REPRISE: EFFICIENT LONG-CONTEXT DIFFUSION INFERENCE VIA DISTILLATION THROUGH REUSED KV CACHES

ABSTRACT

Bidirectional diffusion language models repeatedly refresh key–value (KV) caches over long prompts during generation, incurring substantial computational cost. Sparse attention and cache reuse reduce this cost but alter the context states used for prediction, which can reduce generation quality. To address these limitations, we propose REPRISE, which adapts models to sparse attention and cache reuse through dense-model distillation and selective replay. During training, the student makes predictions using sparse attention and reused caches. A frozen dense teacher supervises these predictions based on the student's current token sequence. Selective replay rebuilds the required cache computations so that later prediction losses can train cache construction without retaining the full generation graph. During inference, REPRISE refreshes prompt states less often than generation states and uses sparse attention for prompt refreshes. Compared with the widely used Prefix Cache and Dual Cache, REPRISE improves average scores by 8.72 and 7.67 points, respectively, while reducing time to first token (TTFT) by 48% and time per output token (TPOT) by 80–82% on average.

1 INTRODUCTION

Diffusion large language models (DLLMs) generate text by repeatedly predicting masked tokens in a partially filled sequence (Nie et al., 2025). Bidirectional attention lets each prediction use context from both sides and allows one forward pass to fill multiple positions. As the answer develops, the model repeatedly processes the prompt to refresh its key–value (KV) states. For long-context tasks, these refreshes recompute representations for tens of thousands of prompt tokens across successive decoding steps, making prompt processing a substantial part of inference cost.

Sparse attention and KV reuse offer complementary ways to reduce this cost. Sparse attention limits the interactions used to compute prompt representations, as illustrated in Fig. 1(a) (Wang et al., 2026b). KV reuse retains states across decoding steps instead of recomputing them, as illustrated in Fig. 1(b). Widely used Prefix Cache and Dual Cache policies refresh cached states at block boundaries (Liu et al., 2026; Wu et al., 2026b). Although combining sparsity and reuse accelerates inference (Song et al., 2026), it creates two linked challenges: **1) Quality under sparse, reused states.** In a bidirectional DLLM, prompt tokens attend to the evolving answer. New answer tokens can therefore change deeper prompt representations, leaving reused KV different from the states computed for the current sequence. Refreshing more often reduces this mismatch but repeatedly processes the long prompt; sparse attention further changes the context used to construct the states. Predictions made with both approximations can lose quality relative to the original dense computation.

2) Training through cache construction. A prompt cache built early in generation supports later predictions after the answer has changed. Training only on freshly recomputed states does not directly supervise those reused-cache predictions. Their losses must also pass through the cache to train the parameters that built it. Retaining the computation graph for every decoding step, however, increases training memory and computation. How can we train a DLLM with sparse, reused caches while preserving gradients through their construction without storing the full generation graph?

We address this question with REPRISE, which combines dense-model distillation and selective replay. During training, the student generates with sparse attention and reused caches, while a frozen dense teacher provides targets for the student's current token sequence. At selected generation refresh

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.

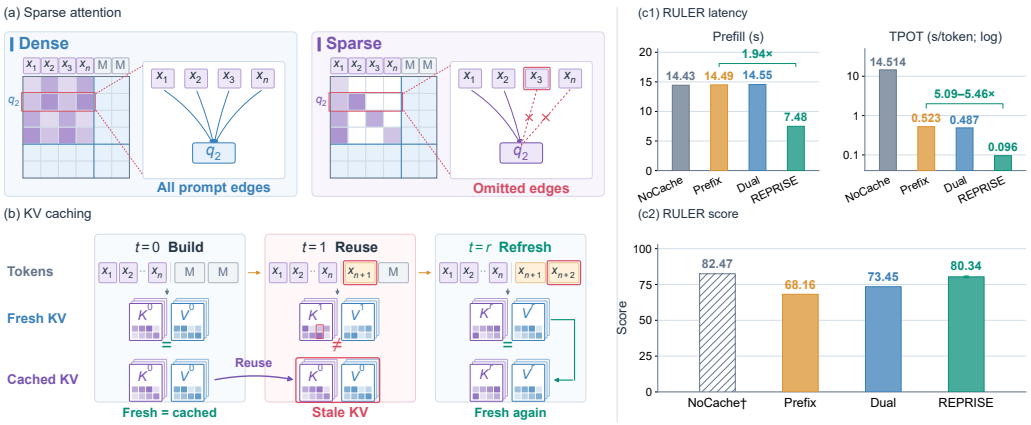


Figure 1: **Adapting DLLMs to sparse cached execution.** (a) Sparse attention omits interactions; both paths recompute KV. (b) Reused KV can become stale as answers change. Tensor values are schematic. (c1) Prefill latency and TPOT for NoCache, Prefix Cache, Dual Cache, and REPRISE on a RULER 64K input over 128 decoding steps. TPOT uses a logarithmic axis. (c2) Scores on RULER 64K. NoCache† uses native threshold decoding; the cached methods use the shared fixed-step protocol.

boundaries, replay rebuilds generation states from the current tokens and retained prompt states. It reconstructs the cache dependencies required by selected predictions without recomputing every earlier generation step, allowing their losses to train the parameters used in earlier cache construction.

During inference, REPRISE refreshes prompt and generation states at different rates. Our measurements show that prompt KV changes less than generation KV between adjacent decoding steps. Because the prompt occupies most positions in a long-context request, refreshing it less often avoids repeatedly processing a much larger region than the active generation block. Generation steps instead read retained prompt KV while updating generation states more frequently, and sparse attention reduces the cost of each prompt refresh. The learned parameter updates are merged into the model weights before deployment.

We evaluate REPRISE on RULER and LongBench V1/V2. On a RULER 64K input, Fig. 1(c1) shows average speedups of 1.94× in time to first token (TTFT) and 5.09–5.46× in time per output token (TPOT) over Prefix and Dual, while maintaining strong task scores. In Fig. 1(c2), its RULER 64K score exceeds Prefix and Dual by 12.18 and 6.89 points, respectively. The main contributions are summarized below.

- **Training for sparse cached inference:** We propose REPRISE to adapt bidirectional DLLMs to sparse attention and KV reuse through dense-teacher supervision of predictions made with reused states. During inference, separate prompt and generation refresh schedules combine with sparse prompt attention to reduce long-context computation.
- **Selective cache replay:** We introduce selective replay to reconstruct the cache dependencies of selected predictions at generation refresh boundaries. This allows later prediction losses to train the parameters used in earlier cache construction without retaining the full generation graph.
- **Quality and efficiency improvements:** We evaluate task quality and inference efficiency, demonstrating average score gains of 8.72 and 7.67 points over Prefix Cache and Dual Cache, respectively, alongside 1.94× TTFT and 5.09–5.46× TPOT speedups on average.

2 RELATED WORK

Efficient DLLM inference. SparseD and MAGE reuse selected attention patterns (Wang et al., 2026b; Kwon et al., 2026); Sparse-dLLM and Focus-dLLM prune cached context (Song et al., 2026; Long et al., 2026). KV reuse reduces recomputation through delayed caching (Ma et al., 2025), blockwise refresh (Wu et al., 2026b), or adaptive updates (Liu et al., 2026; Jiang et al., 2026; Nguyen-

Tri et al., 2026). FlashDLM combines caching with guided unmasking (Hu et al., 2026). Block Diffusion, discrete diffusion forcing, and Fast-dLLM v2 use block-causal attention to support cached decoding (Arriola et al., 2025; Wang et al., 2026a; Wu et al., 2026a). REPRISE retains bidirectional attention and all KV entries, and trains under sparse computation and approximate reuse.

Training for approximate execution. Self-distillation through time and CDLM reduce sampling steps (Deschenaux & Gulcehre, 2025; Kim et al., 2026). MAGE v1 also studies sparse-aware fine-tuning (Kwon et al., 2026). TOPD supervises student-generated traces with reverse KL, and dOPSD uses later denoising states as self-teacher targets (Ren et al., 2026; Dat et al., 2026). Self Gradient Forcing reconstructs differentiable context in video generation so that later losses can train historical KV states (Zhuang et al., 2026). REPRISE trains prompt caches in bidirectional DLLMs, where caches built from a partially generated answer are reused as generation progresses. It combines dense-teacher supervision with sparse cached execution and uses generation refresh boundaries to reconstruct the cache dependencies of selected losses.

3 PRELIMINARIES

Notation and masked decoding. Let P be the prompt length, G the number of generation positions, $N = P + G$, t the forward-call index, and x_t the token sequence before call t. Positions are indexed from zero. Prompt tokens stay fixed, and generation positions start masked. On each forward, a DLLM with weights θ predicts $p_\theta(\cdot \mid x_t, i)$ at masked position i using bidirectional context (Nie et al., 2025). A transfer rule selects which predicted tokens are written into the next sequence x_{t+1}. Blockwise decoding divides the generation region into blocks and uses prediction confidence to select positions to fill within the active block (Wu et al., 2026b).

Sparse attention. For an attention head of dimension d, let $Q_t, K_t, V_t \in \mathbb{R}^{N\times d}$ denote its query, key, and value matrices, with positional transformations included in Q_t, K_t. A nonempty set $S_{t,i} \subseteq \{0, \dots, N-1\}$ specifies the keys retained for query $q_{t,i}$. Scaled dot-product attention (Vaswani et al., 2017) gives

$$o_{t,i} = \mathrm{softmax}\left(\frac{q_{t,i}K_{t,S_{t,i}}^\top}{\sqrt{d}}\right) V_{t,S_{t,i}}. \tag{1}$$

Softmax normalizes over the selected keys. Fresh dense attention retains all N positions and recomputes every row. Sparse attention restricts $S_{t,i}$, often using token blocks.

KV reuse and refresh. At forward t, a refresh set $\mathcal{R}_t$ specifies which rows replace their stored KV; other entries are reused. These stored matrices replace K_t, V_t in Eq. 1. With bidirectional attention, new answer tokens can change deeper prompt states, making reuse approximate. Fast-dLLM's Prefix Cache reuses KV before the active block; Dual Cache also caches the masked suffix. Both refresh at block boundaries (Wu et al., 2026b). Appendix F bounds these local attention effects at a fixed query.

4 ANALYSIS OF SPARSE CACHED GENERATION

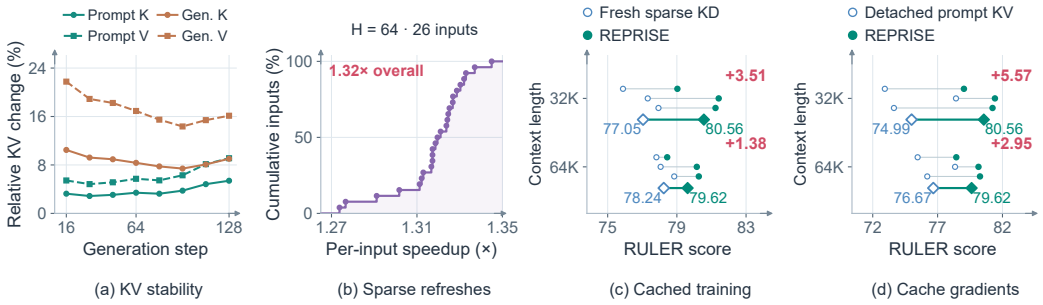
We examine prompt-state stability, the cost of sparse refreshes, and the effects of training with reused states and prompt-cache gradients.

4.1 DIFFERENT UPDATE RATES AND SPARSE REFRESHES

Prompt KV stability. We measure adjacent-step changes in prompt and generation KV during fresh dense decoding on two 4K inputs.

As shown in Fig. 2(a), prompt states change less than generation states for both K and V. Their mean relative changes are 3.73% and 6.26%, compared with 8.66% and 17.15% in the generation region. These measurements support a lower prompt refresh frequency. The measurement definition and sampling procedure are given in Appendix J.

Sparse prompt refreshes. We compare dense and sparse attention at a fixed prompt refresh interval of $H = 64$. Sparse attention retains a nominal 25% of prompt blocks and dense generation interactions. As shown in Fig. 2(b), it reduces mean request latency on 26 RULER 32K inputs from 30.98 to



(a) KV stability (b) Sparse refreshes (c) Cached training (d) Cache gradients

Figure 2: **Design rationale for REPRISE.** (a) KV state stability. Relative changes in the K/V states of the prompt and generation regions between adjacent generation steps during dense decoding on 4K inputs. (b) Speedup from sparse refreshes. Distribution of speedups from sparse attention over dense attention on RULER 32K inputs, with the prompt refresh interval fixed at $H = 64$. (c) Effect of training with cache reuse. RULER 32K and 64K scores of REPRISE and a variant distilled using only fresh sparse states. (d) Effect of cache gradients. RULER 32K and 64K scores of REPRISE and a variant trained with stop-gradient applied to the prompt KV cache.

23.51 seconds, a 1.32× speedup, with lower latency on every input. Cache reuse reduces the number of prompt refreshes, and sparsity reduces the cost of each refresh. The paired timing protocol is specified in Appendix A.5.

4.2 TRAINING FOR CACHE REUSE

Training with cache reuse. Under the shared settings in Section 6.1, we compare REPRISE with fresh sparse distillation, which recomputes all student states at each loss evaluation without reusing previously constructed KV caches. As shown in Fig. 2(c), REPRISE improves the mean RULER score by 3.51 points at 32K and 1.38 points at 64K, with gains for every seed at both lengths.

Train cache construction through later losses. We next retain cache reuse and detach the explicit gradient connections through the incoming prompt KV cache. This control preserves indirect gradients through generation-cache entries. For a fixed token trajectory, let ϕ denote the trainable parameters and $C(\phi)$ an earlier prompt cache. Write a later loss as $\mathcal{L}(\phi) = \ell(\phi, C(\phi))$, and let $J_C(\phi)$ denote the Jacobian of the cache with respect to the parameters. The gradient is

$$\nabla_\phi \mathcal{L} = \partial_\phi \ell + J_C(\phi)^\top \nabla_C \ell. \tag{2}$$

The second term passes gradients from later losses through the cache to the parameters that built it. As shown in Fig. 2(d), retaining this explicit gradient path improves all three seeds at both RULER lengths. Mean gains are 5.57 points at 32K and 2.95 points at 64K. Complete results and confidence intervals are given in Section 6.3 and Appendix D.

5 REPRISE

As shown in Fig. 3, REPRISE uses separate prompt and generation refresh schedules for inference and selective replay for training. The student first generates with sparse attention and cache reuse. Replay then reconstructs selected predictions so that their losses train both the reused prompt cache and the computations that consume it.

5.1 SPARSE CACHED EXECUTION

Refresh prompt and generation states separately. We allocate G token positions for generation and divide them into blocks of $B = 32$ positions. The prompt refresh interval is $H = 128$ generation positions. As shown in Fig. 3(a), full refreshes update both the prompt and generation regions, refreshes at other block boundaries update generation KV, and forward passes within a block recompute only its active rows.

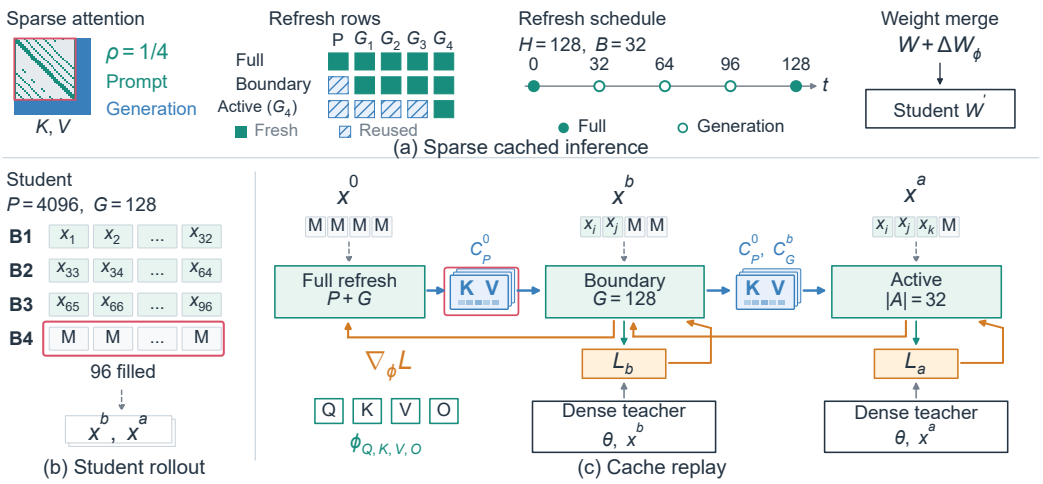


Figure 3: **REPRISE design.** (a) Sparse attention and the refresh schedule for one-token-per-forward decoding. Filled and open markers denote full and generation-wide refreshes; W' contains the merged updates. (b) Student trajectory collection, recording token states and logits. (c) Differentiable replay at initialization x^0, a selected generation boundary x^b, and the first subsequent active-block state x^a. The prompt cache C_P^0 and rebuilt generation cache C_G^b support the predictions supervised by $\mathcal{L}_b$ and $\mathcal{L}_a$. Blue, dashed gray, and orange arrows show cached states, saved tokens, and gradients, respectively. P denotes prompt rows, G_i denotes generation block i, and M denotes a mask.

With one token filled per forward pass, the active block at forward t is $\mathcal{A}_t = [P + B\lfloor t/B \rfloor,\ P + B(\lfloor t/B \rfloor + 1))$. The refresh set is

$$\mathcal{R}_t = \begin{cases} [0, P+G), & t \bmod H = 0, \\ [P, P+G), & t \bmod B = 0 \text{ and } t \bmod H \neq 0, \\ \mathcal{A}_t, & \text{otherwise.} \end{cases} \tag{3}$$

Each layer updates the specified KV rows before computing attention and reuses the remaining rows. Generation-wide refreshes cover both filled and masked positions. Under parallel decoding, the executor advances when the active block is complete and refreshes the prompt every $H/B = 4$ completed blocks.

Reduce the cost of full refreshes. The attention selector partitions the prompt into 128-token blocks and uses pooled Q/K scores to select $\lceil n/4 \rceil$ of the $n = \lfloor P/128 \rfloor$ complete key blocks for each query block. All heads share the selection. Full refreshes retain all KV entries and restrict attention between complete prompt blocks; the prompt tail and generation interactions remain dense. Exact selection and update rules are given in Appendix A.

5.2 TRAINING WITH REUSED STATES

Collect student trajectories. Let θ denote the frozen pretrained weights and ϕ the low-rank updates to the Q/K/V and output projections (Hu et al., 2021). The current student fills $G = 128$ generation positions in four blocks without retaining the sampling graph. Training uses threshold-based token transfer, so each forward pass can fill one or more positions. The model advances to the next block after filling the current block.

As shown in Fig. 3(b), we save token states and logits at each block boundary and at the first subsequent active-block forward pass, when available. Optimization step s, indexed from zero, selects block $s \bmod 4$ and its saved forward passes as loss probes $\mathcal{T}$. The boundary probe supervises predictions during a generation-state refresh; the active-block probe supervises predictions that reuse the resulting cache. Cycling through the four blocks trains on different stages of cache reuse.

Supervise the same token state with fresh context. We compute the distillation loss on student predictions recomputed by differentiable cache replay. At each selected probe t, the frozen teacher

processes the saved token sequence x_t with full dense attention and no cache reuse. The replayed student predicts from the same x_t and the reconstructed cache C_t under the sparse cached policy. For masked positions $\mathcal{M}_t$ in the selected block, the objective is

$$\mathcal{L}(\phi) = \frac{1}{|\mathcal{T}|} \sum_{t \in \mathcal{T}} \frac{1}{|\mathcal{M}_t|} \sum_{i \in \mathcal{M}_t} \mathrm{KL}\Big(p_\theta^{\mathrm{dense}}(\cdot \mid x_t, i) \,\|\, p_{\theta,\phi}^{\mathrm{sparse,cache}}(\cdot \mid x_t, C_t, i) \Big) . \tag{4}$$

We use full-vocabulary KL divergence at temperature 1 as the sole training loss.

5.3 Differentiable Cache Replay

Rebuild the prompt cache. As shown in Fig. 3(c), REPRISE reconstructs the cache computations needed by the selected loss probes, holding the saved tokens, transfer decisions, and block selection fixed. The 128-position training window has one full refresh, at initialization. We rerun this initial forward pass with gradients enabled to rebuild prompt KV. Layerwise activation checkpointing reduces activation storage.

Replay the selected boundary and active-block pass. At the selected generation refresh boundary, each layer overwrites all generation KV entries using the saved token state and rebuilt prompt cache. These computations reconstruct the cache required by the selected probes, allowing replay to skip earlier active-block passes. We then replay the first subsequent active-block pass, when available. For block zero, the initial full forward pass also serves as the boundary probe.

In the fourth-block example, boundary replay rebuilds generation KV after 96 positions have been filled. The next replay updates the 32 active rows. Losses at these states train both the current computation and the parameters that built the initial prompt cache.

Appendix B gives the full algorithm and verifies that replay reproduces the recorded predictions and preserves the intended gradient paths.

5.4 Implementation and Deployment

We train rank-8 updates to the attention projections using 64 WikiText-2 articles (Merity et al., 2016), 4K prompts, and $G = 128$ generation positions. Each run uses 64 AdamW updates with the pretrained weights frozen. The trainable parameters account for 0.105% of the backbone. All training variants use the same data and update budget (Appendix B).

We evaluate the same adapted checkpoints at every context length. Training at 4K and testing on longer inputs measures the transfer of sparse-cache adaptation within UltraLLaDA's existing long-context capability.

We merge the learned updates into BF16 model weights before deployment. As shown in Fig. 3(a), inference follows the sparse cached policy, with teacher supervision and replay confined to training. Trained variants use the same inference modules, refresh schedules, and selection rule; their learned Q/K projections determine the selected attention blocks.

6 Evaluation

6.1 Setup

Model and baselines. We evaluate UltraLLaDA-8B, a bidirectional model that supports contexts up to 128K (Relaxed System Lab, n.d.; Nie et al., 2025). Controlled evaluations use one NVIDIA A800 80GB GPU per request. Prefix and Dual implement the Fast-dLLM cache policies in our evaluation engine (Wu et al., 2026b). The RULER NoCache references in Table 1 use UltraLLaDA's native decoder; other NoCache evaluations use fresh dense attention in the common engine. All baselines use pretrained weights. The main quality comparison and training ablations report the mean and standard deviation across three independently trained REPRISE models, each with its updates merged into the backbone.

Benchmarks and fixed-step decoding. We evaluate RULER at 32K, 64K, and 128K and LongBench V1/V2 under the corresponding input caps (Hsieh et al., 2024; Bai et al., 2024; 2025). Prefix, Dual,

and REPRISE share 104 inputs per setting. RULER and V1 report equally weighted means over 13 and 21 tasks; V2 reports direct-answer accuracy. Fixed-step evaluations transfer one token per forward, execute the full generation budget, and truncate at EOS for scoring. Cached RULER uses 256 forwards. Cached KV uses BF16 at 32K/64K and FP8 at the 128K cap. Full input and decoding specifications are in Appendix L and Appendix M.

Training ablations. We compare REPRISE with fresh sparse distillation and training with detached prompt caches. RULER ablations use 416 inputs at each of 32K and 64K, with a separate 64K set from the main comparison. LongBench ablations cover eight English tasks with 32 inputs each. Trained variants share the teacher, data order, adaptation rank, training seeds, update budget, and inference policy. Each variant collects its own training trajectories. We report source-paired bootstrap intervals for the score differences (Appendix B, Appendix C).

Parallel decoding and latency. We jointly measure task scores and complete-generation latency on 26 RULER 32K inputs and 24 LongBench inputs from eight tasks. Methods share a confidence threshold of 0.9, 32-position blocks, and EOS stopping once preceding positions are filled. Paired requests use BF16 KV on the same GPU. We evaluate one REPRISE checkpoint with one timing measurement per method and input; 13 RULER inputs also include NoCache under this shared decoder (Appendix P). Fixed-step prefill/TPOT measurements and repeated execution timings are reported in Appendix N and Appendix A.5.

6.2 COMPARISON WITH PREFIX AND DUAL

Table 1: Main benchmark results on 104-input subsets. RULER and V1 use equally weighted task scores; V2 reports accuracy (%). REPRISE reports the mean ± SD across three seeds. RULER NoCache uses a native decoder (†) and a separate 208-input set at 128K (‡). Bold marks the best score among comparable methods, excluding these references. LongBench lengths are input caps.

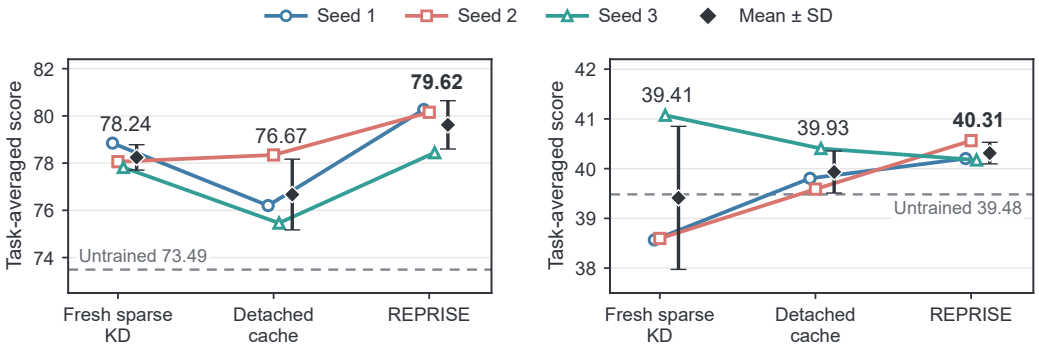
Benchmark	Length	Prefix	Dual	NoCache	REPRISE
RULER	32K	69.79	73.17	83.11†	**78.73** ± 1.47
	64K	68.16	73.45	82.47†	**80.34** ± 0.60
	128K	49.55	48.37	71.39‡	**67.70** ± 0.34
LongBench V1	32K	37.75	37.56	**39.49**	38.32 ± 0.94
	64K	37.68	37.51	**39.52**	38.38 ± 0.99
	128K	36.88	37.28	**39.31**	38.90 ± 0.88
LongBench V2	32K	4.81	5.77	4.81	**19.23** ± 0.96
	64K	4.81	3.85	3.85	**16.99** ± 2.00
	128K	1.92	3.85	4.81	**11.22** ± 2.78

REPRISE achieves higher mean scores than Prefix and Dual in all nine settings, with average gains of 8.72 and 7.67 points, respectively (Table 1). On RULER 64K, it improves scores by 12.18 and 6.89 points, respectively. The largest gains occur in the 128K configuration, where REPRISE scores 67.70, exceeding Prefix by 18.15 points and Dual by 19.33 points.

Across the three independently generated RULER length settings, REPRISE's score decreases by 11.03 points from 32K to 128K, compared with drops of 20.24 and 24.80 points for Prefix and Dual.

On LongBench V1, REPRISE has higher mean scores than both cache baselines at every input cap and remains within 0.41–1.17 points of NoCache. At the 128K cap, it scores 38.90 and recovers 83% of Prefix's gap to NoCache. The source-paired confidence intervals include zero, so the gains remain uncertain on this subset (Appendix L.1). Appendix L defines the gap-recovery metric.

On LongBench V2, REPRISE reaches 11.22–19.23% accuracy. Most additional correct answers occur on questions for which the baselines produce unparseable responses. On questions where both models produce valid answers, correctness differences vary across settings. The evidence therefore supports improved output validity more clearly than improved long-context reasoning. Appendix L.2 gives the paired correctness and failure-mode analysis.



(a) RULER 64K (416 inputs)

(b) LongBench V1 (256 inputs)

Figure 4: **Training controls under a shared inference policy.** (a) RULER 64K, 416 inputs. (b) Eight-task LongBench V1, 256 inputs. Colored lines pair the same three training seeds across objectives. Black diamonds and error bars show the mean and sample SD across seeds; gray dashed lines mark the untrained model.

As shown in Fig. 1(c1), REPRISE reduces TTFT by approximately 48% and TPOT by 80–82% relative to Prefix and Dual on the RULER 64K input. Prefill and decode timings across the evaluated inputs are reported in Appendix N.

Table 2: **Quality and complete-generation latency under shared parallel decoding.** Scores average tasks equally. Forward evaluations (NFE), output tokens excluding EOS, and latency average requests. NoCache is evaluated on 13 of the 26 RULER inputs. The shared decoding and timing settings are specified in Section 6.1.

Benchmark (n)	Method	Score	NFE	Output	Time (s)
RULER 32K (26)	Prefix	68.91	58.7	256.0	44.39
	Dual	74.87	63.5	256.0	43.47
	REPRISE	**83.46**	63.5	256.0	**8.62**
LongBench V1 (24)	Prefix	40.52	63.2	116.0	5.18
	Dual	40.46	64.2	116.0	4.68
	REPRISE	**41.77**	65.8	116.0	**3.03**
Matched RULER 32K (13)	NoCache	81.03	55.5	256.0	291.14
	Prefix	66.92	62.6	256.0	44.83
	Dual	72.69	64.3	256.0	43.71
	REPRISE	**82.05**	66.5	256.0	**8.74**

Under shared parallel decoding, REPRISE achieves a 5.04× speedup over Dual on RULER 32K and improves the score by 8.59 points, with a 95% paired confidence interval of [1.86, 15.32] (Table 2). Both methods average 63.5 forward evaluations. On LongBench V1, REPRISE achieves a 1.55× speedup, with scores of 41.77 versus 40.46. The score-difference interval of [-3.55, 6.95] leaves the quality gain uncertain. All three cached methods reach the output caps on all 50 inputs. Paired intervals and task-level results are in Appendix P.

We also compare REPRISE with SparseD and dLLM-Cache implementations adapted to our evaluation framework on the ablation sets. On RULER 32K, REPRISE scores 80.56, compared with 71.59 for dLLM-Cache and 83.26 for SparseD. On the eight-task LongBench subset, REPRISE, SparseD, and dLLM-Cache score 40.31, 39.68, and 39.33, with mean request latencies of 5.95, 107.69, and 15.66 seconds, respectively. REPRISE has similar mean scores and lower measured latency on this subset. Complete results and implementation details are given in Appendix K and Appendix C.

6.3 Training ablations and cost

As shown in Fig. 4(a), REPRISE improves the RULER 64K score from 73.49 for the untrained sparse model to 79.62, a gain of 6.13 points under the shared inference policy.

REPRISE also exceeds fresh sparse distillation by 1.38 points and detached prompt-cache training by 2.95 points, with gains for every matched seed. The respective 95% paired bootstrap confidence intervals are [0.40, 2.43] and [1.74, 4.23]. These comparisons support training with reused states and retaining prompt-cache gradients under the evaluated budget. Full task and seed results are given in Appendix D.

As shown in Fig. 4(b), REPRISE scores 40.31 on the eight-task LongBench subset, compared with 39.41 for fresh sparse distillation and 39.93 for detached prompt-cache training. The respective gains are 0.90 and 0.38 points, with 95% paired confidence intervals of [-0.36, 2.18] and [-1.34, 2.06]. Both intervals include zero, leaving the quality gains over these controls uncertain. Task and seed results are given in Appendix O.

The 64-update training loop averages 16.64 minutes per seed (Appendix B).

6.4 Refresh frequency and sparse attention

Table 3 uses 26 RULER 32K inputs, two per task, with $G = 256$, $B = 32$, and 256 forwards. Each configuration runs three times per input; we discard the first and average the remaining two. Inputs and physical GPUs are paired (Appendix A.5).

Table 3: **Repeated RULER 32K timing with KV caching.** Refreshes include initialization; 151–153 denote training seeds.

	Dense	Dense	Sparse	REPRISE		
				Seed 151	Seed 152	Seed 153
Refresh interval H	32	64	64	128	128	128
Full refreshes	8	4	4	2	2	2
Mean time (s)	51.29	30.98	23.51	17.15	17.19	17.06

Increasing H from 32 to 64 with dense cached attention halves full refreshes and gives a 1.66× speedup. At fixed $H = 64$, sparsity gives a further 1.32× speedup. REPRISE averages 17.13 seconds, a 2.99× speedup over dense caching at $H = 32$. REPRISE changes both weights and refresh frequency; Section 6.3 evaluates training effects under a shared inference policy.

7 Conclusion and limitations

REPRISE adapts DLLMs to sparse attention and KV reuse through dense-model distillation and selective replay. Selective replay allows later prediction losses to backpropagate through earlier cache computations. Across the evaluated RULER and LongBench settings, average scores improve over Prefix and Dual by 8.72 and 7.67 points, respectively. REPRISE achieves approximately 1.94× TTFT and 5.09–5.46× TPOT speedups on a RULER 64K input. RULER ablations support training with reused states and retaining prompt-cache gradients. Under shared parallel decoding, REPRISE also achieves complete-generation speedups over Dual of 5.04× on RULER 32K and 1.55× on LongBench V1, with an 8.59-point score gain on RULER 32K.

Score gains on LongBench V1 remain uncertain in the paired analyses. The evaluation covers one backbone, fixed input subsets, one adaptation budget, and one parallel-decoding threshold. The parallel-decoding study uses a single checkpoint and one timing measurement per input; its matched NoCache comparison covers 13 RULER inputs. The 128K cached quality results also reflect FP8 KV storage. LongBench V2 gains mainly occur on questions with invalid baseline outputs, limiting conclusions about reasoning ability.

AI USE STATEMENT

Large language models were used to assist with manuscript writing, translation, and language polishing. The authors take full responsibility for the final content of this paper.

REFERENCES

Marianne Arriola, Aaron Gokaslan, Justin Chiu, Zhihan Yang, Zhixuan Qi, Jiaqi Han, Subham Sahoo, and Volodymyr Kuleshov. Block diffusion: Interpolating between autoregressive and diffusion language models. In *International Conference on Learning Representations*, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/hash/7ede97c3e082c6df10a8d6103a2eebd2-Abstract-Conference.html.

Yushi Bai, Xin Lv, Jiajie Zhang, Hongchang Lyu, Jiankai Tang, Zhidian Huang, Zhengxiao Du, Xiao Liu, Aohan Zeng, Lei Hou, Yuxiao Dong, Jie Tang, and Juanzi Li. LongBench: A bilingual, multitask benchmark for long context understanding. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 3119–3137, 2024. doi: 10.18653/v1/2024.acl-long.172. URL https://aclanthology.org/2024.acl-long.172/.

Yushi Bai, Shangqing Tu, Jiajie Zhang, Hao Peng, Xiaozhi Wang, Xin Lv, Shulin Cao, Jiazheng Xu, Lei Hou, Yuxiao Dong, Jie Tang, and Juanzi Li. LongBench v2: Towards deeper understanding and reasoning on realistic long-context multitasks. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 3639–3664. Association for Computational Linguistics, 2025. doi: 10.18653/v1/2025.acl-long.183. URL https://aclanthology.org/2025.acl-long.183/.

Phuong Tuan Dat, Qi Li, and Xinchao Wang. dOPSD: On-policy self-distillation for diffusion language models, 2026. URL https://arxiv.org/abs/2607.04428v1. Version 1, July 2026.

Justin Deschenaux and Caglar Gulcehre. Beyond autoregression: Fast LLMs via self-distillation through time. In *International Conference on Learning Representations*, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/hash/0f93c3e9b557980d93016671acd94bd2-Abstract-Conference.html.

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?, 2024. URL https://arxiv.org/abs/2404.06654. COLM 2024.

Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. LoRA: Low-rank adaptation of large language models, 2021. URL https://arxiv.org/abs/2106.09685.

Zhanqiu Hu, Jian Meng, Yash Akhauri, Mohamed Abdelfattah, Jae-sun Seo, Zhiru Zhang, and Udit Gupta. FlashDLM: Accelerating diffusion language model inference via efficient KV caching and guided diffusion. In *International Conference on Learning Representations*, 2026. URL https://proceedings.iclr.cc/paper_files/paper/2026/hash/f930c6e1292a1160189a0734f22b465b-Abstract-Conference.html.

Yuchu Jiang, Yue Cai, Xiangzhong Luo, Jiale Fu, Jiarui Wang, Chonghan Liu, and Xu Yang. d^2Cache: Accelerating diffusion-based LLMs via dual adaptive caching. In *International Conference on Learning Representations*, 2026. URL https://proceedings.iclr.cc/paper_files/paper/2026/hash/de4f2e3b8263aff39f0dd7571a8dc4b1-Abstract-Conference.html.

Minseo Kim, Chenfeng Xu, Coleman Hooper, Harman Singh, Ben Athiwaratkun, Ce Zhang, Kurt Keutzer, and Amir Gholami. CDLM: Consistency diffusion language models for faster sampling. In *Proceedings of Machine Learning and Systems*, volume 8, 2026. URL https://proceedings.mlsys.org/paper_files/paper/2026/hash/054de805fcceb78a201f5e9d53c85908-Abstract-Conference.html.

Omin Kwon, Yeonjae Kim, Doyeon Kim, Minseo Kim, Yeonhong Park, and Jae W. Lee. MAGE: All-[MASK] block already knows where to look in diffusion LLM, 2026. URL https://arxiv.org/abs/2602.14209v1. Version 1, including the fine-tuning study.

Zhiyuan Liu, Yicun Yang, Yaojie Zhang, Junjie Chen, Chang Zou, Qingyan Wei, Shaobo Wang, Yichen Zhu, and Linfeng Zhang. dLLM-Cache: Accelerating diffusion large language models with adaptive caching, 2026. URL https://arxiv.org/abs/2506.06295v3. Version 3, revised June 2026.

Lingkun Long, Yushi Huang, Shihao Bai, Ruihao Gong, Jun Zhang, Ao Zhou, and Jianlei Yang. Focus-dLLM: Accelerating long-context diffusion LLM inference via confidence-guided context focusing. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 12146–12160, 2026. doi: 10.18653/v1/2026.acl-long.556. URL https://aclanthology.org/2026.acl-long.556/.

Xinyin Ma, Runpeng Yu, Gongfan Fang, and Xinchao Wang. dKV-Cache: The cache for diffusion language models. In *Advances in Neural Information Processing Systems*, volume 38, 2025. doi: 10.52202/085713-4984. URL https://proceedings.neurips.cc/paper_files/paper/2025/hash/db0ee27cb50dd9087b133f6e7d28a90e-Abstract-Conference.html.

Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. Pointer sentinel mixture models, 2016. URL https://arxiv.org/abs/1609.07843.

Quan Nguyen-Tri, Mukul Ranjan, and Zhiqiang Shen. Attention is all you need for KV cache in diffusion LLMs. In *International Conference on Learning Representations*, 2026. URL https://proceedings.iclr.cc/paper_files/paper/2026/hash/3afaa2102fb8ea44cbadc13e45bba718-Abstract-Conference.html.

Shen Nie, Fengqi Zhu, Zebin You, Xiaolu Zhang, Jingyang Ou, Jun Hu, Jun Zhou, Yankai Lin, Ji-Rong Wen, and Chongxuan Li. Large language diffusion models. In *Advances in Neural Information Processing Systems*, volume 38, 2025. doi: 10.52202/085713-1689. URL https://proceedings.neurips.cc/paper_files/paper/2025/hash/48b383b24230e0e6e649d9c98dae4d8c-Abstract-Conference.html.

Relaxed System Lab. UltraLLaDA model card. Hugging Face model repository, n.d. URL https://huggingface.co/relaxe-system-lab/UltraLLaDA. Accessed September 19, 2026.

Haolin Ren, Ziyang Huang, Chenhao Yuan, Jun Zhao, and Kang Liu. Trace-based on-policy distillation for masked diffusion language models, 2026. URL https://arxiv.org/abs/2607.16872v1. Version 1, July 2026.

Yuerong Song, Xiaoran Liu, Ruixiao Li, Zhigeng Liu, Zengfeng Huang, Qipeng Guo, Ziwei He, and Xipeng Qiu. Sparse-dLLM: Accelerating diffusion LLMs with dynamic cache eviction. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 40, pp. 33038–33046, 2026. doi: 10.1609/aaai.v40i39.40586. URL https://ojs.aaai.org/index.php/AAAI/article/view/40586.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems*, volume 30, 2017. URL https://papers.nips.cc/paper/2017/hash/3f5ee243547dee91fbd053c1c4a845aa-Abstract.html.

Xu Wang, Chenkai Xu, Yijie Jin, Jiachun Jin, Hao Zhang, Kai Yu, and Zhijie Deng. Diffusion LLMs can do faster-than-AR inference via discrete diffusion forcing. In *International Conference on Learning Representations*, 2026a. URL https://proceedings.iclr.cc/paper_files/paper/2026/hash/04bb76a98d9f32226b3beba7bd26a51f-Abstract-Conference.html.

Zeqing Wang, Gongfan Fang, Xinyin Ma, Xingyi Yang, and Xinchao Wang. SparseD: Sparse attention for diffusion language models. In *International Conference on Learning Representations*, 2026b. URL https://proceedings.iclr.cc/paper_files/paper/2026/hash/a1598ba0aa7bd7bdf18e0ad04d993edc-Abstract-Conference.html.

Chengyue Wu, Hao Zhang, Shuchen Xue, Shizhe Diao, Yonggan Fu, Zhijian Liu, Pavlo Molchanov, Ping Luo, Song Han, and Enze Xie. Fast-dLLM v2: Efficient block-diffusion LLM. In *International Conference on Learning Representations*, 2026a. URL https://proceedings.iclr.cc/paper_files/paper/2026/hash/d0865cbe51d35ec322f9af9db7806fc7-Abstract-Conference.html.

Chengyue Wu, Hao Zhang, Shuchen Xue, Zhijian Liu, Shizhe Diao, Ligeng Zhu, Ping Luo, Song Han, and Enze Xie. Fast-dLLM: Training-free acceleration of diffusion LLM by enabling KV cache and parallel decoding. In *International Conference on Learning Representations*, 2026b. URL https://proceedings.iclr.cc/paper_files/paper/2026/hash/5d8d4e6061c3ba96c240b7fa1ae3471d-Abstract-Conference.html.

Zihao Ye, Lequn Chen, Ruihang Lai, Wuwei Lin, Yineng Zhang, Stephanie Wang, Tianqi Chen, Baris Kasikci, Vinod Grover, Arvind Krishnamurthy, and Luis Ceze. FlashInfer: Efficient and customizable attention engine for LLM inference serving. In *Proceedings of Machine Learning and Systems*, volume 7, 2025. URL https://proceedings.mlsys.org/paper_files/paper/2025/hash/dbf02b21d77409a2db30e56866a8ab3a-Abstract-Conference.html.

Junhao Zhuang, Shiyi Zhang, Yuxuan Bian, Yaowei Li, Yawen Luo, Yijun Liu, Weiyang Jin, Songchun Zhang, Xianglong He, Xuying Zhang, Haoran Li, Haoyang Huang, Zeyue Xue, and Nan Duan. Self gradient forcing: Native long video extrapolation, 2026. URL https://arxiv.org/abs/2607.20368.

A Inference specification

A.1 Notation and transfer semantics

Table 4: Notation and execution settings. Refresh intervals are measured in generation positions.

Symbol or configuration	Meaning
P, G	Prompt token count and allocated generation token count
$B = 32$	Active generation block size
$H = 128$	Prompt refresh interval
Generation refresh	Update all generation KV at each 32-position boundary
Sparse prompt attention	Retain $\lceil n/4 \rceil$ of n complete prompt key blocks per query block
REPRISE	Explicit reused prompt KV connections remain differentiable
Detached prompt cache	The explicit incoming prompt KV slices are detached at consumers
Fresh sparse distillation	Student loss probes recompute the full current state without old KV
Dense cached	Dense attention with approximate $H = 128$, $B = 32$ caching
Fresh dense	Dense attention and all transformer rows recomputed each forward

During the fixed-budget evaluation, the raw-logit argmax determines a predicted token at each masked position of the active block. Its confidence is computed by an FP64 softmax. The highest-confidence eligible position is transferred, breaking ties by the earliest position. Invalid outputs, nonfinite values, or a still-masked completed canvas fail validation. The fixed generation budget is retained even after an EOS prediction. For scoring, the prefix before the first EOS is decoded with special tokens skipped, then processed by the task's prescribed normalization and metric.

Algorithm 1. Fixed-step sparse cached generation

Input prompt tokens, positive budget G divisible by $B = 32$, $H = 128$.
1. Initialize $x = [\text{prompt}, \text{MASK}^G]$ and empty layer caches.
2. For $t = 0, \ldots, G-1$, identify the active block and $\mathcal{R}_t$ in Eq. (3).
3. Embed $x[\mathcal{R}_t]$ and, in every transformer layer, perform the following.
   a. Project normalized rows to Q/K/V; apply absolute-position RoPE to Q/K.
   b. Replace their KV entries, retaining all other cached entries.
   c. At a full refresh with $P \geq 4096$, select prompt-core blocks and apply sparse prompt-core attention with dense tail and generation interactions.
   d. Otherwise attend densely from $\mathcal{R}_t$ to all cached keys.
   e. Apply output projections, residual updates, and the row-local MLP.
4. Evaluate final-normalized vocabulary logits for the active block only.
5. Transfer one eligible highest-confidence prediction into x.
6. Validate and return all G token IDs; truncate at EOS only for scoring.

A newly filled token affects its representation when its row is next recomputed. Generation-wide refreshes update filled and masked positions under bidirectional attention.

A.2 Model configuration and numerical backends

Base weights and tokenizer use `relaxe-system-lab/UltraLLaDA` revision `a02bb0490ddbe810bc3ac0deb667ed8fac2e8d47`. The model has 32 layers, hidden width 4,096, 32 query and KV heads, MLP width 12,288, and vocabulary size 126,464. Embeddings and the output head are untied. MASK and EOS have IDs 126336 and 126081. RoPE uses base 281,000,000 with FP32 rotations cast back to BF16. The rotary table starts at the configured `max_sequence_length`=4096 and extends to the requested length, without position interpolation or changing the base.

The BF16 control evaluations use PyTorch 2.9.1+cu128, CUDA 12.8, driver 550.54.14, and an A800-SXM4-80GB. Requests run serially on one GPU, with BF16 weights/KV and TF32 matrix multiplication disabled. Sparse prompt attention uses FlashInfer's block-sparse `fa2` backend and a 128 MiB workspace; dense remainder attention and state merging use its prefill and merge operators (Ye et al., 2025). Other dense calls use PyTorch SDPA without a forced backend.

A.3 EXACT POOLED SELECTOR

Let $n = \lfloor P/128 \rfloor$, N_h be the number of query heads, and d their head dimension. The prompt core consists of its first $128n$ positions. Within each complete block, take four contiguous 32-token groups. For head h, query block a, key block b, and groups $u, v \in \{1, 2, 3, 4\}$, denote the FP32 means of post-RoPE Q/K by $\bar{q}_{hau}$ and $\bar{k}_{hbv}$. The block importance is

$$z_{haub} = \max_{v} \frac{\bar{q}_{hau}^{\top} \bar{k}_{hbv}}{\sqrt{d}}, \tag{5}$$

$$w_{ab} = \frac{1}{4N_h} \sum_{h,u} \frac{\exp z_{haub}}{\sum_{b'} \exp z_{haub'}}. \tag{6}$$

For each query block a, the mandatory indices are the valid members of $\{0, n-1, a-1, a, a+1\}$. Set their importance to positive infinity and retain the top $m = \lceil n/4 \rceil$ indices with a stable descending sort. Ties select the lower key-block index. The retained indices are sorted for the sparse operator. The budget of m retained indices includes the mandatory set. With the deployed dense fallback for $P < 4096$, the mandatory set fits in the budget. Ranking is under stop-gradient.

The selector is recomputed at every full refresh and every layer. The selection is shared across heads, while Q/K/V values remain head-specific. The complete prompt tail is retained at its original length. For core query rows, block-sparse attention over core keys and dense attention over tail/generation keys are combined using their log-sum-exp normalizers. Tail and generation query rows use dense attention to all keys. The partition follows an exact softmax identity on the selected graph in real arithmetic.

A.4 STATIC WORK AND RETAINED MEMORY

Under one-token transfer, assume G and H are positive multiples of B, as in our fixed-budget evaluation. Let $L = P + G$. Full refreshes, including initialization, number $F = \lceil G/H \rceil$. The remaining generation-wide refreshes number $A = G/B - F$, and active-block forwards number $U = G - G/B$. The recomputed query rows per layer are

$$W_Q = FL + AG + UB. \tag{7}$$

At a full refresh, the number of logical attention edges per head is

$$E_{\text{full}} = L^2 - n(n-m)128^2, \qquad W_E = FE_{\text{full}} + AGL + UBL. \tag{8}$$

Set $m = n$ for dense fallback. Tails and generation interactions remain dense. At fixed P, G, B, H, W_Q and KV storage remain constant as m varies, and W_E changes with m. Converting these query-row and attention-edge counts to FLOPs or latency also requires the costs of selection, projections, MLPs, vocabulary heads, planning, merging, and memory traffic.

The method stores all KV entries. With N_ℓ layers, N_{KV} KV heads of the same dimension d, and b bytes per element, the cache requires

$$M_{\text{KV}} = 2N_\ell L N_{\text{KV}} d b. \tag{9}$$

For 32 layers, 32 KV heads, $d = 128$, BF16, and an illustrative 130,704-token prompt plus 256 generation positions, this is approximately 63.95 GiB. The 8,015,581,184 BF16 parameters add approximately 14.93 GiB. Together, the weights and KV occupy 78.88 GiB before activations and workspaces, leaving little device memory for the remaining 128K computation.

A.5 REPEATED TIMING DIAGNOSTIC

As shown in Fig. 2(b), sparse attention at $H = 64$ reduces mean request latency from 30.98 to 23.51 seconds on 26 paired RULER inputs, a 1.32× speedup. Prompt refresh intervals of $H = 32$, 64, and 128 require eight, four, and two full refreshes, respectively. The trained $H = 128$ configuration averages 17.13 seconds. Table 5 reports the complete repeated measurements.

The repeated timing experiment uses 26 RULER 32K inputs, two per task, with $G = 256$ and $B = 32$. Each of six configurations runs three times per input. We discard the first run as warm-up and average

the remaining two. The dense configurations use KV caching with their specified refresh intervals. Execution order is specified in Appendix E.

Table 5: **Repeated RULER 32K timing.** Mean request latency on 26 inputs, averaged over the second and third repetitions. Dense configurations use KV caching. Seeds 151–153 abbreviate the three REPRISE training seeds.

	Dense	Dense	Sparse	REPRISE $H = 128$		
Setting	$H = 32$	$H = 64$	$H = 64$	Seed 151	Seed 152	Seed 153
Seconds	51.29	30.98	23.51	17.15	17.19	17.06

B Training details and intervention checks

B.1 Data, parameters, and optimization

The 64 WikiText-2-raw training articles each supply 4,096 contiguous prompt tokens and an initially masked 128-position continuation. Every Q/K/V/output projection in the 32-layer frozen backbone receives a rank-8 FP32 branch with update $\Delta W = W_{\mathrm{up}} W_{\mathrm{down}}$. The down and up matrices have shapes 8×4096 and 4096×8. The scale is one, with no additional division by rank. Down matrices use PyTorch's Kaiming uniform initialization with $a = \sqrt{5}$; up matrices start at zero. Dropout is disabled. Only the 8,388,608 adaptation parameters are optimized, approximately 0.105% of the 8,015,581,184 frozen backbone parameters.

We use the `Salesforce/wikitext` repository, configuration `wikitext-2-raw-v1`, train split, at revision `b08601e04326c79dfdd32d625aee71d232d685c3`. First-level headings reconstruct 629 articles. We exclude an entire article if it shares a contiguous 13-word span with the archived LongBench context/question inputs or either of two archived RULER 128K input sets. Words use NFKC normalization, case folding, and ASCII alphanumeric extraction; answers and scores are not used. This removes 118 articles.

With seed 70908311, SHA256 ordering of whole-article hashes and greedy title/span deduplication select eight admission, 16 development, then 64 training articles. Articles shorter than 4,224 tokens or containing special tokens are skipped. A separate hash selects each contiguous 4,224-token window, without padding or a chat template. Adaptation uses the first 4,096 tokens as its prompt and teacher predictions as its targets. All nine training runs consume the same 64 article IDs once in the same order.

A literal-span check finds no overlap with the 8,437 registered inputs. This check covers exact text matches; semantic overlap, pretraining contamination, and subsequently generated RULER prompts remain unassessed.

Each run performs 64 single-article AdamW updates with constant learning rate 10^{-4}, $(\beta_1, \beta_2) = (0.9, 0.999)$, optimizer $\varepsilon = 10^{-8}$, zero weight decay, and gradient norm clipping at one. AMSGrad is disabled. All nine training runs use PyTorch 2.9.1 with CUDA 12.8. Seeds are 202609151, 202609152, and 202609153. We evaluate every final checkpoint, with no selection by test or validation scores.

B.2 Trajectory collection and bounded replay

The student fills four 32-position blocks. At each boundary or active-block call, it transfers all masked positions with confidence at least 0.9, always including the highest-confidence masked position. Predictions use the raw-logit argmax; confidence uses FP64 softmax, with earliest-position tie breaking. Each block permits at most 32 forwards. Recorded training rollouts use 38–128 forwards. We save boundary states/logits, the first active state/logits when present, and the transfer trace.

Zero-based update $s \in \{0, \ldots, 63\}$ selects block $b = s \bmod 4$. Replay begins at the initial full prompt refresh, the only one in the training window. For $b > 0$, replay reconstructs every generation KV entry at every layer at the selected boundary, using the saved tokens and earlier prompt KV. The

first active-block probe then reads this reconstructed boundary cache. The teacher uses each full saved token state with original weights and fresh dense attention. FP32 full-vocabulary KL first averages masked positions within each probe and then assigns equal weight to each probe mean. Each update uses the boundary probe and, when available, the first active-block probe. The backward pass holds the discrete rollout fixed.

Algorithm 2. One cached-student distillation update

1. Collect a no-gradient rollout using the current student and sparse cache policy.
2. Select block $b = s \bmod 4$; recover its boundary and any first active state.
3. Recompute the initial full forward with gradients and checkpointed layers. For $b = 0$, this is also the boundary probe. For $b > 0$, replay its saved boundary, updating all 128 generation rows while retaining prompt KV.
4. If a first active state exists, replay its 32 query rows using the boundary cache. For the detached control, concatenate detached prompt KV slices with attached generation KV slices at every cache input in steps 3–4.
5. Require replay logits to equal the recorded logits with zero tolerance.
6. Compute fresh dense teacher logits at each identical current token state.
7. Evaluate Eq. (4), backpropagate, clip gradients, and update adapters.

For fresh sparse distillation, a full-input sparse student forward replaces steps 3–5 at each saved loss probe. Each student collects its own trajectory. Teacher and student inputs match within a probe; trajectories, masked-position sets, and teacher distributions can vary across training methods.

B.3 Prompt-cache gradient control

At a cache consumer, the control transforms each KV tensor X as

$$X \longmapsto \mathrm{concat}\big(\mathrm{stopgrad}(X_{:P}), X_{P:}\big). \tag{10}$$

The intervention detaches the explicit incoming prompt KV slices while preserving their forward values, direct fresh-refresh losses, and gradients through generation-cache slices. Generation entries can depend on earlier prompt computation and retain those indirect gradient paths.

Before training, we check the gradient intervention using common pretrained weights and a common saved trajectory, without an optimizer update. Both variants produce loss 0.11530317. The monitored prompt-key gradient norm is 0.03127342 with the cache attached and exactly zero with it detached. Total adapter gradient norms are 0.45732787 and 0.27567557. A Jacobian test confirms that generation-cache slices remain attached. The fresh sparse distillation control is checked separately for finite nonzero gradients and the absence of incoming cache in its student loss calls.

Independent CPU recomputation of the first and last AdamW updates across all nine models gives a maximum parameter discrepancy of $1.862645149 \times 10^{-9}$, below the 2×10^{-7} tolerance. Validation also checks final checkpoint tensors, gradient consistency with and without checkpointing, finite gradients, and exact replay of reference logits.

B.4 Training and deployment differences

Training stores pre-RoPE BF16 KV and uses deterministic math SDPA. Its FP32 adapter branch output is cast to BF16 before addition to the base projection. Deployment forms $\mathrm{BF16}(\mathrm{FP32}(W) + W_{\mathrm{up}}W_{\mathrm{down}})$, with the product and sum computed in FP32, and uses post-RoPE KV and optimized kernels. The auxiliary control studies use BF16 KV; the 128K-cap comparison stores old KV in FP8, as specified in Appendix L. The folded and separate-branch implementations can differ numerically. The training sampler transfers tokens above a confidence threshold; fixed-budget evaluation transfers one token per forward.

Table 6 reports the measured training time for all nine runs. Each time is the sum of per-update wall-clock intervals, including trajectory collection, student and teacher forwards, backward computation, optimizer work, synchronization, the final validation check, and the first/last pre-update snapshot writes. Model loading, pre-run validation, post-update logging and snapshot writes, time between updates, and exploratory runs are excluded. Fresh sparse distillation was measured in a separate runtime batch from the two cached-training variants.

Table 6: Training-loop time for the three training variants and all three seeds. Each run uses 64 updates.

Seed	REPRISE (s)	Detached prompt cache (s)	Fresh sparse distillation (s)
202609151	1010.56	993.40	1150.00
202609152	998.11	1003.00	1148.86
202609153	987.20	998.29	1138.72
Total GPU-hours	0.832	0.832	0.955

C Baseline implementations and ablation protocols

Table 7: Baseline implementations and configurations.

Baseline	Repository, revision, and configuration
SparseD	https://github.com/INV-WZQ/SparseD Revision `52155b4aa78368695f96744cade06d2e0865d085` skip=.2, select=.3, block size=128, total steps=G
dLLM-Cache	https://github.com/maomaocun/dLLM-cache Revision `17235bffc8c5b587a2dc6f7dc76fcd01eab76e3a` prompt interval=100, generation interval=7, transfer ratio=.25

SparseD. The implementation recomputes Q/K/V projections and MLPs for every prompt and generation row at each forward pass and reuses the attention pattern after the dense phase. This phase covers steps 0 through $\lfloor .2G \rfloor + 1$, with pattern construction at the final step. Subsequent passes apply the per-head pattern through FlexAttention. Each selection retains $\max(1, \lfloor .3n_k \rfloor + 1)$ of n_k key blocks, with a separate selection over the suffix. Rounding and suffix selection determine the resulting graph density. We retain the authors' attention and selection code and adapt the sampler and vocabulary-head range.

dLLM-Cache. The original implementation refreshes all layer-zero rows at each forward pass. Deeper layers refresh prompt rows at $t \bmod 100 = 0$ and generation rows at $t \bmod 7 = 0$, with zero-based t. Between generation refreshes, the $G/4$ positions with lowest cosine similarity between current and cached values are selected from the entire generation canvas, including filled and future masked positions. All generation values are recomputed; queries, keys, attention outputs, and MLP outputs update only at selected generation rows. Remaining keys and attention/MLP outputs are reused. The .25 parameter sets the fraction of generation positions selected for these updates. Attention remains dense; the sampler and vocabulary-head range are adapted.

Shared evaluation settings. Untrained methods share pretrained weights, BF16 precision, inputs, output budgets, forward counts, token-transfer rules, and timing boundaries. Trained variants share the deployment policy. Methods are paired by input and physical GPU in the RULER 64K and eight-task LongBench ablations, with requests run serially.

Execution order. The RULER 64K ablation alternates REPRISE and detached prompt-cache models by ascending training seed, followed by untrained sparse and dense cached models. The three GPU slots rotate this fixed list by zero, two, and four entries. Fresh sparse distillation models are evaluated in a separate runtime batch on the same GPUs.

Runtime differences. All methods use a common sampler with their respective numerical backends. Kernels, sparsity, refresh policies, layer-zero treatment, and reused states differ across systems. External baselines use pretrained weights. The 32K evaluations span separate runtime batches, and the fresh sparse distillation runs use a different source-to-GPU assignment from the cached-training runs. Cross-system latency ratios reflect both the algorithms and their implementations.

Context lengths. RULER inputs are constructed with task-specific output reserves. The 64K ablation set has at most 65,453 prompt tokens and 65,709 total positions with the 256-position generation canvas. The additional seed-42 evaluation reaches 65,720 total positions. We retain the generated prompts without truncating them to a 65,536-position total.

C.1 Evaluation protocol

Ablation datasets. RULER training ablations use 32 inputs from each of 13 tasks, giving 416 inputs at each of 32K and 64K (Hsieh et al., 2024). The 64K ablation inputs use generation seed 2026091801. An additional 64K evaluation uses seed 42 and is reported separately in Appendix D. The two 64K sets have no identical prompt-token hashes and can share underlying QA corpora. Every RULER ablation request uses $G = \mathrm{NFE} = 256$. The LongBench ablation subset contains eight English tasks with 32 inputs each (Bai et al., 2024). It uses official prompts, output budgets, and scorers, with an equal-task mean over this subset. The longest prompt has 39,607 tokens under a 40,015-token selection cap. Requests execute the full generation budget; EOS truncation is applied only for scoring.

D Complete seed and task results

Table 8: **Training ablations on RULER.** Equal-task scores on 416 inputs per length under the same deployed sparse/cache policy. Trained entries report the mean ± sample SD across three seeds.

Training	RULER 32K	RULER 64K
Untrained sparse	72.27	73.49
Fresh sparse KD	77.05 ± 1.05	78.24 ± 0.54
Detached prompt cache	74.99 ± 2.98	76.67 ± 1.50
REPRISE	**80.56** ± 1.33	**79.62** ± 1.02

Table 9: **Per-task RULER 64K training ablations.** Scores average 32 inputs per task and three training seeds. Differences are REPRISE minus each control, computed before rounding; best mean scores are bold.

Task	REPRISE	Detached	Fresh sparse	Δ detached	Δ fresh
Common words	**0.00**	**0.00**	**0.00**	0.00	0.00
Frequent words	60.07	58.68	**60.42**	1.39	−0.35
Multiple keys 1	**100.00**	**100.00**	**100.00**	0.00	0.00
Multiple keys 2	**100.00**	**100.00**	**100.00**	0.00	0.00
Multiple keys 3	**100.00**	**100.00**	**100.00**	0.00	0.00
Multiple queries	**97.14**	92.71	95.57	4.43	1.56
Multiple values	**58.07**	50.52	56.77	7.55	1.30
Single needle 1	**100.00**	**100.00**	**100.00**	0.00	0.00
Single needle 2	**100.00**	**100.00**	**100.00**	0.00	0.00
Single needle 3	**100.00**	97.92	97.92	2.08	2.08
QA 1	**80.21**	66.67	71.88	13.54	8.33
QA 2	**44.79**	36.46	41.67	8.33	3.13
Variable tracking	**94.79**	93.75	92.92	1.04	1.88
Equal-task mean	**79.62**	76.67	78.24	2.95	1.38

Task-level effects vary in sign and magnitude (Table 9). Against fresh sparse distillation on the RULER 64K ablation set, the two QA tasks improve by 8.33 and 3.13 points, while frequent-word extraction changes by -0.35. Several retrieval tasks are at ceiling; all trained variants score zero on common-word extraction.

Table 10: Training-seed results on the RULER 64K ablation set, with 416 inputs per model. Entries are score / mean request seconds. Differences are computed before rounding.

Seed	REPRISE	Detached prompt cache	Fresh sparse distillation
202609151	80.2684 / 26.8815	76.2019 / 26.8052	78.8462 / 26.8301
202609152	80.1522 / 26.8280	78.3454 / 26.8400	78.0609 / 26.7554
202609153	78.4415 / 26.8124	75.4607 / 26.7787	77.8165 / 26.9055
Mean	79.6207 / 26.8406	76.6693 / 26.8079	78.2412 / 26.8303

D.1 RULER 64K TRAINING ABLATIONS

REPRISE outperforms both training controls for every seed. The untrained sparse model scores 73.4896 at 26.7319 seconds, and the dense cached model scores 71.3101 at 42.4492 seconds, each on the same 416 inputs.

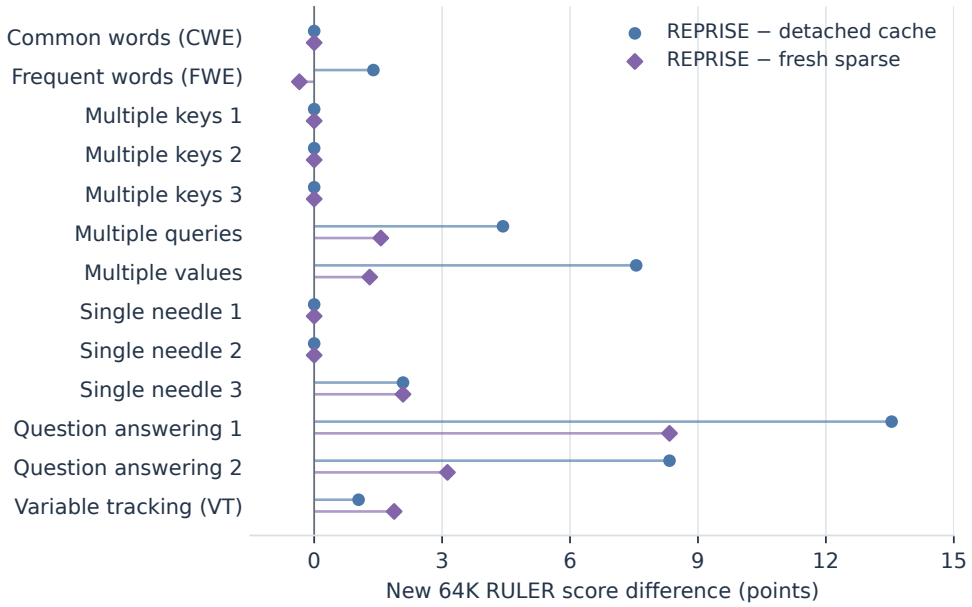


Figure 5: Task-level effects of REPRISE on the RULER 64K ablation set. Circles and diamonds show three-seed mean score differences from the two training controls. Each segment connects zero to the point estimate. Task names follow Table 9.

D.2 RULER 32K AND LONGBENCH RESULTS

Table 11: Seed results as score / mean request seconds. RULER 32K uses 416 inputs per model; the eight-task LongBench subset uses 256. Seed suffixes abbreviate 202609151–202609153. Fresh sparse distillation scores use the same LongBench inputs (Appendix O); timings from separate evaluation batches are omitted.

Set	Seed suffix	REPRISE	Detached prompt cache	Fresh sparse
RULER 32K	151	81.23 / 17.22	73.63 / 17.12	77.94 / 17.27
	152	81.42 / 17.10	78.42 / 17.22	77.32 / 17.12
	153	79.03 / 17.10	72.93 / 17.00	75.89 / 17.21
LongBench subset	151	40.20 / 5.96	39.81 / 5.91	38.57 / –
	152	40.56 / 5.93	39.59 / 5.95	38.60 / –
	153	40.17 / 5.95	40.40 / 5.92	41.07 / –

Table 12: RULER 32K task scores for the gradient comparison, averaged over 32 inputs per task and three training seeds. Differences are REPRISE minus detached cache, computed before rounding.

Task	REPRISE	Detached prompt cache	Difference
cwe	4.90	2.60	+2.29
fwe	63.54	68.06	−4.51
niah_multikey_1	100.00	100.00	0.00
niah_multikey_2	100.00	100.00	0.00
niah_multikey_3	100.00	100.00	0.00
niah_multiquery	99.48	95.57	+3.91
niah_multivalue	71.88	49.74	+22.14
niah_single_1	100.00	100.00	0.00
niah_single_2	100.00	100.00	0.00
niah_single_3	90.63	72.92	+17.71
qa_1	59.38	47.92	+11.46
qa_2	63.54	43.75	+19.79
vt	93.96	94.38	−0.42
Mean	80.56	74.99	+5.57

Table 13: Latency comparison on the eight-task LongBench subset. Scores average tasks equally across 256 shared inputs; trained variants average three seeds. Time / ours is the ratio of mean request latencies. The fresh sparse LongBench control appears in Appendix O.

Method	Score ↑	Seconds ↓	Time / ours
Fresh dense	39.49	111.79	18.80×
SparseD adaptation	39.68	107.69	18.11×
dLLM-Cache adaptation	39.33	15.66	2.63×
Dense attention, cached	39.23	6.15	1.03×
Sparse attention, untrained	39.48	5.98	1.01×
Detached prompt cache	39.93	5.93	1.00×
REPRISE (ours)	40.31	5.95	1.00×

D.3 LongBench task-level results

Table 14: **Task-level quality on the LongBench subset.** All eight tasks, 32 sources each, with official task metrics; best mean scores in bold. Baseline adaptations follow Table 18. Untrained uses our sparse/cache policy without adaptation; detached and REPRISE average all three seeds.

Task	SparseD	dLLM-Cache	Untrained	Detached	REPRISE
Qasper	**17.38**	15.29	10.58	13.41	14.89
MultifieldQA	19.97	25.12	24.40	**26.64**	25.58
HotpotQA	14.71	17.19	21.42	**22.17**	16.12
2WikiMQA	14.26	15.92	**20.23**	17.46	17.32
GovReport	**31.75**	29.23	22.28	25.03	25.05
TREC	71.88	68.75	**78.13**	70.83	66.67
Passage retrieval	**98.44**	95.83	91.67	90.40	96.09
RepoBench-P	49.06	47.31	47.16	53.51	**60.78**
Mean	39.68	39.33	39.48	39.93	**40.31**

On the eight-task LongBench subset, REPRISE exceeds detached prompt-cache training by 0.38 points, with a 95% paired bootstrap confidence interval of [-1.34, 2.06]. One seed has a negative difference. The intervals for comparisons with SparseD and dLLM-Cache also include zero, leaving the quality gains over these methods uncertain.

Task-level changes vary in sign and magnitude (Table 14). Compared with the untrained sparse model, REPRISE gains 13.63 points on RepoBench-P and loses 5.30 points on HotpotQA and 11.46 points on TREC. Prompt lengths and refresh counts for this subset are reported in Appendix I.

D.4 Additional RULER 64K evaluation

This additional evaluation uses RULER generation seed 42 and 32 inputs from each of the same 13 tasks. REPRISE scores 78.4108 at 27.1234 seconds, the detached prompt-cache model scores 76.1779 at 27.0774 seconds, the untrained sparse model scores 73.3494 at 27.0520 seconds, and the dense cached model scores 70.5208 at 42.8684 seconds. REPRISE exceeds detached prompt-cache training by 2.2329 points, with a 95% paired bootstrap confidence interval of [0.9842, 3.4522] and a negative difference for one seed. These inputs are separate from the seed-2026091801 ablation set.

E Statistics, timing, and implementation checks

The statistical procedures and request-level timing below apply to the RULER training ablations and the eight-task LongBench subset. Protocols for the main benchmark comparison are given in Appendix L and Appendix M. **RULER input construction.** Inputs use OpenCompass 0.4.2 generators with Wonderwords 2.2.0. The 32K ablation set and additional 64K evaluation use generation seed 42; the 64K ablation set uses seed 2026091801. Prompts are encoded without added special tokens or a chat wrapper. The generator reserves 128 tokens for needle tasks, 30 for variable tracking, 120 for common-word extraction, and 50 for frequent-word extraction and QA. The frequent-word generator receives the target length minus its 50-token reserve. Each prompt and its task-specific reserve must fit within the target length. Evaluation uses a fixed generation budget of $G = 256$.

Task scores. RULER uses case-insensitive substring matching. QA scores whether any reference answer occurs; other tasks score the fraction of reference strings found within the prediction. LongBench uses QA F1 for its four QA tasks, ROUGE-L for GovReport, and the official classification, retrieval, and code-similarity metrics for the remaining tasks, taking the maximum over reference answers. TREC strips leading newlines and retains the first prediction line before scoring. For both benchmarks, we average unrounded example scores within each task, scale task means to 0–100, and weight tasks equally.

E.1 Paired source bootstrap

For task k, source i, seed s, and score Y_{kis}, the training contrast is

$$d_{ki} = \frac{1}{3}\sum_{s=1}^{3}\left(Y_{kis}^{\text{ours}} - Y_{kis}^{\text{control}}\right), \qquad \widehat{\Delta} = \frac{1}{K}\sum_{k=1}^{K}\frac{1}{n_k}\sum_{i=1}^{n_k} d_{ki}. \tag{11}$$

Each bootstrap replicate samples n_k example IDs with replacement independently within each task and recomputes the equal-task mean. All methods and training seeds for a sampled example remain paired. The interval is the 2.5th–97.5th percentile of 10,000 means. The registered benchmark example is the resampling unit, with its three seeds grouped together. The bootstrap treats distinct example IDs as independent, including IDs that share underlying documents.

For the RULER 64K ablation, bootstrap sampling uses random seed 2026091801 for REPRISE minus detached prompt-cache training and 2026091802 for REPRISE minus fresh sparse distillation. Both contrasts and the fitted models were specified before scoring. The reported 95% intervals are unadjusted and conditional on the three fitted model pairs and benchmark corpora.

On the eight-task LongBench subset, REPRISE exceeds SparseD by 0.63 points, with a 95% paired bootstrap confidence interval of approximately $[-1.63, 2.92]$, and dLLM-Cache by 0.98 points, with an interval of $[-1.08, 3.08]$. Both intervals include zero. A bootstrap interval was not computed for the 32K fresh sparse distillation comparison.

E.2 Request-latency measurement

This protocol covers the BF16 ablation and additional-baseline evaluations and the 26-input repeated timing experiment. Full-request retrieval timings use the separate protocol in Appendix N and average the paired latency ratios for individual inputs. Latency is synchronized wall-clock time, including harness overhead, for one pretokenized request with model weights already on the GPU. The timer starts before prompt transfer and inference-wrapper construction and ends after copying output IDs and decoding the prefix before EOS. It includes prefill, KV allocation, selection/planning, refreshes, generation, per-step process-liveness and deadline checks, and any first-use kernel compilation. Prompt rendering/tokenization, model loading, pre-run validation, adapter folding, scoring, file writes, and post-request cleanup are excluded.

Each request starts with empty KV and sparse-pattern state. A synthetic warm-up with 512 prompt tokens and 32 generation positions precedes evaluation; compiled kernels may persist afterward. All measured dataset requests are included in the configuration mean. Speedup is the ratio of mean request latencies. Requests run serially on one GPU.

For the repeated timing experiment in Appendix A.5, configurations run in the order dense $H = 32$, dense $H = 64$, sparse $H = 64$, then the three trained REPRISE models at $H = 128$, cyclically shifted by GPU slot. The $H = 64$ pair compares attention policies at the same refresh interval. The REPRISE configurations change both trained weights and the refresh interval.

E.3 Evaluation and implementation validation

Official scores are independently recomputed from saved predictions. Validation checks input coverage and identities, prompt hashes, generation traces, numerical settings, and checkpoint identities. Optimizer and replay checks are described in Appendix B.

F Proof and interpretation of the local bound

Proposition 1. *Fix a query $q \in \mathbb{R}^d$, finite fresh keys k_j, values satisfying $\|v_j\|_2 \le R$, and a nonempty retained set S. Let a_D be fresh dense attention, and a_{ST} attention restricted to S with stale keys and values. Suppose the fresh dense probability mass outside S is ϵ, and retained key and value perturbations are bounded by δ_K and δ_V in Euclidean norm. Then*

$$\|a_D - a_{ST}\|_2 \le 2R\epsilon + \frac{2R\|q\|_2\delta_K}{\sqrt{d}} + \delta_V. \tag{12}$$

The bound separates local attention error into omitted-mass and state-perturbation terms for a fixed query and retained set in exact arithmetic.

F.1 Proof of Proposition 1

Let $p = \mathrm{softmax}(qK^\top/\sqrt{d})$ and let p_S be its conditional distribution on S. Finite logits and a nonempty retained set imply $1 - \epsilon > 0$. If $\epsilon > 0$, write

$$a_D = (1-\epsilon)a_S + \epsilon a_{\mathrm{out}}. \tag{13}$$

Both a_S and a_{out} are convex combinations of fresh values with norms at most R. Thus $\|a_D - a_S\|_2 \le 2R\epsilon$. The same conclusion holds when $\epsilon = 0$.

For finite logit vectors z, z' and $h = z' - z$, define $r(t) = \mathrm{softmax}(z + th)$. The softmax derivative gives

$$\sum_j |\dot r_j(t)| = \sum_j r_j(t)\left|h_j - \sum_l r_l(t)h_l\right| \le 2\|h\|_\infty. \tag{14}$$

Integrating over $t \in [0,1]$ yields

$$\|\mathrm{softmax}(z) - \mathrm{softmax}(z')\|_1 \le 2\|z - z'\|_\infty. \tag{15}$$

A retained key perturbation of norm at most δ_K changes its logit by at most $\alpha = \|q\|_2 \delta_K / \sqrt{d}$. Let p'_S be the conditional attention weights formed from the stale retained keys. Then

$$a_S - a_{ST} = \sum_{j \in S} (p_{S,j} - p'_{S,j}) v_j + \sum_{j \in S} p'_{S,j} (v_j - v'_j), \tag{16}$$

$$\|a_S - a_{ST}\|_2 \leq 2R\alpha + \delta_V. \tag{17}$$

The triangle inequality with the truncation bound proves Eq. (12).

At a sparse prompt refresh, $\delta_K = \delta_V = 0$ relative to dense attention on the same layer inputs. At a subsequent dense generation query, $\epsilon = 0$.

F.2 Selection and token-decision margins

If every candidate block score changes by at most e, a gap greater than $2e$ between the weakest selected and strongest unselected optional block preserves the selected set, provided the budget and mandatory set are fixed. The original score difference can decrease by at most $2e$. Without a positive margin, arbitrarily small score perturbations can change a discrete top-k set.

Similarly, if every vocabulary logit changes by at most η, an argmax margin greater than 2η preserves the predicted token at a fixed position. Preserving the transferred position also requires a confidence margin across eligible positions. For one softmax coordinate, the ℓ_1 norm of its gradient is $2p_j(1 - p_j) \leq 1/2$, so its probability changes by at most $\eta/2$. The maximum token probability obeys the same bound. With a fixed eligible-position set and the logit bound at every position, a confidence gap greater than η preserves the winning position. A threshold decision requires distance greater than $\eta/2$ from the threshold. These sufficient conditions assume exact arithmetic.

F.3 Interaction between sparsity and staleness

Consider two fresh values $(1, -1)$ with equal attention weights. The dense output is zero. Retaining only the first value gives a sparse output of one. Add a common stale value shift t to both values. The stale dense output is t and the sparse stale output is $1 + t$. Relative to the fresh dense output, the squared-error interaction is

$$(1 + t)^2 - 1 - t^2 = 2t. \tag{18}$$

It is positive for $t > 0$ and negative for $t < 0$, so the interaction depends on the stale value shift.

F.4 Refresh exposure

Two deterministic executions with identical initial state, weights, graph, arithmetic, transfer rule, and cache updates remain identical until their first differing refresh operation. This follows by induction over the shared operations. A metric determined entirely before that boundary cannot identify a later-horizon cache effect.

G Average gains over dense cached attention

This post-hoc summary compares REPRISE with the untrained dense cached baseline under the same refresh schedule on RULER 32K, the RULER 64K ablation set, and the eight-task LongBench subset (Table 18). Let q_s, q_s^D denote their equal-task scores and t_s, t_s^D their mean request latencies in setting s. We report

$$\overline{\Delta q} = \frac{1}{3} \sum_{s=1}^{3} (q_s - q_s^D), \qquad \overline{r}_{\text{geo}} = \exp\left(\frac{1}{3} \sum_{s=1}^{3} \log \frac{t_s^D}{t_s} \right). \tag{19}$$

Each setting receives equal weight, regardless of its input count. Scores are on a 0–100 scale and use the task metrics of each benchmark. The comparison includes changes in attention execution and trained weights.

REPRISE improves scores by 5.57 points on average, with a geometric mean speedup of 1.26×.

Table 15: Scores and mean request latencies used to calculate the average gains over dense cached attention. Calculations use unrounded values.

	Score		Mean seconds	
Setting	REPRISE	Dense cached	REPRISE	Dense cached
RULER 32K	80.56090	73.24519	17.14135	20.87269
RULER 64K	79.62073	71.31010	26.84062	42.44923
LongBench subset	40.31277	39.22646	5.94540	6.14709

H Supplementary and incomplete evaluations

A separate 256-input LongBench subset gave REPRISE and detached prompt-cache scores of 41.72 and 40.22, a difference of 1.50 points with a 95% paired bootstrap confidence interval of $[-0.17, 3.12]$ and a negative difference for one seed. These results are reported separately from Table 13. A two-seed, 32-input pilot comparing mixed-age and zero-age training gave stale-NLL differences of +0.005580 and -0.004326 nats. Their mean is +0.000627, with an interval of $[-0.006735, 0.008954]$, leaving the effect uncertain.

FP8 KV pilot evaluations use different protocols and are excluded from the BF16 ablation tables. The 128K-cap LongBench results in Table 1 follow Appendix L. The adopted memory-release implementation preserves MLP row shapes and is checked against the preceding inference implementation. Synthetic cases use $P \in \{512, 4096, 32768\}$ and $G = 160$. They require logit agreement within absolute and relative tolerance 0.005, identical token argmax and transfer positions on every forward, matching cache-update events, and KV agreement at four selected forwards. A separate $(P, G) = (65536, 256)$ case checks memory capacity and refresh counts. The dense cached variant uses the same checks with all prompt blocks retained.

Fresh dense RULER 32K completed only 34/416 requests before its deadline; no partial mean is reported. Fresh sparse LongBench results appear in Appendix O.

I Eight-task LongBench subset and workload

We use `THUDM/LongBench` at the following revisions. Code supplies templates, output budgets, and metrics; data supplies the source archive.

Code (GitHub)	`2e00731f8d0bff23dc4325161044d0ed8af94c1e`
Data (Hugging Face)	`5e628be450b7e67fb7ae6e201bd6d8f7056f7672`

The subset uses the eight tasks' official LongBench-E `_e.jsonl` source members. Before selection, we excluded 485 task–example IDs from existing reference, training, and development sets. Underlying text can overlap across distinct IDs. Eight additional candidates exceeded the 40,015-token prompt cap. Prompts use the official templates and model tokenizer with `add_special_tokens=True` and no truncation.

Strata are defined by the raw dataset's `length` field, with boundaries at 4,000 and 8,000. Model-token counts are recorded separately as prompt lengths. Each task initially targets 11, 11, and 10 examples across the three bins, capped by availability after exclusions. A deficit is filled one example at a time from the eligible bin with the smallest current quota, breaking ties in ascending bin order. Candidates within a bin are ordered by the hexadecimal SHA256 of the UTF-8 string formed by joining `overnight-independent-20260916`, `longbench`, the task name, and the raw example ID with colons. We take the first candidates up to each quota.

The resulting raw-bin counts are 95, 95, and 66. Across all 256 prompts, the median native length is 7,505 and 39 use the dense fallback. Exact subset reconstruction requires both the selection rule and the archived exclusion IDs. The registry and exclusion manifest remain in the internal experimental archive pending public release.

Seven tasks, covering 224 examples, have $G \leq 128$ and no second full prompt refresh. Only GovReport, with $G = 512$, reaches four full refreshes. A one-refresh request reuses the initial prompt

Table 16: Eight-task LongBench subset, 32 examples per task. Bins list counts for raw lengths $< 4{,}000$, $[4{,}000, 8{,}000)$, and $\geq 8{,}000$. P lists minimum / median / maximum model prompt tokens. $G = \mathrm{NFE}$ is the fixed official task budget, $F = \lceil G/128 \rceil$ counts full prompt refreshes including initialization, and D counts prompts using the dense fallback because $P < 4096$.

Task	Bins	P min / median / max	G	F	D
qasper	13 / 13 / 6	3511 / 5973 / 21921	128	1	3
multifieldqa_en	16 / 16 / 0	1285 / 6102.5 / 10859	64	1	7
hotpotqa	11 / 11 / 10	838 / 10228.5 / 17228	32	1	6
2wikimqa	11 / 11 / 10	2446 / 7299.5 / 16571	32	1	3
gov_report	11 / 11 / 10	2205 / 7146.5 / 18263	512	4	9
trec	11 / 11 / 10	1132 / 8250.5 / 18146	64	1	6
passage_retrieval_en	11 / 11 / 10	3615 / 7755.5 / 13683	32	1	3
repobench-p	11 / 11 / 10	3708 / 16926 / 39607	64	1	2

KV on subsequent forwards and can expose errors in that representation. Schedule comparisons require requests long enough to reach a differing refresh boundary.

J FRESH KV STABILITY DIAGNOSTIC

We measure KV stability using the unadapted UltraLLaDA model in BF16 on one A800 GPU. The first two training articles each provide 4,096 prompt tokens and 128 initially masked generation positions. Each trajectory contains 128 forward passes, each filling the highest-confidence masked position in the current 32-position block.

Each forward recomputes the full sequence through all 32 layers with dense attention and freshly computed KV. We capture native post-RoPE K and V at layers 8, 16, 24, and 32, sampling 128 evenly spaced prompt positions and all 128 generation positions. The heads are concatenated at each position. For $Z \in \{K, V\}$, the relative change at position i is

$$d_t^Z(i) = \frac{\|Z_t(i) - Z_{t-1}(i)\|_2}{\|Z_{t-1}(i)\|_2}. \tag{20}$$

All denominators are nonzero. For each region, we compute the ratios in FP64 and report their mean over sampled positions. Each plotted point averages four sampled layers, 16 adjacent transitions, and two inputs with equal weights, with K and V reported separately. Table 17 reports each source over its full trajectory.

Table 17: Mean adjacent-state KV change (%) in the fresh dense diagnostic. Unchanged generation excludes the one token directly filled in each transition.

Source	Region	K change	V change
TRAIN 000	Prompt	3.798	6.401
TRAIN 000	Generation	8.527	16.978
TRAIN 000	Unchanged generation	8.215	16.366
TRAIN 001	Prompt	3.658	6.126
TRAIN 001	Generation	8.786	17.315
TRAIN 001	Unchanged generation	8.485	16.730

At each input's initial state, measured logits and sampled KV are bitwise identical to the fresh dense reference.

The two input trajectories are the experimental units, with dependent token and transition observations within each trajectory. The measurements compare prompt and generation state stability on these sampled training inputs.

K Comparison with additional baselines

This comparison uses the same 416-input RULER ablation sets and eight-task, 256-input LongBench subset described in Appendix C. Dense cached uses dense attention with the REPRISE refresh schedule. External baselines are implemented within the evaluation harness using the versions and adaptations specified there. The main 104-input comparisons are reported separately in Appendix L and Appendix M.

Table 18: **Additional baseline comparisons in BF16.** Scores (↑) and mean seconds (↓) on 416 inputs per RULER setting and the separate 256-input LongBench subset. REPRISE reports mean ± sample standard deviation across three training seeds; its times are seed means. Dashes denote unavailable complete results. External baselines are adaptations. Bold marks the best mean score in each setting.

	RULER 32K		RULER 64K		LongBench 256	
Method	Score	Seconds	Score	Seconds	Score	Seconds
Fresh dense	–	–	–	–	39.49	111.79
SparseD adaptation	**83.26**	1057.43	–	–	39.68	107.69
dLLM-Cache adaptation	71.59	122.27	–	–	39.33	15.66
Dense cached	73.25	20.87	71.31	42.45	39.23	6.15
Untrained sparse	72.27	13.61	73.49	26.73	39.48	5.98
REPRISE (ours)	80.56 ± 1.33	17.14	**79.62** ± 1.02	26.84	**40.31** ± 0.22	5.95

Compared with dense cached attention, REPRISE improves quality in all three settings and achieves latency ratios of 1.22×, 1.58×, and 1.03× (Table 18). On the RULER 64K ablation set, REPRISE scores 79.62 at 26.84 seconds, compared with 71.31 at 42.45 seconds for dense cached attention. Training raises the untrained sparse score by 6.13 points at similar latency. SparseD achieves a higher RULER 32K score than REPRISE, 83.26 versus 80.56.

On the eight-task LongBench subset, REPRISE scores 40.31 and is 18.80× faster than fresh dense attention and 1.03× faster than dense cached attention. Its quality differences from SparseD and dLLM-Cache have confidence intervals that include zero. Complete task-level results and training comparisons appear in Appendix D.

The untrained and trained RULER 32K timings were measured in separate runtime batches. Repeated timing within a common experiment is reported in Appendix A.5.

L Main LongBench comparison

Inputs and score aggregation. LongBench V1 and V2 each use 104 fixed inputs at caps of 32,768, 65,536, and 131,072 tokens. V1 includes five inputs from each of 21 tasks except RepoBench-P, which has four, and reports the equal-task mean. V2 reports accuracy over its 104 questions. All methods share the same inputs within each version and cap. REPRISE uses three trained checkpoints; Prefix, Dual, and NoCache use pretrained weights.

Input lengths and truncation. Inputs exceeding a cap retain equal-length token prefixes and suffixes, without padding. The same source examples are used across caps. Table 19 reports model-token lengths. The longest V1 input has 72,570 tokens. Comparisons across caps change truncation and, at 128K, KV precision and FFN chunking.

Execution. V1 retains task budgets $G = \mathrm{NFE} \in \{32, 64, 128, 512\}$; V2 uses $G = \mathrm{NFE} = 128$ with direct answers and no chain of thought. All methods use one generation canvas, 32-position blocks, and deterministic one-token transfer. They compute the full budget without early EOS termination; scoring uses the text before the first EOS. Official prompts, scorers, and the V2 answer extractor are unchanged.

At 32K/64K, cached methods use BF16 KV and compute, with an FFN row chunk of $P + G$. At the 128K cap, they store old KV in FP8 e4m3fn, compute in BF16, and use 4,096-row FFN chunks.

Table 19: Prompt lengths for the fixed LongBench sources, with 104 inputs per row. At cap counts inputs whose processed length equals the cap.

Version / cap	Minimum	Median	Maximum	At cap
V1 / 32K	1,572	9,042.5	32,768	3
V1 / 64K	1,572	9,042.5	65,536	1
V1 / 128K	1,572	9,042.5	72,570	0
V2 / 32K	11,425	32,768	32,768	82
V2 / 64K	11,425	65,536	65,536	68
V2 / 128K	11,425	106,209	131,072	42

Fresh current rows bypass quantization. NoCache has no persistent KV and always computes in BF16; it also uses 4,096-row chunks at 128K. Our evaluation engine implements Prefix/Dual policies with dense PyTorch SDPA and REPRISE sparse full refreshes with FlashInfer BSR. Prefix/Dual quality runs pair sources and physical GPUs; NoCache and REPRISE use different GPU assignments and execution batches.

Scores. Table 20 reports all three seeds. REPRISE has a higher mean than Prefix and Dual in all six settings. Individual V1 seeds show both gains and regressions. The REPRISE means are 1.17, 1.14, and 0.41 points below NoCache at the three V1 caps. Gap recovery is $(q_{\text{ours}} - q_{\text{cache}})/(q_{\text{NoCache}} - q_{\text{cache}})$, giving 83.04% for the Prefix-to-NoCache gap at the V1 128K cap.

Table 20: REPRISE scores for all three checkpoint seeds on the same 104 inputs per setting. Suffixes 151–153 denote seeds 202609151–202609153. SD is computed as the sample standard deviation of the three seed scores.

Version / cap	Seed 151	Seed 152	Seed 153	Mean $\pm$ SD
V1 / 32K	38.4765	39.1640	37.3130	38.318 ± 0.936
V1 / 64K	38.4765	39.3105	37.3414	38.376 ± 0.988
V1 / 128K	39.5065	39.2930	37.8929	38.897 ± 0.877
V2 / 32K	20.1923	18.2692	19.2308	19.231 ± 0.962
V2 / 64K	16.3462	19.2308	15.3846	16.987 ± 2.002
V2 / 128K	9.6154	14.4231	9.6154	11.218 ± 2.776

L.1 Sampling uncertainty on the primary V1 subset

The standard deviations in Table 1 summarize variation across three training seeds. To estimate test-set sampling uncertainty, we average the three REPRISE scores for each source and pair this value with each baseline. We then resample sources within each of the 21 tasks over 10,000 bootstrap draws and weight task means equally. The resulting intervals describe source variation conditional on the selected tasks and three fitted checkpoints.

All nine paired intervals in Table 21 include zero. The score differences and 83% gap-recovery value describe the selected subset, with quality differences between methods remaining uncertain.

L.2 V2 output validity and paired correctness

Table 22 reports valid extracted choices and whitespace-only outputs. The official extractor scores the original predictions, counting whitespace-only and other unparseable responses as incorrect.

Generated failure modes. Among 1,047 whitespace-only predictions, 1,045 contain 128 repetitions of token 198, which decodes to newlines. The audit covers all 1,872 V2 predictions from three baselines and three REPRISE checkpoints at three input caps. Each setting uses the same 104 source IDs, and every trace contains 128 token commits with no earlier EOS. No output is a bare choice letter.

Table 23 separates changes in parseability from correctness on questions for which both models produce parseable answers. At 32K, REPRISE and NoCache each answer five questions correctly within this shared subset for every seed. REPRISE's additional 16, 14, and 15 correct answers occur

Table 21: LongBench V1 source-paired score differences. Pointwise percentile intervals use 10,000 bootstrap resamples within each task and an equal-task mean, conditional on the selected subset and three fitted REPRISE checkpoints.

Input cap	Baseline	REPRISE − baseline	95% CI
32K	NoCache	-1.17	[-3.96, 1.60]
32K	Prefix	+0.57	[-1.57, 2.87]
32K	Dual	+0.76	[-1.34, 3.06]
64K	NoCache	-1.14	[-3.89, 1.63]
64K	Prefix	+0.70	[-1.40, 3.04]
64K	Dual	+0.87	[-1.19, 3.19]
128K	NoCache	-0.41	[-3.39, 2.66]
128K	Prefix	+2.01	[-0.43, 4.78]
128K	Dual	+1.61	[-0.83, 4.39]

Table 22: V2 valid extracted choices / whitespace-only outputs, out of 104. The last three columns are REPRISE seeds. Other unparseable outputs make up the remaining cases.

Cap	Prefix	Dual	NoCache	151	152	153
32K	14 / 73	14 / 76	11 / 70	55 / 35	62 / 33	54 / 39
64K	7 / 74	9 / 75	8 / 68	51 / 39	60 / 36	42 / 48
128K	3 / 70	7 / 68	7 / 71	36 / 57	43 / 50	27 / 65

on questions with unparseable NoCache outputs. At 64K and 128K, REPRISE has fewer correct answers within the shared subset. Comparisons with Prefix and Dual show the same broad pattern: nearly all net gains occur on questions with unparseable baseline outputs.

The shared parseable subset contains only 2–13 questions and depends on both models' outputs. Its conditional accuracy provides a descriptive comparison with limited evidence about changes in reasoning ability or the causal role of formatting. Baseline failures also occur without caching and under BF16 at 32K and 64K, so stale KV states and FP8 storage cannot fully explain them.

The eight-task LongBench ablation uses a separate 256-input subset described in Appendix I.

Table 23: LongBench V2 paired output analysis. Each cell lists training seeds 151/152/153, in order. $n_\cap$ counts the questions on which both the baseline and REPRISE yield an official parseable choice. C_B and C_R count their correct answers within that intersection. C_{new} counts correct REPRISE answers outside the baseline-valid set; C_{lost} counts correct baseline answers outside the REPRISE-valid set. The full-set net correct gain is $C_R - C_B + C_{\text{new}} - C_{\text{lost}}$. All 104 questions remain in official scoring.

Cap	Baseline	$n_\cap$	C_B	C_R	C_{new}	C_{lost}
32K	NoCache	10/11/10	5/5/5	5/5/5	16/14/15	0/0/0
32K	Prefix	13/13/13	5/5/5	6/5/5	15/14/15	0/0/0
32K	Dual	13/13/13	6/6/6	8/6/6	13/13/14	0/0/0
64K	NoCache	6/8/7	4/4/4	2/3/3	15/17/13	0/0/0
64K	Prefix	6/6/6	5/5/5	3/4/4	14/16/12	0/0/0
64K	Dual	8/8/8	4/4/4	3/3/3	14/17/13	0/0/0
128K	NoCache	5/6/6	4/4/4	1/2/2	9/13/8	1/1/1
128K	Prefix	2/2/2	2/2/2	1/1/1	9/14/9	0/0/0
128K	Dual	6/6/6	4/4/4	2/3/3	8/12/7	0/0/0

M MAIN RULER COMPARISON

Inputs and score aggregation. We use 104 inputs at each target length, with eight from each of 13 RULER tasks. Prefix, Dual, and all three REPRISE checkpoints share the same inputs, generated with seed 42 without truncation or padding. Scores average example scores within each task, then average the 13 tasks equally. The training ablations use 416 inputs per length, with a different 64K input set (Appendix C).

Every request uses one 256-position generation canvas, 32-position blocks, and 256 forwards. Each forward fills the highest-confidence masked position in the active block. All forwards are computed, including those after EOS; scoring uses text before the first EOS. Prefix and Dual refresh at block boundaries. Within a block, Prefix updates the active and future suffix, while Dual updates the active block. REPRISE uses its fixed sparse policy and refresh schedule. All three trained checkpoints are retained without length-specific selection.

KV precision and numerical backends follow the settings in Appendix L. Prefix and Dual use pretrained weights, and REPRISE uses the same merged checkpoints as the LongBench comparison.

Evaluation. Prefix and Dual are paired by input and physical GPU, with method order alternating by input index. At 32K/64K, scores are recomputed on the designated 104 inputs from the 416-input evaluations. Quality-run times include diagnostic checks and are excluded from latency comparisons.

Table 24: Matched RULER scores, retaining all three checkpoints. SD is the sample standard deviation of the three trained-seed scores on the fixed input set.

Length	Prefix	Dual	Seed 151	Seed 152	Seed 153	Mean $\pm$ SD
32K	69.7917	73.1731	79.9840	79.0865	77.1154	78.729 ± 1.467
64K	68.1571	73.4455	80.6090	79.6474	80.7532	80.337 ± 0.601
128K	49.5513	48.3654	67.4679	68.0929	67.5321	67.698 ± 0.344

M.1 RULER NOCACHE REFERENCE

The NoCache entries use the pretrained UltraLLaDA native SGLang decoder. The 32K and 64K references were collected on an A100; the 128K reference was collected on an A800. It uses a confidence threshold of 0.9, 128-position canvases and scheduler blocks, 32-position cache blocks, and a 256-token request cap, with EOS stopping enabled. All retained records reach the length cap with 256 output tokens. Prefix, Dual, and REPRISE use a single 256-position canvas with one token transferred per forward.

At 32K and 64K, NoCache is scored on the same 104 inputs as the primary comparison, giving 83.11 and 82.47, respectively (Table 1).

Table 25: Per-task RULER scores on eight shared inputs per task and length. REPRISE averages all three checkpoints.

	32K			64K			128K		
Task	Prefix	Dual	Ours	Prefix	Dual	Ours	Prefix	Dual	Ours
cwe	8.75	7.50	5.83	12.50	12.50	8.33	0.00	0.00	0.00
fwe	54.17	62.50	51.39	41.67	41.67	66.67	16.67	25.00	19.44
niah_multikey_1	100.00	100.00	100.00	100.00	100.00	100.00	75.00	75.00	95.83
niah_multikey_2	100.00	100.00	100.00	100.00	100.00	100.00	87.50	75.00	100.00
niah_multikey_3	100.00	100.00	100.00	100.00	100.00	100.00	75.00	62.50	83.33
niah_multiquery	50.00	59.38	100.00	43.75	68.75	100.00	65.62	68.75	93.75
niah_multivalue	34.38	59.38	77.08	28.12	34.38	71.88	21.88	25.00	38.54
niah_single_1	100.00	100.00	100.00	100.00	100.00	100.00	75.00	75.00	100.00
niah_single_2	100.00	100.00	100.00	100.00	100.00	100.00	87.50	87.50	100.00
niah_single_3	75.00	62.50	70.83	75.00	100.00	100.00	50.00	50.00	75.00
qa_1	75.00	87.50	75.00	62.50	75.00	62.50	37.50	37.50	62.50
qa_2	25.00	25.00	54.17	25.00	25.00	37.50	12.50	12.50	45.83
vt	85.00	87.50	89.17	97.50	97.50	97.50	40.00	35.00	65.83

The 128K NoCache score is 71.39 on a separate set of 208 inputs, with 16 per task. This set overlaps 50 of the primary 104 inputs. NoCache uses a different decoding protocol and, at 128K, a different input set, so its RULER scores do not support isolated cache-error or gap-recovery estimates. NoCache latency is measured separately under the common phase protocol in Appendix N.

N Latency measurement protocols

N.1 Prefill and decode latency

Table 26: **Prefill and decode latency.** Eight representative inputs, one measurement per method and input. Cells show prefill seconds / TPOT seconds per committed token over 128 fixed decode steps. Bold marks the lowest time. These probes do not complete the 256-position generation canvas.

Dataset	Input tokens	NoCache	Prefix	Dual	REPRISE
RULER	29,434	4.58 / 4.542	4.55 / 0.189	4.56 / 0.170	**2.97 / 0.052**
	61,088	14.43 / 14.514	14.49 / 0.523	14.55 / 0.487	**7.48 / 0.096**
LongBench V1	24,756	3.76 / 3.612	3.61 / 0.161	3.61 / 0.142	**2.47 / 0.050**
	37,625	6.62 / 6.641	6.67 / 0.260	6.63 / 0.236	**4.02 / 0.061**
	72,570	19.42 / 19.490	20.06 / 1.009	20.28 / 0.976	**10.16 / 0.432**
LongBench V2	24,345	3.56 / 3.497	3.48 / 0.152	3.48 / 0.137	**2.40 / 0.048**
	48,786	10.02 / 10.047	10.06 / 0.373	10.02 / 0.345	**5.54 / 0.077**
	99,153	33.61 / 33.561	34.34 / 1.573	34.34 / 1.532	**15.56 / 0.596**

We select nine inputs, one from each dataset and actual-length bin, by proximity to 24K, 48K, or 96K tokens, with input ID breaking ties. LongBench candidates come from the 128K-cap subset, with preference given to untruncated inputs. All selected inputs are untruncated and unpadded. Eight inputs complete all four methods and enter the comparison, with one latency measurement per method and input. For the remaining 119,334-token RULER input, NoCache and Prefix complete, Dual reaches the preset time limit, and REPRISE is not run.

The methods run in the fixed order NoCache, Prefix, Dual, and REPRISE on one physical A800 per input. REPRISE uses the merged seed-202609151 checkpoint. Computation is BF16 and feed-forward chunks have 4,096 rows. Cached KV is BF16 when the actual input has at most 65,536 tokens and FP8 above that. NoCache retains no KV between forwards. Model loading, adapter folding, and an independent full-forward warmup are excluded. The input is on the GPU before timing.

All methods use a 256-position generation canvas and 32-position blocks. Prefill latency covers canvas allocation, the initial forward at step 0, and its first token commitment. Decode latency covers steps 1–128, each committing one token, including REPRISE's full refresh at step 128. TPOT is decode latency divided by 128. This measurement fills 129 of the 256 positions with early EOS stopping disabled. Timing includes sampling and excludes final CPU trace copies and checks. GPU synchronization occurs at phase boundaries, without per-step synchronization or progress logging.

N.2 Full-request retrieval latency

We measure full-request latency on retrieval inputs with $G = \mathrm{NFE} = 256$, 32-position blocks, and 4,096-row FFN chunks. REPRISE uses seed 202609151. Each speedup averages per-input latency ratios measured on the same physical GPU, using the REPRISE measurement paired with the corresponding baseline. Timing includes the synchronized request, data transfers, and sampling; it excludes model loading, adapter folding, and one full-forward warmup. Table 27 reports NoCache comparisons, and Table 28 reports Prefix and Dual comparisons.

Table 27: Completed NoCache timing pairs. Times average request seconds within each method. Speedups average the latency ratios computed separately for each matched request pair.

Setting	n	NoCache (s)	Paired REPRISE (s)	Speedup
32K	3	1,369.415	14.304	95.745×
64K	2	4,112.541	25.902	158.772×
128K	2	14,068.019	205.089	68.595×

At 64K and 128K, the first NoCache input reaches the 45-minute limit and is excluded from the completed-request means. Each is attempted once. The reported means at these lengths use the other two completed inputs. These inputs are used only for latency measurement.

Table 28: **Full-request retrieval latency.** Mean request seconds and mean paired speedups over three retrieval inputs per length, with 256 forwards and 4,096-row FFN chunks. REPRISE uses seed 202609151. Each pair shares its input and physical GPU.

Input	Prefix (s)	Dual (s)	REPRISE (s)	Prefix / ours	Dual / ours
32K	52.71	49.84	14.43	3.65×	3.45×
64K	143.38	138.74	26.00	5.51×	5.34×
128K	609.97	602.58	204.78	2.98×	2.94×

O Fresh sparse distillation on natural tasks

We evaluate the three final fresh sparse checkpoints on the same 256 LongBench inputs as REPRISE and the detached-cache controls. Input tokens, source IDs, output budgets, GPU assignments, BF16 arithmetic, sparse selection, refresh schedules, and one-token transfer are matched. All 768 evaluations complete using the existing checkpoints without additional training or selection. The runs were collected in separate batches, so their latency differences cannot identify a training effect.

We score predictions with the official task metrics and average the three seed-specific paired differences for each source. Bootstrap intervals use 10,000 resamples of the 32 sources within each task, weighting the eight tasks equally. The three contrasts use RNG seeds 2026092501–2026092503 in table order, with the detached-cache interval recomputed from the original predictions. Intervals condition on the fitted checkpoints and are unadjusted for multiple contrasts. Independent CPU reconstruction of token traces verifies predictions and source-to-GPU assignments.

Table 29: Matched LongBench training objectives. Each seed is evaluated on all 256 inputs.

Seed	Fresh sparse KD	Detached cache	REPRISE
151	38.5674	39.8052	40.2016
152	38.5959	39.5881	40.5626
153	41.0727	40.4046	40.1741
Mean	39.4120	39.9326	40.3128

Table 30: Paired training contrasts on the same natural-task subset, including all three training seeds.

Contrast	Difference (points)	95% paired interval
REPRISE − fresh sparse	+0.9008	[-0.3604, 2.1836]
REPRISE − detached	+0.3801	[-1.3408, 2.0632]
Fresh sparse − detached	-0.5207	[-2.1679, 1.0538]

Table 31: All eight tasks, with 32 inputs each. Entries average the three training seeds.

Task	Fresh sparse KD	Detached	REPRISE	Full − fresh
qasper	14.09	13.41	14.89	+0.80
multifieldqa_en	28.97	26.64	25.58	-3.38
hotpotqa	17.97	22.17	16.12	-1.86
2wikimqa	14.77	17.46	17.32	+2.56
gov_report	24.45	25.03	25.05	+0.60
trec	68.75	70.83	66.67	-2.08
passage_retrieval_en	91.41	90.40	96.09	+4.69
repobench-p	54.90	53.51	60.78	+5.89

P Complete generation with confidence-based parallel decoding

Inputs and checkpoint. We select two inputs per RULER 32K task and three per task for eight LongBench tasks from the 104-input pools in Table 1. Selection uses SHA256 ordering of source IDs with a fixed prefix, independently of scores. Eligible prompts are untruncated and contain at most 32,768 tokens. Selected RULER prompts range from 29,434 to 32,691 tokens (median 32,044.5), and LongBench prompts from 3,269 to 23,245 (median 8,890). Output caps are 256 for RULER and the official task budgets of 32, 64, 128, or 512 for LongBench. The first selected RULER input per task also receives a NoCache run, giving 13 four-method comparisons. REPRISE uses the checkpoint trained with seed 202609151. All 163 requests complete.

Shared transfer and stopping. Each forward fills all eligible positions whose prediction confidence reaches 0.9, or the highest-confidence position if none qualify. Every method uses 32-position blocks and advances after the active block is filled. Prefix and Dual refresh the full sequence at each block boundary. REPRISE refreshes prompt KV every four blocks, generation KV at other boundaries, and active-block rows between boundaries. NoCache recomputes dense states on every forward. Generation ends at the output cap or at EOS once all preceding positions are filled. Prompts shorter than 4,096 tokens use the dense fallback.

Timing. We measure each method and input once using BF16 weights, computation, and KV in the common executor, with BSR sparse attention, SDPA dense attention, and 4,096-row FFN chunks. Paired inputs run on the same physical A800. Cached methods rotate order by input index; the 13 dense controls follow the cached runs. Each method is warmed with full and active forwards at the measured shapes. Timing covers canvas allocation, forward execution, token transfer, EOS checks, and common per-step guards. It excludes model loading, adapter folding and switching, warmup, prompt transfer, token decoding, scoring, and final trace copies.

We validate the four cache policies against the reference executor using pretrained weights at boundary and active calls, including the next prompt refresh. Independent CPU checks verify confidence decisions, KV update ranges, safe EOS stopping, token traces, official scores, checkpoint identity, and GPU pairing. Bootstrap intervals use 10,000 paired resamples within tasks (RNG seed 2026092503),

conditional on the checkpoint and recorded timings. The 13-input subset has one source per task, so its scores are reported descriptively.

Table 32: Paired uncertainty for complete-generation comparisons. Speedup is the ratio of mean request times.

Set	Baseline	Score gain [95% CI]	Speedup [95% CI]
RULER 32K (26)	Prefix	+14.55 [6.28, 22.82]	5.15 [5.08, 5.22]
RULER 32K (26)	Dual	+8.59 [1.86, 15.32]	5.04 [4.97, 5.12]
LongBench V1 (24)	Prefix	+1.25 [-3.57, 6.85]	1.71 [1.37, 2.16]
LongBench V1 (24)	Dual	+1.32 [-3.55, 6.95]	1.55 [1.29, 1.83]

Table 33: Complete RULER task scores under parallel decoding. The 26-input group has two sources per task; the matched 13-input subset has one per task.

	RULER 32K ($n = 26$)			Matched RULER 32K ($n = 13$)			
Task	Prefix	Dual	REPRISE	NoCache	Prefix	Dual	REPRISE
cwe	20.00	10.00	5.00	40.00	40.00	20.00	0.00
fwe	33.33	33.33	50.00	33.33	0.00	0.00	66.67
niah_multikey_1	100.00	100.00	100.00	100.00	100.00	100.00	100.00
niah_multikey_2	100.00	100.00	100.00	100.00	100.00	100.00	100.00
niah_multikey_3	100.00	100.00	100.00	100.00	100.00	100.00	100.00
niah_multiquery	37.50	37.50	100.00	100.00	25.00	25.00	100.00
niah_multivalue	25.00	62.50	100.00	100.00	25.00	100.00	100.00
niah_single_1	100.00	100.00	100.00	100.00	100.00	100.00	100.00
niah_single_2	100.00	100.00	100.00	100.00	100.00	100.00	100.00
niah_single_3	100.00	100.00	100.00	100.00	100.00	100.00	100.00
qa_1	100.00	100.00	100.00	100.00	100.00	100.00	100.00
qa_2	0.00	50.00	50.00	0.00	0.00	0.00	0.00
vt	80.00	80.00	80.00	80.00	80.00	100.00	100.00

Table 34: Complete LongBench V1 task scores on 24 inputs.

Task	Prefix	Dual	REPRISE
2wikimqa	8.33	8.33	15.08
gov_report	34.59	32.69	25.93
hotpotqa	14.19	14.19	15.25
multifieldqa_en	24.07	25.30	25.72
passage_retrieval_en	100.00	100.00	100.00
qasper	10.95	11.13	8.19
repobench-p	65.33	65.33	77.33
trec	66.67	66.67	66.67