(\pi)$ vs. Theorem
Shallowest-first	$363, 306, 249, 191, 134, 77$	0 (predicted: 0)
Deepest-first	$77, 154, 231, 309, 386, 463$	300 (predicted: $H\binom{K}{2} = 300$)
Random (1 seed)	$249, 191, 403, 77, 154, 386$	140

In the batched regime, all three orders give the identical total 463, matching Proposition 1's order-independent closed form exactly: the union of the six sites' original cones is 343 (the shallowest site's cone alone, since the other five sites – and their downstream nodes – all lie within it), plus $KH = 120$ fresh nodes, giving $343 + 120 = 463$ with no correction term, confirming $C_{\text{batch}} = 463 < C_{\text{iso}} = 1320$, the predicted strict sub-additivity. Two bugs were found and fixed before this number was accepted: an initial version counted leaf nodes (e.g. the graph's input, which has no rule and whose `valid` flag is never touched by `demand!`), producing a constant off-by-one on every interleaved measurement; a second version additionally subtracted one node per selected site from the batched union on the mistaken assumption that every site's own symbol needed excluding, producing a further off-by-five – resolved by recognizing that Proposition 1's union already excludes exactly the sites that are not downstream of any other selected site, by the downstream-cone definition itself, with no separate correction needed.

Bookkeeping drift with mutation count (measurement, not theorem). 100 single-node grafts are applied successively to a fresh 400-node chain (final size 500), timing only the rewiring step (excluding `demand!`); a control arm applies 100 rule-redefinitions *without* adding a node (graph size fixed at 401 throughout), isolating any effect of mutation *count* from the already-established effect of graph *size* (Corollary 1's β term). Trimmed-median (10%) cost rises modestly from 0.033 ms (first half, size $\approx$ 401–451) to 0.036 ms (second half, size $\approx$ 451–501) for the growing chain, consistent with the small size increase; the constant-size control shows no such drift (0.0243 ms in both halves), and grafting is costlier than rule-redefinition alone (0.035 ms vs. 0.024 ms trimmed-median overall) as expected, since it does strictly more work. No history-dependent drift beyond graph size is detected at this scale (100 mutations, single run) – a negative result reported as such, not a proof of its absence.

8 Limitations

The aggregate results concern *recomputation counts*; wall-clock enters only through the two measured constants of Corollary 1, which are specific to the current interpreter (CPU, Float32) — a compilation layer would change β, and with it the wall-clock ceiling, though not the combinatorial limits. The layered cost model of Definition 4 assumes each site's downstream cone coincides with a depth suffix up to a vanishing remainder; architectures with long skip connections violate both this and the ancestry hypothesis of Corollary 3, so their sweep ratios are not covered by the closed forms here (the exact per-site formula of Theorem 3 still applies, since it makes no layering assumption). E7's bookkeeping-drift measurement is a single run of 100 mutations at one graph-size trajectory; absence of detected drift there is evidence at that scale, not a bound valid at the scale of a real multi-thousand-mutation schedule. Finally, Corollary 3 delimits but does not quantify the intermediate regime of architectures with *some* long skips, where the sweep ratio under backpropagation lies strictly between 1 and the forward-only limit; characterizing that interpolation is open.

9 Conclusion

On a reactive graph engine whose invalidation is provably confined to the downstream cone of a mutation, the cost of an entire sweep-shaped workload is a computable function of where computation is concentrated in the network. The aggregate speedup of an exhaustive site-by-site sweep is $(q+2)/(q+1)$ or $q+2$ in the Karamata index of the cost-by-depth profile — 2 only in the depth-uniform case — with a wall-clock ceiling of ≈ 1.79 at the current interpreter's measured constants; a sequence of persistent mutations costs exactly the isolated sum plus a closed-form order penalty, minimized by upstream-first application and eliminated entirely (along with order dependence) by batching; and under backpropagation the aggregate speedup collapses to 1 on architectures without long skip connections, confining the benefit to inference-time sweeps — precisely the regime of the activation-patching and circuit-discovery workloads that motivate this accounting. Every exact identity is validated at zero tolerance, and every asymptotic limit by measured convergence, on the reference implementation.

References

[1] Paszke, A., et al. (2019). PyTorch: An Imperative Style, High-Performance Deep Learning Library. *NeurIPS*.

[2] Bradbury, J., et al. (2018). JAX: Composable Transformations of Python+NumPy Programs. `http://github.com/google/jax`.

[3] Khemais, A. (2026). NeuroDSL: A Reactive Computational Graph Framework for Deep Learning in Julia. *Preprint*.

[4] Khemais, A. (2026). Exact Network Surgery: Functional Invariance and Gradient Plasticity in Reactive Computational Graphs. *Preprint, submitted concurrently*.