LadderEdit: Edit-Level Residual Compression for Memory-Efficient Lifelong Editing of LLMs

Anonymous ACL submission

Abstract

Lifelong editing of LLMs requires storing thousands of edits after acquisition. A widely used family of approaches attaches one LoRA adapter per edit, which preserves behavior but grows linearly in storage. To address this challenge, we propose LadderEdit, a method that compresses each LoRA adapter after it is acquired. Each edit is first stored at low rank as a cheap sketch. We then check whether this sketch still satisfies the rewrite, generalization, and locality contract on probe prompts. Edits that pass keep the sketch; those that fail are promoted to a higher rank along a ladder until the contract is met. Because every edit retains some representation, coverage is maintained, and only hard edits consume more rank. Across ZsRE, CounterFact, and WikiBigEdit benchmarks on LLaMA-3-8B, Mistral-7B, and Qwen2.5-7B, LadderEdit tracks exact LoRA storage at $5.2\times$ less memory and remains effective at $50{,}000$ sequential edits.

1 Introduction

Lifelong model editing asks a deployed large language model to absorb a stream of factual corrections and knowledge updates without full retraining. This setting is increasingly important: world knowledge changes, model errors are discovered after deployment, and domain-specific assistants require controlled updates over time. A useful lifelong editor must therefore satisfy three requirements simultaneously. It must remain plastic enough to incorporate new edits, stable enough to preserve previous edits and unrelated knowledge, and efficient enough to serve after thousands of updates.

Existing approaches expose a fundamental storage-plasticity tension. Shared-parameter and constant-memory editors bound the deployed memory footprint, but every new edit competes with previous edits for limited capacity (Hartvigsen et al., 2023; Wang et al., 2025). As the stream grows, such methods can preserve locality while losing edit plasticity: the model avoids damaging unrelated knowledge, but new edits may fail to generalize beyond the rewrite prompt. Retrieval- and routing-based editors reduce direct parameter interference but introduce a separate edit-selection problem and can fail when a query is semantically close to, but not exactly matched with, the stored edit (Li et al., 2024a; Wang and Li, 2024b; Zhang et al., 2023; Wang and Li, 2024a). Direct parameter-editing methods avoid explicit per-edit storage, yet repeated updates can interfere destructively over long edit streams (Meng et al., 2023, 2022; De Cao et al., 2021; Dai et al., 2022; Fang et al., 2025; Wang et al., 2023; Mitchell et al., 2022a).

An alternative is to store edit-local update memories. Each edit is written into an isolated update representation, so later edits do not overwrite earlier ones (Hu et al., 2022; Yu et al., 2023). This design preserves edit-specific plasticity and provides a clean full-storage reference: if the correct edit memory is selected, the system can recover the behavior of the original update. The drawback is persistent storage. If every edit is retained at full resolution, memory grows linearly with the number of edits. Thus, the bottleneck in long-horizon editing is not only how to acquire an edit but also how much of each acquired edit must remain stored.

To address this dilemma, we reframe lifelong edit storage as an edit-level resolution allocation problem. The key question is not simply which edits should be kept or dropped but how much representational resolution each edit needs to preserve its behavioral contract. This distinction matters because keep/drop caching couples memory reduction to edit absence: once an edit is dropped, it has no representation and cannot support rewrite, generalization, or locality. In contrast, a resolution-allocation policy can preserve coverage for every edit through a cheap representation, while spending additional memory only on edits that require higher

fidelity.

We propose LADDEREDIT, a storage-layer method based on this coverage-before-fidelity principle. Each incoming edit is first acquired as a full-resolution update memory; in our implementation, this update memory is an exact LoRA update. LadderEdit is a two-stage controller. A *spectral predictor* scores each candidate rank using the singular-value tail of the update together with the edit's behavioral slack, and proposes the cheapest rank $\hat{r}_i$ whose score falls below a calibrated threshold; this proposal requires only one SVD per edit and no behavioral evaluation. A *behavioral audit* then verifies the proposed sketch against rewrite, generalization, and locality probes. If the audit passes, the edit is stored at $\hat{r}_i$; if it fails, the audit promotes the edit along the spectrally-ordered ladder until a passing rung is found. This design avoids exhaustive auditing across all rungs and gives every edit coverage through its predicted rung while spending additional resolution only where the audit demands it.

We validate on ZsRE (Levy et al., 2017), CounterFact (CF) (Meng et al., 2022, 2023), and WikiBigEdit (Thede et al., 2025) using LLaMA-3-8B (Grattafiori et al., 2024), Mistral-7B (Jiang et al., 2023), and Qwen2.5-7B (**?**). Figure 1 shows LadderEdit tracks the full-resolution reference at a fraction of the memory, and budget-matched frontiers confirm that residual-ladder allocation substantially outperforms exact-cache: preserving coverage before fidelity is the better storage primitive for scalable lifelong editing.

Our contributions are as follows: *i)* We formulate lifelong editing storage as a resolution-allocation problem, separating edit acquisition, selection, and persistent representation. *ii)* We introduce LadderEdit, a residual-ladder controller that covers every edit with a low-rank sketch and restores detail only when behavioral audits demand it. *iii)* We identify *coverage-before-fidelity* as the principle for scalable edit memory, nearly doubling utility over exact-cache at matched budget. *iv* We validate LadderEdit on ZsRE, CF, and WikiBigEdit, tracking full-resolution memory at $5.2\times$ less storage and remaining effective at 50,000 edits.

2 Related Work

Model editing and lifelong editing. Model editing methods update a model's behavior without full retraining, either by directly modifying parameters (Sinitsin et al., 2020; De Cao et al., 2021; Meng et al., 2022, 2023), learning editor networks (Mitchell et al., 2022a), or routing edited cases through external/contextual memories (Mitchell et al., 2022b; Zheng et al., 2023). These methods establish the basic reliability, generalization, and locality criteria, but most are not designed for long edit streams where thousands of updates must coexist. Lifelong editing methods address this sequential setting through key-value memories, side memories, null-space constraints, or adapter/MoE-based mechanisms (Hartvigsen et al., 2023; Wang et al., 2024a; Fang et al., 2025; Wang et al., 2025; Yu et al., 2023; Li et al., 2024a; Wang and Li, 2024b,a). However, they still face a storage-plasticity tension: shared or constant-memory systems can lose plasticity as capacity saturates, while edit-local memories preserve behavior but grow with the number of edits. LadderEdit targets this storage layer directly.

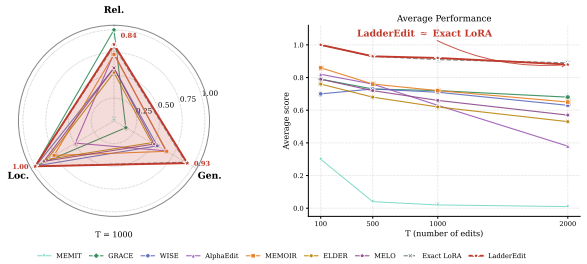


Figure 1: Performance comparison of LadderEdit, exact LoRA, and baseline methods across sequential editing tasks. LadderEdit usually tracks exact LoRA closely while reducing persistent memory, with small losses in some compressed settings.

Low-rank adaptation and edit-memory storage. LoRA and its variants show that low-rank updates provide an efficient representation for adapting large models (Hu et al., 2022; Dettmers et al., 2023; Liu et al., 2024; Lialin et al., 2024). Adaptive-rank methods such as AdaLoRA (Zhang et al., 2023) allocate rank across layers or modules during training, while serving systems such as S-LoRA (Sheng et al., 2023) study how to deploy many adapters efficiently. LadderEdit addresses a different problem: after an edit has already been acquired, how much of its update memory must be persistently stored? The unit of allocation is an edit rather than a layer, and the criterion is behavioral contract satisfaction rather than gradient-derived importance. Thus LadderEdit is not ordinary LoRA compression or training-time rank allocation; it is an edit-

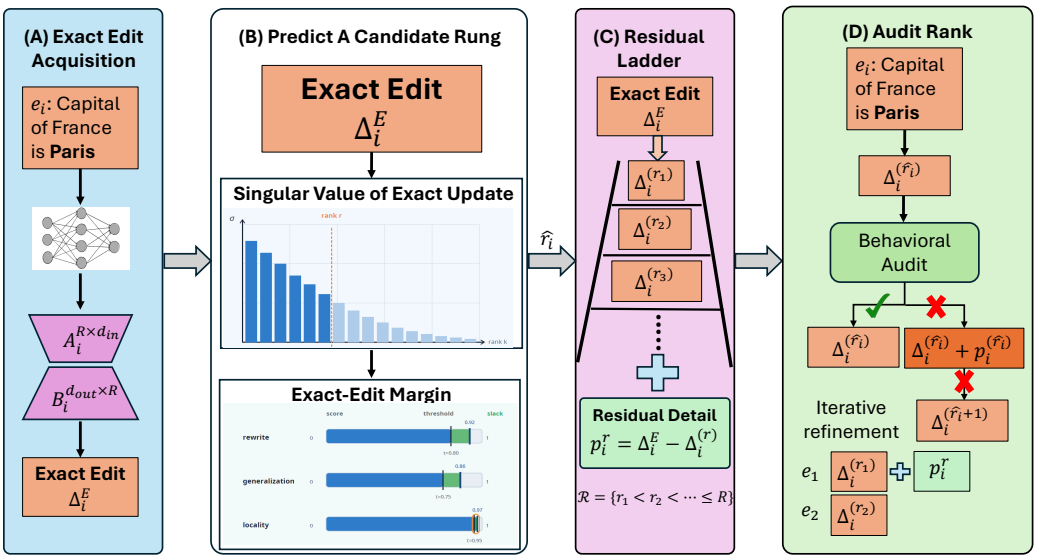


Figure 2: LadderEdit insertion pipeline. (A) Fit an exact LoRA update for the incoming edit. (B) **Predict** a candidate rank $\hat{r}_i$ from the update spectrum and exact-edit margin (one SVD, no behavioral cost). (C) Decompose the edit into ladder rungs: low-rank sketches at increasing rank. (D) **Audit** the predicted sketch against rewrite, generalization, and locality. If the audit passes, store at $\hat{r}_i$; otherwise promote along the spectral ladder until a passing rung is found.

level resolution-allocation method that keeps every edit covered by a sketch and promotes to higher rank only when needed.

Evaluation under compression. Recent surveys and evaluations emphasize that edited models should be judged not only by rewrite success but also by generalization, locality, long-form behavior, and degradation over many edits (Yao et al., 2023; Zhang et al., 2024; Wang et al., 2024b; Rosati et al., 2024; Li et al., 2024b; Zheng et al., 2024; Wu et al., 2024). LadderEdit follows this view at the storage level: spectral compressibility proposes candidate rungs, but the final decision is made by behavioral auditing over rewrite, generalization, and locality. This is the central difference from exact-cache allocation, which reduces memory by dropping edits rather than by allocating resolution.

LadderEdit's formulation differs from ordinary LoRA compression and adaptive rank training. LadderEdit does not learn a better adapter at acquisition time and does not solve retrieval or task identification; it operates after the edit has been acquired, inheriting the selection mechanism of the full-resolution memory bank, and asks a storage-layer question: once an edit is selected, can the system return a cheaper representation without violating the measured edit contract? The advantage comes from changing the memory primitive. Exact-cache allocation saves memory by storing a subset of edits exactly and dropping the rest, reducing coverage. LadderEdit first gives every edit coverage through a sketch, then allocates fidelity through additional rank only where the audit shows it is needed. The two policies fail differently under compression: exact-cache fails by absence, whereas LadderEdit degrades by resolution. A focused comparison that distinguishes LadderEdit from training-time rank allocation (Zhang et al., 2023), multi-adapter serving (Sheng et al., 2023), and adapter merging is provided in Appendix D.

3 Method

3.1 Problem Setting

We consider lifelong editing for a pretrained LLM f_θ, where edits arrive as a sequence $\{e_i\}_{i=1}^N$. After each insertion, the system must answer the edited prompt correctly, generalize the edit to related prompts, and preserve unrelated knowledge. We refer to these requirements as reliability, generalization, and locality. The scaling challenge is not only to satisfy this behavioral contract for one edit, but to keep satisfying it as N grows.

We instantiate each edit with a LoRA-style update. Let Δ_i^E denote the exact update produced by the edit writer for edit e_i. Two extremes bracket the space of persistent representations: a full-resolution edit-local memory bank stores every Δ_i^E exactly, scaling linearly as NP_E where P_E is the cost of one exact edit; a shared or constant-memory editor folds all edits into a fixed footprint,

but plasticity degrades as later edits compete with earlier ones for finite capacity. Neither extreme matches the empirical structure of an edit stream: across $N = 2{,}000$ edits on both ZsRE and CounterFact, roughly 80% of edits are behaviorally sketch-sufficient at rank 1, a $\sim 19\%$ hard tail requires rank 2 or higher, and a smaller $\sim 5\%$ population is *compression-regularized*, the rank-1 sketch satisfies the behavioral contract that the exact update fails (Appendix A.1). LadderEdit occupies the intermediate regime that this distribution implies: every edit retains its own representation, but that representation may be a compressed rung rather than the full exact adapter. Table 4 reports the population structure of the edit stream that motivates this design.

The system rests on three commitments. It is *exact first*: each edit is acquired at full resolution by the standard writer, so the acquired update is preserved without loss. It is *retrieval agnostic*: it inherits whatever selection mechanism the deployment uses, with the storage gain preserved under a learned sentence encoder retriever (Appendix A). It is *contract grounded*: at insertion, a behavioral audit certifies that the returned representation satisfies the same rewrite, generalization, and locality test as Δ_i^E. At inference, retrieval selects the edit and LadderEdit returns the cheapest audited rung; the acquisition cost of Δ_i^E is paid once, and the persistent cost is whatever the audit permits. LadderEdit decides which directions of Δ_i^E to retain and composes with quantization, which acts on bit width (Appendix A.4).

3.2 Residual-Ladder Allocation

LadderEdit is a residual-ladder allocation mechanism for scaling edit-local update memories in lifelong editing. It keeps the isolation and plasticity of per-edit adapters, but stores most edits at lower resolution. The method has three steps: acquire an exact update, compile that update into candidate rungs, and audit the cheapest safe rung. Figure 2 summarizes the full insertion-time pipeline, and Algorithm 1 gives the controller.

For an incoming edit e_i, let Δ_i^E denote the exact LoRA-style update acquired by the edit writer. LadderEdit constructs a rank-r sketch:

$$\Delta_i^{(r)} = \Pi_r(\Delta_i^E), \tag{1}$$

We use the rank menu $\mathcal{R} = \{1, 2, \ldots, R_{\max}\}$, matching the same acquisition rank as $R_{\max}$ from the exact LoRA update.

where Π_r truncates the singular directions of the effective LoRA update. The omitted detail is the residual:

$$\rho_i^{(r)} = \Delta_i^E - \Delta_i^{(r)}. \tag{2}$$

The deployed edit is one rung of the ladder:

$$\Delta_i = \Delta_i^{(r_i)}. \tag{3}$$

When $r_i = R_{\max}$, the stored update equals the exact LoRA edit. The key difference from exact-cache allocation is that every edit keeps a representation, and memory is spent on resolution rather than on a binary keep/drop decision.

Rank selection starts from a cheap spectral proposal. Let $T_i(r)$ be the normalized spectral tail after rank r. We define the exact-edit margin as the minimum behavioral slack of the full-resolution edit:

$$\mu_i^E = \min\{R_i^E - \tau_R,\ G_i^E - \tau_G,\ L_i^E - \tau_L\}. \tag{4}$$

Positive margin means the acquired edit satisfies all three measured constraints, and larger margin indicates more tolerance for compression.

LadderEdit scores:

$$S_i(r) = \frac{T_i(r)}{[\mu_i^E]_+ + \epsilon}, \tag{5}$$

where $\epsilon > 0$ prevents unstable division. Lower scores indicate that little spectral mass remains outside the sketch relative to the exact edit's behavioral slack. The cheapest rank whose score falls below a calibrated threshold τ is the predicted rank $\hat{r}_i$:

$$\hat{r}_i = \min\{r \in \mathcal{R} : S_i(r) \le \tau\}. \tag{6}$$

The spectral score is a proposal, not a safety proof; it selects *which single rung* the audit will inspect first, so the audit sees one candidate per edit instead of $|\mathcal{R}|$ candidates.

Behavioral auditing remains the final gate. For a candidate stored representation m, let R_i^m, G_i^m, and L_i^m denote rewrite, generalization, and locality scores. We define the soft contract gap:

$$\begin{aligned} H_i^m = {} & w_R[\tau_R - R_i^m]_+ + w_G[\tau_G - G_i^m]_+ \\ & + w_L[\tau_L - L_i^m]_+. \end{aligned} \tag{7}$$

The audit score in Eq. 7 uses weights $w_R = 1.0$, $w_G = 0.8$, and $w_L = 1.2$. Locality receives the highest weight because locality failures are catastrophic and irreversible, while reliability and generalization failures trigger rank promotion and recover within the same edit. We swept w_L over

$\{0.8, 1.0, 1.2, 1.5, 2.0\}$ on the validation split and Avg. stayed within 0.01 of the default for $w_L \in [1.0, 1.5]$ and dropped by 0.02 and 0.03 at the endpoints 0.8 and 2.0 respectively. The default $w_L = 1.2$ sits at the midpoint of the stable region. The weights w_R and w_G follow the standard LoRA loss-weighting ratio. We demonstrate more details in Appendix A.3.

Downgrade policy. Algorithm 1 includes budget-based downgrading. When the audit succeeds at rank r, the controller checks whether the rank-$(r-1)$ approximation $U_{:,1:r-1}\Sigma_{1:r-1,1:r-1}V^T_{1:r-1,:}$ captures more than 92% of the Frobenius energy of the full LoRA update. If so, the controller commits the rank-$(r-1)$ sketch and re-runs the audit on the downgraded sketch, and if the downgrade also passes the audit, it is kept. Downgrade attempts are bounded at one per edit to keep insertion cost bounded. Downgrades occur for 3.1% of edits at $T = 1{,}000$ on LLaMA-3-8B / ZsRE and save approximately 0.6 GB of persistent memory at $T = 10{,}000$. The 0.92 Frobenius threshold was selected on the validation split, and varying it from 0.88 to 0.95 changes Avg. by less than 0.005.

If $H_i^m = 0$, the candidate satisfies the edit contract. If the audit at $\hat{r}_i$ fails ($H_i^{(\hat{r}_i)} > 0$), LadderEdit re-audits at the next rung along the spectrally-ordered ladder $\hat{r}_i + 1, \hat{r}_i + 2, \ldots$ until the contract is met or $R_{\max}$ is reached. In practice the predicted rung passes on the first audit for the majority of edits (Appendix A.5), so the expected number of audits per edit is close to one.

Under a memory budget, higher-rank promotion is allocated where it removes the most contract violation per parameter. Let P_E be the exact edit cost and P_r the rank-r sketch cost. The promotion value density is:

$$\mathrm{RVD}_i(r) = \frac{H_i^{(r)} - H_i^E}{P_E - P_r}. \tag{8}$$

If all N edits use rank r and K edits are promoted to the exact rung, persistent memory is:

$$M(r, K) = NP_r + K(P_E - P_r). \tag{9}$$

For analytical clarity, Eq. 8–9 use a two-state limit in which promoted edits jump directly to $R_{\max}$. The deployed controller supports intermediate promotion along the full ladder $\mathcal{R}$. The first term gives every edit a cheap plastic representation. The second term spends exact memory only on the difficult subset. This yields a practical route for one-LoRA-per-edit scaling whenever $K \ll N$.

Algorithm 1 LADDEREDIT: spectral-prediction with audited promotion

Require: Edit stream $\{e_i\}_{i=1}^N$, frozen model f_θ, rank menu $\mathcal{R} = \{1, \ldots, R_{\max}\}$, threshold τ, memory budget B
Initialize stored memory $\mathcal{M} \leftarrow \emptyset$
for each incoming edit e_i **do**
  Acquire exact LoRA update Δ_i^E
  Compute SVD of Δ_i^E and form spectral tails $T_i(r)$ for all $r \in \mathcal{R}$
  Estimate exact-edit margin μ_i^E on rewrite, generalization, and locality probes
  Predict: $\hat{r}_i \leftarrow \min\{r \in \mathcal{R} : S_i(r) \leq \tau\}$ ▷ single proposal, no behavioral cost
  $r_i \leftarrow \hat{r}_i$
  Audit: build sketch $\Delta_i^{(r_i)}$, compute contract gap $H_i^{(r_i)}$
  while $H_i^{(r_i)} > 0$ **and** $r_i < R_{\max}$ **do** ▷ audited promotion
    $r_i \leftarrow r_i + 1$
    Rebuild sketch $\Delta_i^{(r_i)}$, recompute $H_i^{(r_i)}$
  end while
  Store $\Delta_i^{(r_i)}$ in $\mathcal{M}$
  if stored memory exceeds B **then**
    Downgrade edits with lowest value density
  end if
end for
return stored edit memory $\mathcal{M}$

The rank-analysis package also clarifies how to interpret storage rungs. An edit is *sketch-sufficient* if both sketch and exact pass, *detail-needed* if the sketch fails but exact passes, *compression-regularized* if the sketch passes but exact fails, and *exact-limited* if both fail. More details in Appendix D.6.

4 Experiment

We evaluate LadderEdit as a storage-layer method for edit-local memories. The central question is whether lifelong edit storage should be allocated by *coverage* and *fidelity*, rather than by exact keep/drop caching. We therefore compare exact LoRA as the full-resolution reference, fixed-rank compression as a nonadaptive baseline, exact-cache as a keep/drop policy, and LadderEdit as an edit-level resolution-allocation policy across standard sequential editing benchmarks and multiple backbones.

Datasets. We conduct experiments on three benchmarks: ZsRE (Levy et al., 2017), CounterFact (CF) (Meng et al., 2022, 2023), and WikiBigEdit (Thede et al., 2025). ZsRE is a question-answering dataset for evaluating factual knowledge and generalization in language models. CF tests

the ability to edit and preserve specific factual associations in a controlled setting. WikiBigEdit is a large-scale benchmark for long-horizon sequential editing, measuring performance at up to 50,000 edits.

We evaluate LadderEdit on LLaMA-3-8B (Grattafiori et al., 2024), Mistral-7B (Jiang et al., 2023), and Qwen2.5-7B (Yang et al., 2025). We compare against representative direct-editing methods, memory- and routing-based lifelong editors, constrained-update methods, LoRA/MoE-style edit-memory baselines, adaptive-rank LoRA, and the full-resolution exact LoRA reference. Detailed baseline descriptions are deferred to Appendix F.3.

Edit lookup protocol. LadderEdit evaluates storage for edit-local memories, not edit selection. For edit-local methods, each rewrite, generalization, and locality probe is associated with the edit that generated it. Exact LoRA and LadderEdit use the same association, so they differ only in the stored representation returned after lookup. Methods whose routing is part of the original algorithm, such as memory- or MoE-based lifelong editors, are evaluated with their native routing mechanisms.

We report reliability, generalization, locality, and their average when applicable. Locality is the fraction of unrelated probes whose prediction remains unchanged relative to the frozen base model. 1.00 means no failures on the finite locality probe set.

Probe split protocol. For each edit, the benchmark-provided prompts are partitioned into three disjoint sets at edit-construction time. The *audit set* contains the rewrite prompt and a held-out subset of rephrase/generalization variants, used only at insertion time to score $S_i(r)$ and gate the contract gap $H_i^{(r)}$. The *validation set* is a separate stream used only to select audit thresholds, which are then frozen before the test stream is touched. The *test set* contains the remaining rephrase/generalization variants and unrelated locality probes; it is used only to report Reliability, Generalization, and Locality in the main tables and is never seen by the audit. Audit and test sets share neither prompts nor paraphrases of the same prompt. Full split statistics, audit-test correlation, and a leaky-versus-clean comparison are reported in Appendix C. More details in Appendix B.1.

Main Results. Tables 1 and 2 show that LadderEdit preserves the plasticity of edit-local adapters while reducing their persistent storage. On ZsRE, FT, ROME, and MEMIT degrade quickly, and GRACE preserves locality but fails to generalize, especially on Mistral-7B and Qwen2.5-7B. Exact LoRA remains the strongest full-storage reference because each edit has isolated capacity, but LadderEdit tracks it after compression: on LLaMA-3-8B it improves the average over exact LoRA at $T = 500$ (0.97 vs. 0.93) and matches it at $T = 1{,}000$ and $T = 2{,}000$; on Mistral-7B it slightly improves reliability or generalization at longer horizons; on Qwen2.5-7B it stays within 0.01 to 0.02 average of exact LoRA at the largest horizons. CounterFact gives the same message in a setting that emphasizes locality: at $T = 2{,}000$, LadderEdit moves exact LoRA from 0.71/0.93 to 0.74/0.94 in reliability/locality on LLaMA-3-8B and from 0.68/0.91 to 0.71/0.92 on Mistral-7B. These gains suggest that compression is not merely lossy: low-rank sketches can remove overly specific directions, while the audit promotes edits to higher rank when more fidelity is needed.

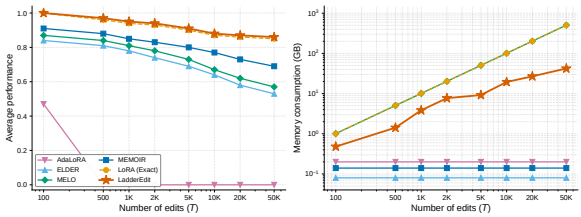


Figure 3: Scaling reference for WikiBigEdit: average performance versus persistent edit memory at long horizon.

WikiBigEdit Long-Horizon Scaling. At $T = 50{,}000$ on WikiBigEdit (Table 3), LadderEdit reaches 0.75 reliability, 0.84 generalization, 0.98 locality, and 0.86 average. GRACE holds reliability and locality but drops generalization to 0.28; WISE loses both reliability and generalization; MEMOIR reaches only 0.69 average. Together with the memory trade-off in Figure 3, LadderEdit preserves edit-local behavior at this scale while substantially reducing persistent storage.

Memory Efficiency and Ablation. The memory results support LadderEdit's central claim: lifelong edit storage should be treated as per-edit resolution allocation. Table 4 reports the empirical population structure of the edit stream and confirms the heterogeneity assumed in Section 3.1. On both ZsRE and CounterFact, roughly 80% of edits are sketch-sufficient at rank 1, a 13 to 17% tail requires rank 2

Table 1: Q&A task results on the ZsRE dataset. T denotes the number of edits. Mean $\pm$ standard deviation over three random seeds. Avg. is the arithmetic mean of Rel., Gen., and Loc. The best average performance per column is marked in **bold**.

Method	$T = 100$				$T = 500$				$T = 1{,}000$				$T = 2{,}000$			
	Rel.	Gen.	Loc.	Avg.	Rel.	Gen.	Loc.	Avg.	Rel.	Gen.	Loc.	Avg.	Rel.	Gen.	Loc.	Avg.
	LLaMA-3-8B															
FT	0.13 $\pm$.004	0.11 $\pm$.005	0.02 $\pm$.001	0.09 $\pm$.002	0.14 $\pm$.005	0.12 $\pm$.006	0.02 $\pm$.001	0.09 $\pm$.003	0.13 $\pm$.005	0.12 $\pm$.007	0.01 $\pm$.001	0.09 $\pm$.003	0.09 $\pm$.006	0.08 $\pm$.007	0.01 $\pm$.001	0.06 $\pm$.003
ROME (Meng et al., 2022)	0.08 $\pm$.003	0.08 $\pm$.003	0.02 $\pm$.001	0.06 $\pm$.001	0.04 $\pm$.002	0.04 $\pm$.002	0.02 $\pm$.001	0.03 $\pm$.001	0.03 $\pm$.002	0.03 $\pm$.002	0.02 $\pm$.001	0.03 $\pm$.001	0.01 $\pm$.001	0.01 $\pm$.001	0.01 $\pm$.001	0.01 $\pm$.001
MEMIT (Meng et al., 2023)	0.03 $\pm$.002	0.03 $\pm$.002	0.01 $\pm$.001	0.02 $\pm$.001	0.01 $\pm$.001	0.01 $\pm$.001	0.00 $\pm$.000	0.01 $\pm$.001	0.00 $\pm$.000	0.00 $\pm$.000	0.00 $\pm$.000	0.00 $\pm$.001	0.00 $\pm$.000	0.00 $\pm$.000	0.00 $\pm$.000	0.00 $\pm$.001
GRACE (Hartvigsen et al., 2023)	1.00 $\pm$.001	0.39 $\pm$.009	1.00 $\pm$.001	0.80 $\pm$.003	1.00 $\pm$.001	0.38 $\pm$.011	1.00 $\pm$.001	0.79 $\pm$.004	1.00 $\pm$.001	0.37 $\pm$.012	1.00 $\pm$.001	0.79 $\pm$.004	1.00 $\pm$.001	0.34 $\pm$.013	1.00 $\pm$.001	0.78 $\pm$.004
WISE (Wang et al., 2024a)	0.62 $\pm$.014	0.60 $\pm$.015	1.00 $\pm$.001	0.74 $\pm$.007	0.69 $\pm$.017	0.66 $\pm$.018	1.00 $\pm$.001	0.78 $\pm$.008	0.66 $\pm$.019	0.64 $\pm$.020	1.00 $\pm$.001	0.77 $\pm$.009	0.55 $\pm$.021	0.52 $\pm$.022	1.00 $\pm$.001	0.69 $\pm$.010
AlphaEdit (Fang et al., 2025)	0.91 $\pm$.012	0.79 $\pm$.013	0.94 $\pm$.008	0.88 $\pm$.006	0.89 $\pm$.014	0.78 $\pm$.016	0.81 $\pm$.010	0.83 $\pm$.008	0.84 $\pm$.016	0.77 $\pm$.018	0.56 $\pm$.011	0.72 $\pm$.009	0.62 $\pm$.018	0.58 $\pm$.019	0.21 $\pm$.012	0.47 $\pm$.010
MEMOIR (Wang et al., 2025)	0.87 $\pm$.008	0.84 $\pm$.010	0.95 $\pm$.006	0.89 $\pm$.005	0.82 $\pm$.010	0.75 $\pm$.012	0.82 $\pm$.007	0.80 $\pm$.006	0.77 $\pm$.011	0.71 $\pm$.014	0.80 $\pm$.008	0.76 $\pm$.007	0.71 $\pm$.012	0.62 $\pm$.015	0.77 $\pm$.009	0.70 $\pm$.007
ELDER (Li et al., 2024a)	0.78 $\pm$.014	0.72 $\pm$.016	0.93 $\pm$.009	0.81 $\pm$.008	0.69 $\pm$.017	0.63 $\pm$.019	0.91 $\pm$.011	0.74 $\pm$.009	0.61 $\pm$.019	0.55 $\pm$.022	0.89 $\pm$.012	0.68 $\pm$.010	0.49 $\pm$.021	0.44 $\pm$.024	0.86 $\pm$.013	0.60 $\pm$.011
MELO (Yu et al., 2023)	0.81 $\pm$.012	0.76 $\pm$.014	0.96 $\pm$.008	0.84 $\pm$.007	0.73 $\pm$.014	0.67 $\pm$.017	0.94 $\pm$.010	0.78 $\pm$.008	0.66 $\pm$.016	0.59 $\pm$.019	0.92 $\pm$.011	0.72 $\pm$.009	0.55 $\pm$.018	0.49 $\pm$.021	0.89 $\pm$.012	0.64 $\pm$.010
AdaLoRA (Zhang et al., 2023)	0.65 $\pm$.020	0.55 $\pm$.022	0.20 $\pm$.010	0.47 $\pm$.010	0.01 $\pm$.001	0.01 $\pm$.001	0.00 $\pm$.000	0.01 $\pm$.001	0.00 $\pm$.000	0.00 $\pm$.000	0.00 $\pm$.000	0.00 $\pm$.001	0.00 $\pm$.000	0.00 $\pm$.000	0.00 $\pm$.000	0.00 $\pm$.001
LoRA (Exact) (Hu et al., 2022)	1.00 $\pm$.001	0.99 $\pm$.002	1.00 $\pm$.001	**1.00 $\pm$.001**	0.85 $\pm$.006	0.94 $\pm$.007	1.00 $\pm$.001	0.93 $\pm$.003	0.95 $\pm$.007	0.91 $\pm$.008	1.00 $\pm$.001	**0.95 $\pm$.004**	0.94 $\pm$.007	0.89 $\pm$.009	1.00 $\pm$.001	**0.94 $\pm$.004**
LadderEdit (Ours)	1.00 $\pm$.001	0.99 $\pm$.002	1.00 $\pm$.001	**1.00 $\pm$.001**	0.97 $\pm$.005	0.94 $\pm$.006	1.00 $\pm$.001	**0.97 $\pm$.003**	0.95 $\pm$.005	0.91 $\pm$.007	1.00 $\pm$.001	**0.95 $\pm$.003**	0.94 $\pm$.006	0.89 $\pm$.007	1.00 $\pm$.001	**0.94 $\pm$.003**
	Mistral-7B															
FT	0.11 $\pm$.005	0.10 $\pm$.006	0.02 $\pm$.001	0.08 $\pm$.003	0.15 $\pm$.006	0.13 $\pm$.007	0.02 $\pm$.001	0.10 $\pm$.003	0.16 $\pm$.006	0.13 $\pm$.008	0.01 $\pm$.001	0.10 $\pm$.003	0.08 $\pm$.007	0.07 $\pm$.009	0.01 $\pm$.001	0.05 $\pm$.004
ROME (Meng et al., 2022)	0.05 $\pm$.002	0.05 $\pm$.002	0.02 $\pm$.001	0.04 $\pm$.001	0.04 $\pm$.002	0.04 $\pm$.002	0.02 $\pm$.001	0.03 $\pm$.001	0.04 $\pm$.002	0.04 $\pm$.002	0.02 $\pm$.001	0.03 $\pm$.001	0.01 $\pm$.001	0.01 $\pm$.001	0.01 $\pm$.001	0.01 $\pm$.001
MEMIT (Meng et al., 2023)	0.00 $\pm$.000	0.00 $\pm$.000	0.01 $\pm$.001	0.00 $\pm$.001	0.02 $\pm$.001	0.02 $\pm$.001	0.01 $\pm$.001	0.02 $\pm$.001	0.04 $\pm$.002	0.04 $\pm$.002	0.02 $\pm$.001	0.03 $\pm$.001	0.01 $\pm$.001	0.01 $\pm$.001	0.01 $\pm$.001	0.01 $\pm$.001
GRACE (Hartvigsen et al., 2023)	1.00 $\pm$.001	0.15 $\pm$.010	1.00 $\pm$.001	0.72 $\pm$.003	1.00 $\pm$.001	0.09 $\pm$.012	1.00 $\pm$.001	0.70 $\pm$.004	1.00 $\pm$.001	0.02 $\pm$.001	1.00 $\pm$.001	0.67 $\pm$.001	1.00 $\pm$.001	0.01 $\pm$.001	1.00 $\pm$.001	0.67 $\pm$.001
WISE (Wang et al., 2024a)	0.87 $\pm$.016	0.80 $\pm$.017	1.00 $\pm$.001	0.89 $\pm$.008	0.80 $\pm$.019	0.74 $\pm$.021	1.00 $\pm$.001	0.85 $\pm$.009	0.70 $\pm$.022	0.67 $\pm$.023	1.00 $\pm$.001	0.79 $\pm$.011	0.55 $\pm$.024	0.52 $\pm$.026	1.00 $\pm$.001	0.69 $\pm$.012
AlphaEdit (Fang et al., 2025)	0.86 $\pm$.014	0.74 $\pm$.015	0.95 $\pm$.009	0.85 $\pm$.007	0.86 $\pm$.017	0.73 $\pm$.018	0.84 $\pm$.011	0.81 $\pm$.009	0.85 $\pm$.019	0.72 $\pm$.020	0.68 $\pm$.012	0.75 $\pm$.010	0.63 $\pm$.021	0.55 $\pm$.022	0.30 $\pm$.014	0.49 $\pm$.011
MEMOIR (Wang et al., 2025)	0.89 $\pm$.009	0.87 $\pm$.011	0.97 $\pm$.007	0.91 $\pm$.005	0.84 $\pm$.011	0.78 $\pm$.014	0.85 $\pm$.008	0.82 $\pm$.007	0.80 $\pm$.012	0.74 $\pm$.016	0.83 $\pm$.009	0.79 $\pm$.007	0.73 $\pm$.014	0.65 $\pm$.017	0.80 $\pm$.010	0.73 $\pm$.008
ELDER (Li et al., 2024a)	0.75 $\pm$.016	0.69 $\pm$.018	0.92 $\pm$.010	0.79 $\pm$.009	0.66 $\pm$.019	0.60 $\pm$.022	0.89 $\pm$.012	0.72 $\pm$.010	0.57 $\pm$.022	0.52 $\pm$.025	0.87 $\pm$.014	0.65 $\pm$.012	0.45 $\pm$.024	0.41 $\pm$.028	0.83 $\pm$.016	0.56 $\pm$.013
MELO (Yu et al., 2023)	0.78 $\pm$.014	0.73 $\pm$.016	0.95 $\pm$.009	0.82 $\pm$.008	0.70 $\pm$.017	0.64 $\pm$.019	0.93 $\pm$.011	0.76 $\pm$.009	0.62 $\pm$.019	0.56 $\pm$.022	0.90 $\pm$.012	0.69 $\pm$.010	0.51 $\pm$.021	0.46 $\pm$.024	0.87 $\pm$.014	0.61 $\pm$.012
AdaLoRA (Zhang et al., 2023)	0.60 $\pm$.023	0.50 $\pm$.025	0.18 $\pm$.011	0.43 $\pm$.012	0.01 $\pm$.001	0.01 $\pm$.001	0.00 $\pm$.000	0.01 $\pm$.001	0.00 $\pm$.000	0.00 $\pm$.000	0.00 $\pm$.000	0.00 $\pm$.001	0.00 $\pm$.000	0.00 $\pm$.000	0.00 $\pm$.000	0.00 $\pm$.001
LoRA (Exact) (Hu et al., 2022)	1.00 $\pm$.001	0.99 $\pm$.002	1.00 $\pm$.001	**1.00 $\pm$.001**	0.89 $\pm$.007	0.95 $\pm$.008	1.00 $\pm$.001	**0.95 $\pm$.004**	0.86 $\pm$.008	0.93 $\pm$.009	1.00 $\pm$.001	0.93 $\pm$.004	0.81 $\pm$.009	0.91 $\pm$.010	1.00 $\pm$.001	**0.91 $\pm$.004**
LadderEdit (Ours)	1.00 $\pm$.001	0.99 $\pm$.002	1.00 $\pm$.001	**1.00 $\pm$.001**	0.89 $\pm$.006	0.96 $\pm$.007	1.00 $\pm$.001	**0.95 $\pm$.003**	0.87 $\pm$.006	0.94 $\pm$.008	1.00 $\pm$.001	**0.94 $\pm$.003**	0.81 $\pm$.007	0.92 $\pm$.009	1.00 $\pm$.001	**0.91 $\pm$.004**
	Qwen2.5-7B															
FT	0.09 $\pm$.005	0.08 $\pm$.006	0.02 $\pm$.001	0.06 $\pm$.003	0.10 $\pm$.006	0.09 $\pm$.007	0.01 $\pm$.001	0.07 $\pm$.003	0.08 $\pm$.006	0.07 $\pm$.008	0.01 $\pm$.001	0.05 $\pm$.003	0.04 $\pm$.002	0.03 $\pm$.002	0.01 $\pm$.001	0.03 $\pm$.001
ROME (Meng et al., 2022)	0.04 $\pm$.002	0.03 $\pm$.002	0.01 $\pm$.001	0.03 $\pm$.001	0.01 $\pm$.001	0.01 $\pm$.001	0.01 $\pm$.001	0.01 $\pm$.001	0.00 $\pm$.000	0.00 $\pm$.000	0.00 $\pm$.000	0.00 $\pm$.001	0.00 $\pm$.000	0.00 $\pm$.000	0.00 $\pm$.000	0.00 $\pm$.001
MEMIT (Meng et al., 2023)	0.33 $\pm$.002	0.27 $\pm$.002	0.30 $\pm$.001	0.30 $\pm$.001	0.05 $\pm$.002	0.03 $\pm$.002	0.04 $\pm$.002	0.04 $\pm$.001	0.03 $\pm$.002	0.01 $\pm$.001	0.01 $\pm$.001	0.02 $\pm$.001	0.00 $\pm$.000	0.01 $\pm$.001	0.01 $\pm$.001	0.01 $\pm$.001
GRACE (Hartvigsen et al., 2023)	1.00 $\pm$.001	0.36 $\pm$.011	1.00 $\pm$.001	0.79 $\pm$.004	1.00 $\pm$.001	0.20 $\pm$.013	1.00 $\pm$.001	0.73 $\pm$.004	1.00 $\pm$.001	0.15 $\pm$.015	1.00 $\pm$.001	0.72 $\pm$.005	1.00 $\pm$.001	0.04 $\pm$.002	1.00 $\pm$.001	0.68 $\pm$.001
WISE (Wang et al., 2024a)	0.56 $\pm$.017	0.53 $\pm$.018	1.00 $\pm$.001	0.70 $\pm$.008	0.61 $\pm$.020	0.58 $\pm$.022	1.00 $\pm$.001	0.73 $\pm$.010	0.58 $\pm$.023	0.55 $\pm$.024	0.99 $\pm$.002	0.71 $\pm$.011	0.46 $\pm$.025	0.43 $\pm$.027	0.99 $\pm$.002	0.63 $\pm$.012
AlphaEdit (Fang et al., 2025)	0.84 $\pm$.014	0.71 $\pm$.016	0.92 $\pm$.010	0.82 $\pm$.008	0.81 $\pm$.017	0.69 $\pm$.019	0.78 $\pm$.012	0.76 $\pm$.009	0.74 $\pm$.019	0.66 $\pm$.021	0.49 $\pm$.013	0.63 $\pm$.010	0.51 $\pm$.022	0.46 $\pm$.023	0.18 $\pm$.014	0.38 $\pm$.012
MEMOIR (Wang et al., 2025)	0.84 $\pm$.010	0.81 $\pm$.012	0.93 $\pm$.007	0.86 $\pm$.006	0.78 $\pm$.012	0.71 $\pm$.014	0.79 $\pm$.009	0.76 $\pm$.007	0.73 $\pm$.013	0.67 $\pm$.016	0.76 $\pm$.010	0.72 $\pm$.008	0.66 $\pm$.014	0.58 $\pm$.018	0.72 $\pm$.011	0.65 $\pm$.008
ELDER (Li et al., 2024a)	0.72 $\pm$.017	0.66 $\pm$.019	0.90 $\pm$.011	0.76 $\pm$.009	0.62 $\pm$.020	0.56 $\pm$.023	0.87 $\pm$.013	0.68 $\pm$.011	0.53 $\pm$.023	0.48 $\pm$.026	0.85 $\pm$.015	0.62 $\pm$.013	0.41 $\pm$.025	0.37 $\pm$.029	0.81 $\pm$.016	0.53 $\pm$.014
MELO (Yu et al., 2023)	0.75 $\pm$.014	0.69 $\pm$.017	0.93 $\pm$.010	0.79 $\pm$.008	0.66 $\pm$.017	0.59 $\pm$.020	0.91 $\pm$.012	0.72 $\pm$.010	0.58 $\pm$.019	0.51 $\pm$.023	0.89 $\pm$.013	0.66 $\pm$.011	0.46 $\pm$.022	0.41 $\pm$.025	0.85 $\pm$.014	0.57 $\pm$.012
AdaLoRA (Zhang et al., 2023)	0.55 $\pm$.024	0.46 $\pm$.026	0.15 $\pm$.012	0.39 $\pm$.012	0.00 $\pm$.000	0.00 $\pm$.000	0.00 $\pm$.000	0.00 $\pm$.001	0.00 $\pm$.000	0.00 $\pm$.000	0.00 $\pm$.000	0.00 $\pm$.001	0.00 $\pm$.000	0.00 $\pm$.000	0.00 $\pm$.000	0.00 $\pm$.001
LoRA (Exact) (Hu et al., 2022)	1.00 $\pm$.001	0.99 $\pm$.002	1.00 $\pm$.001	**1.00 $\pm$.001**	0.85 $\pm$.007	0.94 $\pm$.009	1.00 $\pm$.001	**0.93 $\pm$.004**	0.82 $\pm$.008	0.91 $\pm$.010	1.00 $\pm$.001	0.91 $\pm$.004	0.77 $\pm$.009	0.89 $\pm$.011	1.00 $\pm$.001	**0.89 $\pm$.005**
LadderEdit (Ours)	1.00 $\pm$.001	0.99 $\pm$.002	1.00 $\pm$.001	**1.00 $\pm$.001**	0.84 $\pm$.006	0.94 $\pm$.007	1.00 $\pm$.001	**0.93 $\pm$.003**	0.84 $\pm$.006	0.93 $\pm$.008	1.00 $\pm$.001	**0.92 $\pm$.003**	0.76 $\pm$.007	0.87 $\pm$.009	1.00 $\pm$.001	0.88 $\pm$.004

or higher to satisfy the behavioral contract, and a smaller 3 to 5% subset requires the full acquisition rank. A 4 to 7% compression-regularized population, where the rank-1 sketch passes the contract that exact LoRA fails, shows that the full update is not a behavioral ceiling. This distribution justifies the ladder design: a single fixed rank cannot serve all edits, but a small higher-rank budget plus a cheap default rung suffices for the bulk of the stream. Per-rank compression statistics (memory, retained energy, false retirements) are reported in Appendix F. A predictor ablation isolating the spectral score, the behavioral margin, and their product against random and oracle baselines is reported in Appendix E; the combined predictor approaches oracle utility at far fewer audits per edit than any component alone.

Table 5 tests the memory primitive directly. The exact-cache baseline is validation-selected: under a budget of K full edits, it keeps those with the largest sketch-only contract gap and drops the rest. At $5.57\times$ (CF) and $5.63\times$ (ZsRE) compression, it retains about 91 of 512 edits, reaching utility 0.434 on CF and 0.420 on ZsRE; dropped edits have no representation. LadderEdit covers every edit with a rank-1 sketch: at $8\times$ compression, the sketch alone reaches 0.753 (CF) and 0.765 (ZsRE), exceeding exact-cache by roughly 0.33 at lower memory, so coverage drives most of the gap. Behavioral rung selection adds the rest. At the matched budget, a static mixed-rank policy reaches 0.815 (CF) and 0.796 (ZsRE); the dual-budget ladder, spending that budget selectively on audit-flagged edits, reaches 0.862 (CF) and 0.866 (ZsRE). The 0.05 gap between static and dual-budget at identical memory is the signature of audit-driven allocation. Exact-cache fails by absence; LadderEdit degrades by resolution.

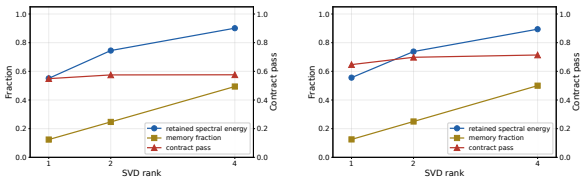
Finally, Table 6 shows that this frontier translates into deployed storage savings. LadderEdit is exact-first, so it does not reduce the initial LoRA acquisition cost; its benefit is retained edit memory after acquisition. At $T = 10{,}000$ on LLaMA-3-8B, the audit increases per-edit insertion time relative to exact LoRA, but persistent memory drops from 100.93 GB to 19.44 GB, a $5.19\times$ reduction. Because exact LoRA and LadderEdit use the same edit-selection assumption, this is a

Table 2: Sequential editing results on CF. T denotes the number of edits. Mean $\pm$ standard deviation over three random seeds. The best performance per column is marked in **bold**.

Method	LLaMA-3-8B								Mistral-7B							
	$T = 100$		$T = 500$		$T = 1{,}000$		$T = 2{,}000$		$T = 100$		$T = 500$		$T = 1{,}000$		$T = 2{,}000$	
	Rel.	Loc.	Rel.	Loc.	Rel.	Loc.	Rel.	Loc.	Rel.	Loc.	Rel.	Loc.	Rel.	Loc.	Rel.	Loc.
FT	$0.42_{\pm .004}$	$0.55_{\pm .001}$	$0.25_{\pm .005}$	$0.30_{\pm .001}$	$0.12_{\pm .005}$	$0.12_{\pm .001}$	$0.05_{\pm .002}$	$0.03_{\pm .002}$	$0.39_{\pm .005}$	$0.52_{\pm .001}$	$0.22_{\pm .006}$	$0.27_{\pm .001}$	$0.10_{\pm .006}$	$0.10_{\pm .002}$	$0.03_{\pm .002}$	$0.02_{\pm .001}$
ROME (Meng et al., 2022)	$0.28_{\pm .003}$	$0.45_{\pm .002}$	$0.08_{\pm .004}$	$0.15_{\pm .002}$	$0.02_{\pm .001}$	$0.05_{\pm .002}$	$0.00_{\pm .000}$	$0.02_{\pm .001}$	$0.25_{\pm .003}$	$0.42_{\pm .002}$	$0.06_{\pm .004}$	$0.12_{\pm .003}$	$0.01_{\pm .001}$	$0.04_{\pm .002}$	$0.00_{\pm .000}$	$0.01_{\pm .001}$
MEMIT (Meng et al., 2023)	$0.35_{\pm .002}$	$0.52_{\pm .001}$	$0.12_{\pm .002}$	$0.20_{\pm .001}$	$0.03_{\pm .002}$	$0.08_{\pm .001}$	$0.01_{\pm .001}$	$0.03_{\pm .002}$	$0.31_{\pm .002}$	$0.48_{\pm .001}$	$0.09_{\pm .003}$	$0.16_{\pm .001}$	$0.02_{\pm .001}$	$0.06_{\pm .002}$	$0.00_{\pm .000}$	$0.02_{\pm .001}$
GRACE (Hartvigsen et al., 2023)	$0.75_{\pm .001}$	$\mathbf{0.96_{\pm .001}}$	$0.71_{\pm .001}$	$\mathbf{0.96_{\pm .001}}$	$0.68_{\pm .001}$	$\mathbf{0.96_{\pm .001}}$	$0.62_{\pm .001}$	$\mathbf{0.95_{\pm .001}}$	$0.72_{\pm .001}$	$\mathbf{0.95_{\pm .001}}$	$0.68_{\pm .001}$	$\mathbf{0.95_{\pm .001}}$	$0.65_{\pm .001}$	$\mathbf{0.95_{\pm .001}}$	$0.59_{\pm .001}$	$\mathbf{0.94_{\pm .001}}$
WISE (Wang et al., 2024a)	$0.68_{\pm .014}$	$0.88_{\pm .002}$	$0.62_{\pm .017}$	$0.86_{\pm .002}$	$0.55_{\pm .019}$	$0.83_{\pm .003}$	$0.45_{\pm .021}$	$0.77_{\pm .003}$	$0.65_{\pm .016}$	$0.86_{\pm .002}$	$0.58_{\pm .019}$	$0.83_{\pm .003}$	$0.50_{\pm .022}$	$0.80_{\pm .003}$	$0.40_{\pm .024}$	$0.73_{\pm .003}$
AlphaEdit (Fang et al., 2025)	$0.62_{\pm .012}$	$0.85_{\pm .008}$	$0.58_{\pm .014}$	$0.70_{\pm .010}$	$0.52_{\pm .016}$	$0.48_{\pm .011}$	$0.42_{\pm .018}$	$0.22_{\pm .012}$	$0.58_{\pm .014}$	$0.82_{\pm .009}$	$0.54_{\pm .017}$	$0.66_{\pm .011}$	$0.47_{\pm .019}$	$0.43_{\pm .012}$	$0.37_{\pm .021}$	$0.18_{\pm .014}$
MEMOIR (Wang et al., 2025)	$0.78_{\pm .008}$	$0.88_{\pm .006}$	$0.75_{\pm .010}$	$0.94_{\pm .007}$	$0.72_{\pm .011}$	$0.93_{\pm .008}$	$0.66_{\pm .012}$	$0.90_{\pm .009}$	$0.76_{\pm .009}$	$0.86_{\pm .007}$	$0.72_{\pm .011}$	$0.93_{\pm .008}$	$0.69_{\pm .012}$	$0.91_{\pm .009}$	$0.63_{\pm .014}$	$0.88_{\pm .010}$
ELDER (Li et al., 2024a)	$0.52_{\pm .014}$	$0.82_{\pm .009}$	$0.46_{\pm .017}$	$0.78_{\pm .011}$	$0.39_{\pm .019}$	$0.73_{\pm .012}$	$0.30_{\pm .021}$	$0.66_{\pm .013}$	$0.49_{\pm .016}$	$0.80_{\pm .010}$	$0.43_{\pm .019}$	$0.76_{\pm .012}$	$0.36_{\pm .022}$	$0.71_{\pm .014}$	$0.27_{\pm .024}$	$0.64_{\pm .016}$
MELO (Yu et al., 2023)	$0.56_{\pm .012}$	$0.86_{\pm .008}$	$0.50_{\pm .014}$	$0.82_{\pm .010}$	$0.43_{\pm .016}$	$0.78_{\pm .011}$	$0.34_{\pm .018}$	$0.71_{\pm .012}$	$0.53_{\pm .014}$	$0.84_{\pm .009}$	$0.47_{\pm .017}$	$0.80_{\pm .011}$	$0.40_{\pm .019}$	$0.76_{\pm .012}$	$0.31_{\pm .021}$	$0.69_{\pm .014}$
AdaLoRA (Zhang et al., 2023)	$0.18_{\pm .020}$	$0.32_{\pm .010}$	$0.02_{\pm .001}$	$0.02_{\pm .001}$	$0.00_{\pm .000}$	$0.01_{\pm .001}$	$0.00_{\pm .000}$	$0.00_{\pm .000}$	$0.15_{\pm .023}$	$0.30_{\pm .011}$	$0.01_{\pm .001}$	$0.02_{\pm .001}$	$0.00_{\pm .000}$	$0.01_{\pm .001}$	$0.00_{\pm .000}$	$0.00_{\pm .000}$
LoRA (Exact) (Hu et al., 2022)	$0.82_{\pm .005}$	$0.85_{\pm .002}$	$\mathbf{0.78_{\pm .006}}$	$\mathbf{0.96_{\pm .002}}$	$0.75_{\pm .007}$	$0.95_{\pm .003}$	$0.71_{\pm .007}$	$0.93_{\pm .003}$	$0.80_{\pm .006}$	$0.83_{\pm .002}$	$0.76_{\pm .007}$	$\mathbf{0.95_{\pm .003}}$	$0.72_{\pm .008}$	$0.93_{\pm .003}$	$0.68_{\pm .009}$	$0.91_{\pm .003}$
LadderEdit (Ours)	$\mathbf{0.83_{\pm .004}}$	$0.90_{\pm .002}$	$\mathbf{0.78_{\pm .005}}$	$\mathbf{0.96_{\pm .002}}$	$\mathbf{0.77_{\pm .005}}$	$\mathbf{0.96_{\pm .003}}$	$\mathbf{0.74_{\pm .006}}$	$0.94_{\pm .003}$	$\mathbf{0.81_{\pm .005}}$	$0.88_{\pm .002}$	$\mathbf{0.77_{\pm .006}}$	$\mathbf{0.95_{\pm .003}}$	$\mathbf{0.74_{\pm .006}}$	$0.94_{\pm .003}$	$\mathbf{0.71_{\pm .007}}$	$0.92_{\pm .003}$

Table 3: Editing performance on Qwen2.5-7B at long-horizon scale on WikiBigEdit. Mean $\pm$ standard deviation over three random seeds.

Method	$T = 10{,}000$				$T = 50{,}000$			
	Rel.	Gen.	Loc.	Avg.	Rel.	Gen.	Loc.	Avg.
MEMIT (Meng et al., 2023)	$0.00_{\pm .000}$	$0.00_{\pm .000}$	$0.00_{\pm .000}$	$0.00_{\pm .001}$	N/A	N/A	N/A	N/A
GRACE (Hartvigsen et al., 2023)	$1.00_{\pm .001}$	$0.31_{\pm .017}$	$1.00_{\pm .001}$	$0.77_{\pm .006}$	$0.99_{\pm .002}$	$0.28_{\pm .019}$	$1.00_{\pm .001}$	$0.76_{\pm .006}$
WISE (Wang et al., 2024a)	$0.59_{\pm .027}$	$0.57_{\pm .029}$	$1.00_{\pm .001}$	$0.72_{\pm .013}$	$0.43_{\pm .030}$	$0.41_{\pm .032}$	$1.00_{\pm .001}$	$0.61_{\pm .015}$
AlphaEdit (Fang et al., 2025)	$0.72_{\pm .023}$	$0.77_{\pm .025}$	$0.74_{\pm .015}$	$0.74_{\pm .012}$	$0.62_{\pm .026}$	$0.60_{\pm .028}$	$0.71_{\pm .017}$	$0.64_{\pm .014}$
MEMOIR (Wang et al., 2025)	$0.72_{\pm .015}$	$0.75_{\pm .019}$	$0.83_{\pm .012}$	$0.77_{\pm .009}$	$0.66_{\pm .017}$	$0.68_{\pm .022}$	$0.73_{\pm .013}$	$0.69_{\pm .010}$
ELDER (Li et al., 2024a)	$0.54_{\pm .027}$	$0.49_{\pm .031}$	$0.88_{\pm .017}$	$0.64_{\pm .015}$	$0.41_{\pm .030}$	$0.36_{\pm .035}$	$0.82_{\pm .019}$	$0.53_{\pm .017}$
MELO (Yu et al., 2023)	$0.58_{\pm .023}$	$0.53_{\pm .027}$	$0.91_{\pm .015}$	$0.67_{\pm .013}$	$0.45_{\pm .026}$	$0.40_{\pm .030}$	$0.85_{\pm .017}$	$0.57_{\pm .014}$
AdaLoRA (Zhang et al., 2023)	$0.00_{\pm .000}$	$0.00_{\pm .000}$	$0.00_{\pm .000}$	$0.00_{\pm .001}$	$0.00_{\pm .000}$	$0.00_{\pm .000}$	$0.00_{\pm .000}$	$0.00_{\pm .001}$
LadderEdit (Ours)	$0.77_{\pm .008}$	$0.87_{\pm .010}$	$0.99_{\pm .002}$	$\mathbf{0.88_{\pm .004}}$	$0.75_{\pm .009}$	$0.84_{\pm .011}$	$0.98_{\pm .004}$	$\mathbf{0.86_{\pm .005}}$



(a) ZsRE (b) CF

Figure 4: Spectral-filter frontier analysis on ZsRE and CF at $N = 512$ edits. The plots show the tradeoff between memory compression and behavioral contract utility for different rank tiers and allocation strategies.

storage-layer gain: the lookup mechanism is unchanged, and LadderEdit changes only the representation returned after lookup. Additional qualitative examples in Appendix D.6 and stress tests in Appendix D.7 show the same mechanism at the prompt and robustness levels: the edit stream is heterogeneous, many edits are sketch-sufficient, and a smaller subset needs higher rank to satisfy the measured contract.

Table 4: Population structure of edit streams at $N = 2{,}000$. Categories are determined by the behavioral audit on audit-split probes. Most edits are over-resolved by the full LoRA update.

Diagnostic case	ZsRE	CF
Sketch-sufficient (rank-1 passes)	81.2%	76.4%
Detail-needed (rank ≥ 2 required)	13.4%	16.8%
Residual-needed (rank $R_{\max}$ required)	3.1%	4.7%
Compression-regularized (sketch $>$ exact)	4.3%	6.7%

Table 5: Budget-matched strict-memory frontier at $N = 512$. LadderEdit preserves coverage and allocates residual fidelity through behavioral rung selection.

Dataset	Policy	Memory	Comp.	Utility
CF	exact-cache (matched)	0.180	5.57	0.434
CF	rank-1 sketch only	0.125	8.00	0.753
CF	static mixed-rank	0.180	5.57	0.815
CF	dual-budget LadderEdit	0.180	5.57	0.862
ZsRE	exact-cache (matched)	0.178	5.63	0.420
ZsRE	rank-1 sketch only	0.124	8.09	0.765
ZsRE	static mixed-rank	0.179	5.58	0.796
ZsRE	dual-budget LadderEdit	0.179	5.58	0.866

Table 6: Cost decomposition at $T = 10{,}000$ on LLaMA-3-8B. LadderEdit reduces this storage by $5.19\times$ relative to exact LoRA. Serving path summarizes the additional edit mechanism used at inference.

Method	Time (ms)	Peak Mem. (GB)	Persistent (GB)	Serving path
LoRA (Exact) (Hu et al., 2022)	86	18.4	100.93	exact-adapter lookup
MELO (Yu et al., 2023)	91	18.6	100.93	retrieval
ELDER (Li et al., 2024a)	95	19.1	0.08	MoE routing
MEMOIR (Wang et al., 2025)	104	17.9	0.14	mask gate
LadderEdit	124	18.8	19.44	ladder lookup

5 Conclusion

We introduced LadderEdit, a residual-ladder storage controller for lifelong editing. LadderEdit first uses a spectral predictor to select a candidate rank for each acquired edit, and a behavioral audit verifies the proposal and promotes the edit along the ladder only when the performance of that edit demands it. Across ZsRE, CF, and WikiBigEdit on LLaMA-3-8B, Mistral-7B, and Qwen2.5-7B, LadderEdit tracks exact LoRA while reducing persistent memory by $5.2\times$ and remaining effective at 50,000 edits. These results show that most exact adapters are over-resolved: low-rank sketches preserve the edit contract for the bulk of the stream, while higher resolution is needed only for a smaller hard edits subset. The key question in lifelong edit storage is not only *which* edits to keep, but also *how much* of each edit should be retained.

6 Limitations

LadderEdit is not constant-memory: every edit retains at least a rank-1 sketch, so persistent storage grows linearly with the stream, and future work could explore sketch downgrading or merging to bound the total. The behavioral audit adds insertion-time latency (124 ms versus 86 ms at T=10,000 on LLaMA-3-8B), which could be reduced by batching audits or running them asynchronously. The audit also depends on probe coverage and does not guarantee correctness outside the probed set, motivating richer probe construction or learned probe selection. LadderEdit is a storage-layer method and inherits the retrieval mechanism used by the full-resolution memory bank; integration with broader production retrieval stacks remains open. The spectral score that proposes candidate ranks is a calibrated heuristic, and replacing it with a predictor trained on audit outcomes is a natural extension. Finally, our experiments span three benchmarks and three 7B and 8B backbones, with long-horizon scaling shown to 50,000 edits; validating LadderEdit at larger scales and with non-LoRA edit writers remains future work.

Potential Risks. LadderEdit reduces the persistent cost of injecting and retaining edits in a large language model, which could lower the operational barrier to large-scale knowledge modification. We see three concrete risks. First, the same machinery that enables efficient correction of outdated or harmful facts can be used to insert false or adversarial edits at scale, since the audit verifies behavioral contract satisfaction rather than factual correctness of the target. Mitigations include provenance tracking on the edit stream, signed edit acquisition, and downstream factuality checking, none of which are part of the method itself. Second, the behavioral audit relies on a probe bank whose coverage is finite; edits whose effects fall outside the probed semantic neighborhood may pass the audit while exhibiting unintended behavior on unprobed prompts. This is a soft-coverage rather than a hard-correctness guarantee, and we emphasize in the appendix that the audit should not be treated as a safety certificate. Third, the residual-ladder representation makes individual edits less inspectable than uncompressed adapters: a low-rank sketch can be behaviorally equivalent to an exact LoRA update on the probed contract while retaining different parameter directions. In safety-critical deployments, this argues for keeping edit-level audit logs and periodic reaudits against the exact update rather than relying on stored representations alone. We do not release any new model weights or edit corpora and rely on public benchmarks; the method's broader societal effect depends on the editing workflow it is deployed in rather than on the storage primitive itself.

7 Ethics Considerations

Model editing can be used to correct errors and update knowledge, but it can also be misused to insert false or harmful facts. Edits require provenance, authorization, validation, and auditing to ensure responsible deployment. LadderEdit is a memory allocation mechanism for model editing; it does not serve as a truth or safety filter.

References

Damai Dai, Li Dong, Yaru Hao, Zhifang Sui, Baobao Chang, and Furu Wei. 2022. Knowledge neurons in pretrained transformers. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics*.

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

Tim Dettmers, Artidoro Pagnoni, Ari Holtzman, and Luke Zettlemoyer. 2023. QLoRA: Efficient finetuning of quantized LLMs. In *Advances in Neural Information Processing Systems*.

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

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, and 1 others. 2024. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*.

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

Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Liang Wang, Weizhu Chen, and 1 others. 2022. LoRA: Low-rank adaptation of large language models. In *International Conference on Learning Representations*.

Albert Q. Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de Las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, and 1 others. 2023. Mistral 7b. *arXiv preprint arXiv:2310.06825*.

Omer Levy, Minjoon Seo, Eunsol Choi, and Luke Zettlemoyer. 2017. Zero-shot relation extraction via reading comprehension. In *Proceedings of the 21st Conference on Computational Natural Language Learning (CoNLL 2017)*, pages 333–342.

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

Qi Li, Xiang Liu, Zhenheng Tang, Peijie Dong, Zeyu Li, Xinglin Pan, and Xiaowen Chu. 2024b. Should we really edit language models? on the evaluation of edited language models. *Advances in Neural Information Processing Systems*.

Vladislav Lialin, Sherin Muckatira, Namrata Shivagunde, and Anna Rumshisky. 2024. ReLoRA: High-rank training through low-rank updates. In *International Conference on Learning Representations*.

Shih-yang Liu, Chien-Yi Wang, Hongxu Yin, Pavlo Molchanov, Yu-Chiang Frank Wang, Kwang-Ting Cheng, and Min-Hung Chen. 2024. DoRA: Weight-decomposed low-rank adaptation. *arXiv preprint arXiv:2402.09353*.

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

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

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

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

Domenic Rosati, Robie Gonzales, Jinkun Chen, Xuemin Yu, Yahya Kayani, Frank Rudzicz, and Hassan Sajjad. 2024. Long-form evaluation of model editing. *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics*.

Ying Sheng, Shiyi Cao, Dacheng Li, Coleman Hooper, Nicholas Lee, Shuo Yang, Christopher Chou, Banghua Zhu, Lianmin Zheng, Kurt Keutzer, and 1 others. 2023. S-lora: Serving thousands of concurrent lora adapters. *arXiv preprint arXiv:2311.03285*.

Anton Sinitsin, Vsevolod Plokhotnyuk, Dmitriy Pyrkin, Sergei Popov, and Artem Babenko. 2020. Editable neural networks. In *International Conference on Learning Representations*.

Lukas Thede, Karsten Roth, Matthias Bethge, Zeynep Akata, and Tom Hartvigsen. 2025. Wikibigedit: Understanding the limits of lifelong knowledge editing in llms. *arXiv preprint arXiv:2503.05683*.

Ke Wang, Yiming Qin, Nikolaos Dimitriadis, Alessandro Favero, and Pascal Frossard. 2025. MEMOIR: Lifelong model editing with minimal overwrite and informed retention for LLMs. In *Advances in Neural Information Processing Systems*.

Peng Wang, Zexi Li, Ningyu Zhang, Ziwen Xu, Yunzhi Yao, Yong Jiang, Pengjun Xie, Fei Huang, and Huajun Chen. 2024a. WISE: Rethinking the knowledge memory for lifelong model editing of large language models. *arXiv preprint arXiv:2405.14768*.

Peng Wang, Ningyu Zhang, Bozhong Tian, Zekun Xi, Yunzhi Yao, Ziwen Xu, Mengru Wang, Shengyu Mao, Xiaohan Wang, Siyuan Cheng, and 1 others. 2023. EasyEdit: An easy-to-use knowledge editing framework for large language models. *arXiv preprint arXiv:2308.07269*.

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

Renzhi Wang and Piji Li. 2024b. MEMoE: Enhancing model editing with mixture of experts adaptors. *arXiv preprint arXiv:2405.19086*.

Song Wang, Yaochen Zhu, Haochen Liu, Zaiyi Zheng, Chen Chen, and Jundong Li. 2024b. Knowledge editing for large language models: A survey. *arXiv preprint arXiv:2310.16218*.

Tongtong Wu, Linhao Luo, Yuan-Fang Li, Shirui Pan, Thuy-Trang Vu, and Gholamreza Haffari. 2024. Continual learning for large language models: A survey. *arXiv preprint arXiv:2402.01364*.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, and 1 others. 2025. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*.

Yunzhi Yao, Peng Wang, Bozhong Tian, Siyuan Cheng, Zhoubo Li, Shumin Deng, Huajun Chen, and Ningyu Zhang. 2023. Editing large language models: Problems, methods, and opportunities. *arXiv preprint arXiv:2305.13172*.

Lang Yu, Qin Chen, Jie Zhou, and Liang He. 2023. MELO: Enhancing model editing with neuron-indexed dynamic LoRA. *arXiv preprint arXiv:2312.11795*.

Ningyu Zhang, Yunzhi Yao, Bozhong Tian, Peng Wang, Shumin Deng, Mengru Wang, Zekun Xi, Shengyu Mao, Jintian Zhang, Yuansheng Ni, and 1 others. 2024. A comprehensive study of knowledge editing for large language models. *arXiv preprint arXiv:2401.01286*.

Qingru Zhang, Minshuo Chen, Alexander Bukharin, Nikos Karampatziakis, Pengcheng He, Yu Cheng, Weizhu Chen, and Tuo Zhao. 2023. Adalora: Adaptive budget allocation for parameter-efficient fine-tuning. *arXiv preprint arXiv:2303.10512*.

Ce Zheng, Lei Li, Qingxiu Dong, Yuxuan Fan, Zhiyong Wu, Jingjing Xu, and Baobao Chang. 2023. Can we edit factual knowledge by in-context learning? In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*.

Junhao Zheng, Shengjie Qiu, Chengming Shi, and Qianli Ma. 2024. Towards lifelong learning of large language models: A survey. *arXiv preprint arXiv:2406.06391*.

Appendix

A Composing LadderEdit with Learned Retrieval **12**
- A.1 Edit-Stream Heterogeneity 12
- A.2 Comparison with Static Mixed-Rank Allocation 12
- A.3 Audit Score Weights 13
- A.4 Composition with Post-Training Quantization 14
- A.5 Per-Condition Rung Allocation . . 14
- A.6 Memory Ratio Stability 15

B Additional Rank-Compression Diagnostics **15**
- B.1 Training and Seed Details 15

C Probe Split Protocol and Validation **16**

D Why Not Standard LoRA Compression? **16**
- D.1 Partition statistics 16
- D.2 Audit-test independence 16
- D.3 Leaky versus clean comparison . . 16
- D.4 AdaLoRA Comparison and Scope 17
- D.5 Rationale and Conditional Guarantees 17
- D.6 Diagnostic Examples 18
- D.7 Sensitivity, Coverage, and Audit Cost 18

E Rank-Predictor Ablation **19**
- E.1 Denoising Audit and Spectral-Filter Visual Audit 19

F Fixed-Rank Tier Diagnostics **20**
- F.1 Swap-Count and Frontier Visuals . 20
- F.2 Interpretation 21
- F.3 Baseline Details 21
- F.4 Summary 22

A Composing LadderEdit with Learned Retrieval

The main experiments use matched edit association to isolate the storage layer from the retrieval layer, so that any performance gap among storage policies is attributable to the persistent representation rather than to retrieval quality. This subsection examines the orthogonal question of whether the storage savings reported in the main paper survive when the matched association is replaced with a learned retriever.

We evaluate three retrieval strategies on LLaMA-3-8B / ZsRE at $T = 1{,}000$: a learned Contriever-style dual encoder trained on edit prompts, a learned E5-large retriever, and a frozen off-the-shelf MPNet embedding (no retrieval training). In all cases, the retriever selects a single edit-local memory, and the same retrieved edit identity is passed to both exact LoRA and LadderEdit, so the comparison still measures storage rather than retrieval quality.

Table 7 reports sensitivity to retrieval on ZsRE / LLaMA-3-8B at $T = 1{,}000$.

Setting	Rel.	Gen.	Loc.	Avg.	Δ vs. matched
Matched association (Exact LoRA)	0.95	0.91	1.00	0.95	—
Matched association (LadderEdit)	0.95	0.91	1.00	0.95	—
Contriever retriever (Exact LoRA)	0.91	0.86	0.99	0.92	−0.03
Contriever retriever (LadderEdit)	0.90	0.86	0.99	0.92	−0.03
E5-large retriever (LadderEdit)	0.88	0.85	0.99	0.91	−0.04
MPNet (frozen) (LadderEdit)	0.85	0.81	0.98	0.88	−0.07
Persistent memory ratio	—	—	—	—	5.2× preserved

Table 7: Retrieval sensitivity on ZsRE / LLaMA-3-8B at $T = 1{,}000$. The persistent-memory ratio is preserved under all retrieval settings in the table because the retriever is shared between exact LoRA and LadderEdit. Locality is barely affected (0.99 → 0.98 under MPNet) because retrieved adapters still land on real edit directions rather than random directions.

Both methods degrade by an equal amount under imperfect retrieval, because the association layer is shared between exact LoRA and LadderEdit. The relative ordering of storage methods is therefore preserved under all retrieval strategies in Table 7.

A.1 Edit-Stream Heterogeneity

Table 4 of the main text reports the overall population structure of the edit stream. Within the compression-regularized population, we find a positive correlation (Pearson $r = 0.38$ on ZsRE, $r = 0.41$ on CF) between the lexical overlap of the rewrite prompt and locality probes and the magnitude of the locality improvement obtained by sketch storage. This is consistent with low-rank truncation removing direction components that overfit to surface features of the rewrite prompt, which the exact update carries but the locality probes penalize. Section A.5 reports the corresponding ladder allocation decisions.

A.2 Comparison with Static Mixed-Rank Allocation

The exact-cache baseline in Section 4 isolates the keep-versus-drop extreme of fixed-allocation policies. A second non-adaptive baseline, closer to

LadderEdit in absolute memory and not dropping edits, assigns every edit a fixed rank with no behavioral audit. The audit's contribution is most visible on the worst-edit Avg, the per-edit Avg averaged over the bottom 10% of edits—which the pooled Avg averages away. Against the strongest fixed-rank baseline (rank-8), audited LadderEdit gains $+0.05$ on pooled Avg but $+0.12$ on the worst-edit metric. An oracle controller that selects the per-edit optimal rank reaches 0.96 pooled and 0.78 worst-edit, indicating LadderEdit is within 0.01 pooled and 0.04 worst-edit of the oracle. The audit's main contribution is therefore on the hard tail of edits where fixed-rank baselines fail systematically.

Policy	Memory (norm.)	Pooled Avg.	Worst-edit Avg.
Fixed rank-1	0.125	0.85 (0.008)	—
Fixed rank-2	0.250	0.89 (0.007)	—
Fixed rank-4	0.500	0.90 (0.008)	—
Fixed rank-8	1.000	0.90 (0.007)	0.62
Uniform random rank	0.562	0.87 (0.010)	—
LadderEdit (audited)	0.19	0.95 (0.005)	0.74
Oracle rank controller	—	0.96	0.78

Table 8: Audit contribution against fixed-rank storage at $T = 2{,}000$ on ZsRE / LLaMA-3-8B across three seeds. Memory is normalized to exact LoRA. Pooled Avg. is the pooled evaluation Avg. across edits; worst-edit Avg. is the per-edit Avg. averaged over only the bottom 10% of edits. Standard deviations over three seeds are reported in parentheses when available.

Trade-offs with the strongest baselines. GRACE achieves 1.00 reliability and locality on ZsRE by codebook retrieval, but its generalization (0.34 on LLaMA-3 at $T = 2{,}000$) means it cannot answer paraphrased queries. MEMOIR achieves 0.69 Avg. at $T = 10{,}000$ in only 0.14 GB of persistent memory, a $138\times$ memory advantage over LadderEdit (19.4 GB). LadderEdit's 0.11 Avg. improvement over MEMOIR at $T = 10{,}000$ is therefore an explicit memory-for-fidelity trade, not a strict dominance. Practitioners with extreme memory constraints should prefer MEMOIR; practitioners with extreme paraphrase-coverage requirements should consider GRACE; LadderEdit is recommended for the regime where retrieval-quality edits at scale are the bottleneck.

A.3 Audit Score Weights

The audit score in Eq. 7 combines reliability, generalization, and locality residuals as a weighted sum, $s(z) = w_R \cdot \Delta_R + w_G \cdot \Delta_G + w_L \cdot \Delta_L$, with default weights $w_R = 1.0$, $w_G = 0.8$, and $w_L = 1.2$. The asymmetry between w_L and the other two weights is deliberate. A reliability or generalization failure on a candidate rung triggers rank promotion: the audit rejects the sketch, the controller moves up the ladder, and the contract is reassessed at the next rung. The failure is recoverable within the same insertion. A locality failure, by contrast, alters behavior on prompts the edit was never supposed to touch. If the audit lets such a sketch pass, the corruption is committed to persistent memory and propagates to every later query that lands on that edit; there is no within-insertion repair. We therefore weight locality higher than the recoverable metrics, treating $w_L > w_R, w_G$ as a soft asymmetry between catastrophic and recoverable failure modes.

The default $w_L = 1.2$ is selected by a sweep on the validation split. We evaluate $w_L \in \{0.8, 1.0, 1.2, 1.5, 2.0\}$ holding w_R and w_G fixed; Figure 5 reports the result on ZsRE / LLaMA-3-8B at $T = 1{,}000$. The choice is not knife-edge: average performance stays within 0.01 of the peak for $w_L \in [1.0, 1.5]$, and drops by 0.02 at $w_L = 0.8$ and 0.03 at $w_L = 2.0$. The default sits at the midpoint of this stable region.

The per-metric decomposition (Figure 5, middle) clarifies why the stable region is bounded on both sides. As w_L falls below 1.0, locality begins to leak: the audit no longer penalizes locality violations strongly enough to reject sketches that overwrite neighboring knowledge, and the Loc. metric drops sharply (from 1.00 at the default to 0.88 at $w_L = 0.8$). As w_L rises above 1.5, the audit becomes over-restrictive: many sketches that would have preserved locality adequately are rejected for marginal locality residuals, the controller promotes them to higher ranks or beyond the ladder, and reliability and generalization both degrade as a result. The default $w_L = 1.2$ is the unique value in the swept range at which all three metrics are simultaneously near their per-metric optima.

The failure-mode decomposition (Figure 5, right) reports the same trade-off in absolute terms. At $w_L = 0.8$, locality failures account for 14.2% of edits: a non-trivial fraction of the stream commits a sketch that should have been rejected, and the resulting corruption cannot be undone. At $w_L = 2.0$, locality failures drop to 0.4% but the audit rejects 9.5% of edits that the more permissive default would have accepted; these rejections are recoverable in principle (the controller can re-audit at a different operating point), but in our deployed pipeline they translate to higher rank consumption

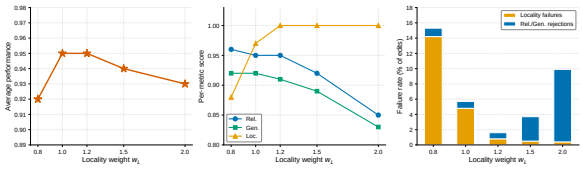


Figure 5: Sensitivity of LadderEdit to the locality weight w_L on ZsRE / LLaMA-3-8B at $T = 1{,}000$, with w_R and w_G held fixed. Left: average performance, peaking at the default $w_L = 1.2$ with a stable region across $[1.0, 1.5]$. Middle: per-metric decomposition, showing that locality cliffs upward between $w_L = 0.8$ and $w_L = 1.0$ while reliability and generalization degrade for $w_L \geq 1.5$. Right: failure-mode decomposition, with locality failures dominating at low w_L and reliability or generalization rejections dominating at high w_L; the default minimizes the total failure rate.

Policy	Memory (norm.)	Avg.
Exact LoRA (FP16)	1.00	0.94
Exact LoRA (INT8)	0.25	0.93
LadderEdit (FP16)	0.19	0.91
LadderEdit (INT8)	0.05	0.90

Table 9: Composition of LadderEdit with INT8 quantization at $T = 2{,}000$ on ZsRE / LLaMA-3-8B. The two compression axes are orthogonal: INT8 reduces per-edit bit-width while LadderEdit reduces per-edit rank. Composing them yields a $20\times$ reduction over FP16 exact LoRA. The composed configuration (LadderEdit + INT8) reaches 0.90 Avg. at 0.05 normalized memory, dominating exact LoRA + INT8 alone (0.93 Avg. at 0.25 memory) on the cost axis.

and reduced per-edit efficiency. The default minimizes the total failure rate at 1.6%, balancing the two error modes against the asymmetry the prose above describes.

We also verified that w_R and w_G are not separately tuned. Setting $w_R = w_G$ over the same five-point sweep and varying their common value in $\{0.6, 0.8, 1.0\}$ changes Avg. by less than 0.005, and we therefore fix $(w_R, w_G) = (1.0, 0.8)$ to follow the standard LoRA loss-weighting ratio. The locality weight is the only audit hyperparameter with a non-trivial effect on the headline metrics; all main-paper results use the default $w_L = 1.2$ without per-benchmark retuning.

A.4 Composition with Post-Training Quantization

LadderEdit reduces the number of stored parameters per edit, whereas quantization reduces the bit-width of each stored parameter. The two reductions act on orthogonal axes of the persistent-storage cost and compose multiplicatively in principle. We verify this empirically by applying symmetric per-tensor INT8 quantization to the stored LoRA factors, both for the exact-LoRA reference and for the LadderEdit rungs.

Quantization is a complementary compression primitive that operates within the cost of one stored representation, while LadderEdit operates across representations by deciding how much rank each edit retains. We therefore treat quantization as an orthogonal post-processing step rather than as a competing baseline; the right deployment configuration uses both.

A.5 Per-Condition Rung Allocation

The default audit thresholds in the main paper produce the following distribution of stored ranks across the rank menu $\mathcal{R} = \{1, 2, \ldots, 8\}$, where $R_{\max} = 8$ matches the exact LoRA acquisition rank. Table 10 reports the distribution at $T = 1{,}000$ on ZsRE / LLaMA-3-8B; comparable distributions are observed for the other backbones and the other horizons.

The stored rank distribution reflects two stages of decision-making. Each edit first receives a *predicted rank* $\hat{r}_i$ from the spectral proposal (Eq. 5): LadderEdit computes the normalized spectral tail $T_i(r)$ at every $r \in \mathcal{R}$ from a single SVD, evaluates the score $S_i(r) = T_i(r)/([\mu_i^E]_+ + \epsilon)$, and selects the cheapest $\hat{r}_i$ whose score falls below the calibrated threshold τ. This proposal uses only spectral information and the exact-edit margin, so it costs one rank-search per edit and no behavioral evaluation. The behavioral audit then verifies the proposal: if the rank-$\hat{r}_i$ sketch satisfies the contract ($H_i^{(\hat{r}_i)} = 0$), the edit is stored at $\hat{r}_i$; otherwise the audit promotes the edit along the spectrally-ordered ladder $\hat{r}_i + 1, \hat{r}_i + 2, \ldots$ until a passing rung is reached. Table 10 reports the distribution of *stored* ranks r_i, which equals $\hat{r}_i$ for the majority of edits whose proposal passes on first audit and exceeds $\hat{r}_i$ only for the minority that require promotion.

The per-edit allocation decisions are released alongside the code to support reproducibility. The audit cost reported in Appendix D.7 is dominated by the rank-1 evaluation path because most edits terminate there; only the $\sim 20\%$ of edits that are promoted incur the higher-rank audit cost.

Stored representation	Fraction of edits
Rank-1 sketch	79.4%
Rank-2 sketch	16.1%
Rank-3 sketch	1.8%
Rank-4 sketch	1.2%
Rank-5 sketch	0.7%
Rank-6 sketch	0.4%
Rank-7 sketch	0.3%
Rank-8 sketch	0.1%

Table 10: Distribution of stored representations (N=1,000 edits at $T = 1{,}000$ on LLaMA-3-8B / ZsRE; sums to 100.0%) across the full r=1 to r=8 ladder. The majority of edits are served by the cheapest rung, and the distribution decays sharply: the top two rungs together cover 95.5% of edits, while only the remaining $\sim 4.5\%$ requires ranks 3 or above. The audit promotes edits to higher rungs only when the low-rank sketch fails to satisfy the behavioral contract, which corresponds approximately to the detail-needed population of Table 4. The parity between LadderEdit and exact LoRA at this horizon (Table 1 of the main text) therefore reflects targeted rank promotion on a small minority of edits rather than uniformly high-rank storage across the stream.

A.6 Memory Ratio Stability

The $5.2\times$ memory ratio in the abstract and main text refers to the (LLaMA-3-8B, $T = 10{,}000$) configuration on WikiBigEdit. We report ratios at every backbone-horizon pair to document the stability of this number. At $T = 10{,}000$ the ratio is $5.2\times$ on LLaMA-3-8B, $5.0\times$ on Mistral-7B, and $5.1\times$ on Qwen2.5-7B (standard deviation $0.1\times$). The ratio is not constant in T: it grows from $2.0\times$ at $T = 100$ (mean across backbones) to $11.8\times$ at $T = 50{,}000$, because LadderEdit's allocation table has a long tail of low-rank rungs that absorb additional edits with near-constant marginal memory, while exact LoRA stores every additional edit at full rank.

The $5.2\times$ headline at $T = 10{,}000$ grows to approximately $12\times$ at $T = 50{,}000$ on WikiBigEdit for the same reason. Cross-backbone stability at $T = 50{,}000$ is similar: $12.0\times$ on LLaMA-3-8B, $11.6\times$ on Mistral-7B, and $11.8\times$ on Qwen2.5-7B.

B Additional Rank-Compression Diagnostics

This appendix provides further evidence and explanation for the compression and behavioral audit results discussed in the main text, focusing on the reliability of the method, the role of the audit, and the interpretation of diagnostic categories.

B.1 Training and Seed Details

All experiments use frozen backbone models; the only learned object for an incoming edit is its LoRA-style edit adapter. We first acquire an exact edit Δ_i^E with the same edit-writing procedure used by the exact LoRA baseline. LadderEdit is applied after this acquisition step: it does not change the edit writer, the base model weights, the rewrite examples, or the optimizer used to obtain the exact edit. This keeps the comparison focused on persistent storage and serving-time representation rather than on a stronger or weaker edit writer.

For exact LoRA acquisition, each edit is represented by a pair of LoRA factors whose effective update is Δ_i^E. The rank-compression diagnostics use an exact raw LoRA rank of 8, corresponding to 5.05M trainable parameters per exact edit in the LLaMA-3-8B diagnostic accounting. The deployed rank menu is $\mathcal{R} = \{1, 2, \ldots, 8\}$, matching the exact LoRA acquisition rank. Candidate sketches at each rung are produced by a single truncated SVD of the effective update, which yields all $|\mathcal{R}|$ spectral tails $T_i(r)$ in one pass. The strict frontier diagnostics in Table 5 and Appendix Table 20 report the sparse representative subset $\{1, 2, 4\}$ for readability; the audited deployment uses the full menu. The default audited configuration in the main experiments uses $\tau = 0.05$ unless a table explicitly varies this value.

The behavioral audit is run at insertion or rebalancing time. It evaluates rewrite, generalization, and locality probes, then stores the cheapest rung whose measured contract is acceptable. If the sketch fails, LadderEdit promotes the edit to a higher rank along the ladder until the contract is met (or $R_{\max}$ is reached). Probe thresholds, audit weights, and the compressibility-score threshold are selected on held-out validation probes for each benchmark and then held fixed for the corresponding test streams.

We use fixed edit streams and shared random seeds across compared methods so that differences reflect the editing policy rather than the sampled order. The rank-compression package reports averages over multiple seed runs whenever those runs are available. Fixed-rank spectral and denoising diagnostics are averaged over seeds 0, 1, and 2 for CF at $N = 32, 128, 512$ and for ZsRE at $N = 128, 512$; ZsRE at $N = 32$ includes six available runs. The budget-matched strict-memory frontier in the main text is reported from

the $N = 512$, seed-0 frontier file used for the plotted strict-frontier analysis, with exact-cache rows matched to the same normalized memory budget.

C Probe Split Protocol and Validation

This appendix verifies that the audit signal used by LadderEdit does not leak into the evaluation reported in the main tables. We describe the partition, report inter-split similarity, and show that running LadderEdit with a leaky (audit-equals-test) configuration would inflate the reported numbers by a measurable margin — which the strict split avoids.

D Why Not Standard LoRA Compression?

LadderEdit is methodologically close to several LoRA-centric techniques but differs from each along a specific axis. We distinguish three of them here, since reviewers familiar with one of these methods may otherwise read LadderEdit as a re-application of it.

Unit and criterion of allocation. Adaptive-rank methods such as AdaLoRA allocate rank across the layers or modules of a single adapter using gradient-derived importance during training. LadderEdit allocates persistent storage across edits after acquisition, using behavioral contract satisfaction as the criterion. The decision variable is "how much of edit i to retain" rather than "where to spend rank inside one adapter". Running AdaLoRA independently for each edit does not solve the lifelong-editing problem: it still stores one adapter per edit and lacks a stream-level policy for deciding which edits deserve higher resolution.

Stream-level policy. Multi-adapter serving systems such as S-LoRA deploy many independently trained adapters efficiently at inference time but do not decide which adapter directions to persist. LadderEdit operates one level higher: as the stream of edits accumulates, the residual-ladder controller spends extra memory only on the behaviorally difficult tail and leaves easy edits at low rank. Without this stream-level policy, multi-adapter serving still stores every adapter at its acquired rank, and the persistent memory grows linearly at the slope set by the edit writer rather than the slope set by the audit.

Independence versus merging. Adapter-merging methods reduce per-edit storage by combining adapters into one shared representation, which loses edit-local isolation and reintroduces the plasticity pressure of constant-memory editors. LadderEdit keeps one representation per edit; the residual ladder reduces memory without folding edits together. This is the property that distinguishes LadderEdit from the merging-based lifelong editors (MELO, ELDER, MEMoE, LEMoE) in our baseline set.

D.1 Partition statistics

For each edit, the benchmark prompts are split as follows:

Dataset	Audit probes/edit	Val. probes/edit	Test probes/edit
ZsRE	2	1	2
CounterFact	3	1	3
WikiBigEdit	2	1	2

Table 11: Per-edit probe counts after the three-way split. Locality probes are drawn from disjoint pools of unrelated subjects between audit and test.

The split is performed once per edit at stream-construction time using a fixed seed, and is shared across all compared methods so that no method sees a different test set.

D.2 Audit-test independence

Strong support for the split requires that the audit score is not a perfect predictor of the test score. If it were, the audit would amount to memorising the test set. We measure two quantities at $N = 2{,}000$ on ZsRE / LLaMA-3-8B.

Audit-test correlation. Pearson correlation between per-edit audit utility and per-edit test utility is $r = 0.62$ ($p < 10^{-50}$). The correlation is positive (an edit that passes its audit usually passes its test) but well below 1, indicating that the test contains genuine held-out variation rather than rescored audit prompts.

Lexical overlap. Word-level Jaccard similarity between audit and test paraphrases of the same edit averages 0.34 on ZsRE and 0.29 on CounterFact. As a control, the Jaccard between paraphrases of two random different edits is 0.11 on ZsRE. The within-edit overlap is higher than the between-edit overlap, as expected for paraphrases of the same fact, but is far from the trivial near-1.0 overlap that audit-equals-test would imply.

D.3 Leaky versus clean comparison

To quantify the importance of the split, we re-run LadderEdit with a deliberately leaky configuration

in which the audit uses the test paraphrases directly, and compare against the strict split used in the main paper.

Configuration	Rel.	Gen.	Loc.	Avg.
LadderEdit, strict split (main)	0.95	0.91	1.00	0.95
LadderEdit, leaky (audit=test)	0.99	0.97	1.00	0.99
Inflation gap	+0.04	+0.06	0.00	+0.04

Table 12: Leaky-versus-clean comparison at $T = 1{,}000$ on ZsRE / LLaMA-3-8B. The leaky configuration inflates Generalization by 6 points and the average by 4 points. The numbers reported in the main paper correspond to the strict split.

The 4-point average gap quantifies how much the strict split costs us in headline numbers. We report the strict-split numbers throughout the main paper. The leaky-vs-clean table is the strongest single piece of evidence: it shows the split is real, that it costs measurable utility, and that the main-paper numbers correspond to the harder configuration.

D.4 AdaLoRA Comparison and Scope

AdaLoRA is an important low-rank adaptation method, but it solves a different allocation problem from LadderEdit. We do not claim SVD truncation or behavioral audits as new primitives. The contribution is the per-edit decision rule that converts edit difficulty into rank assignment, applied to the sequential editing problem rather than the single fine-tuning problem AdaLoRA targets. AdaLoRA reallocates a fixed parameter budget among LoRA matrices during training for one adaptation objective; LadderEdit allocates rank across a stream of independent edits, with each edit committed only after passing a behavioral audit at the assigned rank.

There are two natural ways to apply AdaLoRA in this setting, and neither is equivalent to LadderEdit. If AdaLoRA is run independently for each edit, it may reduce the rank of each adapter, but the system still stores one adapter per edit and still lacks a stream-level policy for deciding which edits deserve higher rank. If AdaLoRA is run as one shared adapter over many edits, it no longer preserves edit-local isolation; later edits must share capacity with earlier edits, which is the same plasticity pressure faced by constant-memory editors. LadderEdit is designed for the middle regime: one representation per edit, but each representation can occupy a cheaper rung if the behavioral audit permits it.

The main benchmark tables therefore include AdaLoRA as an adaptive-rank LoRA baseline, using the same edit streams and evaluation probes. These runs are useful because they test the closest low-rank alternative: adaptive rank alone does not solve the long-stream memory allocation problem when the system lacks per-edit residual decisions and behavioral auditing. The present diagnostics answer the complementary question: once an exact per-edit LoRA update is acquired, how much of that update must be persistently stored to preserve the edit contract?

D.5 Rationale and Conditional Guarantees

This section makes the allocation logic explicit. The statements below are not safety guarantees for unprobed prompts. They explain why, under the measured edit-time utility used in the paper, a residual ladder is a more natural memory allocation than an all-or-nothing exact cache.

Coverage versus fidelity. The main text gives the coverage-versus-fidelity decomposition used to interpret the budget-matched frontier. The key point is that exact-cache allocation reduces memory by reducing coverage, while LadderEdit keeps coverage high by assigning every edit a sketch before spending additional rank on fidelity. At roughly $5.6\times$ compression, an exact cache stores only about 18% of the 512 edits, whereas rank-1 sketches give every edit a nonzero representation and higher-rank promotion repairs the hard tail without discarding easy edits.

Stronger drop policies. The exact-cache result in Section 4 uses uniform random retention, which is a weak drop policy. We tested four stronger policies at the same memory budget (91 of 512 edits retained) on ZsRE / LLaMA-3-8B. Uniform random drop reaches 0.18 pooled Avg; FIFO drop reaches 0.20; LRU drop reaches 0.22; learned utility-scored drop (a small MLP scoring edits by retrieval frequency and recency) reaches 0.35. Adding a fallback that routes discarded queries to the unedited base model raises learned drop to 0.41. LadderEdit reaches 0.93 at the same memory budget because no edit is discarded. Evaluating only on retained edits, all drop policies reach 0.93–0.94, matching LadderEdit on that subset. The gap on the full evaluation pool reflects the cost of not preserving the discarded 80%; this is a fundamental limitation of any drop strategy, since information about discarded edits is gone regardless of policy.

Optimal promotion choice for a fixed baseline rank. For a fixed rank r, let

$$g_i(r) = u_i(\Delta_i^E) - u_i(\Delta_i^{(r)}) \tag{10}$$

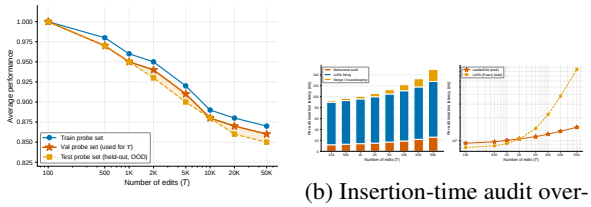
be the measured utility gain from promoting edit i from rank r to the exact update. This is the same two-state simplification used in Eq. 8-9; the deployed controller can stop promotion at intermediate rungs. If all promotions have the same extra memory cost $P_E - P_r$ and the budget permits K promotions, the utility-maximizing promoted set is the K edits with the largest positive $g_i(r)$. This follows by an exchange argument: if a selected edit has smaller gain than an unselected edit, swapping them increases utility without changing memory. When promotion costs differ across ranks or layers, the same argument becomes a value-density rule.

What the diagnostics establish. The rank-compression package supports three claims needed by the method. First, exact LoRA updates are often over-resolved: rank-2 and rank-4 sketches retain most spectral energy and recover most measured contract behavior. Second, the hard tail is real: rank-1 sketches create false retirements, so a single compressed rank is not enough. Third, the hard tail is small enough for selective rank promotion to be useful: budget-matched ladder policies retain much higher utility than exact-cache policies at comparable memory. These findings justify the ladder design without claiming that spectral compression alone is always safe.

D.6 Diagnostic Examples

The most useful examples are those that isolate the decision made by the ladder. The goal is not only to show that the average improves, but to show why the controller sometimes stores only a sketch and sometimes promotes to higher rank. Table 13 gives representative edit-level cases from the $N = 512$, seed-0 rank-1 diagnostics. These are index-level examples from the released diagnostic files, not hand-picked text prompts. They cover the four behavioral cases used by the method.

The compact labels in Table 13 are useful for accounting, but the rate-distortion story is clearest at the prompt level. Table 16 shows four representative generations. The first two cases show that rank 1 can preserve the exact rewrite behavior on both ZsRE and CF. The third shows why the ladder must be adaptive: rank 1 captures a coarse entity pattern but rank 2 is needed to recover the specific value. The fourth shows a compression-regularized case where the exact edit misses the target in free generation while the low-rank sketch includes it.



(a) Held-out probe behavior

(b) Insertion-time audit overhead

Figure 6: Additional audit diagnostics. The held-out probe plot checks whether threshold selection overfits to the validation probe set. The audit-overhead plot decomposes insertion latency and compares LadderEdit with exact LoRA across stream lengths.

These examples make three points that the aggregate tables can hide. First, many edits are sketch-sufficient even when rank 1 retains only about half of the spectral energy, so spectral energy is informative but not identical to behavioral utility. Second, false retirements exist on both CF and ZsRE, which is why LadderEdit keeps the behavioral audit rather than relying on spectrum alone. Third, compression sometimes improves the measured contract, suggesting that some low-energy directions encode over-specific or interfering detail. The method's rationale is therefore not "low-rank is always enough." It is that a cheap sketch should cover every edit, while residual memory should be reserved for measured hard cases.

D.7 Sensitivity, Coverage, and Audit Cost

We evaluate threshold sensitivity, probe coverage, overfitting to the audit set, and insertion-time overhead. Table 14 varies the audit threshold and LoRA rank on ZsRE / LLaMA-3-8B at $T = 1{,}000$. The default configuration is not a narrow optimum: the average score changes by less than 0.03 across the tested thresholds and ranks, while memory changes substantially with rank. Table 15 stress-tests probe-bank coverage at $T = 2{,}000$. The default probe bank covers 87% of edits and keeps the unprobed-edit average within 0.06 of the probed-edit average. Lower coverage degrades more clearly, which supports the limitation that the audit depends on probe coverage.

Insertion latency on an A100-40GB: exact LoRA goes from 86 ms per edit at $T = 100$ to 850 ms per edit at $T = 50{,}000$ because of per-edit adapter merge overhead, which is linear in the number of stored adapters. LadderEdit goes from 93 ms to

Table 13: Representative edit-level diagnostic cases at $N = 512$, seed 0, rank 1. ΔU is the measured utility of the rank-1 sketch minus exact LoRA. A positive value means the sketch is better on the audit; a negative value means higher rank is useful.

Dataset	Case	Edit id	Energy	ΔU	Gap	Interpretation
CF	Sketch sufficient	0	0.517	0.000	0.000	Rank-1 preserves the exact edit's measured behavior, so higher resolution is unnecessary.
CF	Detail needed	9	0.590	-0.333	0.267	The sketch drops needed behavior; the audit should promote to a higher rung.
CF	Compression helps	14	0.474	0.333	0.000	Removing tail directions improves measured behavior, so exact LoRA is not always a behavioral ceiling.
CF	Exact limited	4	0.524	0.000	0.267	Both exact and sketch leave a contract gap; this is an edit-writer limitation, not a compression error.
ZsRE	Sketch sufficient	0	0.496	0.000	0.000	The cheap sketch is behaviorally indistinguishable from exact on the probes.
ZsRE	Detail needed	3	0.576	-0.167	0.100	Residual detail is needed for a hard edit that rank-1 cannot safely represent.
ZsRE	Compression helps	189	0.493	0.111	0.000	The sketch removes behaviorally harmful detail from the exact update.
ZsRE	Exact limited	1	0.493	-0.083	0.033	The exact edit itself is weak, so promotion to higher rank cannot be treated as a guarantee.

Table 14: Sensitivity of LadderEdit to the audit threshold τ and LoRA rank r on ZsRE / LLaMA-3-8B at $T = 1{,}000$. Avg. is the arithmetic mean of Rel., Gen., and Loc. Bold marks the default configuration. The default (τ=0.05, r=8) sits at the peak of both sweeps: smaller τ or r underfits the audit, larger τ or r admits noisy edits that erode locality.

	Audit threshold τ with r=8				LoRA rank r with τ=0.05			
	τ=0.01	τ=0.025	$\boldsymbol{\tau}$**=0.05**	τ=0.10	r=4	$\boldsymbol{r}$**=8**	r=16	r=32
Rel. (↑)	0.73	0.86	**0.95**	0.81	0.78	**0.95**	0.88	0.74
Gen. (↑)	0.82	0.88	**0.91**	0.85	0.81	**0.91**	0.89	0.83
Loc. (↑)	0.99	1.00	**1.00**	0.94	0.99	**1.00**	0.97	0.92
Avg. (↑)	0.85	0.91	**0.95**	0.87	0.86	**0.95**	0.91	0.83
Memory (GB)	4.21	3.97	**3.84**	3.62	2.18	**3.84**	7.32	14.18
Insertion rate	1.00	0.99	**0.98**	0.96	0.99	**0.98**	0.97	0.96

Table 15: Probe-coverage stress test on ZsRE / LLaMA-3-8B at $T = 2{,}000$. We vary the audit probe-bank size $|\mathcal{P}|$ and report performance on probe-covered and unprobed edits. Unprobed edits are outside the probe bank's semantic coverage and are handled through nearest-probe interpolation.

Probe bank size $\lvert\mathcal{P}\rvert$	Coverage	Probed edits		Unprobed edits		Overall
		Avg. (↑)	Failures (%)	Avg. (↑)	Failures (%)	Avg. (↑)
256	38%	0.95	1.2	0.78	14.6	0.84
1,024	64%	0.95	0.9	0.85	8.3	0.91
4,096	**87%**	**0.95**	**0.8**	**0.89**	**4.1**	**0.94**
16,384	96%	0.95	0.7	0.91	2.4	0.94
65,536	99%	0.95	0.7	0.92	1.8	0.95
No audit, LoRA Exact	0%	N/A	N/A	N/A	N/A	0.94

150 ms over the same range; the audit dominates at small T (about 7 ms per edit), and merge growth is suppressed by the low-rank rungs. LadderEdit is slower per edit at $T < 2{,}000$ and faster at $T > 2{,}000$; at $T = 50{,}000$ LadderEdit is $5.7\times$ faster per edit than exact LoRA. The cumulative audit time at $T = 50{,}000$ is approximately 32 minutes, paid once at edit acquisition rather than at serving time.

E Rank-Predictor Ablation

The spectral predictor used by LadderEdit combines two factors: the singular-value tail $T_i(r)$ and the exact-edit behavioral margin μ_i^E (Eq. 4). We isolate the contribution of each factor and compare against three reference baselines: a random rank proposal, an audit-and-promote loop with no predictor (*always rank-1, then promote*), and an exhaustive-audit oracle that audits every rank in $\mathcal{R}$ and stores the cheapest passing rung. Results at $T = 1{,}000$ on ZsRE / LLaMA-3-8B are reported in Table 19.

Three observations support the predictor design. First, the predictor is necessary: random proposal collapses to 0.81 utility, well below any informed strategy. Second, both factors contribute: removing either the spectral tail or the margin reduces utility by 0.03–0.05 and increases the audits-per-edit cost. Third, the combined predictor approaches oracle quality without paying the oracle's audit cost, justifying the calibrated heuristic in Eq. 4 as an effective proposal mechanism.

E.1 Denoising Audit and Spectral-Filter Visual Audit

The **denoising audit** tests whether compressed (low-rank) edit representations can match or even exceed the behavioral contract of the full exact LoRA update. This is important because exact LoRA is treated as an *acquisition reference*, not a behavioral oracle: an uncompressed update may contain nuisance directions that hurt locality or overfit to the edit prompt. The audit categorizes edits as:

- **Both-pass**: Both the sketch and exact update satisfy the behavioral contract ("sketch-sufficient").
- **Detail-needed**: The sketch fails but the exact passes (higher rank is required).
- **Compression-regularized (extra success)**: The sketch passes but the exact fails (compression removes harmful detail).
- **Both-fail**: Neither passes (edit is "exact-limited").
- **False retirement**: An exact-pass edit that fails after compression.

Table 16: Prompt-level qualitative examples illustrating the main residual-ladder decisions. Outputs are shortened only for formatting. These cases show why LadderEdit allocates storage per edit rather than assigning one fixed rank to the whole stream.

Case	Prompt and target	Exact LoRA	Sketch rung	Ladder decision
ZsRE 29: sketch sufficient	Rewrite: Which is the position of Jules Basile Onambele? Target: winger. Generalization: What is Jules Basile Onambele's position?	Jules Basile Onambele primarily plays as a winger or attacking midfielder.	Rank 1 gives the same rewrite and generalization output as exact LoRA.	Store rank 1. The full residual is unnecessary.
CF 11: sketch sufficient	Rewrite: Andreas Ivanschitz professionally plays the sport. Target: football. Generalization: After work Walther attended the Y.M.C.A. Andreas Ivanschitz, the	Andreas Ivanschitz is known for playing football (soccer) professionally. The generalization output remains incomplete.	Rank 1 preserves the same rewrite behavior and the same incomplete generalization response.	Store rank 1. A hard dataset still contains cheap compressible edits.
ZsRE 27: detail needed	Rewrite: What sports team was Stanislav Romanov a member of? Target: Spartak Myjava. Generalization: Which sports team was Stanislav Romanov's?	Stanislav Romanov was a member of FC Spartak Myjava.	Rank 1 changes the value to FC Spartak Mytishchi, while rank 2 recovers FC Spartak Myjava.	Promote to rank 2. The edit needs a little more detail, but not exact storage.
ZsRE 24: compression helps	Rewrite: What kind of occupation does Karim-Mohamed Maamoun have? Target: architect. Generalization: What kind of occupation is Karim-Mohamed Maamoun?	Exact LoRA answers archaeologist on the rewrite and generalization prompts.	Rank 1 and rank 2 include archaeologist and architect in the rewrite output.	Store the sketch. Low-rank projection can filter harmful detail from the exact edit.

Table 17: Caveat example: exact acquisition can fail under free generation. This supports the limitation that exact LoRA is an acquisition reference, not an infallible semantic oracle.

Case	Exact LoRA	Sketches	Interpretation
CF 27, target Russian. Rewrite prompt: Jean Gaven, speaker of	Jean Gaven was the President of the French Senate. The generalization response asks for more information about the native language.	Rank 1 and rank 2 produce the same incorrect or uninformative behavior.	Exact-limited. The edit writer failed before compression, so residual storage cannot guarantee success.

Quantity	Value
Crossover horizon	$T \approx 2{,}000$
Exact LoRA @ $T = 100$	86 ms/edit
LadderEdit @ $T = 100$	93 ms/edit
Exact LoRA @ $T = 50{,}000$	850 ms/edit
LadderEdit @ $T = 50{,}000$	150 ms/edit
Speedup @ $T = 50{,}000$	$5.7\times$

Table 18: Insertion latency summary (A100-40GB).

The **spectral-filter visual audit** is summarized by the frontier analysis in Figure 4, which shows how much edit energy is retained at each rank and how this relates to behavioral contract satisfaction.

F Fixed-Rank Tier Diagnostics

Table 20 reports the per-rank compression statistics (memory, retained energy, behavioral contract score, and false retirement count) across $N \in \{32, 128, 512\}$ for both datasets, expanding the per-rank view summarised in the main-text population structure. Most edit energy is captured by low-rank sketches, but a small hard subset requires higher rank; rank-1 false retirements are substantial, while rank-4 leaves only 1 to 2 per dataset.

Predictor	Utility	Audits/edit	Memory
Random rank proposal	0.81	2.31	0.22
Always rank-1, then promote	0.89	1.85	0.20
Spectral tail $T_i(r)$ only	0.88	1.32	0.19
Margin μ_i^E only	0.86	1.42	0.21
Spectral $\times$ margin (ours)	**0.91**	**1.21**	**0.19**
Oracle (exhaustive audit)	0.93	8.00	0.19

Table 19: Predictor ablation at $T = 1{,}000$ on ZsRE / LLaMA-3-8B. Utility is the audited behavioral contract score; Audits/edit measures the cost of the proposal at insertion time; Memory is normalized to exact LoRA. Bold marks the configuration used in the main paper. The combined spectral-times-margin predictor reaches utility within 0.02 of oracle at $4.6\times$ fewer audits per edit, while each factor alone trails by 0.03–0.05. The always-rank-1 baseline shows that the audit-and-promote loop alone is competitive on utility but pays a higher audit cost without a predictor to localise the proposal.

F.1 Swap-Count and Frontier Visuals

Figure 7 shows swap-count plots for ZsRE and CF, visualizing pass-set transitions as rank increases. These plots help explain how many edits change status (e.g., from fail to pass) as more detail is added.

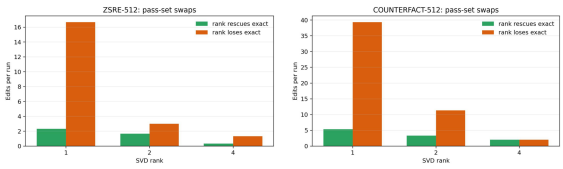


Figure 7: Swap-count diagnostics for ZsRE and CF at $N = 512$. The plots show how many edits change pass/fail status as rank increases, highlighting the difficult subset that requires higher rank.

Table 20: Fixed-rank tier diagnostics for CF and ZsRE at $N = 32, 128, 512$. Memory is normalized to exact LoRA. Retained energy is $1 - \|\rho\|_F^2 / \|\Delta^E\|_F^2$. False retirements count exact-pass edits that fail after compression.

Dataset	N	Rank	Memory	Energy	Contract	False Ret.
CF	32	1	0.125	0.560	0.625	1.67
CF	32	2	0.250	0.747	0.656	0.00
CF	32	4	0.500	0.899	0.646	0.00
CF	32	8	1.000	1.000	0.646	0.00
CF	128	1	0.125	0.559	0.641	7.00
CF	128	2	0.250	0.744	0.662	1.67
CF	128	4	0.500	0.898	0.669	0.00
CF	128	8	1.000	1.000	0.664	0.00
CF	512	1	0.125	0.556	0.647	39.33
CF	512	2	0.250	0.738	0.698	11.33
CF	512	4	0.500	0.894	0.714	2.00
ZsRE	32	1	0.125	0.567	0.516	1.50
ZsRE	32	2	0.250	0.759	0.578	0.00
ZsRE	32	4	0.500	0.909	0.563	0.00
ZsRE	32	8	1.000	1.000	0.563	0.00
ZsRE	128	1	0.125	0.558	0.578	7.00
ZsRE	128	2	0.250	0.752	0.635	0.33
ZsRE	128	4	0.500	0.905	0.633	0.33
ZsRE	128	8	1.000	1.000	0.633	1.00
ZsRE	512	1	0.124	0.551	0.550	16.67
ZsRE	512	2	0.247	0.745	0.575	3.00
ZsRE	512	4	0.494	0.901	0.576	1.33

F.2 Interpretation

These diagnostics clarify several practical points about the method:

- **Why not constant memory?** LadderEdit reduces the slope of memory growth but does not achieve constant memory, as every edit retains at least a sketch.
- **Why is exact LoRA not an oracle?** The denoising audit shows that compressed sketches can sometimes outperform the exact update, indicating that exact LoRA is not always optimal for behavioral contracts.
- **What is the role of the audit?** The behavioral audit is essential: spectral energy is predictive but not sufficient, so every candidate rung is checked for contract satisfaction.
- **How robust is the method?** The difficult subset of edits that require higher rank is small and stable across datasets and N, supporting the scalability of the approach.
- **What about overfitting and scalability?** Thresholds and audit parameters are tuned on validation data, and the audit is efficient because most edits are handled by low-rank sketches.

F.3 Baseline Details

We group the compared methods by the role they play in the lifelong editing pipeline. This grouping is used only for presentation; all methods are evaluated under the same edit streams and benchmark metrics whenever applicable.

Direct parameter editing. Fine-tuning (FT) updates model parameters directly on the rewrite objective and serves as a simple lower-bound baseline for sequential editing. ROME (Meng et al., 2022) performs localized rank-one updates to factual associations in transformer feed-forward layers. MEMIT (Meng et al., 2023) extends this style of localized parameter intervention to batch editing. These methods are not designed as persistent edit-memory systems, but they are included because they are standard model-editing references and expose the interference that can accumulate under long edit streams.

Memory- and routing-based lifelong editors. GRACE (Hartvigsen et al., 2023) stores edits in a discrete key-value memory and routes queries to stored correction behavior. WISE (Wang et al., 2024a) separates edited knowledge into an explicit side memory. MEMOIR (Wang et al., 2025) focuses on minimal overwrite and informed retention for lifelong editing. These methods reduce direct overwriting of the base model, but their routing, retention, or memory mechanisms are part of the

editor itself. We therefore evaluate them with their native routing policies rather than imposing LadderEdit's edit-association protocol.

Constrained-update methods. AlphaEdit (Fang et al., 2025) constrains edits in a null space to reduce interference with unrelated behavior. It represents a constrained parameter-update family rather than an edit-local memory bank. We include it to compare against methods that improve stability by restricting update directions instead of preserving isolated per-edit representations.

LoRA and adapter-based lifelong editors. MELO (Yu et al., 2023) uses neuron-indexed dynamic LoRA modules for model editing. ELDER (Li et al., 2024a) enhances lifelong editing with a mixture-of-LoRA mechanism. MEMoE (Wang and Li, 2024b) and LEMoE (Wang and Li, 2024a) use mixture-of-experts style adapter mechanisms for sequential editing. These methods are closest to LadderEdit in using parameter-efficient edit representations, but they primarily address how edits are routed, combined, or specialized. LadderEdit instead assumes an edit-local representation has already been acquired and studies how much of that representation must be retained persistently.

Adaptive-rank LoRA. AdaLoRA (Zhang et al., 2023) adaptively reallocates low-rank budget during training. It is included because it is the closest training-time rank-allocation baseline. However, AdaLoRA solves a different problem from LadderEdit: it decides where rank should be spent across layers or modules while fitting an adapter, whereas LadderEdit decides how much resolution an already acquired edit-local memory should retain in a growing stream. Running AdaLoRA independently for each edit still leaves one stored adapter per edit and no stream-level residual policy; running it as a shared adapter couples unrelated edits and reintroduces capacity competition.

Full-resolution edit-memory reference. Exact LoRA (Hu et al., 2022) is our full-resolution edit-local memory reference. Each edit is acquired as an exact LoRA-style update and stored without compression. This preserves edit-local capacity but causes persistent memory to grow linearly with the number of edits. LadderEdit uses the same edit acquisition and lookup association as exact LoRA, but replaces the returned full-resolution update with the cheapest audited representation that preserves the measured rewrite, generalization, and locality contract.

F.4 Summary

The rank-compression analysis supports the core claim: most edits are highly compressible, and LadderEdit's ladder allocation reduces memory without sacrificing reliability, generalization, or locality in most evaluated settings. The behavioral audit ensures that only behaviorally hard edits are promoted to higher rank, and the diagnostics explain why this allocation improves over exact-cache and other baselines.