FAST AND MEMORY-EFFICIENT LIFELONG MODEL EDITING WITH CERTIFIED EVALUATION REUSE

Anonymous authors
Paper under double-blind review

ABSTRACT

Model editing writes new knowledge into a frozen large language model without retraining. In lifelong editing, edits arrive continually and must be re-verified against the accumulated history. A single edit in a memory-based editor changes one active value row, yet the standard execution path recomputes edit-invariant prefixes and retrieval at every optimization step, allocates training state for the whole table, and re-tests the full history at every checkpoint. We propose eHoReN, which removes this mismatch in editors that retrieve before they inject and write one row, with a single principle: the execution scope of every computation and every piece of storage should equal its dependency scope on mutable state. Each technique carries an equivalence proposition fixing when the original behavior is preserved. On the editor that holds the quality frontier at scale, eHoReN edits up to 3.5x faster, with outputs identical to the baseline on Llama-3.1-8B and quality within seed variability on Qwen2.5-7B. Its training state shrinks from 4.58 GiB to 48 KiB and stops growing with the memory. The same dependency records make reuse of historical evaluation results checkable, and a cost model predicts when checking pays off. Certified reuse speeds up evaluation by 1.89x per checkpoint without lag, and periodic re-certification reaches 1.6–2.4x over exact checking with exact agreement at every anchor. Transferred to a second editor with different retrieval and injection mechanisms, the editing speedup reproduces and the evaluation gain follows the benefit condition's prediction. Code is available here.

1 INTRODUCTION

A deployed language model answers from knowledge frozen when its pre-training ended. Facts change and errors surface, and retraining as a maintenance strategy is expensive and slow. Model editing instead writes target knowledge through localized updates, and its quality is measured by Reliability, Generalization, and Locality, the standard triple (Yao et al., 2023; Wang et al., 2024a). Requests arrive continually in deployment, and in this lifelong setting (Hartvigsen et al., 2023) each new edit must take effect alongside all earlier ones, with their coexistence re-verified at every checkpoint for as long as the deployment lasts.

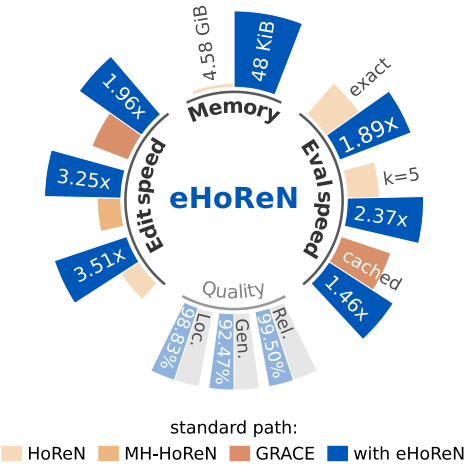


Figure 1: eHoReN against the standard path of each editor, with quality unchanged and the gains appearing in edit speed, training-state memory, and evaluation speed.

Existing methods divide by whether an edit changes the backbone. Parameter-modifying editors write each fact into the weights, altering a local part of the model per edit (Fang et al., 2025). Parameter-preserving editors leave the weights untouched and accumulate knowledge in an external memory, whose form may be a routed side model (Wang et al., 2024a), a discrete codebook (Hartvigsen et al., 2023), or a Hopfield memory with its multi-head extension MH-HoReN (Fang et al., 2026; Che et al., 2026); each edit adds or updates one memory entry.

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

Comparison across these families has centered on quality. In the lifelong setting, efficiency decides whether a system can keep running at all. Write cost grows linearly in the number of edits and verification cost quadratically. Verification has not been an optimization target, and as edits get faster it becomes the bottleneck sooner. Underneath both lies one mismatch. A single edit changes one active value row, yet generic execution recomputes the whole backbone, updates the whole key–value memory, and re-tests the whole edit history at every checkpoint.

We propose eHoReN, which removes this mismatch by scheduling execution from dependency. Nothing is computed or stored at a wider scope than the state it depends on. The principle yields one group of techniques at each of three timescales: (i) within an edit, computations independent of the active value row move out of the optimization loop; (ii) across edits, training state shrinks from $O(nd_v)$ to $O(d_v)$ and no longer depends on memory size; (iii) across checkpoints, verifying the effective-injection sequence suffices to reuse a historical output. Each technique comes with a proposition fixing exact equivalence, so a violated condition names where it must not apply; Figure 1 previews the gains in edit time, training state, and evaluation time.

Dependency-recorded reuse is established for general computation (Acar et al., 2013); to our knowledge, this work is the first to bring it to lifelong model evaluation as exact certified reuse. Checking at a checkpoint certifies the results it covers, while re-certifying periodically trades a bounded number of differences between anchors for speed. Because the derivation rests on three structural properties rather than on Hopfield retrieval, it carries to a codebook-memory editor, where the editing speedup reproduces and the evaluation gain follows the benefit condition.

2 Related Work

Parameter-modifying and -preserving editors. Parameter-modifying methods rewrite weights by rank-one or batched updates (Meng et al., 2022; 2023), with null-space projection against interference (Fang et al., 2025), while hypernetworks predict the update (De Cao et al., 2021; Mitchell et al., 2022a; Wang et al., 2026) or simulate the batched ideal shift (Guo et al., 2025); all are assessed per edit, none against re-verification. Parameter-preserving methods differ in routing and in what a write touches: SERAC and WISE (Mitchell et al., 2022b; Wang et al., 2024a) gate into jointly updated side weights, GRACE (Hartvigsen et al., 2023) matches a codebook key within a radius and writes one value, HoReN (Fang et al., 2026; Che et al., 2026) retrieves by Hopfield attention and injects a residual, and prompt or LoRA memories freeze one entry per edit (Wang et al., 2025; Luo et al., 2026), unlike trained-in memories (Wang et al., 2024b). They compete on retention over half a million updates (Thede et al., 2025), yet the memory's execution cost goes unmeasured, and a parallel line lowers cost by changing the algorithm itself (Gu et al., 2025; Li et al., 2024; Wang & Li, 2024; Li et al., 2025; Atri et al., 2025), coupling efficiency to behavior. eHoReN changes no algorithm and shrinks execution scope under equivalence propositions, preserving behavior under their premises, exactly on fixed-step Llama runs and within seed variability on Qwen; jointly trained routing (Chen et al., 2024) or multi-row writes fall outside them.

Computation reuse and sparse training state. Outside editing, frozen-layer activation caching exploits invariance across passes (Liu et al., 2021), fused loss kernels avoid materializing logits (Wijmans et al., 2025), and row-wise optimizers update sparsely but store full-table moments (PyTorch Contributors, 2024), all within one run. Incremental computation reuses across runs by recording dependencies on mutable state (Acar et al., 2013). Lifelong evaluation has not adopted this view: benchmarks re-test fully at intervals or subsample a fixed set (Thede et al., 2025; Jang et al., 2022), the former quadratic, the latter uncertifiable, and Sort & Search (Prabhu et al., 2024) reuses evaluations across models approximately, by ranking test samples; eHoReN brings exact, certified reuse of one model's history to editing.

3 Setting and Method

3.1 Lifelong Sequential Editing and Its Two Costs

Given a frozen base model f_θ and a stream of edit requests $E = \{e_1, \ldots, e_N\}$ arriving in order, the t-th request rewrites the model's answer to a subject–relation query. All editing information lives in an external key–value memory, whose contents after e_t we denote S_t. A checkpoint every p edits

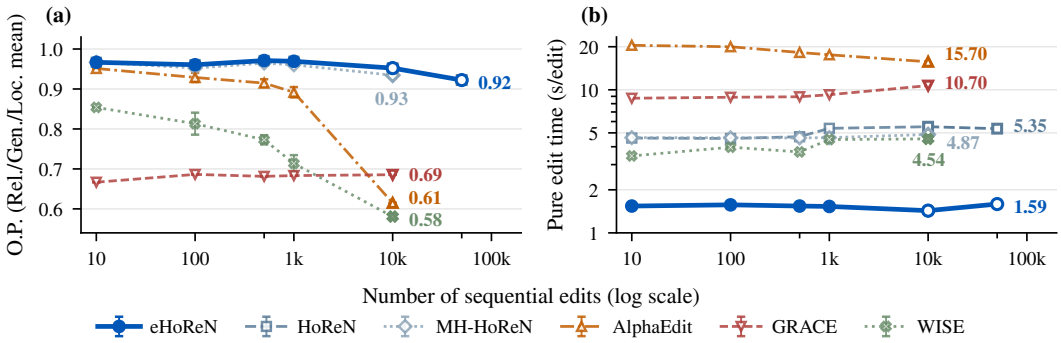


Figure 2: Quality–cost frontier on Llama-3.1-8B/ZsRE. **(a)** O.P. and **(b)** pure edit time per edit, against edit count. Error bars in (a) are three-seed standard deviations at $N \leq 1000$, and open markers are single-seed points ($N \geq 10^4$, seed 42), including the 50K runs, which the baselines did not run. HoReN coincides with eHoReN in (a). Values in Table 1, per-metric spreads in Table 7.

re-evaluates all edits then in effect, each contributing its original question, rephrasings, and locality questions, scored by Reliability, Generalization, and Locality, whose mean is O.P. Existing lifelong benchmarks report quality at the end of the stream or at a few points, which samples the history; we re-verify all of it at every checkpoint, because a later edit can silently change an earlier answer and only re-testing detects it. An edit retrieves a node by its key and injects its value. Its optimization loop therefore updates one active value row while everything before the injection layer stays frozen.

Two costs follow. Writing runs an optimization loop for every request, and its cost accumulates linearly in N. Verification re-evaluates the accumulated history at every checkpoint, and under the protocol above its cumulative query count grows as $\Theta(N^2/p)$. This paper reduces both at once, without changing the editor's behavior or the completeness of verification.

3.2 Execution Scope Exceeds Dependency Scope

Both costs have the same origin. The only mutable state of an edit is the active value row. Generic execution nevertheless repeats the frozen prefix forward, the retrieval, and the full-vocabulary projection at every optimization step, and it maintains gradients and Adam state for the entire table. Both quantities scale with input length and table size, and a one-row update touches neither. Figure 2 shows what the mismatch costs and what removing it gains, with per-edit time dropping by a constant factor while the O.P. curve stays unchanged, and only the Hopfield-memory family holding quality at 10K edits, where the nearest other family sits 26.7 points lower.

On the verification side, the correctness of a historical output depends only on the effective injections during its generation, yet the protocol regenerates every historical query at every checkpoint. The waste is measurable. At the 1K checkpoint half of HoReN's 3,000 queries are carried over and all but twelve are unchanged, while on GRACE all 900 certifiable carried-over queries stay valid (Appendix H.6). The checkpoint schedule sets the size of that share; whether it can be reclaimed depends only on the cost of checking. A certified dependency check reclaims that share when checking is cheaper than generation. The criterion has to be exact, because a write that touches no old parameter can still change an old query's answer and subsampling dilutes such changes rather than locating them.

Removing it requires no new editing algorithm, only shrinking execution scope to dependency scope, so that an edit allocates computation and training state for the active value row alone and a checkpoint verifies only whether each output's effective-injection sequence changed.

Stated as a principle, the execution scope of every computation and every piece of storage should equal its dependency scope on mutable state. Section 3.6 places the three technique groups in one cost model with the benefit condition for evaluation reuse. Figure 3 shows the three timescales; pseudocode and symbols are in Appendices E and A.

Editor abstraction. The derivation treats the editor as an object with three structural properties that HoReN and GRACE both instantiate. Retrieval precedes injection, so frozen computation produces the retrieval input $q = \psi_\ell(z)$; selection is discrete and injection gated over keys K, values

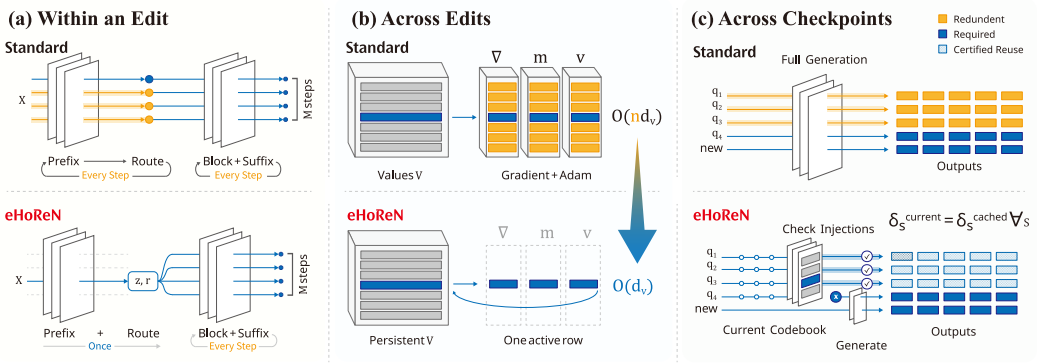


Figure 3: Execution matched to dependency at three timescales. Upper rows: the standard path; lower rows: eHoReN. Orange marks computation the standard path repeats, blue what any path must do, light blue certified reuse. Panels (a)–(c) correspond to Props. 1–3.

V, and versions ν; and writes touch one node. HoReN instantiates retrieval by Hopfield iteration, building on modern Hopfield networks (Ramsauer et al., 2021),

$$i^* = \arg\max_i \langle q^{(H)}, k_i \rangle, \qquad g = \mathbf{1}\left[\max_i \langle q^{(H)}, k_i \rangle > \tau\right], \qquad \widetilde{h}_\ell = h_\ell + g\, v_{i^*}, \tag{1}$$

where h_ℓ is the output of the wrapped projection layer ℓ.

3.3 Editing-side Computation Reuse

During an edit the input, the target, and the backbone stay fixed while the active value row is updated repeatedly, which yields three within-edit invariants. (a) Frozen prefix. The cache boundary sits at the entry of the injection-layer decoder block. The representation $z = F_{<\ell}(x)$ does not depend on the active value row, so it is computed once during preparation and every optimization step replays only that block and the network above it. (b) Within-edit routing. The retrieval input $q = \psi_\ell(z)$ precedes injection, and once node insertion is complete and keys are fixed, the hit node i^* and gate g no longer change with value updates. Caching them as the fixed route of this edit reduces retrieval from once per step to a constant number of preparation-time calls. (c) Supervised-position projection. Let $u_s(v)$ be the final hidden state at position s when the active value row holds v, let $\mathcal{A}$ ($|\mathcal{A}| = A$) be the supervised set, and let $\mathcal{L}(v)$ be the mean cross-entropy over $\mathcal{A}$ (Appendix C). The vocabulary projection is position-wise independent once final hidden states are formed, so selecting supervised positions before projecting saves the $O((T-A)\, d_v|\mathcal{V}|)$ computation spent at unsupervised positions, while the network above still processes all T tokens.

Proposition 1 (Editing computation reuse)**.** *Fix the input, positional encoding, and masks; freeze the prefix; assume deterministic computation and an identical supervised set. Then (i) reusing the block-entry representation z and (ii) projecting only on $\mathcal{A}$ preserve $\mathcal{L}(v)$ and the gradient $\nabla_v \mathcal{L}$ for every v. If additionally there is a single injection position and the retrieval input does not depend on v, then caching the final route (i^*, g) after keys are fixed preserves the same conclusion.*

Route caching needs the proposition's two further premises, a single injection position and a v-independent retrieval input, because a second injection position or a v-dependent retrieval input would let the route move within the edit (proof in Appendix C).

3.4 Active-row Training

The current route selects a single node i^*, and the loss propagates only through row i^* of V. The full-table path nevertheless builds a dense gradient for the whole $n \times d_v$ table, and Adam maintains two moments of the same shape. eHoReN copies v_{i^*} into an independent temporary parameter, freezes the persistent table, creates Adam for that parameter only, and on completion writes back to row i^* and advances the node's version number. The two kinds of storage now have distinct roles. The persistent table accumulates knowledge and grows with the memory, while gradients and moments serve only the current edit and are sized by the active row (Appendix B).

Proposition 2 (Active-row update equivalence)**.** *Suppose both paths start from the same post-preparation state, the hit node i^* and gate g are fixed within the edit, Adam's moments and step count are re-initialized at the start of every edit, and there is no weight decay and no cross-row coupling. Then under the same number of optimization steps and hyperparameters, the active-row path and the full-table path produce the same final v_{i^*}, and all other rows remain unchanged.*

Gradients of inactive rows are identically zero, and re-initializing the moments at every edit keeps their Adam state at zero, so the two paths execute the same update on row i^* (proof in Appendix C).

3.5 Route-aware Evaluation

Within-edit reuse rests on computation that does not depend on mutable state. Across checkpoints the question is which historical outputs still arise from the same effective modifications after the memory changes. A single write can change the value an old query hits or redirect it to a newly added node, so the dependency record must cover which value was used and why it was selected.

Route-aware evaluation associates each cached output with the sequence of effective injections from its generation. The query cache stores output tokens, scores, and a per-step trace (gate, hit-node identity, and value version at each step); write the effective injection at step s as $\delta_s = g_s v_{i_s}$. A new checkpoint checks, along the old generation prefix, whether the current memory produces the same effective injections as cached; queries that pass reuse their old output and score directly, and the rest are regenerated together with newly added queries.

Proposition 3 (Historical output reuse)**.** *Assume a frozen backbone, identical tokenizer and prompts, identical injection positions and computation configuration, and deterministic decoding and stopping rules. If, under the current memory state, the effective injection at every step s on the old prefix $(x, y_{<s})$ equals that of the old execution, then the complete generation under the current state coincides with the cached output, including length.*

Induction over generation steps: an unchanged injection sequence reproduces each step's input and logits, so the whole output is reproduced (proof in Appendix C). The condition is sufficient, not necessary, and the archived records bound the slack: none of the twelve ZsRE invalidations and 49–50 of the 150 WikiBigEdit ones regenerate a result whose output had not changed, so a necessary-and-sufficient criterion could reuse at most 3.3% more of the candidates.

The criterion admits two implementations, whose cost ratio R/G enters the benefit condition of Section 3.6. Full replay recomputes every old prefix and trusts no cached intermediate. Cached-query checking stores the pre-injection retrieval input q_s, which is invariant for a fixed old prefix, and re-runs acceptance on the current memory without any backbone forward, at $O(nd_k)$ per step.

3.6 Cost Model, Benefit Condition, and Transferability

Editing-side cost. The cost of a single edit with M fixed steps is

$$\begin{aligned} C_{\text{full}} &\approx C_0 + M\big[P(T) + S_{\text{net}}(T) + S_{\text{head}}(T) + \mathcal{R}(n) + \mathcal{D}(n)\big], \\ C_{\text{eHoReN}} &\approx C_0' + \underbrace{P(T)}_{M\to 1} + \underbrace{\kappa\,\mathcal{R}(n)}_{M\to\kappa} + M\big[S_{\text{net}}(T) + \underbrace{S_{\text{head}}(A)}_{T\to A} + \underbrace{\mathcal{D}(1)}_{n\to 1}\big] + O(d_v). \end{aligned} \tag{2}$$

Orange marks the terms the standard path repeats and blue those any path must do, as in Figure 3. Appendix B defines the symbols and Table 4 compares the two paths term by term. Reuse is not free, since preparing the route and the cached representation adds a fixed per-edit overhead whose share grows as M shrinks. Accumulated over the stream, the optimizer term drops from $O(Md_v \sum_t n_t)$ to $O(MNd_v)$, while cumulative retrieval remains $O(Hd_k \sum_t n_t)$. The dense sums are quadratic whenever the memory grows with the stream, which holds for these streams to within five percent, so the dense path is $O(MN^2 d_v)$ here (Appendix H.7).

Evaluation benefit condition. Let a checkpoint contain Q queries, of which U are new and C historical queries fail the check; let G be the mean cost of full generation with scoring, R the mean check cost, and B the net maintenance cost. Comparing $T_{\text{full}} \approx QG$ with $T_{\text{route}} \approx (Q-U)R + (U+C)G + B$, reuse yields a positive time gain iff

$$(Q-U)\,R + B \;<\; (Q-U-C)\,G. \tag{3}$$

Table 1: Quality and cost across editing scales: O.P. (%) and pure edit time (s/edit) at each scale; bold best, underline second best; a descriptive operating-point view, the paired arms of Table 2 carry the efficiency claims. fail: run aborted before final evaluation. Cell conventions and per-metric values: Table 7, Appendix H.1.

Model	**Method**	$N = 10$		$N = 100$		$N = 500$		$N = 1$K		$N = 10$K	
		O.P.	Time	O.P.	Time	O.P.	Time	O.P.	Time	O.P.	Time
Llama-3.1 8B / ZsRE	AlphaEdit	95.13	20.47	92.88	20.00	91.47	18.24	89.18	17.57	61.49	15.70
	WISE	85.39	3.45	81.32	3.97	77.29	3.68	71.37	4.49	58.08	4.54
	GRACE	66.67	8.73	68.63	8.87	68.14	8.94	68.33	9.24	68.55	10.70
	HoReN	**96.67**	4.60	**96.10**	4.57	**97.12**	4.70	**96.93**	5.38	95.16	5.52
	MH-HoReN	**96.67**	4.63	95.23	4.62	96.46	4.61	96.05	4.63	93.44	4.87
	eHoReN	**96.67**	**1.54**	96.06	**1.57**	97.11	**1.54**	**96.93**	**1.53**	**95.22**	**1.43**
Qwen2.5 7B / WikiBigEdit	AlphaEdit	**84.89**	32.77	**88.72**	33.12	**82.37**	31.52	**77.30**	31.01	fail	
	WISE	52.44	11.58	56.75	12.33	47.60	12.10	25.54	10.12	fail	
	GRACE	66.67	8.81	67.06	8.87	67.15	8.98	67.16	9.01	fail	
	HoReN	68.17	4.56	75.63	4.62	72.83	4.69	71.97	4.63	67.70	5.71
	MH-HoReN	68.17	4.61	77.44	4.63	76.04	5.21	74.61	4.64	**69.58**	4.91
	eHoReN	68.17	**1.90**	75.63	**2.00**	72.70	**1.96**	71.96	**2.02**	67.68	**2.40**

The gain separates into two independent factors, check efficiency (R/G) and the workload's reuse opportunity ($u = U/Q$, $c = C/Q$, reused share $\rho = 1 - u - c$); Appendix B gives the ratio in closed form. Certified reuse is capped by $1/u$; skipping the check can exceed that cap, at the price of replacing the guarantee with measurable error, which gives a fidelity–speed spectrum. With $R/G \approx 1$, HoReN's full replay gains certification itself, zero-miss reuse without running an oracle. Cached-query checking pushes R/G far below 1 and converts the same criterion into acceleration.

Transferability. The derivation uses only the three structural properties and no detail of Hopfield retrieval, so it applies to any editor that has them: Appendix H.6 checks the premises for GRACE, whose acceptance depends only on retrieval inputs, keys and radii, and for MH-HoReN, whose heads split only the retrieval scoring and still select a single row, so the premises hold unchanged.

4 Experiments

Models, data, and baselines. Llama-3.1-8B-Instruct (L8) is the primary backbone and Qwen2.5-7B-Instruct (Q7) a cross-architecture control. The main editing stream is ZsRE; a second panel uses WikiBigEdit. Baselines are AlphaEdit, WISE, GRACE, and HoReN with its multi-head variant MH-HoReN, one per family. eHoReN is applied to HoReN, and the same optimizations are ported to GRACE for the transfer study. Quality is Reliability, Generalization, Locality, and their mean O.P., measured at every checkpoint over all edits then in effect.

Protocol. Within a comparison both arms share data order, stopping rule, hyperparameters, precision, and hardware, so a ratio isolates execution; seed conventions, the two timing bases that are never compared, table conventions, and a map from each headline ratio to its basis, setting, and denominator (Table 5) are in Appendix F.

4.1 Quality and Cost across Editing Scales

The families separate by how quality moves with scale rather than by their level at small N, with parameter-modifying falling to 61.49% at 10K, side-memory decaying throughout, and codebook-memory staying flat but low, its radius protecting stored entries at a cost in Generalization (Table 7). The Hopfield family stays above 93% at every scale, which is why it is the base editor here.

On the cost axis eHoReN is the frontier point at every scale, and its advantage widens with the stream. Its per-edit time stays within 1.43–1.57 s and does not grow over the measured range, while HoReN's rises by 20% from N=10 to 10K as preparation and retrieval grow with the memory. At N=1K eHoReN scores 7.75 points above AlphaEdit in 11.5x less time per edit, and at 10K the gap widens to 33.73 points in 11.0x less time. Against its own unoptimized path the largest O.P. difference at any scale is 0.06 points, inside the three-seed spread, and MH-HoReN, the nearest configuration outside the pair, ties it at N=10 and trails by 0.65 to 1.78 points thereafter.

The second panel repeats the pattern under a different backbone and a harder distribution. At every scale eHoReN tracks HoReN's quality to within 0.13 points in less than half the per-edit time, and at 10^4 edits the pair agrees to 0.01 O.P. points, 67.70 and 67.68%, at 5.71 and 2.40 s/edit, a 2.38x ratio, with Locality carrying the decay as in the ZsRE stream. The multi-head variant is the strongest Hopfield configuration here, 77.44 at $N{=}100$ in 4.63 s/edit and 69.58 at 10^4 in 4.91 s/edit, so the pattern is not specific to ZsRE; MH-eHoReN carries the same reuse over to it at 3.25x (Section 4.5).

4.2 Paired Speedup and Its Attribution

Both arms of Table 2 share model, data order, stopping rule, and hyperparameters, so the ratio is attributable to execution alone: 3.51 ± 0.13x on L8 and 2.77 ± 0.34x on Q7, every quality difference inside one three-seed standard deviation. On L8 the fixed-step arms agree at the level of output hashes; on Q7 that identity is not established (Appendix D). The cost model predicts the gap between backbones, since boundary depth, supervision ratio, and vocabulary size set how much of the per-step cost is reusable (Appendix B) and all three differ across the two backbones.

The techniques contribute in different currencies, isolated arm by arm in Figure 6 of Appendix H.2. Frozen-prefix reuse carries the speedup on its own, cutting per-edit time from 5.10 to 1.84 s (2.77x) in the isolation runs, and the route cache and preparation cleanup add three to six percent on top (Appendix H.2). Supervised-position projection returns no wall-time gain at this supervision length, since its value is the scale term (Appendix B), and active-row training pays in space. Step counts are identical across ablation configurations at 50.00 ± 0.00, so no ratio comes from optimizing less, and pairing invariance holds at five further scales on Q7/WikiBigEdit (Appendix H.1).

Table 2: Paired editing results; within each pair the arms differ only in execution (Section 4.2). Per-edit time from the harness's internal timer, the same timer in both arms; three seeds; quality in %; O.P. and memory columns in Appendices H.2 and H.6, and a tuned GRACE pair in Table 23.

Backbone	Method	Rel.	Gen.	Loc.	API Time/Edit (s)	Speedup
L8/ZsRE	HoReN (Hopfield)	99.49±0.09	92.47±0.16	98.83±0.00	5.38±0.02	1.00x
	eHoReN	99.50±0.11	92.47±0.16	98.83±0.00	1.53±0.06	**3.51±0.13x**
Q7/ZsRE	HoReN (Hopfield)	99.56±0.26	92.89±0.23	98.16±0.00	5.14±0.32	1.00x
	eHoReN	99.37±0.29	92.62±0.21	98.16±0.00	1.89±0.31	**2.77±0.34x**
L8/ZsRE	GRACE (codebook)	100.00	5.01	100.00	4.98±0.04	1.00x
	GRACE + ours	100.00	5.01	100.00	2.54±0.04	**1.96±0.05x**
Q7/ZsRE	GRACE (codebook)	100.00	1.92	100.00	5.19±0.16	1.00x
	GRACE + ours	100.00	1.92	100.00	3.13±0.38	**1.68±0.22x**

4.3 Space Efficiency

The claim about training state concerns growth order and is therefore measured where memory size can be varied freely. Figure 4(a) plots a synthetic codebook at five scales from 10^2 to 10^5 nodes; the flat line is the behavior Proposition 2 predicts, and the measurements fall into the two predicted regimes. Full-table gradients and moments grow linearly in n, reaching 4.58 GiB and 42.08 ms per step at $n = 10^5$, while the active-row path holds 48 KiB and 0.91–0.95 ms at every scale, a state ratio of about 10^5 at the largest point. Retrieval time is identical on both paths, leaving retrieval as the only per-step cost that still grows with memory size.

At the device level the training state at 10^5 nodes differs by 4.58 GiB in FP32 (Table 11), so optimizer state alone occupies about one seventh of a 32 GiB device. Table 11 accounts for every other storage class and shows why allocation at $N{=}10^3$ barely moves beside a 14.96 GiB backbone.

4.4 Route-aware Evaluation: Criterion and Cost

Four strategies share one re-test budget in Figure 4(c): full replay, unchecked reuse, route-aware checking, and a random budget of equal size. Route-aware checking holds zero output and score differences against full replay at CP1000 across three seeds, while the two shortcuts leave 11 and 10.3 score-changing results (14 and 13.3 changed outputs) unregenerated among the $1{,}500$ and $1{,}488$ queries they reuse. At CP1000 unchecked reuse reads 0.74 points high on Generalization and 0.28

on O.P., the random budget 0.69 and 0.26, while route-aware checking matches exactly. The re-test set therefore has to follow current routes rather than a sample. Across eight invalidation scenarios, from competing-key insertion to precision switches, it misses nothing (Appendices G and H.4).

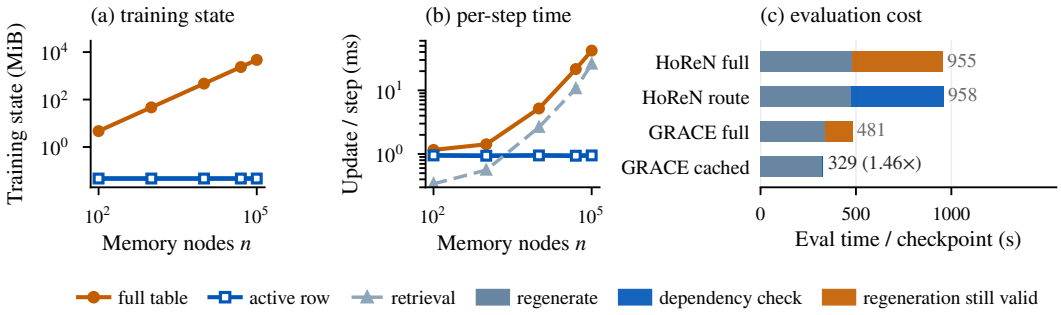


Figure 4: Measured cost profiles. (a) allocated training state and (b) per-step update time against memory size (FP32 microbenchmark; 4.58 GiB against 48 KiB of state and 42.08 against 0.95 ms per step at n=10^5; Tables 12 and 13). (c) end-to-end evaluation wall time per checkpoint at CP1000, the still-valid regeneration apportioned by query count (1,512 of 3,000 for HoReN, 70% for GRACE; Tables 16 and 22).

Full replay operates at cost parity with full evaluation, 1.00 ± 0.00x, and the decomposition says why (Table 16): reuse removes 482 s of generation per checkpoint while the check costs 486 s, so checking one query (0.32 s) costs what generating it costs. This is the $R \approx G$ boundary of the benefit condition in measured seconds, a property of that implementation, not of the criterion.

The second implementation removes the boundary. Caching the pre-injection retrieval input and replaying retrieval on the current memory decides acceptance from gate state, node identity and value version with no backbone forward. On this workload it returns the same verdicts, 12 of 1,500 candidates invalidated and 1,488 reused, with zero output, score and realized-reuse differences in three seeds, and on a dedicated device it spends 6.78 s on the check where full replay spends 485.77 s. Certification stops being the bottleneck and the reused share ρ sets the gain: the measured 1.20x at ρ=0.199 over three seeds sits just below the 1.25x of the equal-cost approximation $1/(1-\rho)$.

At CP1000, where ρ=0.496, the gain measures $1.89_{\pm 0.02}$x on a dedicated device with one strategy per process over three seeds, 95% of the equal-cost approximation, and $1.43_{\pm 0.02}$x cumulatively over the stream, lower because early checkpoints have little history to reuse (Appendix H.5). The benefit follows routing stability across distributions. On Qwen2.5-7B/WikiBigEdit at N=1000 the route predicate invalidates 10% of candidates against 0.8% on ZsRE, and both implementations hold zero output and score differences against the oracle in three seeds; full replay flags 186 candidates to cached-query checking's 150 and again lands at parity, 0.96x. The extra flags are conservative: in a per-query-logged rerun all 34 are next-token inequalities with unchanged output and score, MP16 noise the discrete predicate of gate, key and version never sees. On a dedicated device the cached-query gain measures $1.73_{\pm 0.02}$x per checkpoint and $1.37_{\pm 0.03}$x cumulatively over three seeds, the grid's first checkpoint having no history to reuse. Skipping the check is faster, 2.03x, but leaves 102 changed outputs unregenerated, 36 score-changing, and reads O.P. 0.46 points high. A criterion that stores no retrieval inputs and invalidates a result when its node appears in the interval's write set is faster than either and unsound on both distributions: beside cached-query checking on ZsRE it reaches 2.06x against 1.82x per checkpoint, but it flags 0 of 1,500 candidates on ZsRE, leaving all 12 changed outputs unregenerated, and on WikiBigEdit it is blind to 119 of the 150 invalidations, since these editors insert more often than they rewrite, and a new node winning a route or a gate flipping touches no old row.

Amortizing the check moves the operating point along the spectrum of Figure 8: full replay at anchors, reuse in between. It yields 1.6–2.4x, is exact at every anchor, and diverges between anchors by at most 5 of 3,000 outputs at k=2 and 16 at k=5, moving O.P. by at most 0.20 and 0.37 points against 0.53 for unchecked reuse in the same runs. Skipping the check reaches 5.49x while accumulating about 25 unregenerated differences by CP1000, never reset. Certified reuse is exact at every checkpoint, anchored reuse at every anchor with bounded lag, unchecked reuse guarantees nothing.

4.5 CROSS-MECHANISM TRANSFER

GRACE tests the transferability claim with a different mechanism on both sides, nearest-neighbor retrieval with an acceptance radius and value replacement instead of injection. On the editing side every per-item score is unchanged while edit time falls by 1.96 ± 0.05x on L8 and 1.68 ± 0.22x on Q7 (Table 2). GRACE's absolute quality is set by its acceptance radius: a radius tuned on a held-out slice raises Generalization from 5.01 to 13.90% on L8, and on that configuration the pair again agrees on every output and score while edit time falls 4.21 ± 0.05x (Table 23). On the evaluation side the two editors differ in the price of validity rather than in how often routes change. Under the same criterion both invalidate almost nothing, 12 of $1{,}500$ candidates on HoReN and none of GRACE's 900, with zero differences in both. GRACE's acceptance test reads stored retrieval inputs, keys and radii in 3.96 ± 0.13 s per checkpoint and turns reuse into 1.46 ± 0.06x, where HoReN's full replay spends 486 s and lands at parity (Appendix H.6). The tuned configuration reuses 49.0% of its queries and measures 1.98 ± 0.01x, matching the 1.96x equal-cost approximation, so on GRACE the gain tracks the reuse opportunity at both radii. Cheap validity is paid for in storage, 331 MB of retrieval inputs for GRACE and 663 MB for HoReN's cached-query path and none for full replay (Table 11). Likewise, MH-eHoReN with $H{=}64$ heads reaches 3.25x with three-seed O.P. matching MH-HoReN seed by seed, so computation reuse is independent of the number of retrieval heads.

4.6 LONG-HORIZON BEHAVIOR

The guarantees also hold over long streams: in the independent 10K stream the route-aware arm holds zero differences against the full oracle at every checkpoint, in three seeds each for both editors. Both 50K arms end at O.P. 92.23%, within 0.003 points on every metric at every shared checkpoint, Reliability holding above 99.3% while Generalization and Locality carry the decay (Appendix H.7). Cost separates them. The standard path spends 4.76 s/edit at 5K and 5.35 at 50K while eHoReN stays flat at 1.59, so the ratio widens from $3.16\times$ at 25K to $3.37\times$ at 50K, because Eq. 2's retrieval term is paid every optimization step on one path and once per edit on the other (Proposition 1). At a fixed daily budget that flatness separates 18K from 54K edits.

5 DISCUSSION AND CONCLUSION

eHoReN reorganizes parameter-preserving editing around one principle, that execution scope should equal dependency scope, and the speedups it derives reproduce as predicted on a codebook-memory editor with different retrieval and injection and on a multi-head variant.

The evaluation results are one benefit condition read at different (R, u, C): the gain vanishes where checking costs what generation costs, varies with the price of the acceptance test, and rises with the recertification interval at a stated fidelity price. With acceptance decidable without regenerating tokens, on GRACE natively and on the Hopfield path through cached queries, the reuse opportunity is the remaining bound on the gain. Correctness after editing becomes auditable, not spot-checked.

Limitations. Routing stability is distribution-dependent where the propositions and paired speedups are not; a third distribution could move the reuse share. The guarantee assumes deterministic decoding, a fixed protocol, and single injection positions; sampling, precision switches, and multi-point injection need separate treatment (Appendix D). Cached retrieval inputs reach 331 MB for GRACE and 663 MB for HoReN at $N{=}1000$ and grow with the history, which Table 11 sets against the training-state saving, and a criterion storing none of them misses every route change on ZsRE and most on WikiBigEdit (Appendix H.5). Retrieval remains $O(nd_k)$ per step, and the $\Theta(N^2/p)$ growth persists until a recertification interval is bounded.

Two consequences reach past this family and its one-entry-per-edit relatives (Wang et al., 2025; Luo et al., 2026). For parameter-modifying editors the cost accounting applies, with dependency records naming weights rather than rows, though the equivalence propositions do not transfer as stated; long-horizon stability stays with the update rule (Ma et al., 2026), which eHoReN leaves unchanged. For benchmarks, historical results that carry validity conditions remove the choice between quadratic re-testing and uncertifiable subsampling: quality at scale becomes checkable, and how much of an accumulated history is reusable becomes the quantity worth measuring.

Reproducibility Statement

All formal runs record seeds, data order, precision configuration, stopping rules, and per-checkpoint raw timing; scaler attempt/overflow/fallback counts are logged for every MP16 stream. We release the eHoReN implementation at `https://anonymous.4open.science/r/eHoReN/`, together with the test and evaluation source code used for every result: the dependency-recording and route-aware checking harness, the timing instrumentation with its scope definitions (which operations each timer encloses and where synchronization points sit), the paired-arm and one-strategy-per-process drivers, and the frozen run configurations, so that both the quality and the time rows of Table 1 can be reproduced independently. Protocol tables, configurations, and hardware records are in Appendix F; proofs are in Appendices C.1–C.3.

Use of Large Language Models

Generative AI tools were used in this work only to polish the writing: they improved the grammar, readability, and concision of text drafted by the authors. They were not used to generate data, develop the theoretical model or conceptual framework, formulate or prove mathematical claims, propose or refine hypotheses, design or give feedback on the methodology or experiments, implement methods, clean or reformat data, analyze or interpret results, or search for, summarize, or format literature. Every AI-assisted edit was reviewed by the authors against the original draft, and every claim, number, and reference in the paper was checked by the authors against the experimental records and the cited sources. We take responsibility for the final content of this work.

References

Umut A. Acar, Matthias Blume, and Jacob Donham. A consistent semantics of self-adjusting computation. *Journal of Functional Programming*, 2013.

Yash Kumar Atri, Ahmed Alaa, and Thomas Hartvigsen. Model editing with graph-based external memory. *arXiv preprint arXiv:2505.18343*, 2025.

Yue Che, Chen Guoyi, Xijian Deng, Yujie Wu, Jiang Cai, Xuming Ran, and Mingkun Xu. MH-HoReN: Mitigating memory interference in lifelong model editing through multi-head hopfield retrieval. GitHub repository, `https://github.com/YUECHE77/MH-HoReN`, 2026. Accessed 2026-09-26.

Qizhou Chen, Taolin Zhang, Xiaofeng He, Dongyang Li, Chengyu Wang, Longtao Huang, and Hui Xue. Lifelong knowledge editing for LLMs with retrieval-augmented continuous prompt learning. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pp. 13565–13580, 2024.

Nicola De Cao, Wilker Aziz, and Ivan Titov. Editing factual knowledge in language models. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, 2021.

Junfeng Fang, Houcheng Jiang, Kun Wang, Yunshan Ma, Jie Shi, Xiang Wang, Xiangnan He, and Tat-Seng Chua. AlphaEdit: Null-space constrained knowledge editing for language models. In *International Conference on Learning Representations*, 2025.

Yuan Fang, Yi Xie, and Xuming Ran. HoReN: Normalized hopfield retrieval for large-scale sequential model editing. *arXiv preprint arXiv:2605.08143*, 2026.

Xiaojie Gu, Ziying Huang, Jia-Chen Gu, and Kai Zhang. Ultraedit: Training-, subject-, and memory-free lifelong editing in language models. *arXiv preprint arXiv:2505.14679*, 2025.

Yaming Guo, Siyang Guo, Hengshu Zhu, and Ying Sun. Towards lifelong model editing via simulating ideal editor. In *Proceedings of the 42nd International Conference on Machine Learning*, 2025.

Thomas Hartvigsen, Swami Sankaranarayanan, Hamid Palangi, Yoon Kim, and Marzyeh Ghassemi. Aging with GRACE: Lifelong model editing with discrete key-value adaptors. In *Advances in Neural Information Processing Systems*, 2023.

Joel Jang, Seonghyeon Ye, Changho Lee, Sohee Yang, Joongbo Shin, Janghoon Han, Gyeonghun Kim, and Minjoon Seo. TemporalWiki: A lifelong benchmark for training and evaluating ever-evolving language models. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, 2022.

Jiaang Li, Quan Wang, Zhongnan Wang, Yongdong Zhang, and Zhendong Mao. Elder: Enhancing lifelong model editing with mixture-of-lora. *arXiv preprint arXiv:2408.11869*, 2024.

Zherui Li, Houcheng Jiang, Hao Chen, Baolong Bi, Zhenhong Zhou, Fei Sun, Junfeng Fang, and Xiang Wang. Reinforced lifelong editing for language models. *arXiv preprint arXiv:2502.05759*, 2025.

Yuhan Liu, Saurabh Agarwal, and Shivaram Venkataraman. AutoFreeze: Automatically freezing model blocks to accelerate fine-tuning. *arXiv preprint arXiv:2102.01386*, 2021.

Haihua Luo, Xuming Ran, Tommi Kärkkäinen, Zhonghua Chen, Jiangrong Shen, Qi Xu, and Fengyu Cong. Reversible lifelong model editing via semantic routing-based LoRA. *arXiv preprint arXiv:2603.11239*, 2026.

Xin Ma, Wei Chen, Qi Liu, Derong Xu, Zhi Zheng, Tong Xu, and Enhong Chen. More edits, more stable: Understanding the lifelong normalization in sequential model editing. *arXiv preprint arXiv:2605.11836*, 2026.

Kevin Meng, David Bau, Alex Andonian, and Yonatan Belinkov. Locating and editing factual associations in GPT. In *Advances in Neural Information Processing Systems*, 2022.

Kevin Meng, Arnab Sen Sharma, Alex Andonian, Yonatan Belinkov, and David Bau. Mass-editing memory in a transformer. In *International Conference on Learning Representations*, 2023.

Eric Mitchell, Charles Lin, Antoine Bosselut, Chelsea Finn, and Christopher D. Manning. Fast model editing at scale. In *International Conference on Learning Representations*, 2022a.

Eric Mitchell, Charles Lin, Antoine Bosselut, Christopher D. Manning, and Chelsea Finn. Memory-based model editing at scale. In *International Conference on Machine Learning*, 2022b.

Ameya Prabhu, Vishaal Udandarao, Philip H. S. Torr, Matthias Bethge, Adel Bibi, and Samuel Albanie. Efficient lifelong model evaluation in an era of rapid progress. In *Advances in Neural Information Processing Systems*, volume 37, 2024.

PyTorch Contributors. torch.optim.SparseAdam. PyTorch documentation, `https://pytorch.org/docs/stable/generated/torch.optim.SparseAdam.html`, 2024.

Hubert Ramsauer, Bernhard Schäfl, Johannes Lehner, Philipp Seidl, Michael Widrich, Thomas Adler, Lukas Gruber, Markus Holzleitner, Milena Pavlović, Geir Kjetil Sandve, et al. Hopfield networks is all you need. In *International Conference on Learning Representations*, 2021.

Lukas Thede, Karsten Roth, Matthias Bethge, Zeynep Akata, and Thomas Hartvigsen. WikiBigEdit: Understanding the limits of lifelong knowledge editing in LLMs. In *International Conference on Machine Learning*, 2025.

Jinke Wang, Zenan Ying, Qi Liu, Wei Chen, Tong Xu, Huijun Hou, and Zhi Zheng. Think and recall: Layer-level prompting for lifelong model editing. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 14487–14502, 2025.

Peng Wang, Zexi Li, Ningyu Zhang, Ziwen Xu, Yunzhi Yao, Yong Jiang, Pengjun Xie, Fei Huang, and Huajun Chen. WISE: Rethinking the knowledge memory for lifelong model editing of large language models. In *Advances in Neural Information Processing Systems*, 2024a.

Renzhi Wang and Piji Li. Lemoe: Advanced mixture of experts adaptor for lifelong model editing of large language models. *arXiv preprint arXiv:2406.20030*, 2024.

Yangfan Wang, Tianyang Sun, Chen Tang, Jie Liu, Wei Cai, and Jingchi Jiang. HiEdit: Lifelong model editing with hierarchical reinforcement learning. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 39924–39942, 2026.

Yu Wang, Yifan Gao, Xiusi Chen, Haoming Jiang, Shiyang Li, Jingfeng Yang, Qingyu Yin, Zheng Li, Xian Li, Bing Yin, Jingbo Shang, and Julian McAuley. MemoryLLM: Towards self-updatable large language models. In *International Conference on Machine Learning*, 2024b.

Erik Wijmans, Brody Huval, Alexander Hertzberg, Vladlen Koltun, and Philipp Krähenbühl. Cut your losses in large-vocabulary language models. In *International Conference on Learning Representations*, 2025.

Yunzhi Yao, Peng Wang, Bozhong Tian, Siyuan Cheng, Zhoubo Li, Shumin Deng, Huajun Chen, and Ningyu Zhang. Editing large language models: Problems, methods, and opportunities. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pp. 10222–10240, 2023.

A NOTATION

Symbols used throughout the paper are collected in Table 3.

Table 3: Notation, grouped by what each symbol describes, namely the edit stream, the per-edit optimization budget, the memory dimensions, the retrieval mechanism, the cost-model terms, and the evaluation workload. Dimensions are model constants and counts are per edit unless stated.

Symbol	Meaning
Rel., Gen., Loc., O.P.	Reliability, Generalization, Locality, and their mean
$N,\ n_t$	Stream length; node count after edit t ($n_t < t$ under node reuse)
$M,\ T,\ A$	Optimization steps, input tokens, and supervised tokens per edit
$d_k,\ d_v,\ \lvert\mathcal{V}\rvert$	Key dim., value/hidden dim., vocabulary size
H	Hopfield iterations per retrieval (complexity uses the configured cap)
$K,\ V,\ \nu$	Keys, values, node versions
$i^*,\ g,\ \ell$	Selected node, binary gate, injection layer
$P(T)$	Frozen-prefix forward cost (before the cache boundary)
$S_{\mathrm{net}}(T)$	Network cost from boundary to final hidden states, incl. required backward
$S_{\mathrm{head}}(k)$	Projection and supervised loss at k positions, incl. required backward
$\mathcal{R}(n),\ \mathcal{D}(n)$	Retrieval cost $O((H{+}1)nd_k)$; full-table gradient and Adam cost $O(nd_v)$
$f,\ s$	Reusable fraction of the original path; its local speedup
p (protocol)	Checkpoint interval: full historical evaluation every p edits
$Q,\ U,\ C$	Total queries; new (uncached); historically invalidated after checking
$G,\ R,\ B$	Mean full generation-and-scoring cost; mean check cost; net maintenance cost vs. the full path
$u,\ c,\ \rho$	New fraction U/Q; invalidated fraction C/Q; reused share $1-u-c$
$L_j,\ P_j$	Output and prompt lengths of query j
$b_k,\ b_v,\ b_q$	Bytes per element of keys, values, cached retrieval inputs

B COMPLEXITY COMPARISON

With reusable fraction f and local speedup s, the total speedup ignoring added maintenance is $1/[(1-f)+f/s]$, which is why it varies with backbone and editor structure.

An edit runs M optimization steps on an input of T tokens with A supervised positions, over a memory of n nodes; P, S_{net}, S_{head}, $\mathcal{R}$, $\mathcal{D}$ denote the costs of the frozen prefix forward, the network replay, the vocabulary projection, retrieval, and the optimizer update. Let C_0, C_0' collect preparation and maintenance costs outside the explicit terms, and κ the constant number of retrievals needed for node preparation. Reuse is not free: $C_0' > C_0$, because preparing the route and the cached representation costs κ retrievals and $O(Td_v)$ storage per edit, a fixed overhead whose share grows as M shrinks. With $u = U/Q$ and $c = C/Q$, the evaluation cost ratio is $T_{\mathrm{route}}/T_{\mathrm{full}} \approx (1-u)R/G + u + c + B/(QG)$, and online storage for route-aware evaluation is $O(\sum_j L_j + Q + n)$, plus $O(\sum_j L_j d_k)$ when retrieval inputs are cached.

Table 4: Per-edit compute and space, standard path versus eHoReN (projection rows list forward complexity; the required backward is of the same order).

Item	Full/full-table path	eHoReN path	Deciding variables
Frozen prefix forward	$M\,P(T)$	$P(T)$	boundary depth, T, M
Output projection	$O(MTd_v\|\mathcal{V}\|)$	$O(MAd_v\|\mathcal{V}\|)$	ratio A/T, vocabulary
Within-edit retrieval	$O(M(H{+}1)nd_k)$	$O((H{+}1)nd_k)$	n, iterations H
Gradient + Adam update	$O(Mnd_v)$	$O(Md_v)$	n; one active row
Gradient + moments (FP32)	$12nd_v$ bytes	$12d_v$ bytes	decoupled from n
Persistent keys/values	$n(b_kd_k + b_vd_v)$ bytes	same	bytes b_k, b_v
Per-edit representation cache	—	$O(Td_v)$	constant space of reuse

C Proofs and Correctness Conditions

C.1 Proof of Proposition 1

Fix the edit request; write the frozen computation before the cache boundary as $z = P_\theta(x)$, the computation after it as $u = S_\theta(z, v)$, and the supervised-position selector as $\Pi_\mathcal{A}$. The full path's objective is

$$\mathcal{L}_{\text{full}}(v) = \ell(\Pi_\mathcal{A}\left[W_{\text{out}}S_\theta(P_\theta(x), v)\right], y)\,.$$

Since P_θ's parameters and input are fixed and execution is deterministic, z is identical across optimization steps and $\partial z/\partial v = 0$. The vocabulary projection is position-wise independent, so $\Pi_\mathcal{A}(W_{\text{out}}u) = W_{\text{out}}\Pi_\mathcal{A}(u)$. Substituting both relations yields the cached, selected-projection objective, so the two paths have equal loss for every v; where differentiable, the chain rule gives the same $\nabla_v\mathcal{L}$. The conclusion requires identical supervised positions, label shift, loss normalization, and positional encoding, and that the objective uses only these positions' logits; constraints on unsupervised positions must include the corresponding outputs. For routing, the single-injection-position query $q = \psi_\ell(z)$ is produced before value injection; after node preparation, keys, iteration rule, threshold, and tie-breaking are fixed, so i^* and g are constant in v and caching the final route preserves the injection function. When multiple editing layers interact, one must re-confirm that upstream updates cannot change downstream queries. The proof is over real arithmetic; floating-point shapes, reduction orders, and attention paths can change rounding, so implementations require step-wise numerical comparison, and states affected by injection are never treated as frozen cache.

C.2 Proof of Proposition 2

Let both paths share $V^{(0)}$, the final node i^*, and the gate after node preparation. Write row j of the full-table loss gradient at step r as $g_j^{(r)}$; single-row dependency gives $g_j^{(r)} = 0$ for all $j \neq i^*$. Adam's states satisfy

$$m_j^{(r)} = \beta_1 m_j^{(r-1)} + (1-\beta_1)g_j^{(r)},\;\; u_j^{(r)} = \beta_2 u_j^{(r-1)} + (1-\beta_2)(g_j^{(r)})^{\odot 2},\;\; v_j^{(r)} = v_j^{(r-1)} - \eta\frac{\widehat{m}_j^{(r)}}{\sqrt{\widehat{u}_j^{(r)}}+\epsilon}.$$

With $m_j^{(0)} = u_j^{(0)} = 0$ at every edit start, induction gives all-zero moments and updates for inactive rows. For the active row, the temporary parameter starts at $a^{(0)} = v_{i^*}^{(0)}$ and receives the same loss gradients; if parameters and states agree through step $r{-}1$, the step-r gradient, bias correction, and update also agree, so the final write-back values coincide. When the gate stays closed, all rows have zero gradient and the conclusion still holds. Premises: identical learning rate, Adam hyperparameters, and step count; no weight decay; no cross-row coupling. If moments persisted across edits, an inactive row could carry non-zero momentum and be updated at zero gradient; creating a fresh Adam per edit satisfies the reset condition. Under exact arithmetic both paths produce the same loss sequence, so identical deterministic early stopping stops at the same step; floating-point perturbations can move stops near the threshold, so fixed-step comparisons isolate per-step cost.

C.3 Proof of Proposition 3

For query j with old output $y_1, \ldots, y_L$, fix tokenizer, prompt, backbone, injection positions, computation configuration, and deterministic decoding. Assume that under the current memory state,

feeding the old prefix $(x, y_{<s})$ yields the same effective injection at every step s as the old execution. At $s{=}1$ the inputs coincide and the frozen computation before injection coincides; the same injection at the same position makes the subsequent network outputs equal, and deterministic decoding selects the same y_1. If the first $s{-}1$ outputs coincide, the step-s input equals the old prefix used by the check; by assumption the injection, logits, and next token coincide. Induction gives full output equality, and identical EOS, maximum length, and other deterministic stopping rules trigger on the same sequence, so lengths agree. For single-value injection $g_s v_{i_s}$: with both gates closed the injection is zero on both sides; with the gate open, identical node identity and version give the same value, under stable node IDs and version advance on every write-back. A newly added key can change the whole Hopfield trajectory, so version records must be used jointly with current retrieval; conversely a version advance may leave the value numerically unchanged, in which case re-testing is conservative invalidation.

C.4 Boundary Conditions and the Executed Checker

Generation and checking both use `use_cache=False`; the injection position is fixed at the prompt's last token and retrieval inputs come from the prompt region. The checker replays a full forward along every old prefix, then compares trace length, per-step gate, effective node identity/version, and predicted token; any disagreement sends the query to re-test. The trace stores chosen key, gate, max score, key version, and node count; max score and node count are diagnostic only. Direct token comparison provides an execution-level verification of the induction chain in addition to the trace. For the check's logits argmax to correspond to the generator's choice, the generation config's logits processors, repetition penalties, forced tokens, and stopping settings must match; custom generation rules require independent validation. Formal evaluation compares caching strategies against the full oracle under identical precision configuration; switching model, tokenizer, target version, generation settings, or precision rebuilds the cache. On the editing side, the propositions' boundary conditions are: losses with unsupervised-position constraints must enter the projection (Prop. 1); moments persisting across edits, weight decay, or cross-row regularizers break equivalence (Prop. 2); abnormal exits clean up active parameters, optimizer state, and hooks, but node insertions already performed are not transactionally rolled back, so runs after a failed edit must be marked and resumed from a verifiable state.

D Numerical Equivalence under Mixed Precision

Propositions 1 and 2 guarantee behavioral equivalence in exact arithmetic; under MP16, floating-point non-associativity and data-dependent dynamic loss scaling define the boundary between that guarantee and the implementation. The measured boundary is as follows. On L8/ZsRE N1K under fixed-step editing, the paired arms reproduce identical per-request scores and output hashes across all three seeds: at the hash level, every prediction is exactly identical. Dynamic-scaler telemetry is logged per stream (on Q7 N1K, $50{,}000$ scaling attempts with zero fallbacks in both arms), so scaler behavior is observable rather than assumed. On Q7, hash-level identity is not established: the paired arms agree within one three-seed standard deviation on every metric (largest gap: Reliability 99.56% vs. 99.37%, against sds of 0.26 and 0.29), and Locality coincides exactly. Fixed-step and early-stop configurations are reported separately throughout (Table 9), because a stopping rule thresholds on loss and can convert an ULP-level difference into a different step count; the per-sample count of such threshold crossings is not instrumented in the current logs, so we delimit this boundary rather than quantify it.

E Algorithms

Algorithm 1 writes one edit request; the comments name the optimization each line carries.

Algorithm 1: eHoReN edit (one request)

Input: Edit request (x, y^*); codebook $\mathcal{C} = \{(k_i, v_i, \nu_i)\}$; injection layer ℓ and position π; step cap M
Output: Updated codebook $\mathcal{C}$
Build inputs, shifted labels, supervised set $\mathcal{A}$, and position π; `// once per edit`
$z \leftarrow F_{<\ell}(x)$ without gradients; finish node preparation; `// frozen prefix, once`
$q \leftarrow \psi_\ell(z)$; $(i^*, g) \leftarrow \textsc{Retrieve}(q, \mathcal{C})$;
if *a node was added* **then**
  re-retrieve and overwrite (i^*, g);
Cache the route (i^*, g) for this edit; `// route cache`
Freeze the persistent table V; $a \leftarrow \mathrm{copy}(V[i^*])$;
Initialize Adam state for a alone; `// O(d_v) training state`
for $u \leftarrow 1$ **to** M **do**
  Replay the injection block and the network above it from z, injecting a under the cached route;
  Project and compute $\mathcal{L}_{\mathrm{CE}}(f_{\theta_\mathcal{C}}(x), y^*)$ at the positions of $\mathcal{A}$ only; `// supervised projection`
  Backpropagate and update a only;
  if *early stopping criterion is met* **then**
    break;
$V[i^*] \leftarrow a$; $\nu[i^*] \leftarrow \nu[i^*] + 1$; release all within-edit state;
return $\mathcal{C}$;

Algorithm 2 is the checkpoint side: it decides per query whether the cached result still holds, and re-tests only the rest.

Algorithm 2: eHoReN evaluation at a checkpoint

Input: Checkpoint state S; cached outputs, scores and traces of the historical queries $\mathcal{H}$; new queries $\mathcal{U}$; fixed protocol
Output: Rel, Gen, Loc and their mean O.P.
$\mathcal{R} \leftarrow \mathcal{U}$; `// new queries are never reused`
foreach *cached query* $j \in \mathcal{H}$ **do**
  if *Hopfield memory* **then**
    Replay the old generation prefix under S to obtain the current trace and predicted tokens;
    if *trace length, per-step gate, effective node identity or version, or predicted tokens differ* **then**
      $\mathcal{R} \leftarrow \mathcal{R} \cup \{j\}$;
  else
    Read the cached retrieval inputs q_s and recompute nearest neighbour and acceptance on the current codebook; `// no backbone forward`
    if *the accepted node identity or version differs at any step* **then** $\mathcal{R} \leftarrow \mathcal{R} \cup \{j\}$;
  ; `// either test is the sufficient condition of Prop. 3`
Generate and score $\mathcal{R}$; replace the cached outputs, scores and traces of $\mathcal{R}$;
return the aggregate of all per-query scores over the current valid set;

Queries that pass the check in Algorithm 2 keep their cached output and score unchanged, so the re-tested set is exactly $\mathcal{U}$ together with the queries whose effective injections moved.

F Experimental Protocol

Runs at $N \leq 1000$ report means $\pm$ sample standard deviation over seeds 11/22/33, except the WikiBigEdit panel, where GRACE, HoReN and eHoReN are single seed (Table 8); quality and cost streams of 10^4 edits and longer use seed 42, and the 10K evaluation-reuse streams seeds 22/33/42.

Evaluation times are executed wall-clock; the harness records the basis with every run and no archived run used a reconstruction from oracle query times. Within one run the strategies execute sequentially in a single process, so a strategy that runs later starts from a warmer device: on the WikiBigEdit workload the same cached-query strategy measured 1.87x as the third of four strategies and 1.61x as the second of two, with identical verdicts. Speedups quoted in Section 4.4 therefore come from dedicated-device reruns timing one strategy per process (Tables 18 and 19).

Table 5 maps each headline ratio to what it measures. API time is the harness's internal per-edit timer; pure edit time is checkpoint-recorded edit seconds excluding evaluation; evaluation time is end-to-end wall-clock per checkpoint.

Table 5: Headline ratios and their bases. Each ratio divides two arms sharing model, data order, stopping rule, precision, and hardware; three seeds throughout except the 50K stream (seed 42).

Ratio	Where	Basis	Setting	Numerator / denominator
3.51x	Table 2	API time	paired arms, one process each	HoReN / eHoReN, L8/ZsRE N=1000
2.77x	Table 2	API time	paired arms, one process each	HoReN / eHoReN, Q7/ZsRE N=1000
1.96x, 4.21x	Tables 2, 23	API time	paired arms	GRACE / GRACE+ours, reference and tuned
1.89x	Table 18	eval. wall-clock	dedicated device, one strategy per process	full oracle / cached query, CP1000
1.43x	Table 18	eval. wall-clock, cumulative	dedicated device, one strategy per process	full oracle / cached query, CP500+CP1000
1.6–2.4x	Table 15	eval. wall-clock	one process, four strategies	exact check / recertify at k
1.46x	Table 22	eval. wall-clock	one process	GRACE full oracle / cached query, CP1000
3.37x	Figure 10	pure edit time	independent 50K streams	HoReN / eHoReN at 50K

In these runs the request order and edit-key sequence are identical across seeds, so the cached-query invalidation count has zero spread and the quality spread comes from the full generated outputs. In the tables, fail marks a run that crashed before its terminal evaluation, − a scale that was not run for that configuration, a blank cell a panel it does not apply to, and n/a a quantity that was not archived for an otherwise valid run. All formal runs use MP16 (frozen backbone FP16; values, training state, losses, and routing-sensitive operations FP32) with a real dynamic loss scaler whose attempt/success/overflow/non-finite/fallback counts are logged per stream. Stopping rules: F50 (fixed 50 steps), ES50 (early stop, cap 50), G100 (GRACE fixed 100). Quality is $\mathrm{O.P.} = (\mathrm{Rel} + \mathrm{Gen} + \mathrm{Loc})/3$ from per-query scores over the currently valid set. Timing bases: cross-method tables report the per-edit time recorded in the checkpoint records, total edit time divided by the edit count and excluding checkpoint evaluation, while paired tables report the harness's internal per-edit timer, the same timer in both arms of a pair. The two coincide on the Hopfield rows, where 5.38 s/edit appears in both Table 1 and Table 2 for the same configuration, and differ on GRACE, 9.24 against 4.98 s/edit, which is why no ratio is formed across the two tables. Job wall-clock divided by the edit count is not a per-edit time and is never reported as one: on the configuration just named it reads 8.94 s/edit where the edit loop reads 5.38. Speed ratios never mix hardware, precision, data order, or stopping protocol. GRACE's radius $\epsilon = 1$ with implementation-frozen hyperparameters makes its Generalization near zero and Reliability/Locality full; the transfer section's claims are therefore paired speedups and zero-difference checks, with GRACE's absolute quality on the frontier read under this qualification. Seeds 11/22/33 at $N \leq 1000$, seed 42 for long streams; identical data order within every comparison. Baseline budgets differ across methods, so cross-method timing differences are not pure algorithmic advantages; the paper's efficiency claims rest on within-editor paired comparisons sharing hyperparameters item by item.

Token-prefix KV caching is available as an independent switch: it caches the frozen KV before the injection token, but maintaining that cache can cost more than it saves on short inputs, so it is off by default in every run reported here.

G Negative Results and Boundary Cases

Token-prefix KV cache. Building the frozen KV before the injection token shortens each replay but adds cache construction and container maintenance; on short inputs the net effect is negative, so the main algorithm ships with the switch off, and its records are retained as a negative result.

Controlled invalidation scenarios. Eight adversarial scenarios stress-test the criterion of Proposition 3: competing-key insertion with old values unchanged; hit-node write-back with version ad-

vance; gates closed on both sides; multi-token, EOS, and generation-config boundaries; radius expansion, contraction, and conflict splitting (GRACE); nearest-neighbor insertion with old versions unchanged (GRACE); cache miss and protocol/precision switches; step-wise optimization divergence localization (GRACE). Across all eight, activation-route misses no actual output change, while activation-only misses changes in the competing-key scenario, consistent with the natural-stream result of Section 4.4.

Component-to-cost mapping. Table 6 pairs each component with its theoretical term and measured paired time difference; differences are of the same order as three-seed sd and are read directionally, with the isolated attribution of Section 4.1 as the primary time evidence.

Table 6: Component-to-cost mapping (Fixed 50, V100 MP16, seeds 11/22/33). Paired time difference is core time without minus with the component on the same request; positive means saving. Preparation and self-check time was 0.00 ± 0.00 s for every component and is omitted.

Component	Theory term	Per-step cost	Paired time diff/edit (s)	API mean (s)	Peak alloc / reserved (GiB)
Active row	$\mathcal{D}(n) \to \mathcal{D}(1)$	0.03	+0.04±0.03	1.75±0.01	15.09 / 15.24
Route cache	$M\mathcal{R}(n) \to \kappa\mathcal{R}(n)$	0.04	+0.01±0.03	1.77±0.03	15.10 / 15.25
Edit prep	Init, index, sync	0.04	+0.03±0.04	1.77±0.03	15.10 / 15.25
Token cache	Prefix-KV build, replay	0.03	+0.02±0.04	1.75±0.01	15.09 / 15.24

H Extended Tables

H.1 Frontier Panels

The $N{=}10^4$ columns of Table 1 are single-seed (seed 42) and carry no spread; their markings follow the same column rule as the three-seed columns. Both external-memory baselines reach the second panel only after a numerical fix of their own: deterministic decoding without a key-value cache in GRACE's scoring, and an FP32 token-level loss in WISE.

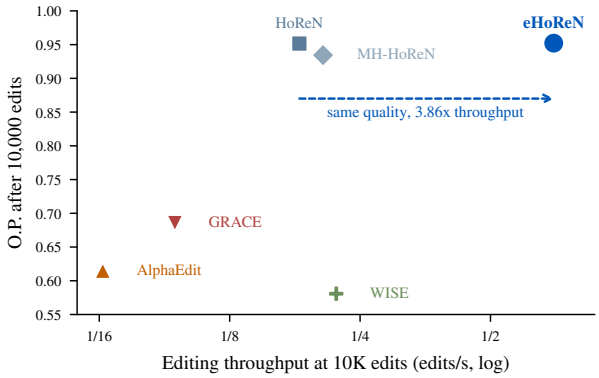


Figure 5: Single-scale view of Table 1: quality vs. throughput at 10K edits (L8/ZsRE; throughput is inverse pure edit time, log axis). eHoReN holds the Hopfield-family O.P. at the lowest cost.

A shortcut that quietly changed the optimization would surface as a quality gap widening with N, because errors compound in a sequential editor; neither backbone shows such divergence. Table 7 decomposes the main results per metric, and Table 8 repeats the panel on Qwen2.5-7B with WikiBigEdit, where two baselines fail numerically. On that panel AlphaEdit leads on quality at about 15x eHoReN's cost, and the two external-memory baselines land below the Hopfield rows, GRACE flat near 67 and WISE at 56.75 to eHoReN's 75.63 at $N{=}100$. Figure 5 reads one scale as a quality–throughput plane.

The provenance of Table 8 is as follows. GRACE's earlier $N{=}100$ failure was a device-side assertion in scoring, removed by decoding deterministically without a key-value cache; its entries are single seed 11 with per-edit times of 8.87, 8.98 and 9.01 s at $N{=}100$, 500 and 1000. Its

Table 7: Per-metric decomposition of Table 1 on Llama-3.1-8B/ZsRE: three-seed mean$_{\pm sd}$ (%) at each scale.

N	Method	Rel.	Gen.	Loc.	O.P.
100	AlphaEdit	$97.65_{\pm 0.38}$	$90.81_{\pm 1.23}$	$90.19_{\pm 2.29}$	$92.88_{\pm 1.16}$
	WISE	$75.53_{\pm 3.58}$	$68.43_{\pm 4.70}$	$100.00_{\pm 0.00}$	$81.32_{\pm 2.75}$
	GRACE	$100.00_{\pm 0.00}$	$5.90_{\pm 0.00}$	$100.00_{\pm 0.00}$	$68.63_{\pm 0.00}$
	HoReN	$99.93_{\pm 0.12}$	$88.37_{\pm 0.64}$	$100.00_{\pm 0.00}$	$96.10_{\pm 0.24}$
	MH-HoReN	$99.93_{\pm 0.12}$	$85.77_{\pm 0.58}$	$100.00_{\pm 0.00}$	$95.23_{\pm 0.18}$
	eHoReN	$99.80_{\pm 0.35}$	$88.37_{\pm 0.64}$	$100.00_{\pm 0.00}$	$96.06_{\pm 0.30}$
500	AlphaEdit	$98.89_{\pm 0.19}$	$93.91_{\pm 0.94}$	$81.61_{\pm 2.82}$	$91.47_{\pm 0.97}$
	WISE	$68.40_{\pm 1.52}$	$63.54_{\pm 1.89}$	$99.93_{\pm 0.01}$	$77.29_{\pm 1.14}$
	GRACE	$99.99_{\pm 0.02}$	$4.44_{\pm 0.00}$	$100.00_{\pm 0.00}$	$68.14_{\pm 0.01}$
	HoReN	$99.71_{\pm 0.17}$	$92.61_{\pm 0.09}$	$99.05_{\pm 0.00}$	$97.12_{\pm 0.08}$
	MH-HoReN	$99.66_{\pm 0.24}$	$91.22_{\pm 0.22}$	$98.48_{\pm 0.00}$	$96.46_{\pm 0.15}$
	eHoReN	$99.67_{\pm 0.19}$	$92.61_{\pm 0.16}$	$99.05_{\pm 0.00}$	$97.11_{\pm 0.12}$
1000	AlphaEdit	$98.35_{\pm 0.03}$	$93.21_{\pm 0.44}$	$75.98_{\pm 3.82}$	$89.18_{\pm 1.30}$
	WISE	$58.96_{\pm 3.33}$	$55.49_{\pm 2.88}$	$99.65_{\pm 0.09}$	$71.37_{\pm 2.08}$
	GRACE	$100.00_{\pm 0.01}$	$5.01_{\pm 0.00}$	$100.00_{\pm 0.00}$	$68.33_{\pm 0.00}$
	HoReN	$99.49_{\pm 0.09}$	$92.47_{\pm 0.16}$	$98.83_{\pm 0.00}$	$96.93_{\pm 0.08}$
	MH-HoReN	$99.56_{\pm 0.11}$	$90.86_{\pm 0.19}$	$97.73_{\pm 0.01}$	$96.05_{\pm 0.08}$
	eHoReN	$99.50_{\pm 0.11}$	$92.47_{\pm 0.16}$	$98.83_{\pm 0.00}$	$96.93_{\pm 0.09}$

O.P. is carried by Reliability and Locality with Generalization near zero, $99.00/2.19/100.00\%$ at $N{=}100$ and $99.80/1.67/100.00\%$ at $N{=}1000$, the acceptance radius again setting its absolute quality. MH-HoReN reports seeds 11/22/33 with Rel/Gen/Loc $= 100.00/94.25/38.07\%$ at $N{=}100$ and $99.17_{\pm 0.06}/88.94_{\pm 0.08}/35.72_{\pm 0.05}\%$ at $N{=}1000$, Locality again carrying the gap. WISE's $N{=}100$ failure was its token-level cross-entropy diverging to NaN in MP16 on Qwen; computing that loss in FP32 repairs it, and the $N{=}100$ entry is the three-seed mean over seeds 11/22/33 with Rel/Gen/Loc $= 74.71_{\pm 3.87}/68.11_{\pm 1.43}/27.44_{\pm 4.18}\%$, Locality carrying the gap to the Hopfield rows. All WISE entries from $N{=}10$ on use the FP32 loss with a finite-value guard that rolls back to the last finite weight: $N{=}10$ is seed 42 with no guard event, $N{=}500$ the three-seed mean over 11/22/33 with one to two guard events per run, and $N{=}1000$ the three-seed mean over the same seeds, where seeds 11 and 33 had two guard events each and reached O.P. 32.72 and 35.34% while on seed 22 the guard fired 492 times and the run ended at 8.57% in 6.23 s/edit, which the standard deviation reflects; WISE is unstable at this scale on WikiBigEdit even with the guard. The $N{=}1$ entry predates the FP32 fix. At $N{=}10^4$ (seed 42) HoReN and eHoReN reach O.P. 67.70 and 67.68% at 5.71 and 2.40 s/edit, Locality carrying the decay at 18.67%, and MH-HoReN reaches 69.58% at 4.91 s/edit with Locality at 24.03%; the AlphaEdit, WISE and GRACE runs at that scale terminated with a CUDA assertion error before their terminal evaluation, so those cells are marked fail.

Table 8: Second frontier panel (Q7/WikiBigEdit), same protocol as Table 1; time is pure edit s/edit at $N{=}500$; quality metrics in %. Single-seed entries carry no spread. Run provenance and the per-metric values behind each row are given in the text above.

Method	O.P. @1	O.P. @10	O.P. @100	O.P. @500	O.P. @1000	Time/Edit @500 (s)
AlphaEdit	100.00	$84.89_{\pm 1.92}$	$88.72_{\pm 0.88}$	$82.37_{\pm 0.40}$	$77.30_{\pm 0.59}$	$31.52_{\pm 1.11}$
WISE	100.00	52.44	$56.75_{\pm 2.19}$	$47.60_{\pm 1.18}$	$25.54_{\pm 14.75}$	12.10
GRACE	66.67	66.67	67.06	67.15	67.16	8.98
HoReN	100.00	68.17	75.63	72.83	71.97	4.69
MH-HoReN	–	$68.17_{\pm 0.00}$	$77.44_{\pm 0.00}$	$76.04_{\pm 0.12}$	$74.61_{\pm 0.02}$	$5.21_{\pm 0.50}$
eHoReN	100.00	68.17	75.63	72.70	71.96	1.96

H.2 Main Editing and Ablations

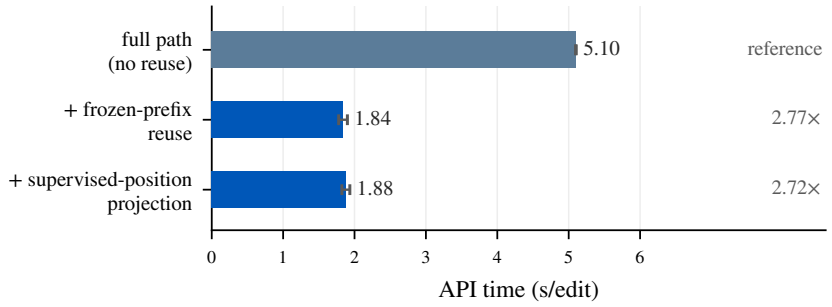


Figure 6: Isolated attribution of the editing speedup (L8/ZsRE, N=100, fixed 50 steps, MP16, three seeds; mean per-edit API time). The frozen prefix carries it; supervised-position projection moves wall time by less than the seed spread, and peak allocated memory is unchanged across the arms at 15.12/15.11/15.10 GiB.

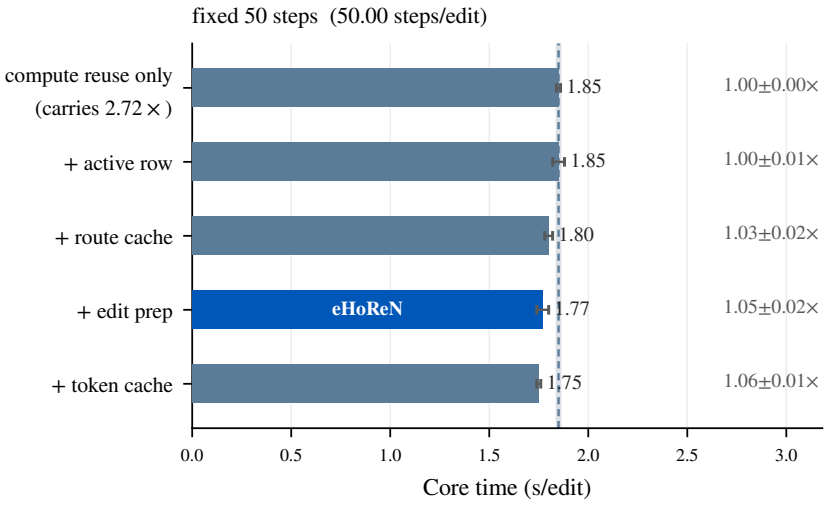


Figure 7: Editing ablation (L8/ZsRE, N=100, MP16, fixed 50 steps, three seeds): core time per edit as components are added above the compute-reuse configuration, whose 2.72x over the unoptimized path is not part of these bars. The dashed line and band mark that baseline and one seed standard deviation, so rows inside the band are not resolved from it.

Table 9: Records behind Figure 7 (L8/ZsRE, N=100, MP16, fixed 50 steps, three seeds), with Rel/Gen/Loc identical throughout ($99.70_{\pm 0.40}/87.50_{\pm 0.10}/100.00_{\pm 0.00}\%$).

Config	Active row	Route cache	Edit prep	Token cache	Core time (s)	Steps	Speedup
compute reuse only	off	off	off	off	1.85±0.01	50.00±0.00	1.00±0.00x
+ active row	on	off	off	off	1.85±0.03	50.00±0.00	1.00±0.01x
+ route cache	on	on	off	off	1.80±0.02	50.00±0.00	1.03±0.02x
eHoReN	on	on	on	off	1.77±0.03	50.00±0.00	1.05±0.02x
eHoReN + token cache	on	on	on	on	1.75±0.01	50.00±0.00	1.06±0.01x

Table 10 holds the full paired editing records with total time and peak memory; the component ablation is Figure 7 with its records in Table 9, and Figure 6 isolates the two reuse mechanisms.

Table 10: Main editing results, paired arms with full timing and memory columns. Speed ratios computed within identical model, dataset, scale, stopping rule, and environment; API time per edit. Quality differences between paired arms lie within one three-seed sd (Appendix D). Quality metrics in %. Both panels are N=1000, ZsRE, seeds 11/22/33.

	Llama-3.1-8B-Instruct		Qwen2.5-7B-Instruct	
	HoReN	eHoReN	HoReN	eHoReN
Rel.	99.49±0.09	99.50±0.11	99.56±0.26	99.37±0.29
Gen.	92.47±0.16	92.47±0.16	92.89±0.23	92.62±0.21
Loc.	98.83±0.00	98.83±0.00	98.16±0.00	98.16±0.00
O.P.	96.93±0.08	96.93±0.09	96.87±0.16	96.72±0.17
API Time / Edit	5.38±0.02	1.53±0.06	5.14±0.32	1.89±0.31
Total API Time	5375.15±22.99	1534.23±64.41	5142.01±321.03	1884.98±313.51
Speedup	1.00x	3.51±0.13x	1.00x	2.77±0.34x
Peak GPU Mem (GiB)	15.13±0.00	15.10±0.00	15.0–15.3	15.0–15.3

The three isolation arms differ only in which reuse is enabled, and all run the dynamic loss scaler at 50 attempted updates per edit with 211, 211 and 212 overflow skips in total, so their times are matched. The third arm is the configuration of the ablation baseline, and the two independent measurements agree to one percent (1.88 against 1.85 s/edit).

H.3 Space Scaling

Table 11 groups every storage class by the quantity it depends on. Only the class whose execution scope exceeded its dependency scope changes: persistent keys and values depend on the memory and are untouched, while gradients and moments depend on one row and now cost one row. The evaluation caches depend on the evaluated history and are optional, and they are where the two editors differ, since a cheap acceptance test stores retrieval inputs where full replay stores none, the storage face of the R/G ratio in Eq. 3: HoReN's cached-query check pays 221 kB per evaluated query, one pre-retrieval query per generation step, for the replay it no longer runs.

Table 11: Storage by dependency scope (L8, MP16). Two-value cells are n=10^3 and n=10^5 from Table 12; single values come from N=10^3 runs. Caches hold $\approx$ 4 kB per query for outputs, scores and traces and $\approx$ 221 kB per query for retrieval inputs, both linear in the evaluated history.

Storage class	Depends on	Standard path	eHoReN
Backbone parameters	nothing (frozen)	14.96 GiB	14.96 GiB
Persistent keys and values	memory size n	42.96 MiB / 4.20 GiB	unchanged
Gradients, Adam moments	the active value row	46.87 MiB / 4.58 GiB	48 KiB at both
Cached outputs and scores	evaluated history	none (no reuse)	11.7 MB at CP1000
Cached retrieval inputs	evaluated history	none	663 MB HoReN, 331 MB GRACE
Peak GPU while editing	—	15.13 GiB	15.10 GiB

Table 12 lists all 40 synthetic-codebook rows, including the MP16 and GRACE arms.

Table 12: Synthetic-codebook space and update cost, 36 of the 40 grid cells (five scales $\times$ two editors $\times$ two precisions $\times$ two methods); GRACE was not measured at n=50,000. Three timing rounds, mean $\pm$ sd. Space columns; timing columns are in Table 13. Theoretical training state is $12nd_v$ bytes for the dense table and $12d_v$ for the active row (gradients plus two Adam moments); the microbenchmark loads no editing model.

Nodes n	Editor	Dtype	Method	Persistent keys	Persistent values	Gradients	Adam moments	Temp. active parameter
100	HoReN	FP32	Dense table	5.47 MiB	1.56 MiB	1.56 MiB	3.12 MiB	0.00 MiB
100	HoReN	FP32	Active row	5.47 MiB	1.56 MiB	0.02 MiB	0.03 MiB	0.00 MiB
100	HoReN	MP16	Dense table	2.73 MiB	1.56 MiB	1.56 MiB	3.12 MiB	0.00 MiB
100	HoReN	MP16	Active row	2.73 MiB	1.56 MiB	0.02 MiB	0.03 MiB	0.00 MiB
1000	HoReN	FP32	Dense table	54.69 MiB	15.62 MiB	15.62 MiB	31.25 MiB	0.00 MiB
1000	HoReN	FP32	Active row	54.69 MiB	15.62 MiB	0.02 MiB	0.03 MiB	0.00 MiB
1000	HoReN	MP16	Dense table	27.34 MiB	15.62 MiB	15.62 MiB	31.25 MiB	0.00 MiB
1000	HoReN	MP16	Active row	27.34 MiB	15.62 MiB	0.02 MiB	0.03 MiB	0.00 MiB
10000	HoReN	FP32	Dense table	546.88 MiB	156.25 MiB	156.25 MiB	312.50 MiB	0.00 MiB
10000	HoReN	FP32	Active row	546.88 MiB	156.25 MiB	0.02 MiB	0.03 MiB	0.00 MiB
10000	HoReN	MP16	Dense table	273.44 MiB	156.25 MiB	156.25 MiB	312.50 MiB	0.00 MiB
10000	HoReN	MP16	Active row	273.44 MiB	156.25 MiB	0.02 MiB	0.03 MiB	0.00 MiB
50000	HoReN	FP32	Dense table	2.67 GiB	781.25 MiB	781.25 MiB	1.53 GiB	0.00 MiB
50000	HoReN	FP32	Active row	2.67 GiB	781.25 MiB	0.02 MiB	0.03 MiB	0.00 MiB
50000	HoReN	MP16	Dense table	1.33 GiB	781.25 MiB	781.25 MiB	1.53 GiB	0.00 MiB
50000	HoReN	MP16	Active row	1.33 GiB	781.25 MiB	0.02 MiB	0.03 MiB	0.00 MiB
100000	HoReN	FP32	Dense table	5.34 GiB	1.53 GiB	1.53 GiB	3.05 GiB	0.00 MiB
100000	HoReN	FP32	Active row	5.34 GiB	1.53 GiB	0.02 MiB	0.03 MiB	0.00 MiB
100000	HoReN	MP16	Dense table	2.67 GiB	1.53 GiB	1.53 GiB	3.05 GiB	0.00 MiB
100000	HoReN	MP16	Active row	2.67 GiB	1.53 GiB	0.02 MiB	0.03 MiB	0.00 MiB
100	GRACE	FP32	Dense table	5.47 MiB	1.56 MiB	1.56 MiB	3.12 MiB	0.00 MiB
100	GRACE	FP32	Active row	5.47 MiB	1.56 MiB	0.02 MiB	0.03 MiB	0.02 MiB
100	GRACE	MP16	Dense table	2.73 MiB	1.56 MiB	1.56 MiB	3.12 MiB	0.00 MiB
100	GRACE	MP16	Active row	2.73 MiB	1.56 MiB	0.02 MiB	0.03 MiB	0.02 MiB
1000	GRACE	FP32	Dense table	54.69 MiB	15.62 MiB	15.62 MiB	31.25 MiB	0.00 MiB
1000	GRACE	FP32	Active row	54.69 MiB	15.62 MiB	0.02 MiB	0.03 MiB	0.02 MiB
1000	GRACE	MP16	Dense table	27.34 MiB	15.62 MiB	15.62 MiB	31.25 MiB	0.00 MiB
1000	GRACE	MP16	Active row	27.34 MiB	15.62 MiB	0.02 MiB	0.03 MiB	0.02 MiB
10000	GRACE	FP32	Dense table	546.88 MiB	156.25 MiB	156.25 MiB	312.50 MiB	0.00 MiB
10000	GRACE	FP32	Active row	546.88 MiB	156.25 MiB	0.02 MiB	0.03 MiB	0.02 MiB
10000	GRACE	MP16	Dense table	273.44 MiB	156.25 MiB	156.25 MiB	312.50 MiB	0.00 MiB
10000	GRACE	MP16	Active row	273.44 MiB	156.25 MiB	0.02 MiB	0.03 MiB	0.02 MiB
100000	GRACE	FP32	Dense table	5.34 GiB	1.53 GiB	1.53 GiB	3.05 GiB	0.00 MiB
100000	GRACE	FP32	Active row	5.34 GiB	1.53 GiB	0.02 MiB	0.03 MiB	0.02 MiB
100000	GRACE	MP16	Dense table	2.67 GiB	1.53 GiB	1.53 GiB	3.05 GiB	0.00 MiB
100000	GRACE	MP16	Active row	2.67 GiB	1.53 GiB	0.02 MiB	0.03 MiB	0.02 MiB

Table 13: Synthetic-codebook update and retrieval time, same 40 rows as Table 12 (three timing rounds, mean ± sd).

Nodes n	Editor	Dtype	Method	Update Time / Step	Retrieval Time
100	HoReN	FP32	Dense table	1.16±0.01 ms	0.34±0.01 ms
100	HoReN	FP32	Active row	0.95±0.00 ms	0.34±0.01 ms
100	HoReN	MP16	Dense table	1.13±0.04 ms	0.52±0.01 ms
100	HoReN	MP16	Active row	0.93±0.03 ms	0.52±0.01 ms
1000	HoReN	FP32	Dense table	1.42±0.02 ms	0.56±0.01 ms
1000	HoReN	FP32	Active row	0.94±0.01 ms	0.56±0.01 ms
1000	HoReN	MP16	Dense table	1.44±0.09 ms	0.86±0.04 ms
1000	HoReN	MP16	Active row	0.91±0.04 ms	0.86±0.04 ms
10000	HoReN	FP32	Dense table	5.19±0.01 ms	2.63±0.03 ms
10000	HoReN	FP32	Active row	0.95±0.00 ms	2.63±0.03 ms
10000	HoReN	MP16	Dense table	5.12±0.10 ms	3.95±0.02 ms
10000	HoReN	MP16	Active row	0.91±0.05 ms	3.95±0.02 ms
50000	HoReN	FP32	Dense table	21.58±0.03 ms	10.66±0.03 ms
50000	HoReN	FP32	Active row	0.94±0.00 ms	10.66±0.03 ms
50000	HoReN	MP16	Dense table	21.52±0.11 ms	16.78±0.01 ms
50000	HoReN	MP16	Active row	0.91±0.07 ms	16.78±0.01 ms
100000	HoReN	FP32	Dense table	42.08±0.05 ms	25.92±0.01 ms
100000	HoReN	FP32	Active row	0.95±0.01 ms	25.92±0.01 ms
100000	HoReN	MP16	Dense table	41.91±0.07 ms	33.55±0.03 ms
100000	HoReN	MP16	Active row	0.92±0.03 ms	33.55±0.03 ms
100	GRACE	FP32	Dense table	0.85±0.07 ms	0.23±0.00 ms
100	GRACE	FP32	Active row	0.79±0.06 ms	0.23±0.00 ms
100	GRACE	MP16	Dense table	0.86±0.07 ms	0.25±0.01 ms
100	GRACE	MP16	Active row	0.79±0.07 ms	0.25±0.00 ms
1000	GRACE	FP32	Dense table	1.18±0.14 ms	0.79±0.01 ms
1000	GRACE	FP32	Active row	1.00±0.07 ms	0.78±0.01 ms
1000	GRACE	MP16	Dense table	1.28±0.07 ms	0.89±0.00 ms
1000	GRACE	MP16	Active row	1.00±0.07 ms	0.89±0.00 ms
10000	GRACE	FP32	Dense table	4.66±0.07 ms	7.43±0.02 ms
10000	GRACE	FP32	Active row	1.02±0.08 ms	8.46±0.11 ms
10000	GRACE	MP16	Dense table	4.65±0.06 ms	8.60±0.01 ms
10000	GRACE	MP16	Active row	1.00±0.07 ms	9.78±0.01 ms
100000	GRACE	FP32	Dense table	40.46±0.06 ms	77.57±0.01 ms
100000	GRACE	FP32	Active row	1.06±0.10 ms	77.79±0.04 ms
100000	GRACE	MP16	Dense table	40.40±0.01 ms	89.48±0.04 ms
100000	GRACE	MP16	Active row	0.91±0.17 ms	89.56±0.07 ms

H.4 Evaluation Detail

Table 14 records evaluation quality and output consistency per strategy.

Table 14: Evaluation quality and output consistency at CP1000 (HoReN, L8/ZsRE N=1000, three seeds; 3,000 re-test queries). Quality metrics in %. At CP100 all four strategies are zero-difference, since no historical invalidation source exists yet.

CP	Mode	Rel.	Gen.	Loc.	O.P.	Output diffs	Score diffs	Missed / reused
1000	oracle	$98.95_{\pm 0.18}$	$91.83_{\pm 0.10}$	$98.83_{\pm 0.00}$	$96.54_{\pm 0.09}$	0	0	n/a
1000	act.-only	$98.95_{\pm 0.18}$	$92.57_{\pm 0.09}$	$98.92_{\pm 0.00}$	$96.82_{\pm 0.08}$	14	11	11 / 1500
1000	act.-route	$98.95_{\pm 0.18}$	$91.83_{\pm 0.10}$	$98.83_{\pm 0.00}$	$96.54_{\pm 0.09}$	0	0	0 / 1500
1000	rand.-budget	$98.95_{\pm 0.18}$	$92.52_{\pm 0.09}$	$98.92_{\pm 0.00}$	$96.80_{\pm 0.09}$	$13.3_{\pm 0.6}$	$10.3_{\pm 0.6}$	$10.3_{\pm 0.6}$ / 1488

Table 15 follows the same runs across checkpoints, trading fidelity for speed as the full-replay anchor thins.

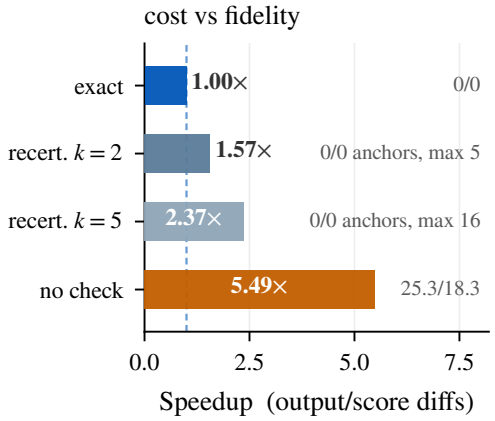


Figure 8: Evaluation cost against fidelity at CP1000, the four strategies of Table 15. Bars give cumulative speedup, the right-hand labels the output and score differences per 3,000 queries.

Table 15: Fidelity–speed spectrum (eHoReN, L8/ZsRE N=1000, early stop, checkpoints every 100, MP16, three seeds; cumulative speedup over CP100–1000 against an activation-route denominator). hybrid-k anchors a full replay every k checkpoints and reuses in between; activation-only performs no route check. Table 16 uses a different basis: HoReN, fixed step, full-oracle denominator.

Strategy	Eval. speedup	Output/score diffs (per 3,000)	Recertification
activation-route (exact)	1.00	0 / 0	every checkpoint
hybrid-k=2	1.57	0 / 0 at anchors	every 2nd checkpoint
hybrid-k=5	2.37	0 / 0 at anchors	every 5th checkpoint
activation-only (no check)	5.49	25.3 / 18.3	never; error accumulates

Intermediate-checkpoint output differences, of 3,000 queries across three seeds, grow as the anchor thins: hybrid-k=2 averages 2.8 with a maximum of 5, hybrid-k=5 averages 5.3 with a maximum of 16, and activation-only averages 10.3 with a maximum of 26 and never resets.

Figure 4(c) composes the measured cost of one checkpoint from its three parts, for both editors.

Table 16 gives the CP1000 workload with every cost column.

Table 16: Evaluation workload and time decomposition at CP1000 (HoReN, L8/ZsRE N=1000, fixed step, three seeds). Speedup is the full-oracle time over the strategy's time at the same checkpoint; every mode generates and scores the new queries U. Figure 4(c) uses a different basis: eHoReN, early stop, activation-route denominator.

Quantity	Full oracle	Activation-only	Activation-route	Random budget
New queries U	1500	1500	1500	1500
Checked	0	0	1500	0
Invalidated C	n/a	n/a	12	n/a
Retested	3000	1500	1512	1512
Retested fraction	1.00	0.50	0.50	0.50
Check (s)	0.00±0.00	0.00±0.00	485.77±2.79	0.00±0.00
Generate (s)	951.88±16.53	457.63±2.23	469.39±13.71	463.71±7.22
Score (s)	0.86±0.06	0.41±0.00	0.44±0.05	0.43±0.03
Cache/maint. (s)	2.51±0.06	1.23±0.04	2.61±0.04	1.26±0.03
Point time (s)	955.25±16.64	459.27±2.20	958.22±13.12	465.40±7.28
Cumulative (s)	1568.40±24.07	955.90±2.91	1563.15±13.25	959.30±19.32
Speedup	1.00±0.00x	2.08±0.03x	1.00±0.00x	2.05±0.01x

H.5 CACHED-QUERY CERTIFICATION ON HOREN

Table 18 reports the cached-query implementation of the reuse criterion on the same edit streams as Table 16, against a full oracle that regenerates and rescores every query. Runs are L8/ZsRE in MP16 with 50 fixed steps, seeds 11/22/33 at $N{=}1000$ on the checkpoint grid $1, 10, 30, 100, 500, 1000$; the rows marked † ran on a dedicated device with one strategy per process. The cheapest conceivable criterion needs no query cache: invalidate a historical result when the node it depended on appears in the write set of the interval, which follows from the edit log alone and needs only the dependency record, not the 221 kB of retrieval inputs per query. Recomputing it offline from the per-edit logs and the CP500 dependency records of the runs in Tables 18 and 19 shows why the retrieval inputs are worth their space. On ZsRE it flags none of the 1,500 candidates and therefore reuses all 12 results whose route changed, of which 12 differ in output and 9 in score; on WikiBigEdit 119 of the 150 invalidations arise without any write to the node the historical result depended on, so it would reuse at least that many invalid results. Cached-query checking reuses slightly less and leaves none. The reason is structural: only 3 of the 500 edits in the ZsRE interval re-select an existing node and 14 of the WikiBigEdit ones do, while 9 of the 12 ZsRE invalidations and 97 of the 150 WikiBigEdit ones come from a new node winning the route, 3 and 22 from the gate flipping, and the remaining 31 WikiBigEdit ones from a re-selected node whose value version advanced, identical across the three seeds. Neither mode touches the node the old result depended on, and a query whose gate was previously closed has no dependency to test at all. On ZsRE the criterion was also run as a third strategy alongside the full oracle and cached-query checking, three seeds in one process per dedicated device (Table 17). It costs under 1 ms per checkpoint against 7.0 s for the cached-query check and stores no cache, and it reaches 2.06x per checkpoint and 1.51x cumulative where cached-query checking in the same runs reaches 1.82x and 1.42x. Every seed reuses the same 12 changed results, 9 of them score-changing, and the strategy reports O.P. 0.24 points above the oracle. The WikiBigEdit verdicts above are recomputed offline and carry no timing. Cached-query checking reuses 45.0% of the WikiBigEdit queries and full replay 43.8%. On ZsRE the cost model, given the measured CP1000 check cost, puts the cumulative gain at 1.44x; the measured 1.43x is within 1% of it.

Table 17: Write-set criterion against cached-query checking (L8/ZsRE, MP16, $N{=}1000$, CP500 and CP1000; full oracle, cached-query check and write-set check run sequentially in one process per seed, one seed per device). Invalidations are out of 1,500 candidates; speedups are wall-clock over the full oracle at CP1000 and cumulative over both checkpoints. Diffs count reused results whose output or score differs from the oracle.

Seed	Strategy	O.P. (%)	Invalid.	Check (s)	Cache (MB)	Speedup	Cumul.
11	Cached query	96.57	12	7.29	662.7	1.81	1.42
11	Write set	96.81	0	0.00	0	2.07	1.51
22	Cached query	96.60	12	7.47	662.7	1.81	1.42
22	Write set	96.84	0	0.00	0	2.08	1.52
33	Cached query	96.43	12	6.37	662.7	1.84	1.43
33	Write set	96.67	0	0.00	0	2.04	1.51
Cached query, output / score diffs		0 / 0 in every seed					
Write set, output / score diffs		12 / 9 in every seed					

Figure 9 reads the benefit condition as a curve: the gain a deployment can expect follows from its checkpoint density and its invalidation rate, and the cached-query check is cheap enough that the reuse opportunity, not the check, sets the bound. The pre-retrieval query cache scales with the evaluated history rather than the distribution: 663 MB over 3,000 ZsRE queries and 682 MB over the same number on WikiBigEdit, 221 and 227 kB per query. Diffs count per-query output, score and realized-reuse differences against the oracle. The route changes are events, not distinct queries: a query appearing at several checkpoints is counted once per appearance.

Output, score and realized-reuse differences against the oracle are $0/0/0$ for full replay and cached-query checking in every seed. The $N{=}100$ prototype (seed 42, checkpoints at 50 and 100) shows the same separation at CP100: both checkers find the single route change with $0/0/0$ differences, at 39.24 s for full replay against 0.40 s for cached-query checking, while no check misses it ($1/1/1$).

Its totals were measured in one process and are not comparable across strategies, so only check times are reported for it.

Table 18: Cached-query checking on the evaluation workload of Table 16 (L8/ZsRE, MP16, N=1000, seeds 11/22/33, checkpoint grid $1, 10, 30, 100, 500, 1000$). Verdicts and reuse counts match that table cell for cell, and the check costs 6.78 s on a dedicated device where full replay costs 485.77 s. Rows marked † are a three-seed rerun of the 500–1000 grid on a dedicated device, one strategy per process, and are the basis for the CP1000 check and end-to-end times quoted in the text; the unmarked CP1000 rows ran while the devices were shared, which inflates both. Verdicts and reuse counts are identical in every case. Cache bytes are the measured pre-retrieval query cache, one entry per generation step.

Checkpoint	Seed	O.P. (%)	Checked	Invalid.	Reused	Check (s)	Cache (MB)	Speedup
CP100	11	96.03	90	0	90	0.24	60.7	1.31
CP100	22	96.10	90	0	90	0.25	60.7	1.31
CP100	33	95.90	90	0	90	0.27	60.7	1.30
CP500	11	—	300	2	298	1.06	339.5	1.10
CP500	22	—	300	2	298	1.02	339.5	1.33
CP500	33	—	300	2	298	1.11	339.5	1.18
CP1000	11	96.57	1500	12	1488	11.57	662.7	shared
CP1000	22	96.60	1500	12	1488	11.62	662.7	shared
CP1000	33	96.43	1500	12	1488	8.10	662.7	shared
CP1000†	11	96.57	1500	12	1488	7.07	662.7	1.90
CP1000†	22	96.60	1500	12	1488	6.35	662.7	1.87
CP1000†	33	96.43	1500	12	1488	6.91	662.7	1.89

In Table 18, output, score and realized-reuse differences against the full oracle are zero at every checkpoint and seed. The CP500 ratio of means is 1.20x with per-seed values 1.10, 1.33 and 1.18; CP100 carries no invalidations, so its reuse is exact by construction of the grid.

Table 19: Certified reuse on a second distribution (Q7/WikiBigEdit, MP16, N=1000, seeds 11/22/33, checkpoint grid $1, 10, 30, 100, 500, 1000$; CP1000 shown). O.P. uses rewrite, rephrase and locality. Reuse and invalidation counts without a spread are identical across the three seeds under the fixed request order, not a statistic over seeds. The four strategies run sequentially in one process, so a later strategy starts warmer, and the speedup column states what this protocol measured rather than a device-level guarantee: two reruns with two strategies per process read 1.61x and 1.60x against the 1.87x here, and a three-seed rerun timing one strategy per process on a dedicated device over the CP500–CP1000 grid measures $1.73_{\pm 0.02}$x per checkpoint and $1.37_{\pm 0.03}$x cumulatively, with the same verdicts and counts throughout. Both checkers are sound against the oracle here. Cached-query checking applies the narrower route predicate (gate, key and value version from stored pre-injection queries) and flags 150 of 1,500 candidates, while full replay also compares next tokens and flags 186 (184 in a rerun with per-query records, whose 34 extra flags are all next-token inequalities with unchanged output and score); the risk arm reuses the most and pays for it in unregenerated changes.

Strategy	O.P. (%)	Out./score diffs	Reused	Invalidated	Speedup	Cumulative
Full oracle	$71.90_{\pm 0.02}$	—	—	—	1.00x	1.00x
Full replay	$71.90_{\pm 0.02}$	0 / 0	$43.80_{\pm 0.07}$%	$12.40_{\pm 0.13}$%	$0.96_{\pm 0.00}$x	$0.97_{\pm 0.00}$x
Cached query	$71.90_{\pm 0.02}$	0 / 0	45.00%	10.00%	$1.87_{\pm 0.00}$x	$1.56_{\pm 0.00}$x
No check	$72.36_{\pm 0.02}$	102 / 36	48.57%	—	$2.03_{\pm 0.01}$x	$1.64_{\pm 0.00}$x

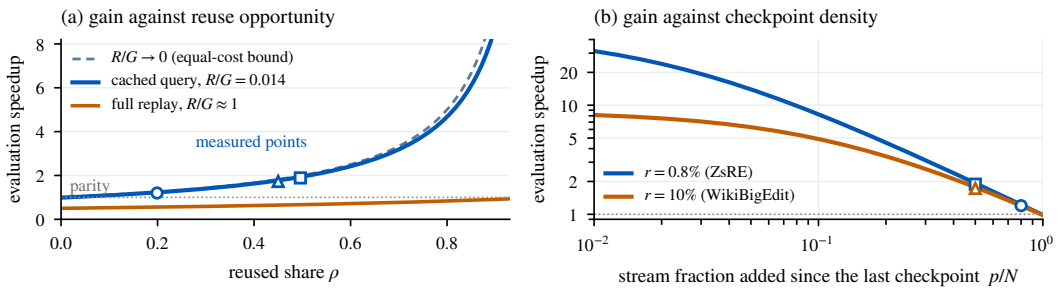


Figure 9: Where the evaluation gain comes from, and what bounds it. (a) per-checkpoint speedup against the reused share ρ: for a check whose cost vanishes against generation ($R/G \to 0$, the equal-cost bound $1/(1-\rho)$), for cached-query checking at its measured $R/G = 0.014$ (4.5 ms per check against 324 ms per regenerated query), and for full replay, whose check costs what generation costs and which therefore stays below parity at every ρ. (b) the same expression against checkpoint density, with the protocol fixing $\rho = 1 - p/N - (1 - p/N)r$ at candidate invalidation rate r; the expression assumes a fixed number of queries per edit, so it is unchanged when that number scales uniformly and moves only if the mix of rephrase and locality queries drifts along the stream. Markers are the three dedicated-device measurements of Tables 18 and 19 (ZsRE CP500 and CP1000, WikiBigEdit CP1000); they fall 3–4% below the expression, which omits scoring and cache bookkeeping. Neither curve is fitted: R/G and r are measured quantities.

Table 19 repeats the comparison on WikiBigEdit. An $N{=}100$ smoke run at seeds 11 reproduced the same pattern before the formal streams, with zero differences for both checkers and a non-empty query cache.

H.6 GRACE Transfer

GRACE realizes the same abstraction with Euclidean nearest-neighbor retrieval, per-node radius acceptance, and representation replacement. Its retrieval input precedes injection and routes are fixed after node preparation, so the premises of Propositions 1 and 2 hold and the editing-side techniques apply unchanged; acceptance depends only on retrieval inputs, keys, and radii, so checking takes the cached-query path. GRACE also exhibits invalidation modes HoReN lacks, and Proposition 3 covers them by definition. In MH-HoReN the heads partition the key dimension for scoring only: the head-wise match scores are averaged into one score per node, a single argmax selects one node, and one value row is injected and trained, so the mutable state of an edit is the same single active row as in HoReN and Propositions 1–3 apply without modification. No head can select a different row from another, and no per-head write-back exists.

The invalidation modes GRACE adds over HoReN: radius expansion lets a historical query start accepting a replacement, while radius contraction and competition from newly inserted nodes remove or redirect an acceptance. All three alter the effective injection sequence without touching any old value, which is what the criterion of Proposition 3 tests.

Excluding the Check column while keeping its verdicts, activation-route's generation-reuse total is 472.4 s against full-oracle's 955.2 s at CP1000, a route-validity-oracle upper bound of $2.02\times$. The figure is counterfactual: the check is what certifies the zero difference, so the bound states what a cheaper checker could recover, not a measured speedup.

Full column sets for the GRACE transfer follow, quality first in Table 20.

Table 20: GRACE editing efficiency and quality, full version of Table 2: quality columns in % (ZsRE, seeds 11/22/33, GRACE's fixed 100 steps; GRACE + ours allocates active-row state with route and preparation reuse). Ratios are within identical model, data, scale and seed on the paired API basis, not comparable with the pure-edit basis of Table 1.

Backbone	N	Method	Rel.	Gen.	Loc.
L8	100	GRACE	100.00±0.00	5.90±0.00	100.00±0.00
L8	100	GRACE + ours	100.00±0.00	5.90±0.00	100.00±0.00
Q7	100	GRACE	100.00±0.00	1.20±0.00	100.00±0.00
Q7	100	GRACE + ours	100.00±0.00	1.20±0.00	100.00±0.00
L8	1000	GRACE	100.00±0.00	5.01±0.00	100.00±0.00
L8	1000	GRACE + ours	100.00±0.00	5.01±0.00	100.00±0.00
Q7	1000	GRACE	100.00±0.00	1.92±0.00	100.00±0.00
Q7	1000	GRACE + ours	100.00±0.00	1.92±0.00	100.00±0.00

Table 21 carries the cost and memory columns of the same runs.

Table 21: Cost columns of Table 20: API seconds per edit and peak memory (allocated / reserved).

Backbone	N	Method	Mean	P50	P95	Peak (GiB)	Speedup
L8	100	GRACE	5.00±0.20	4.98±0.18	5.53±0.45	15.11 / 15.27	1.00x
L8	100	GRACE + ours	2.43±0.08	2.50±0.07	2.67±0.04	15.10 / 15.25	2.06±0.15x
Q7	100	GRACE	5.32±0.15	5.30±0.16	5.88±0.09	14.40 / 14.54	1.00x
Q7	100	GRACE + ours	3.15±0.12	3.20±0.12	3.56±0.06	14.39 / 14.52	1.69±0.11x
L8	1000	GRACE	4.98±0.04	4.94±0.05	5.49±0.19	15.21 / 15.79	1.00x
L8	1000	GRACE + ours	2.54±0.04	2.54±0.03	2.84±0.09	15.18 / 15.77	1.96±0.05x
Q7	1000	GRACE	5.19±0.16	5.14±0.11	5.80±0.33	14.53 / 15.92	1.00x
Q7	1000	GRACE + ours	3.13±0.38	3.15±0.43	3.60±0.11	14.50 / 15.92	1.68±0.22x

Table 22 compares full evaluation with the cached-query check, including the cache sizes each strategy holds. At CP1000 GRACE has $1{,}500$ historical queries, of which 900 carry an accepting route dependency that can be certified; the remaining 600 have none and are regenerated together with the $1{,}500$ new queries, which gives the 0.70 re-test fraction and a reused share $\rho{=}0.30$, so the equal-cost approximation $1/(1-\rho)$ puts the gain at 1.43x against the measured 1.46x.

Table 22: GRACE full evaluation vs. cached-query checking (per checkpoint; quality in %, re-aggregated from all valid queries; time covers preparation, check, transfer, generation, scoring, and cache maintenance).

Quantity	Full oracle	Cached-query route check
Rel.	100.00±0.00	100.00±0.00
Gen.	5.01±0.00	5.01±0.00
Loc.	100.00±0.00	100.00±0.00
Output / score diffs	0 / 0	0 / 0
Retest Fraction	1.00±0.00	0.70±0.00
Backbone Forwards	11557±0	8086±0
Distance Calls	11557±0	11557±0
Check + Transfer	0.00±0.00	3.96±0.13
CPU / GPU query cache	0 / 0	331.36 MB / 0
Total Wall Time	480.66±15.07	328.94±7.94
Speedup	1.00±0.00x	1.46±0.06x

Tuned GRACE. The GRACE rows above use the reference hyperparameters ($\varepsilon{=}1$, layer 29). A second configuration was selected on a held-out ZsRE slice (requests 1000–1100, disjoint from the formal stream 0–1000) over $\varepsilon \in \{1, 2, 4, 8, 16\}$ and layers $\{23, 27, 29, 31\}$ with fixed learning rate 1.0 and 100 steps, by filtering to Rel. $\geq 98\%$ and Loc. $\geq 95\%$ with finite loss and then maximizing Gen.; the selection gave $\varepsilon{=}4$ at layer 29. Table 23 reports the formal three-seed runs at $N{=}1000$. The same selection on Q7/WikiBigEdit (validation $N{=}100$, 22 candidates, 10 passing the filter) found no configuration that raises Generalization while keeping Reliability and Locality: the se-

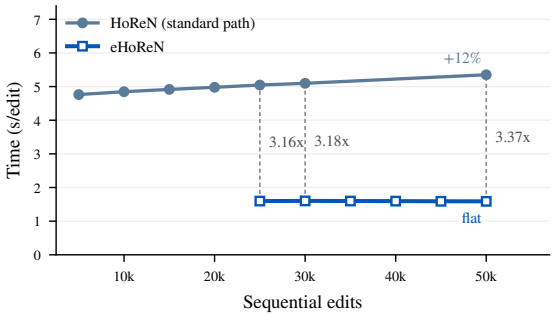


Figure 10: Per-edit cost over the 50K stream (L8/ZsRE, seed 42, checkpoint-basis means, not comparable with the per-edit times of Table 1). The standard path spends more per edit as the memory grows while eHoReN does not, so the ratio widens with N; eHoReN was evaluated from 25K onward.

lected ε=1 at layer 24 has Gen. 0.00%, and every candidate with higher Gen. (10.1% at best) fails the Rel. or Loc. filter, so no formal Q7 pair was run and the reference configuration stands for that panel. Each editing arm ran in its own process; the earlier pair ran both arms in one process, and the two optimized-path times (2.54 against 1.14 s/edit) are not compared, only the within-pair ratios. The four evaluation strategies ran sequentially in one process, as in Table 22.

Table 23: GRACE with tuned hyperparameters (ε=4, layer 29) on L8/ZsRE at N=1000, seeds 11/22/33, MP16. Top: paired editing, per-edit time from the harness's internal timer, with zero per-query output and score differences between the arms at CP1000 over 3,000 queries; reuse counts are out of 3,000. Bottom: evaluation at CP1000 against the full oracle; wall time is executed wall-clock, and the cached-query check stores 316 MiB of retrieval inputs.

Arm	Rel.	Gen.	Loc.	O.P.	Time/edit (s)	Speedup
GRACE (tuned)	97.59±0.01	13.90±0.01	100.00±0.00	70.50±0.00	4.80±0.01	1.00x
GRACE (tuned) + ours	97.59±0.01	13.90±0.01	100.00±0.00	70.50±0.00	1.14±0.01	4.21±0.05x
Strategy	**O.P.**	**Reused**	**Output diffs**	**Score diffs**	**Wall (s)**	**Speedup**
Full oracle	70.50	0	0	0	501.40±10.50	1.00x
Full replay	70.50	1471	0	0	500.00±10.30	1.00±0.00x
Cached-query check	70.50	1471	0	0	253.20±4.90	1.98±0.01x
No check	70.46	1485	14	13	243.90±5.30	2.06±0.01x

H.7 Long-run Archive

Per-checkpoint records along the 50K streams archive per-edit and inference times, the aggregate metrics, and the editor parameter count, which grows from 2.05×10^7 at 5K to 1.94×10^8 at 50K. That count is an exact multiple of 4096 at every checkpoint, so the node trajectory reads off it directly: 4,993 nodes at 5K edits, 24,296 at 25K and 47,338 at 50K, that is n_t/t from 1.00 to 0.95, with 0.91 new nodes per edit over the last interval. Stream finals report the terminal metrics. Both 50K arms are closed and end together: Rel/Gen/Loc/O.P. $= 99.33/85.97/91.38/92.23\%$ for both HoReN and eHoReN, the O.P. values 0.0001 apart, with the arms within 0.003 points on every metric at the 25K and 30K checkpoints. Per-edit cost separates them throughout (Figure 10): $4.76 \rightarrow 5.35$ s/edit for the standard path against $1.60 \rightarrow 1.59$ for eHoReN, a ratio of $3.16\times$ at 25K and $3.37\times$ at 50K.

I Positioning

Three adjacent lines of work optimize different objects, and this paper's position differs from each. **Editing methods** optimize editing quality: interference, retention, and capacity. This paper holds the method and its quality fixed and restructures the execution and verification cost of a given memory structure. **Computation-reuse techniques** optimize cross-request caching, kernel implementations, and the number of trainable parameters. This paper's reuse operates inside the edit loop, aligns state allocation to the single-row dependency, and carries equivalence proofs for each step. **Incremental computation** provides result-reuse semantics for general programs over mutable state. This paper instantiates that semantics to generated sequences with indirect routing dependencies, and makes verification cost itself a term in the benefit condition. The combination, historical evaluation results as certified reusable products in lifelong editing, is the position no prior line occupies.