WHAT TO WRITE: STRUCTURED SPARSE WRITING FOR TEST-TIME TRAINING

Anonymous authors
Paper under double-blind review

ABSTRACT

Test-time training turns an input stream into fast weights through successive memory writes. Each write carries a decision that is usually left implicit: when only a fraction of the available contributions can enter memory, which ones should, and which ones should enter together? This allocation shapes the composition of a write alongside the update rule and chunk schedule: contributions enter memory jointly, and their combination determines the state available to subsequent reads. We introduce *Structured Sparse Writing* (SSW), a model-independent operator for TTT write allocation. It scores contiguous blocks of contributions with signals the update path already produces, selects the highest-scoring blocks under a budget, and gathers their contributors in temporal order before invoking the original write. SSW uses existing update signals and preserves the full read stream, target construction and commit schedule, so one interface precedes both optimizer-based and additive memories while exposing the selection unit, the ranking signal and the budget as separate design choices. We prove a tight worst-case omission guarantee for norm-scored additive contributions under a block budget and characterize how selection granularity changes the feasible allocations. A complementary pairwise decomposition describes the joint effects of retained contributions. We instantiate the same operator in optimizer-based and additive TTT memories. In the optimizer-based evaluation, SSW retains one eighth of the contributions and improves retrieval over dense writing by 6.56 and 2.12 points at 16K and 32K. Matched-budget controls in that evaluation show the importance of ranking and grouping: block selection outperforms token-wise score maximization. Accuracy and answer likelihood favor different budgets on identical inputs. These results establish write allocation as a design axis of test-time training, where the value of a write budget depends on the retained combinations and the intended readout.

1 INTRODUCTION

Test-time training (TTT) represents context through fast weights that change as a model processes an input stream (Sun et al., 2025; Behrouz et al., 2025). A write incorporates information from the available context into a carried memory state, and later reads use that state to make predictions. Efficient update rules make this form of adaptation practical over long contexts (Zhang et al., 2026; Feng et al., 2026). They also expose a decision within each update: when only a fraction of the available contributions can be retained, which ones should enter memory? The relevant budget concerns the write computation even when every token remains available to the model's read path.

Write allocation shapes both retrieval quality and computational work. Contributions combine in a finite state, so their composition affects what subsequent reads recover. A backend fixes a training objective, chunk schedule, and optimizer; within one scheduled write, activations, targets, and contribution weights form a collection of candidate updates, and selecting among them changes the evidence entering that write while leaving the surrounding computation intact. Keeping the full read path available lets us study these write policies under the same access to context. At successive write sites, these local decisions compose into a memory trajectory. The resulting design question is how to allocate each available write so that the carried state serves predictions made later in the stream. Related approaches select adaptation examples or question-relevant spans (Hu et al., 2025; Zhu et al., 2026); our subject is the internal contributors of an existing update, which lets allocation be studied independently of a particular fast-weight parameterization.

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

Two design choices make allocation concrete: the ranking signal and the selection unit. Contribution weights and activation statistics provide priorities using quantities already computed at the write site. We evaluate those priorities through predictions made from the resulting memory state. The selection unit determines how those priorities translate into a retained set. Token-wise ranking chooses contributors individually, while block-wise ranking retains neighboring contributors jointly. This distinction matters because an update combines contributions before any read measures its effect: changing one member changes the aggregate formed with its neighbors. Individual selection permits any subset at the specified count. Grouping restricts this choice to unions of blocks, making neighborhood membership part of the policy. Block size controls how strongly this joint-retention preference shapes the admissible subsets. Exposing both choices lets us compare individual priorities and joint retention under a common write budget.

We introduce *Structured Sparse Writing* (SSW), which allocates each current chunk's write budget to contiguous blocks of contributions. SSW aggregates token priorities within each block, selects the highest-scoring blocks, and gathers their contributors in temporal order before invoking the original update operator. This score-based rule uses the priorities already supplied by the backend. The retained set may contain separated blocks, with joint retention applying within each block, and the block size controls this restriction from individual tokens to larger neighboring groups. The allocation uses only the current complete chunk and the available state, and preserves the original read path, target construction, and commit boundaries while selected contributors keep their original weights, so one interface can precede either an optimizer-based or an additive write. Figure 1 places the operation inside a stream of carried memory states; score adapters appear in Appendix A.

We evaluate SSW across optimizer-based and additive TTT writes, using LaCT and In-Place TTT as two concrete backends. The experiments study how write allocation affects memory quality. Three findings emerge. In the main retrieval comparisons, SSW attains higher scores than dense writing while using a fraction of its contributions. At a fixed contributor count, the LaCT comparisons show that block selection outperforms token-wise score maximization; SSW also outperforms reverse ranking on their shared execution path. Thus local scalar-score optimality alone does not select the strongest retrieval policy. Finally, retrieval accuracy and answer likelihood prefer different budgets on identical inputs. These findings identify the retained set and the intended readout as central choices in allocating memory writes.

Our contributions are the following. We formulate write allocation as a decision internal to an existing memory update, separate from the update rule and the chunk schedule, and give it an operator—SSW—that realizes the decision through block scoring, budgeted selection, and ordered gathering while leaving the read path and commit boundaries intact. We prove a tight worst-case omission guarantee under a block budget, characterize the feasible-set relation between token and block selection, and express joint contribution effects through pairwise interactions. We study selection unit, ranking direction, and budget through controlled comparisons, show that the preferred budget depends on the readout on identical inputs, and instantiate the same selection interface with optimizer-based and additive TTT writes.

2 Structured Sparse Writing

Structured Sparse Writing (SSW) allocates a TTT write budget to ordered groups of update contributions through block scoring, budgeted selection, and ordered gathering. The resulting contributors enter the existing update operator at its original commit boundary. The definition requires only a contributor interface and a priority signal, leaving the memory architecture and its optimizer to the backend.

2.1 Problem formulation

We adopt the memorization/retrieval view of associative sequence models (Wang et al., 2026a). At write opportunity t, let σ_t contain the memory and any optimizer state. The ordered collection $\mathcal{C}_t = (c_{t,1}, \ldots, c_{t,N_t})$ contains the eligible contributors at that site. Each contributor packages one token's activations, targets, and weights for the update. The backend determines which sites are eligible and constructs these quantities through its original computation. From a common state and

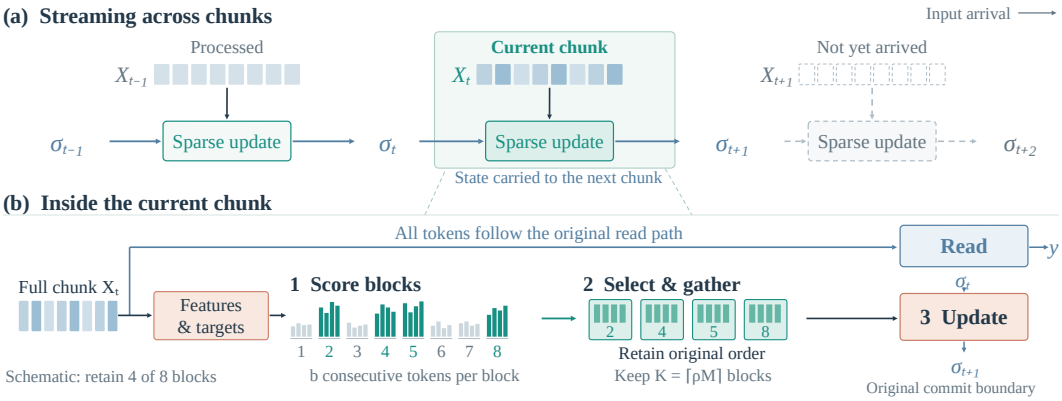

Figure 1: **Structured Sparse Writing in a stream.** (a) At one write site, successive complete input chunks X_t update a carried state σ_t; dashed elements show processing after future chunks arrive. (b) Block scoring, selection, and ordered gathering implement $\Pi_{\rho,b,\psi}$; step 3 invokes the original update operator $\mathcal{U}$ at the existing commit boundary. All tokens retain their original read and target-construction paths; y denotes the readout, whose timing remains backend-specific. The four-of-eight selection is schematic.

contributor collection, its dense transition is $\sigma_{t+1}^{\text{dense}} = \mathcal{U}(\sigma_t; \mathcal{C}_t)$. A sparse transition supplies a subset to the same operator.

This local comparison distinguishes the allocation decision from the trajectory it induces. SSW preserves the read computation and its access to tokens, while the selected write shapes subsequent states and representations. All context tokens can still influence activations, targets, and predictions through the full read path. The budget measures direct participation in the write. This distinction is especially useful in a stream: a token can contribute to the representation used to form a write even when its own update term is omitted from that write.

Given a target ratio ρ, allocation chooses both a retained subset and its selection unit. For write sites $\mathcal{T}$ and retained index sequences S_t, we report $r_{\text{write}} = (\sum_{t\in\mathcal{T}} |S_t|)/(\sum_{t\in\mathcal{T}} N_t)$, whose denominator counts eligible dense-write contributions. Matched-budget policies have equal contributor counts at the compared sites, so the identities and organization of their retained subsets can be evaluated separately from the nominal budget.

2.2 Block scoring, selection, and gathering

SSW is specified by a score map ψ, block size b, and target ratio $\rho \in (0, 1]$. The map assigns a nonnegative priority $a_{t,i} = \psi(c_{t,i}; \sigma_t)$ using quantities available in the update path. A backend might supply contribution weights or activation statistics. Each signal defines a ranking within its backend, and SSW uses that ranking directly to select blocks. The selection rule depends on the ordering of block scores, while the adapter defines what those scores summarize. This separation lets the same budget and grouping rule use signals from different update mechanisms. The concrete score adapters used in our experiments are given in Appendix A.

Allocate the budget to blocks. Partition a complete eligible chunk of $N_t = M_t b$ contributors into M_t consecutive, nonoverlapping blocks $B_{t,j}$ of size b. Define the block score $s_{t,j} = \sum_{i\in B_{t,j}} a_{t,i}$ and block budget $K_t = \lceil \rho M_t \rceil$. SSW selects

$$J_t^\star \in \operatorname*{arg\,max}_{J\subseteq\{1,\dots,M_t\},\,|J|=K_t} \sum_{j\in J} s_{t,j}. \tag{1}$$

Top-K_t solves this score-allocation objective exactly. Equal block sizes make sum and mean rankings equivalent. The realized write retains $K_t b$ contributors and has ratio $K_t/M_t \geq \rho$ due to rounding; we report the realized contribution count when discussing a budget.

The selection block and the update chunk play separate roles. A chunk defines when contributors become eligible for a memory transition; a block defines which contributors are admitted together

within that transition. All retained blocks enter the same scheduled write, so changing b controls joint admission while preserving the chunk boundary and frequency of adaptation.

Gather and apply the update. Let $S_t = \mathrm{SortByPosition}(\bigcup_{j \in J_t^\star} B_{t,j})$ and define $\Pi_{\rho,b,\psi}(\mathcal{C}_t; \sigma_t) = \mathrm{Gather}_{S_t}(\mathcal{C}_t)$. Using the same ordered indices for every contributor-associated tensor, SSW composes selection with the original update operator:

$$\sigma_{t+1}^{\mathrm{SSW}} = \mathcal{U}(\sigma_t; \Pi_{\rho,b,\psi}(\mathcal{C}_t; \sigma_t)) \,. \tag{2}$$

The selected activations, targets, and contribution weights remain paired, sorting preserves their temporal order, and gathering presents them as compact tensors to the write. The backend constructs contributors and their targets from the full context before selection. Algorithm 1 summarizes the operation at a complete-chunk write site.

The position of gathering matters for an optimizer-based memory. The compact contributors are inputs to $\mathcal{U}$, so the backend computes its update from the selected evidence before applying its original optimizer transformation; scaling an already transformed dense update is a different operation when that transformation is nonlinear. Masking full-size tensors can likewise change numerical reductions relative to gathering, so a comparison must specify its execution path.

Algorithm 1 Structured Sparse Writing at a write opportunity

Require: State σ, ordered contributors $\mathcal{C} = (c_1, \ldots, c_N)$, each with d associated scalar entries, block size b with $N = Mb$, target ratio $\rho \in (0, 1]$, score map ψ, write operator $\mathcal{U}$

1: $a_i \leftarrow \psi(c_i; \sigma)$ for each contributor i
2: Partition $\{1, \ldots, N\}$ into consecutive blocks $B_1, \ldots, B_M$
3: $s_j \leftarrow \sum_{i \in B_j} a_i$ for each block j ▷ $O(N)$
4: $K \leftarrow \lceil \rho M \rceil$, $J \leftarrow \mathrm{TopK}((s_1, \ldots, s_M), K)$ ▷ $O(M \log(K+1))$
5: $S \leftarrow \mathrm{SortByPosition}(\bigcup_{j \in J} B_j)$ ▷ $O(K \log(K+1) + Kb)$
6: $\widetilde{\mathcal{C}} \leftarrow \mathrm{Gather}_S(\mathcal{C})$ ▷ $O(Kbd)$
7: **return** $\mathcal{U}(\sigma; \widetilde{\mathcal{C}})$ at the original commit boundary

Selected contributions keep their original weights and learning rate. The retained contributors are passed directly to the backend, so selection can change both the direction and magnitude of a write. Section 3 characterizes this relationship for additive updates and motivates the matched-budget and reduced-rate controls.

2.3 Streaming semantics and allocation complexity

Available information and carried state. At an eligible boundary, the policy uses the complete current chunk and the state available at that point. After committing the selected write, the backend carries σ_{t+1} forward as usual, including any optimizer state. SSW preserves memory continuity between chunks, the original update opportunities, the handling of incomplete chunks, and sequence-reset boundaries. Selection therefore fits the existing streaming transition using information available at the current write site. In particular, a selected write becomes part of the state used when the next eligible chunk is processed. Applying the rule repeatedly yields a sequence of locally budgeted transitions connected by the backend's carried memory. Target construction and read/write ordering also follow the backend's timing convention, allowing one operator to serve memories with different internal implementations of a chunked update interface.

Allocation complexity. Once token scores are available, Algorithm 1 can allocate the budget in $O(N + M \log(K+1) + Kbd)$ work. A size-K heap selects the blocks; sorting the retained block indices and expanding their positions costs $O(K \log(K+1) + Kb)$, absorbed in this bound because $K \leq M$ and $d \geq 1$. The Top-K step solves Equation 1 within this cost and, for norm scores, attains the minimax guarantee of Proposition 1. Score construction and the update operator $\mathcal{U}$ have backend-specific costs. The resulting write uses Kb contributors at ratio K/M, with the original update schedule.

3 Analysis of the Allocation Objective

The Top-K rule solves its score objective exactly. We establish a tight worst-case omission guarantee under a block budget, relate token and block selection through their feasible sets, and describe joint contribution effects through pairwise interactions. We analyze additive raw matrices $A_i \in \mathbb{R}^{p\times q}$ at a common pre-write state with scores $a_i = \|A_i\|_F$, omitting the time index because this is a single-write comparison before optimizer transformations and subsequent state evolution.

3.1 A tight guarantee under a block constraint

Proposition 1 (Minimax allocation under a block budget)**.** *Let $B_1, \dots, B_M$ partition $\{1, \dots, N\}$ into size-b blocks, so $N = Mb$. Fix $1 \le K \le M$ and a nonnegative norm vector $a = (a_1, \dots, a_N)$. Let $S(J) = \bigcup_{j\in J} B_j$ and $s_j = \sum_{i\in B_j} a_i$. Among all raw matrix directions consistent with these norms, the worst-case omission obeys*

$$R(J; a) := \sup_{\|\widetilde{A}_i\|_F = a_i \ \forall i} \left\| \sum_{i\notin S(J)} \widetilde{A}_i \right\|_F = \sum_{j\notin J} s_j. \tag{3}$$

Any set $J^\star$ of the K largest block scores minimizes $R(J; a)$ over $|J| = K$.

The triangle inequality bounds omission by the sum of omitted norms, aligning their directions attains this bound, and minimizing that sum is equivalent to maximizing the retained block score; the full argument and its omission-envelope lemma appear in Appendix B. The result characterizes the best worst-case allocation across matrix directions consistent with the observed norms. For a realized tuple, cancellation can reduce actual omission below this envelope. The score budget thus controls an upper envelope on omitted raw-update mass, giving a precise allocation criterion for the norm-scored case.

3.2 Grouping and contribution interactions

The token and block policies optimize over different feasible sets. Every union of K size-b blocks is a valid Kb-token subset, so even for arbitrary nonnegative scores,

$$\min_{|S|=Kb} \sum_{i\notin S} a_i \le \min_{|J|=K} \sum_{j\notin J} s_j. \tag{4}$$

Token-wise Top-Kb optimizes this scalar objective over the larger feasible set. When scores are norms, it achieves an equal or tighter omission envelope. Section 4.3 shows that this per-write optimality can coexist with lower retrieval accuracy. Grouping, in turn, specifies which contributors enter the update jointly. Their pairwise relationships give a complementary description of the retained sum: for $\Delta_S = \sum_{i\in S} A_i$ and the Frobenius inner product $\langle A, B\rangle_F = \mathrm{tr}(A^\top B)$,

$$\|\Delta_S\|_F^2 = \sum_{i\in S} \|A_i\|_F^2 + 2 \sum_{i,j\in S,\, i<j} \langle A_i, A_j\rangle_F, \tag{5}$$

where positive pair terms reinforce the sum and negative terms cancel it. For example, two equal-norm contributions reinforce when aligned and cancel when opposed. Their scalar scores are identical in both cases, while the aggregate depends on their relative direction. Joint admission therefore specifies a concrete geometric choice at the write interface. Block selection keeps within-block pairs together, while individual selection may retain only one endpoint, and it also changes which pairs survive across blocks. SSW implements this preference for joint retention through block membership. The pair terms describe the resulting update geometry; their signs and magnitudes depend on the actual contributions. With fixed norms, different matrix directions can favor either policy in actual omission (Appendix B).

Selection also changes a write's magnitude. For additive updates, uniformly sampling K of M blocks has the same mean raw update as reducing the dense learning rate by the realized retention ratio (Appendix B). This identity describes the expected raw matrix update under uniform random selection at a fixed pre-write state. A matched-budget comparison fixes contributor count while allowing update direction and magnitude to vary. A reduced-learning-rate dense control provides an additional empirical reference for comparing update policies.

4 EXPERIMENTS

4.1 EVALUATION DESIGN

TTT backends and evaluation tasks. We instantiate SSW with optimizer-based and additive TTT writes. The LaCT 760M checkpoint (Zhang et al., 2026) uses an inner optimizer for memory writes. Its main retrieval comparison covers all eight needle-in-a-haystack tasks from RULER (Hsieh et al., 2024), with 50 new inputs per task at each of 16K and 32K. The fixed SSW configuration uses 64-token blocks and retains 12.5% of write contributions. Structure and budget sweeps use separate development inputs, and the main comparison evaluates the fixed configuration on new inputs. For additive writes, Qwen3-4B In-Place TTT (Yang et al., 2025a; Feng et al., 2026) supplies an additive memory update, and Table 1 reports all 13 RULER tasks with 100 inputs per task at 50% writes, evaluating the 32K checkpoint at 32K and the 64K checkpoint at 64K. Each checkpoint is evaluated at its matched context length. We compare policy scores within each backend, since LaCT uses modified RULER prompts and a different task population.

Selection and budget controls. Matched-budget controls compare retained sets at 12.5% writes in LaCT; the In-Place extension uses 50%. Random blocks average three selector seeds, reverse ranking inverts the score order, token-wise selection removes joint block admission using the same scoring rule and contributor count, dispersed groups collect nonadjacent contributors into equally sized units, and block16 versus block64 compares grouping scales. All hold the write budget fixed and so compare selection policies. The archived In-Place public-code study additionally evaluates a reduced-learning-rate dense arm to compare sparse allocation with a uniformly smaller dense update (Appendix A.3). SSW, reverse ranking, and the grouping comparisons use gathered contributors, whereas LaCT's supplied random control uses full-size masking, so reverse ranking carries our ranking claim. Adapters and full-execution anchors appear in Appendices A and C.1.

Readouts and uncertainty. Retrieval uses task-balanced macro scores on a 0–100 scale. LaCT's budget study also measures answer negative log-likelihood (NLL) on the same retrieval inputs. For continuation, 96 nonoverlapping prose segments from one author each provide 6144 context tokens and a 1024-token read-only window, shared across policies, with sparse ratios $1/16$, $1/8$, $1/4$, and $1/2$. LaCT retrieval intervals use 20,000 bootstrap draws over its eight tasks and continuation intervals resample the 96 paired segments. These descriptive 95% intervals are conditional on the evaluated checkpoints and unadjusted across comparisons. The In-Place extension is summarized by aggregate point estimates. Appendix A.2 specifies the populations and aggregation conventions.

4.2 RETRIEVAL AT A FIXED WRITE BUDGET

Structured allocation retains effective retrieval updates while using only a fraction of the available contributions (Table 1). In LaCT, SSW retains one eighth and improves on the three-seed random mean by $22.44\,[7.85, 41.08]$ points at 16K and $16.62\,[3.25, 39.58]$ at 32K. Because that random control uses full-size masking, we also compare against reverse ranking, which shares SSW's gathered path at the same budget and block size: it scores 10.31 and 4.31, at the level of random selection (Table 3). SSW's advantage over reverse ranking supports the utility of the score order on their shared execution path.

The selected subset also exceeds the evaluated LaCT dense update, by $6.56\,[0.00, 14.38]$ and $2.12\,[0.25, 4.38]$ points at 16K and 32K, with separate full-execution comparisons in the same direction (Appendix C.1). In these comparisons, allocating one eighth of the contributions attains higher retrieval scores than writing all of them. This comparison changes contributor count; the matched-budget comparisons below hold that count fixed and evaluate alternative retained sets.

Additive TTT writes. In-Place applies the same interface to additive writes. At 50% retention, SSW scores 76.93 at 32K and 69.35 at 64K, against dense scores of 63.47 and 68.47 and random scores of 56.17 and 59.30 — improvements of 13.46 and 0.89 points over dense and of 20.75 and 10.05 over matched-budget random selection. This comparison uses supplied aggregate extension scores, with reporting scope specified in Table 1 and Appendix A.3. The following controlled comparisons examine selection structure and write budgets in the optimizer-based backend.

Table 1: **Retrieval quality under a fixed write budget.** Task-balanced RULER scores (%, higher is better). SSW retains 12.5% of contributions on LaCT and 50% on In-Place; random retains the same fraction.

	LaCT (760M)		In-Place (Qwen3-4B)	
Policy	16K	32K	32K	64K
Dense	26.94	19.31	63.47	68.47
Random	11.06	4.81	56.17	59.30
SSW	**33.50**	**21.44**	**76.93**	**69.35**
Δ_{dense} (pp)	+6.56	+2.12	+13.46	+0.89
Δ_{random} (pp)	+22.44	+16.62	+20.75	+10.05

Random denotes the three-seed mean and is the matched-budget control. Δ_{dense} and Δ_{random} denote SSW minus dense and random, respectively, in percentage points, computed before display rounding. In-Place evaluates the 32K checkpoint at 32K and the 64K checkpoint at 64K. LaCT intervals and controls are described in Section 4.1 and Appendix C. In-Place uses supplied updated aggregates without verified new per-input records or paired intervals; archived diagnostics are a separate result source (Appendix A.3).

4.3 Selection structure and grouping scale

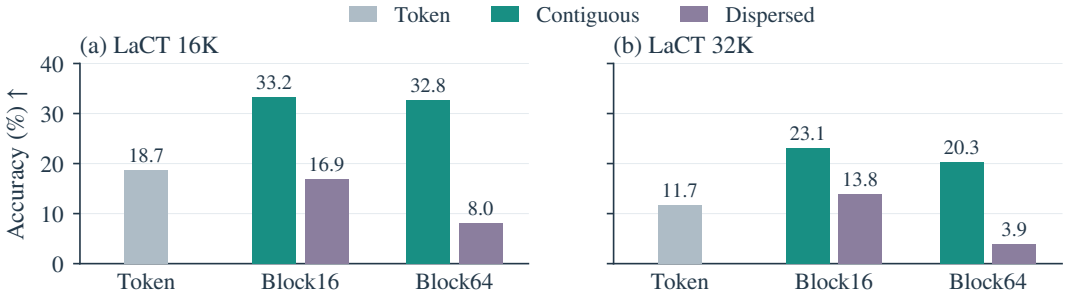

Figure 2: **Selection structure at a fixed write budget.** LaCT development inputs: eight tasks with 50 inputs each per length, at $\rho = 1/8$. Policies use the same scoring rule and contributor budgets, retaining tokens individually or in contiguous or dispersed groups of size 16 or 64. Block64 is the fixed main configuration.

The selection unit changes quality at a fixed budget. In LaCT, intact blocks improve on token-wise selection by $14.06\,[6.62, 22.88]$ points at 16K and $8.62\,[3.50, 14.44]$ at 32K under the same scoring rule and contributor count (Figure 2, Table 3). At each write, token-wise selection maximizes the retained scalar score over the larger feasible set (Equation 4), yet its retrieval scores are lower than those of block selection here. An exact optimizer of the per-write scalar objective can therefore yield a weaker retrieval policy than block-constrained selection. The additive score objective has no explicit pairwise term. Equation 5 describes how retained contributions combine through such terms in the raw update.

Group composition also affects quality. At equal group size and retention, contiguous groups outperform dispersed groups on both development and new-input populations (Table 3). These results support contiguous grouping in LaCT. Both block16 and block64 outperform token-wise selection; block16 has the higher point estimates in these panels. We retain the fixed block64 configuration and report block16 as a sensitivity comparison.

4.4 Write budgets and the intended readout

The preferred write budget depends on the readout, even on identical inputs (Figure 3). At 16K, increasing retention from $1/4$ to $1/2$ lowers retrieval accuracy while improving answer NLL; at 32K, accuracy is highest at $1/8$ and NLL lowest at $1/2$. These observed optima measure different uses of the same memory: free-generation accuracy scores what the model produces, whereas answer

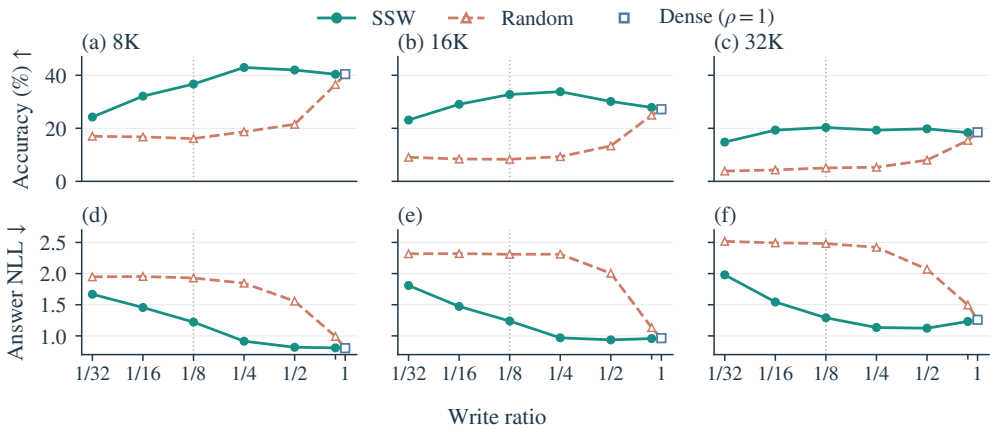

Figure 3: **Accuracy and answer likelihood favor different budgets on the same inputs.** LaCT development retrieval: eight tasks with 50 inputs each per length. Top: accuracy; bottom: answer NLL from the same runs, with axes shared within rows. Both policies retain every block at $\rho = 1$ and therefore meet the dense write, marked by the square; the curves converge towards it as the budget grows. Vertical guides mark the fixed $1/8$ policy. Accuracy peaks away from full retention at every length, whereas the NLL optimum lies at $1/2$ or at dense itself.

likelihood scores a reference continuation. Holding inputs and context length fixed exposes this difference between criteria: write budgets should be selected for the intended readout.

Natural-text continuation shows a ranking benefit at every sparse budget. On the 96 prose windows, SSW lowers NLL relative to random by 0.00296–0.00519 nats with paired intervals below zero, while dense attains the lowest NLL at 2.48431 (Table 4). Ranking improves the use of each sparse budget; full retention gives the best overall likelihood on this readout.

5 Related Work

Fast weights, test-time memory, and update rules. Fast weights store recent inputs in parameters (Ba et al., 2016), an interpretation also available for linear attention (Schlag et al., 2021), and TTT extends self-supervised adaptation under distribution shift (Sun et al., 2020) to a trainable recurrent state (Sun et al., 2025). LaCT uses large-chunk updates, In-Place adapts existing MLP output projections, and Titans introduces neural memory with momentum and data-dependent forgetting (Zhang et al., 2026; Feng et al., 2026; Behrouz et al., 2025); MIRAS and test-time regression organize memory objectives and update rules (Behrouz et al., 2026; Wang et al., 2026a), and TTCD trains shorter-window fast weights with a longer-window teacher's target (Wang et al., 2026b). A parallel line reshapes the update itself, through input-dependent dynamics and state-space duality (Gu & Dao, 2024; Dao & Gu, 2024), content-dependent gates (Yang et al., 2024a), delta-rule updates and their combination with retention (Yang et al., 2024b; 2025b), and orthogonal updates that suppress already-represented directions (Karami et al., 2025). All of these change memory dynamics, whereas SSW constrains which contributors enter an unchanged update rule. At the scheduling level, LaCT already supports updating on selected chunks while applying the state to all chunks (Zhang et al., 2026, Section 4.1 and Figure 3(d)), and TNT uses local and global memories with two-stage chunk-wise training (Li et al., 2026); SSW addresses allocation *within* each eligible chunk, preserving the existing update opportunities while selecting which contributors enter each write.

Selecting adaptation evidence or what the model reads. TLM prioritizes high-perplexity examples for LoRA adaptation (Hu et al., 2025); query-only TTT updates query projections on sampled spans while reusing a frozen KV cache (Bansal et al., 2026); EASE-TTT selects evidence for query-side supervision (Yuan et al., 2026); and Self-Guided TTT annotates question-relevant spans for next-token adaptation (Zhu et al., 2026). SSW shares their full-context reading and selects internal contributions directly from the current chunk's update signals at streaming write opportunities. A further line budgets the read instead, retaining KV positions from a prompt observation window

(Li et al., 2024) or selecting attention read pages from page-level key statistics (Tang et al., 2024). Because SSW budgets memory-update contributors while preserving the original reads and targets, its write ratio describes a different resource.

6 Discussion and Limitations

Write composition is a design choice for test-time training alongside the update rule and contributor count. In our matched-budget experiments, grouped writes retrieve more accurately than token-wise score maximization. The scalar allocation objective thus serves as a local surrogate whose retrieval value depends on the selection unit. Ranking direction and grouping scale provide further ways to change that retained composition. The budget sweep reinforces the role of the intended readout: retrieval accuracy and answer likelihood attain their optima at different retention levels. Write policies should consequently be evaluated through the memory behavior they induce, alongside their local allocation objective.

The analysis suggests studying selectors that track relationships between contributions. The omission guarantee characterizes allocation from scalar norms, while Equation 5 expresses the geometry of joint retention. Combining these descriptions with subsequent state evolution offers a direction for designing priorities around the readout served by memory.

Limitations. The evaluation covers two text backends at 760M and 4B and one author's prose; broader scales, modalities, and text distributions remain targets for evaluation. The write ratio measures contribution participation at the existing update opportunities. LaCT's 32K comparison provides the clearer dense-control evidence: the 16K interval reaches zero after rounding. In-Place uses distinct checkpoints at 32K and 64K; its updated aggregate scores lack per-input records and verified execution settings for paired analysis. Separate unclipped public-code records show that a half-learning-rate dense arm recovers much of the dense deficit, with an unresolved difference from SSW. That comparison applies to the archived protocol, and a corresponding LaCT control remains to be evaluated. LaCT's random contrast includes a masking–gathering difference; reverse ranking supplies the comparison on a shared execution path. Unavailable LaCT raw predictions limit exact numerical reproduction. Exploratory gate-based selectors yield higher NLL than random selection in TTT-Linear/MLP (supplementary records), motivating backend-specific validation of ranking signals.

7 Conclusion

We formulate write allocation as a design decision within test-time training. Structured Sparse Writing states that decision as an operator over an existing update—score contiguous blocks, select under a budget, gather in order—leaving the update rule, the schedule, and the read path untouched while making the selection unit, the ranking signal, and the budget explicit. Top-K solves the allocation objective exactly, with a tight worst-case omission guarantee for norm-scored additive contributions under a block budget. Pairwise interactions complement this scalar objective by describing joint contribution effects. The matched-budget experiments show that block selection outperforms token-wise score maximization at a fixed budget, exposing a gap between local scalar optimality and retrieval quality. Ranking direction also affects retrieval, and accuracy and likelihood favor different budgets on the same inputs. The same interface is instantiated with optimizer-based and additive TTT writes. Together, these results establish write allocation as a design axis of test-time training, with the retained set and budget chosen for the intended readout.

Reproducibility Statement

The appendix specifies numerical semantics, evaluation protocols, and support selection. The accompanying records contain fixed result snapshots, task-level scores, paired statistics, and evaluation settings. Anonymous repository: https://anonymous.4open.science/status/ssw-9082.

AI Use Statement

Generative AI tools were used for manuscript editing, scientific figure preparation, and coding assistance for experiment execution. All research ideas, methodology, experimental design, result interpretation, and the overall structure and narrative of the paper were developed by the authors. All AI-assisted artifacts were reviewed and verified by the authors, who take full responsibility for the final content of this work.

References

Jimmy Ba, Geoffrey E. Hinton, Volodymyr Mnih, Joel Z. Leibo, and Catalin Ionescu. Using fast weights to attend to the recent past. In *Advances in Neural Information Processing Systems*, volume 29, 2016. URL https://papers.nips.cc/paper_files/paper/2016/hash/9f44e956e3a2b7b5598c625fcc802c36-Abstract.html.

Rachit Bansal, Aston Zhang, Rishabh Tiwari, Lovish Madaan, Venkata Sai Surya Subramanyam Duvvuri, Devvrit Khatri, David Brandfonbrener, David Alvarez-Melis, Prajjwal Bhargava, Mihir Kale, and Samy Jelassi. Let's (not) just put things in context: Test-time training for long-context LLMs. In *International Conference on Learning Representations*, 2026. URL https://proceedings.iclr.cc/paper_files/paper/2026/hash/b619cd6dcc986856b8a8da2b08d89396-Abstract-Conference.html.

Ali Behrouz, Peilin Zhong, and Vahab Mirrokni. Titans: Learning to memorize at test time. In *Advances in Neural Information Processing Systems*, volume 38, pp. 113506–113543, 2025. URL https://proceedings.neurips.cc/paper_files/paper/2025/hash/a4ca07aa108036f80cbb5b82285fd4b1-Abstract-Conference.html.

Ali Behrouz, Meisam Razaviyayn, Peilin Zhong, and Vahab Mirrokni. It's all connected: A journey through test-time memorization, attentional bias, retention, and online optimization. In *International Conference on Learning Representations*, 2026. URL https://proceedings.iclr.cc/paper_files/paper/2026/hash/d55f39791f04745b2e0c8abebf3dd5d7-Abstract-Conference.html.

Tri Dao and Albert Gu. Transformers are SSMs: Generalized models and efficient algorithms through structured state space duality. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 10041–10071, 2024. URL https://proceedings.mlr.press/v235/dao24a.html.

Guhao Feng, Shengjie Luo, Kai Hua, Ge Zhang, Wenhao Huang, Di He, and Tianle Cai. In-place test-time training. In *International Conference on Learning Representations*, 2026. URL https://proceedings.iclr.cc/paper_files/paper/2026/hash/ba6f5879d7f382241316f7dda7e6b8ac-Abstract-Conference.html.

Albert Gu and Tri Dao. Mamba: Linear-time sequence modeling with selective state spaces. In *First Conference on Language Modeling*, 2024. URL https://arxiv.org/abs/2312.00752.

Cheng-Ping Hsieh, Simeng Sun, Samuel Kriman, Shantanu Acharya, Dima Rekesh, Fei Jia, Yang Zhang, and Boris Ginsburg. RULER: What's the real context size of your long-context language models? In *First Conference on Language Modeling*, 2024. URL https://openreview.net/forum?id=kIoBbc76Sy.

Jinwu Hu, Zitian Zhang, Guohao Chen, Xutao Wen, Chao Shuai, Wei Luo, Bin Xiao, Yuanqing Li, and Mingkui Tan. Test-time learning for large language models. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 24823–24849. PMLR, 2025. URL https://proceedings.mlr.press/v267/hu25z.html.

Keller Jordan, Yuchen Jin, Vlado Boza, Jiacheng You, Franz Cesista, Laker Newhouse, and Jeremy Bernstein. Muon: An optimizer for hidden layers in neural networks, 2024. URL https://kellerjordan.github.io/posts/muon/.

Mahdi Karami, Razvan Pascanu, and Vahab Mirrokni. Lattice: Learning to efficiently compress the memory. *arXiv preprint arXiv:2504.05646*, 2025. URL https://arxiv.org/abs/2504.05646.

Yuhong Li, Yingbing Huang, Bowen Yang, Bharat Venkitesh, Acyr Locatelli, Hanchen Ye, Tianle Cai, Patrick Lewis, and Deming Chen. SnapKV: LLM knows what you are looking for before generation. In *Advances in Neural Information Processing Systems*, volume 37, 2024. URL https://proceedings.neurips.cc/paper_files/paper/2024/hash/28ab418242603e0f7323e54185d19bde-Abstract-Conference.html.

Zeman Li, Ali Behrouz, Yuan Deng, Peilin Zhong, Praneeth Kacham, Mahdi Karami, Meisam Razaviyayn, and Vahab Mirrokni. TNT: Improving chunkwise training for test-time memorization. In *International Conference on Learning Representations*, 2026. URL https://proceedings.iclr.cc/paper_files/paper/2026/hash/bdb30687f1c2255c29b11b0b45204ebe-Abstract-Conference.html.

Pranav Rajpurkar, Robin Jia, and Percy Liang. Know what you don't know: Unanswerable questions for SQuAD. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pp. 784–789, 2018. URL https://aclanthology.org/P18-2124/.

Imanol Schlag, Kazuki Irie, and Jürgen Schmidhuber. Linear transformers are secretly fast weight programmers. In *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pp. 9355–9366. PMLR, 2021. URL https://proceedings.mlr.press/v139/schlag21a.html.

Yu Sun, Xiaolong Wang, Zhuang Liu, John Miller, Alexei Efros, and Moritz Hardt. Test-time training with self-supervision for generalization under distribution shifts. In *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pp. 9229–9248. PMLR, 2020. URL https://proceedings.mlr.press/v119/sun20b.html.

Yu Sun, Xinhao Li, Karan Dalal, Jiarui Xu, Arjun Vikram, Genghan Zhang, Yann Dubois, Xinlei Chen, Xiaolong Wang, Sanmi Koyejo, Tatsunori Hashimoto, and Carlos Guestrin. Learning to (learn at test time): RNNs with expressive hidden states. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267, pp. 57503–57522, 2025. URL https://proceedings.mlr.press/v267/sun25h.html.

Jiaming Tang, Yilong Zhao, Kan Zhu, Guangxuan Xiao, Baris Kasikci, and Song Han. QUEST: Query-aware sparsity for efficient long-context LLM inference. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 47901–47911, 2024. URL https://proceedings.mlr.press/v235/tang24l.html.

Ke Alexander Wang, Jiaxin Shi, and Emily B. Fox. Test-time regression: a unifying framework for designing sequence models with associative memory. *Journal of Machine Learning Research*, 27 (169):1–41, 2026a. URL https://www.jmlr.org/papers/v27/25-0903.html.

Zixuan Wang, Xingyu Dang, Rui-Jie Zhu, Zixin Wen, Hengyu Fu, Wenhao Chai, and Jason D. Lee. Learning what to remember: Test-time training via context distillation. *arXiv preprint arXiv:2608.01672*, 2026b. URL https://arxiv.org/abs/2608.01672.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, Chujie Zheng, Dayiheng Liu, Fan Zhou, Fei Huang, Feng Hu, Hao Ge, Haoran Wei, Huan Lin, Jialong Tang, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jing Zhou, Jingren Zhou, Junyang Lin, Kai Dang, Keqin Bao, Kexin Yang, Le Yu, Lianghao Deng, Mei Li, Mingfeng Xue, Mingze Li, Pei Zhang, Peng Wang, Qin Zhu, Rui Men, Ruize Gao, Shixuan Liu, Shuang Luo, Tianhao Li, Tianyi Tang, Wenbiao Yin, Xingzhang Ren, Xinyu Wang, Xinyu Zhang, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yinger Zhang, Yu Wan, Yuqiong Liu, Zekun Wang, Zeyu Cui, Zhenru Zhang, Zhipeng Zhou, and Zihan Qiu. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025a. URL https://arxiv.org/abs/2505.09388.

Songlin Yang, Bailin Wang, Yikang Shen, Rameswar Panda, and Yoon Kim. Gated linear attention transformers with hardware-efficient training. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 56501–56523, 2024a. URL https://proceedings.mlr.press/v235/yang24ab.html.

Songlin Yang, Bailin Wang, Yu Zhang, Yikang Shen, and Yoon Kim. Parallelizing linear transformers with the delta rule over sequence length. In *Advances in Neural Information Processing Systems*, volume 37, pp. 115491–115522, 2024b. doi: 10.52202/079017-3668. URL https://proceedings.neurips.cc/paper_files/paper/2024/hash/d13a3eae72366e61dfdc7eea82eeb685-Abstract-Conference.html.

Songlin Yang, Jan Kautz, and Ali Hatamizadeh. Gated delta networks: Improving mamba2 with delta rule. In *International Conference on Learning Representations*, 2025b. URL https://arxiv.org/abs/2412.06464.

Zhilin Yang, Peng Qi, Saizheng Zhang, Yoshua Bengio, William W. Cohen, Ruslan Salakhutdinov, and Christopher D. Manning. HotpotQA: A dataset for diverse, explainable multi-hop question answering. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pp. 2369–2380, 2018. URL https://aclanthology.org/D18-1259/.

Xiaopeng Yuan, Zebin Wang, Suwen Wang, Zongxin Yang, Haohan Wang, and Yushun Dong. EASE-TTT: Evidence-aligned selective test-time training for long-context question answering. *arXiv preprint arXiv:2606.06906*, 2026. URL https://arxiv.org/abs/2606.06906.

Tianyuan Zhang, Sai Bi, Yicong Hong, Kai Zhang, Fujun Luan, Songlin Yang, Kalyan Sunkavalli, William T. Freeman, and Hao Tan. Test-time training done right. In *International Conference on Learning Representations*, 2026. URL https://proceedings.iclr.cc/paper_files/paper/2026/hash/ffbfff81fb78e4bb558273b91e12e318-Abstract-Conference.html.

Xinyu Zhu, Zhe Xu, Xiaohan Wei, Yunchen Pu, Fei Tian, Chonglin Sun, Kaushik Rangadurai, Hua Zhi, Frank Shyu, Sandeep Pandey, Luke Simon, Yu Meng, and Xi Liu. Self-guided test-time training for long-context LLMs. *arXiv preprint arXiv:2607.09415*, 2026. URL https://arxiv.org/abs/2607.09415.

A Implementation and Evaluation

A.1 Score adapters and execution

SSW changes the contributors entering a scheduled write. The read path, target construction, update operator, sequence resets, and treatment of incomplete chunks remain those of the backend. Table 2 specifies the two adapters; their scores need not have comparable scales or interpretations.

Table 2: **Two instances of the same within-write policy.**

Property	LaCT	In-Place
Backbone	760M LaCT	Qwen3-4B-based
Update rule	Inner-loop optimizer (Muon)	MLP output-projection update
Token score	Learning-rate signal $\bar{\eta}_i$	$\|P^\top t_i\|_2 \|h_i\|_2$
Update chunk / selection block	2048 / 64	1024 / 64
Retained blocks / eligible blocks	4 / 32	8 / 16
Main write ratio	12.5%	50%

LaCT. The token score is the supplied learning-rate statistic $a_i = \bar{\eta}_i$ (Zhang et al., 2026). Selected keys and associated quantities enter the original inner optimizer, including Muon (Jordan et al., 2024). This statistic is a contribution-weight proxy, not the norm of the optimizer-transformed update, so the norm assumption of Proposition 1 need not hold for it. The available configuration and study description for the reported LaCT checkpoint specify four fast heads. The exact head/inner-step reduction of the learning-rate tensor and original run manifests are unavailable in the supplied record. We specify the recorded token-score interface without inferring those missing implementation details.

In-Place. We use column vectors: target $t_i \in \mathbb{R}^{d_{\text{tar}}}$, activation $h_i \in \mathbb{R}^q$, and target projection $P \in \mathbb{R}^{d_{\text{tar}} \times p}$. The projected target is $v_i = P^\top t_i \in \mathbb{R}^p$; its additive contribution $A_i \in \mathbb{R}^{p \times q}$ and score are

$$A_i = v_i h_i^\top, \qquad a_i = \|A_i\|_F = \|v_i\|_2 \|h_i\|_2. \tag{6}$$

The rank-one factorization avoids materializing the outer product. In the archived implementation, norm products are computed in float32 and updates are committed in BF16. The recorded learning rate is 3, with adaptation at layers 0, 6, 12, 18, 24, and 30. The archived studies use the public In-Place TTT implementation. That code path does not apply the original paper's 10^{-5} clipping setting; all policies in the archived studies retain this unclipped convention. Available checkpoint metadata does not certify identity with the weights used for the original paper's final scores.

Budgets and numerical conventions. LaCT retains 4 of 32 blocks per complete chunk (256 of 2048 contributors); In-Place retains 8 of 16 (512 of 1024). Both use blocks of size 64. All contributor-associated tensors share the sorted gather indices, with original weights and learning rates and no inverse-retention rescaling. SSW gathers compact tensors; LaCT random uses full-size masking, while its reverse-ranking and grouping comparisons gather. Archived In-Place random blocks also gather. Masking and gathering can differ in contraction shapes and summation order. The full-retention In-Place wrapper was checked against the unmodified implementation before those comparisons; this check does not verify the later aggregate extension.

A.2 Inputs and statistical procedures

Retrieval populations. LaCT's new-input comparison uses all eight RULER NIAH tasks with 50 inputs each at 16K and 32K. The fixed policy ranks 64-token blocks by the learning-rate signal, retains one eighth of the contributors, and gathers them before the original update. The original pre-registration covers 16K, and 32K extends that configuration to new inputs. Structure (16K/32K) and budget (8K/16K/32K) studies use separate development panels of eight tasks with 50 inputs each. Streaming packets contain 511 tokens; generation is capped at 128 tokens. The recorded prompt

changes replace `Some` with `A` and `answers` with `answer` in the affected templates, identically across paired policies. The full eight-task macro is used throughout.

The In-Place main-table extension reports 13 RULER tasks with 100 inputs per task at each matching checkpoint length. Earlier main records (B) have the same task counts but are a separate result source; the earlier design study (C) uses 20 inputs per task drawn from B and reuses B's controls. Archived evaluations generate 64 greedy tokens. RULER QA conditions draw answerable questions from SQuAD 2.0 development (Rajpurkar et al., 2018) and questions from HotpotQA (Yang et al., 2018); these are not full evaluations of either QA dataset. The separate selection and grouping populations are documented in the supplementary records and analyzed separately from B/C and the extension.

Aggregation and uncertainty. For each policy we average official input scores within each task and then weight tasks equally. For random selection, the three seeds are first averaged within each input; differences use paired task/input identities and unrounded scores on the 0–100 scale. LaCT selector seeds 11, 23, and 37 yield macros 11.2500, 10.6875, and 11.2500 at 16K and 4.5625, 4.8750, and 5.0000 at 32K. Seed SD, where reported in the supplementary records, is the sample SD of complete seed-level macros, not uncertainty over new tasks.

LaCT retrieval intervals use 20,000 percentile task-cluster bootstrap draws over the eight tasks, with analysis seed 0. Continuation uses 96 nonoverlapping same-author segments, each with 6144 context tokens and a 1024-token read-only window. Its intervals resample paired segments for 20,000 draws, also with seed 0; this assumes exchangeable segments and does not quantify variation across authors or corpora. Archived In-Place comparisons resample inputs within each of 13 fixed tasks, using source-question clusters for QA, with 20,000 draws and analysis seed 20260919. Unless explicitly stated otherwise, intervals are descriptive 95% intervals without multiplicity adjustment. Crossing zero does not establish equivalence or non-inferiority.

A.3 Result sources and reproducibility scope

The In-Place extension uses the updated aggregate scores supplied for Table 1, including random scores 56.17 and 59.30. The earlier B records instead have random means 76.17 and 69.30. Updated per-input outputs and execution settings are absent from the current artifact, so neither the archived protocol checks nor its paired intervals verify the new comparison. This also prevents attributing the new ranking gap to a specific implementation change. LaCT supplies aggregate tables and paired statistics, but raw predictions and the exact score reduction remain unavailable. These gaps limit complete numerical reproduction and are not filled by the archived In-Place records.

The supplementary records separate the extension snapshot, LaCT summaries, earlier In-Place B/C results, and the fresh-input studies. The two archived 32K studies use 1,300 selection inputs and 650 grouping inputs. The selection study reports a block–token gap; its contrasts with random, periodic, no-write, and half-learning-rate dense include zero in their intervals, as do all four primary grouping contrasts. Table and figure regeneration from supplied summaries is distinct from reproducing model executions.

B Proofs and Scope

The results concern additive raw matrices at one fixed pre-write state. They do not assert optimality after an optimizer transformation or for a later prediction loss.

Lemma 1 (Tight norm-only omission envelope)**.** *For nonnegative $a_1, \dots, a_N$, a retained set S, and matrix dimensions $p, q \geq 1$,*

$$\sup_{\|\widetilde{A}_i\|_F = a_i \ \forall i} \left\| \sum_{i \notin S} \widetilde{A}_i \right\|_F = \sum_{i \notin S} a_i. \tag{7}$$

The equality also holds with constraints $\|\widetilde{A}_i\|_F \leq a_i$.

Proof. The triangle inequality gives the upper bound. For any unit-Frobenius matrix E, choosing $\widetilde{A}_i = a_i E$ attains it, including when nothing is omitted. □

Proof of Proposition 1. For $S(J) = \bigcup_{j \in J} B_j$, the lemma gives $R(J; a) = \sum_{j \notin J} s_j = \sum_j s_j - \sum_{j \in J} s_j$. The total is fixed. Exchanging a retained block for any higher-scoring omitted block improves the retained sum, so Top-K minimizes R. Equal block sizes make every feasible set retain exactly Kb contributors. The uncertainty set permits all directions with the specified norms, including directions a particular model may not reach. □

Grouping and interactions. Every union of K size-b blocks is a valid Kb-token subset. Minimizing the omitted-score sum over the larger token feasible set proves Equation 4. This ordering holds for any nonnegative scores; its omission-envelope interpretation requires norm scores or valid norm upper bounds. Expanding $\| \sum_{i \in S} A_i \|_F^2 = \sum_{i,j \in S} \langle A_i, A_j \rangle_F$ and separating diagonal and off-diagonal terms proves Equation 5. This identity does not identify the cause of a downstream accuracy difference.

Uniform random selection. For a shared learning rate η, let $D_j = \eta \sum_{i \in B_j} A_i$ and $D_{\text{dense}} = \sum_j D_j$. For a uniformly sampled K-element block set J_{random}, $D_{\text{random}} = \sum_{j \in J_{\text{random}}} D_j$. Each block is included with probability K/M, giving

$$\mathbb{E}[D_{\text{random}} \mid D_1, \ldots, D_M] = \frac{K}{M} \sum_j D_j = \frac{K}{M} D_{\text{dense}}. \tag{8}$$

This is an expectation of the raw update, not of its norm, and does not pass unchanged through nonlinear optimization, clipping, rounding, or state evolution. Nor does it constrain score-based selection: with two scalar blocks $(10, 0)$ and $K = 1$, Top-1 returns 10 whereas the random expectation is 5. Equal budgets fix contributor count, not update magnitude. A reduced-learning-rate dense arm tests a different policy.

Unchanged scores can reverse actual-error order. Take norms $a = (3, 1, 2, 1)$, blocks $\{1, 2\}$ and $\{3, 4\}$, and a budget of two contributors. Block scores select $S_{\text{blk}} = \{1, 2\}$ and token scores select $S_{\text{tok}} = \{1, 3\}$. For omission error $e(S) = |\sum_{i \notin S} A_i|$, the pairs $(e(S_{\text{blk}}), e(S_{\text{tok}}))$ are

$$A = (3, -1, 2, 1) : \ (3, 0), \qquad A = (3, 1, -2, 1) : \ (1, 2).$$

The norm scores, selected sets, and bounds (3 for blocks, 2 for tokens) are unchanged. Cancellation reverses actual omission error, ruling out a universal fidelity ordering from the norm scores alone; neither case establishes a task-accuracy ordering.

C Supporting Results

C.1 LaCT ranking and grouping

Table 3 reports matched-budget ranking and grouping controls. SSW and reverse ranking share the gathered path; random uses full-size masking. Contiguous and dispersed groups preserve group size while changing membership, comparing selection policies at a fixed contributor count.

Table 3: **LaCT ranking and grouping controls at 12.5% writes.**

Policy	Dev. 16K	Dev. 32K	New 16K	New 32K
Block64 (SSW)	32.75	20.31	33.50	21.44
Block16	33.25	23.06	34.06	24.75
Dispersed groups64	8.00	3.88	10.94	4.38
Token-wise selection	18.69	11.69	–	–
Reverse blocks64	–	–	10.31	4.31
Random blocks64	–	–	11.06	4.81
Block64 − token (pp)	+14.06	+8.62	–	–
95% interval	[6.62,22.88]	[3.50,14.44]	–	–

Block16 has higher point estimates than block64 on development and new inputs. Block64 remains the fixed main configuration, with block16 reported as a grouping-scale sensitivity comparison.

Full-execution anchors. Separate full-execution comparisons score SSW/dense at 33.81/27.31 for 16K and 20.44/19.31 for 32K. Their paired differences are $+6.50\,[0.25, 14.25]$ and $+1.12\,[0.31, 2.12]$ points, respectively.

C.2 Write budgets and continuation

Figure 3 and `results/paired_readout_budgets.csv` report accuracy and answer NLL on the same retrieval inputs. Accuracy peaks at $1/4$, $1/4$, and $1/8$ writes for 8K, 16K, and 32K, whereas answer NLL is lowest at 1, $1/2$, and $1/2$. These are point-estimate optima over the measured grid.

Table 4: **LaCT natural-text continuation NLL.** Lower is better.

ρ	SSW	Random	Δ	95% interval
$1/16$	2.50349	2.50868	-0.00519	[-0.0088, -0.0022]
$1/8$	2.49622	2.50096	-0.00475	[-0.0079, -0.0020]
$1/4$	2.49103	2.49541	-0.00438	[-0.0070, -0.0022]
$1/2$	2.48758	2.49054	-0.00296	[-0.0048, -0.0014]

96 nonoverlapping same-author prose segments; 6144 context tokens and a 1024-token read-only scoring window. Intervals resample the 96 paired segments, using 20,000 draws. Δ is SSW minus random. Dense NLL is 2.48431; these are not retrieval answer-token NLLs.

On continuation, SSW lowers NLL relative to random at each sparse budget, while dense gives the lowest NLL (Table 4). Intervals resample the 96 same-author segments (Appendix A.2). Additional score-direction and block-size contrasts are provided in `results/prose_controls.csv`.

C.3 Write participation

In the LaCT 16K development record, gathering retains 172,032 of 1,376,256 token–layer–head contributions (12.5%) in the counted key-forward and inner-backward operations. Full-size random masking retains the dense operation counts. These counts exclude reading, target preparation, selection, and optimizer work; the gathered-call counter is not a count of every state commit.