RISA: RECOVERABLE SPARSE ATTENTION FOR BLOCK-DIFFUSION LANGUAGE MODELS

Anonymous authors
Paper under double-blind review

ABSTRACT

Block-wise diffusion language models (DLMs) refine multiple tokens in parallel, offering an alternative to token-by-token autoregressive decoding. Their long-context inference, however, remains bottlenecked by memory-bound attention: every denoising step repeatedly reads a growing historical key–value (KV) cache. Directly applying sparse attention does not fully solve this problem, because different queries can select different pages, inflating the block's collective read set, while repeated support construction adds overhead and permanent eviction can discard information needed later. We present RISA, a recoverable sparse-attention method with shared read plans for block-wise DLMs. RISA exploits the fact that historical KV remains fixed within a block while queries evolve across denoising steps. Newly generated history is organized into exact priority and recovery tiers; seed queries then build a protected, bounded read plan whose indices are reused across the block while attention is recomputed with fresh queries and current-block KV. This separates history preservation from repeated reading and amortizes sparse-support construction without changing the backbone or decoder. On DreamReasoner-8B, RISA retains comparable task performance and delivers $2.32\times$ sparse-attention and $1.31\times$ complete-step speedup over dense at a 32K-token prefix. In a 128K scalability study on H200, sparse-attention speedup rises to $5.53\times$ as the prefix grows under the same model and decoding stack.

1 INTRODUCTION

Diffusion language models (DLMs) generate text by iteratively refining multiple token positions in parallel, offering an alternative to token-by-token autoregressive decoding (Austin et al., 2021; Lou et al., 2023; Sahoo et al., 2024). Large-scale models such as LLaDA and Dream demonstrate the potential of this paradigm for language modeling, instruction following, and reasoning (Nie et al., 2026; Ye et al., 2025). Block-wise DLMs combine parallel refinement within each block with autoregressive progression across blocks, allowing completed blocks to serve as cached context for subsequent generation (Arriola et al., 2025; Wu et al., 2026b). This parallelism, however, does not remove the cost of accessing history. Caching avoids *recomputing* committed key–value (KV) states, but every denoising step still *reads* the growing prefix to compute attention. For long prompts and extended reasoning traces, these repeated KV accesses become increasingly memory-bound and limit the decoding speedup that block-wise parallelism can deliver (Kwon et al., 2026; Xi et al., 2026).

Sparse attention is a natural way to reduce this cost by reading only a subset of the historical KV (Zhang et al., 2023; Tang et al., 2024; Xiao et al., 2024a). Yet sparsity designed for ordinary autoregressive decoding does not directly translate into efficient block diffusion. A block contains many evolving queries, and independently selecting a small support for each query can produce a much larger union of accessed pages—the *KV inflation* problem identified by Xi et al. (2026). Reconstructing these supports across denoising steps further introduces repeated selection overhead. An alternative is to permanently evict low-priority KV, but doing so can remove information that later blocks need. Recent diffusion-specific methods therefore amortize attention computation through pattern, index, or output reuse (Wang et al., 2026; Kwon et al., 2026; Xi et al., 2026). The resulting design challenge is to bound repeated KV reads without permanently discarding historical information that later blocks may still need under different queries.

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.

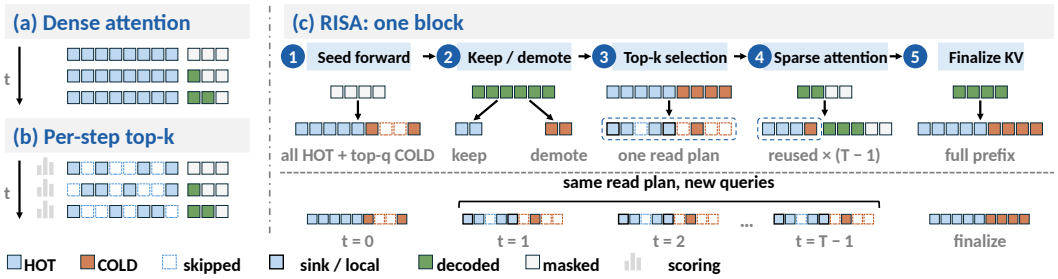


Figure 1. Dense rereads history; RISA reuses one bounded read plan. (a) Dense attention revisits the historical prefix at every denoising step. (b) Per-step top-k reduces each attention support but repeats scoring and selection. (c) RISA uses the seed forward to observe available history, admits aging KV, constructs one protected read plan, and reuses its historical indices for later steps while queries continue to evolve; finalization still reads the full legal prefix.

Block diffusion provides a useful structure for addressing this question. During the denoising of one block, historical KV remains fixed while the queries associated with the unfinished block continue to evolve. This creates a natural block-level scope for selecting historical addresses once and reusing them across subsequent denoising steps, while still recomputing attention with fresh queries. At the same time, the repeated queries issued before a generated block ages out provide multiple observations of how its tokens are used. These observations can inform longer-term priority without forcing an irreversible decision immediately after generation, giving the block a natural scope for read selection and history management without assuming static attention.

We introduce RISA, a sparse-attention method that separates *preserving historical KV* from *deciding which history a block reads* (Figure 1). Newly generated KV first remains in a grace window, where repeated denoising queries provide observations before it is assigned to a compact priority tier or an addressable recovery tier; both retain exact KV values on the GPU. For each new block, seed queries jointly select a protected, bounded page plan across the available history. Subsequent denoising steps reuse these indices while recomputing attention with their current queries and current-block KV, and later blocks may recover history omitted from an earlier plan. The backbone, native decoder, commit rule, and full-prefix KV finalization remain unchanged throughout generation.

Our main contributions are:

- We introduce a **recoverable history organization** that uses repeated denoising observations to prioritize generated KV while keeping lower-priority history available for later blocks.
- We develop a **block-static sparse-attention executor** that bounds older-history reads and reuses one page plan across denoising steps while recomputing attention with fresh queries.
- We provide a **cross-model quality and system evaluation** across reasoning, code, retrieval, and long-context settings. On DreamReasoner-8B, RISA retains comparable task performance while reaching $2.32\times$ sparse-attention and $1.31\times$ complete-step speedup at 32K; a 128K scalability study reaches $5.53\times$ on H200 GPUs.

2 Related Work

Diffusion language models. Discrete and masked diffusion generate text by reversing token corruption (Austin et al., 2021; Lou et al., 2023; Sahoo et al., 2024); LLaDA and Dream scale this approach to language modeling (Nie et al., 2026; Ye et al., 2025). Block Diffusion combines within-block denoising with autoregressive progression and prefix caching (Arriola et al., 2025), while DreamReasoner develops block-size curricula for reasoning (Wu et al., 2026b). We target the repeated attention reads that remain after historical representations have been cached.

KV compression and sparse attention. FlashAttention optimizes exact attention through IO-aware execution and work partitioning (Dao et al., 2022; Dao, 2024). KV compression uses observed attention (Zhang et al., 2023; Liu et al., 2023; Li et al., 2024), sinks and recency (Xiao et al., 2024b),

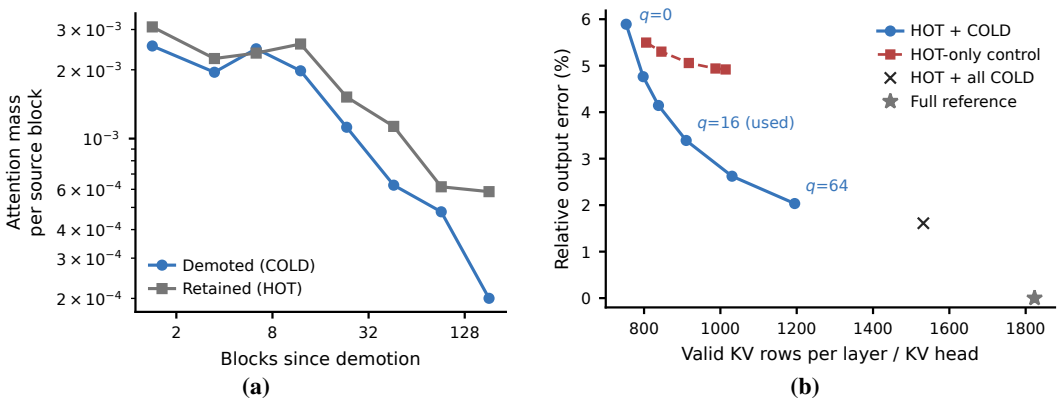


Figure 2. Low-priority history can remain useful. (a) Full-domain attention continues to revisit tokens after they have been demoted from the priority tier. (b) At a comparable read budget, retrieving from the recovery tier yields lower attention-output error than spending the same budget only on additional priority pages. Error bars in (a) resample requests.

or head specialization (Xiao et al., 2025). Retrieval methods preserve a larger history and select query-relevant KV (Tang et al., 2024; Ribar et al., 2024; Xiao et al., 2024a). TidalDecode reuses indices across layers (Yang et al., 2024); HiSparse combines host storage with bounded GPU residency (Xie et al., 2026). RISA instead keeps both tiers on the GPU: priority determines access preference, not deletion, and the objective is reduced read traffic rather than KV-memory compression.

Efficient diffusion inference. Fast-dLLM introduces approximate block-wise KV caching for bidirectional DLMs and confidence-aware parallel decoding (Wu et al., 2026a); dKV-Cache likewise targets redundant representation computation (Ma et al., 2026). Caching reduces what must be recomputed, while parallel decoding changes which tokens are committed together. Cached KV can nevertheless be read in full at every step. RISA instead reduces these reads through sparse attention while preserving the backbone and native commit and stopping rules. It addresses a complementary bottleneck rather than replacing the cache or decoder.

Within sparse diffusion inference, SparseD reuses head-specific patterns (Wang et al., 2026), MAGE derives token indices from exact all-mask attention and reuses them within a block (Kwon et al., 2026), and Sparse-dLLM combines eviction with block-boundary refresh (Song et al., 2025). LoSA instead reuses prefix-attention outputs for stable tokens (Xi et al., 2026). RISA recomputes attention for every current-block query and selects jointly from page summaries. Its distinction is not index reuse alone, but combining a protected, bounded read plan with observation-guided, recoverable generated history that remains addressable across later blocks.

3 Motivation: Preserve History, Avoid Repeated Work

Sparse attention is useful only when the cost it removes is not replaced by irreversible cache decisions or repeated support construction. We therefore ask two questions: should low-priority history remain recoverable, and must a block rebuild its support at every denoising step? The two decisions operate at different time scales: storage priority persists across blocks, while a read plan serves only the block currently being refined. Diagnostic alternatives reuse the same sampled queries, KV values, and masks, so they isolate these mechanisms without changing the generated trajectory; protocol details appear in the appendix.

3.1 Low Priority Does Not Mean No Future Use

Attention-based eviction estimates importance from observations available so far (Zhang et al., 2023; Li et al., 2024). Such evidence can prioritize what should be read frequently, but it is weaker evidence for permanent deletion because later blocks issue different queries. We therefore treat priority as a scheduling decision rather than a claim that omitted history has become irrelevant.

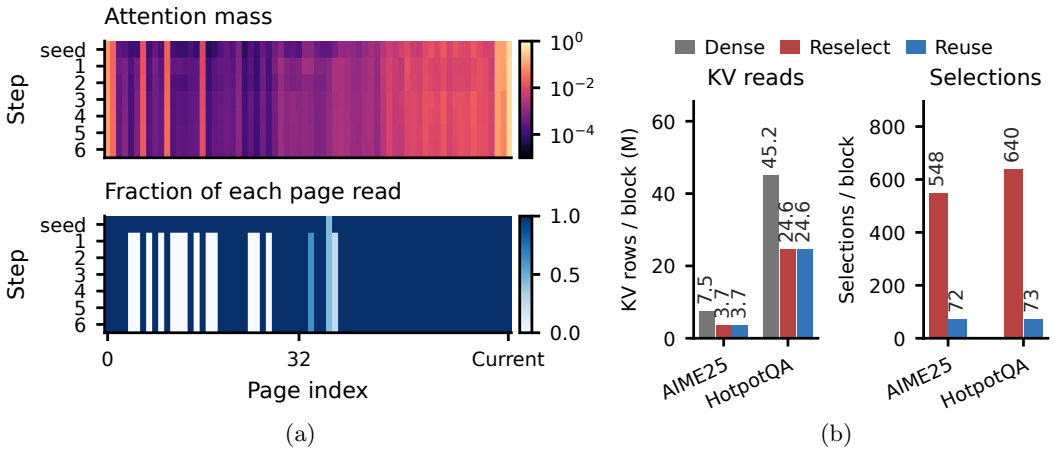


Figure 3. Read sparsity and index reuse remove different work. (a) Attention mass changes across denoising, while a block-static execution can keep the same historical support. (b) Sparse execution reduces requested KV rows, and reusing the plan avoids repeatedly rebuilding that support. Counts include seed and mandatory work and should not be interpreted as direct HBM-traffic measurements.

Figure 2a shows that demoted cohorts continue to receive attention from later blocks. Moreover, redirecting the same extra read budget to additional HOT pages leaves a larger approximation error than retrieving COLD pages (Figure 2b). Recent use is therefore informative for prioritization, but not a reliable certificate for irreversible eviction; recoverable storage preserves this distinction.

3.2 Bound Reads and Amortize Selection

Query-aware retrieval reduces KV reads but introduces scoring, ranking, and dispatch (Tang et al., 2024). In block diffusion these costs can repeat even though historical KV is unchanged: attention weights evolve, but one historical address set can often serve several denoising queries.

Figure 3 illustrates the two savings. Sparse support bounds the historical rows consumed by attention, whereas plan reuse removes repeated selection without reusing stale attention outputs. The controlled ablation in Section 5.2 shows the same separation in latency. Sparsity and reuse therefore address complementary costs: what is read, and how often those reads must be selected.

Reuse also requires a bounded read set: a shared plan does not scale if it retains a fixed fraction of an ever-growing prefix. We therefore protect recent or otherwise mandatory context separately and cap eligible older HOT reads. This separates two complementary sources of savings within the same block: sparse support limits recurring KV reads, while plan reuse amortizes the work of finding that support. The main ablation confirms that the cap lowers attention cost without changing the observed greedy score. The resulting plan bounds recurring reads while leaving lower-priority history recoverable for later blocks and longer prefixes, rather than turning a read-priority decision into permanent eviction.

4 RISA

RISA separates the lifetime of stored history from the lifetime of its read plan (Figure 4). The three panels expose this separation directly: observations determine storage priority as blocks age, seed queries compose a bounded plan, and later steps reuse only its historical addresses while recomputing attention. Thus, storage state persists across blocks, whereas the sparse plan is temporary state for the block being denoised. Each model uses one fixed budget, with no change to its backbone or decoder.

4.1 Observe Recent History Before Prioritizing It

Newly committed blocks enter a fully readable *grace window*. Repeated queries supply observations before aging entries enter compact HOT storage or addressable COLD storage. Both preserve their

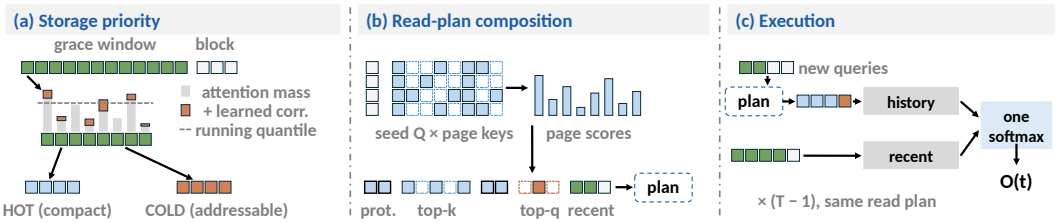


Figure 4. Storage priority, read-plan composition, and execution. (a) Grace-window observations determine whether generated KV enters compact HOT or addressable COLD storage; both keep exact values. (b) Seed queries score page summaries, then protected HOT, selected HOT, recovered COLD, and mandatory recent rows form one read plan. (c) Later steps reuse the historical plan with new queries and current recent KV, combining all contributions under one softmax.

stored KV values on the GPU; prompt KV stays in HOT but remains subject to read selection. Future plans can read COLD entries directly without promotion or deletion.

For each token and KV head, a fixed priority rule maps accumulated attention observations to a priority score. A running quantile of prior scores targets eight retained HOT entries per 32-token source block. Observations come from the executed attention distribution, with no additional teacher forward. This score controls long-term storage priority rather than the current block's read set.

With 32-token blocks, 16-slot pages, and $G = 8$, the seed observes the outgoing eighth block before admission. Later steps read the seven more recent blocks, when present, and the active block. Short prefixes below 512 tokens retain the carrier's original dense-attention path.

Storage state versus read state. HOT/COLD assignment persists across blocks and determines how historical KV is organized, whereas the read plan is temporary state for the block currently being denoised. A page omitted from one block's plan is therefore not deleted: it remains addressable and can be selected again by a later block. Conversely, retaining a page in storage does not force every step to read it. This separation bounds recurring reads without turning sparsification into irreversible KV eviction, so omitted pages remain eligible for future block plans.

4.2 Select One Protected, Bounded Plan

The seed reads full HOT, mandatory rows, and up to $q = 16$ query-selected COLD pages, producing logits and saved queries. HOT eligibility and protection are prepared before this forward. After observations are updated and aging entries are admitted, the native rule commits seed proposals. If denoising continues, saved queries build the reuse plan. Newly admitted HOT rows remain mandatory for this block and join the ranked candidate set when the next block begins.

Each layer and KV head has one plan shared by its positions and mapped query heads. For each query, we take the larger dot product with a page's minimum or maximum key vector, then sum these scores over the block before selecting pages. This yields one bounded support rather than a union of per-query sets. The max-after-dot heuristic uses page-wise key extrema (Tang et al., 2024), but differs from QUEST's coordinate-wise bound in how relevance is aggregated across the page.

Let H_b^{prot} contain the first four and last eight sealed HOT pages, protecting initial and recent resident context (Xiao et al., 2024b). From the remaining eligible pages E_b, we select a fraction ρ, subject to a 512-page cap; $\rho = 0.5$ for DreamReasoner and 1.0 for SDAR. The budget k_b rounds $\rho|E_b|$, clips it to the cap and candidate count, and selects at least one page when available. Using pooled page scores u, we combine mandatory and selected rows into the block's read support:

$$S_b = R_b \cup \mathrm{rows}\big(H_b^{\mathrm{prot}} \cup \mathrm{Top}_{k_b}(u, E_b) \cup \mathrm{Top}_q(u, C_b)\big)\,, \tag{1}$$

where C_b contains COLD candidates and R_b contains mandatory recent/current, partial-page, and newly admitted HOT rows. The cap excludes protected pages, COLD, and mandatory rows. Each valid row appears once; unselected HOT pages stay available for subsequent block plans.

Table 1. Accuracy results of DreamReasoner-8B with 32-token blocks. Mathematics uses answer accuracy, LCB execution pass@1, and HotpotQA token F1. Bold marks the highest sparse-method score, including ties; Dense is the unbolded reference.

Method	GSM8K Acc.	MATH-500 Acc.	AIME25 Acc.	LCB pass@1	HotpotQA F1
Dense	85.0	88.2	58.3	50.0	40.2
QUEST	86.0	88.2	59.4	47.5	38.9
MAGE	85.5	**89.0**	59.2	42.5	35.2
RISA	**86.0**	88.8	**60.4**	**50.0**	**39.9**

4.3 Reuse Addresses, Not Attention Outputs

Later steps reuse S_b with fresh queries, current-block KV, and attention outputs. Historical and recent/current partials share one softmax normalization; they are not independently normalized outputs that are averaged. Persistent device buffers hold indices and valid lengths for CUDA-graph replay; a block completed by its seed skips plan construction entirely.

Reuse addresses, recompute attention. The shared plan fixes only which historical rows are gathered. Each denoising step still recomputes query–key scores, normalization, and value aggregation using its current queries and current-block state. Thus the optimization does not assume that attention probabilities are static across denoising; it amortizes support construction while preserving step-specific attention computation. Mandatory recent/current rows remain outside the eligible-HOT cap so that newly evolving context is not frozen by the seed plan.

After commitment, the native full-prefix finalization pass reads all legal KV to write the block's representations, independently of S_b. Seed attention, maintenance, and finalization therefore remain context-dependent costs. Recent protection and later recovery mitigate omitted-support error without making attention lossless. Exact storage does not imply dense-equivalent trajectories.

5 Evaluation

Experimental setup. We evaluate DreamReasoner-8B(Wu et al., 2026b) and SDAR-8B-Chat (Cheng et al., 2025) in BF16 on NVIDIA H200 GPUs, using 32-token blocks and one request per device. RISA uses $G = 8$, $K_{\max} = 512$, and $q = 16$, with a fixed read fraction per model: $\rho = 0.5$ on DreamReasoner and 1.0 on SDAR. Efficiency is measured at 16K, 32K, 64K, and 128K prefixes. Both models are configured for contexts up to 32K; we use the longer prefixes to characterize long-context scalability, while task-quality evaluation remains within the configured range.

Baselines. Dense uses each carrier's native full-attention path. QUEST performs query-dependent sparse retrieval, while MAGE forms a block-level sparse pattern from its all-mask observation and reuses that pattern during later denoising steps. As no official MAGE implementation was publicly available at the time of our evaluation, we reimplement it following the algorithm and configuration described in the paper (Kwon et al., 2026). Dense, QUEST, MAGE, and RISA share the same model weights, tokenizer, prompts, native decoder, and block size. QUEST and MAGE use nominal 1,024-token budgets. RISA's cap applies only to eligible HOT pages, with protected HOT, recovered COLD, and mandatory recent/current rows handled separately. Because these budget semantics differ, we compare the implementations under their stated configurations and a common timing boundary rather than treating equal nominal token counts as identical read sets.

5.1 End-to-End Accuracy

Benchmarks. DreamReasoner-8B is evaluated on GSM8K, MATH-500, AIME25, LiveCodeBench v6, and HotpotQA; SDAR-8B-Chat is evaluated on GSM8K, MATH-500, and HotpotQA. All methods use identical prompts, decoding settings, and generation limits within each model and benchmark. Mathematics uses answer accuracy, LiveCodeBench uses execution pass@1, and HotpotQA uses

Table 2. Accuracy results of SDAR-8B-Chat with 32-token blocks. All tasks use greedy decoding; metrics and boldface follow Table 1.

Method	GSM8K Acc.	MATH-500 Acc.	HotpotQA F1
Dense	88.5	65.4	53.6
QUEST	87.5	66.2	52.3
MAGE	88.0	66.2	51.4
RISA	**89.5**	**66.8**	**53.4**

Table 3. Core mechanism ablation. Accuracy uses greedy decoding on an AIME25 subset used only for this controlled study. All latency values use the same 32K implementation and timing boundary; lower is better.

Variant	Acc. (%)	Attn. (ms)	Step (ms)
Full RISA	56.7	2.307	9.365
No COLD recovery	50.0	2.312	9.357
Per-step selection	53.3	3.710	11.152
No HOT cap	56.7	2.754	9.872

token F1. GSM8K, MATH-500, LiveCodeBench, and HotpotQA use greedy decoding; AIME25 uses temperature 0.6 with prompt and output together limited to 32,768 positions. Output caps are 24,576 tokens for mathematics/code and 1,024 for HotpotQA.

DreamReasoner-8B. Table 1 reports competitive scores across reasoning, code, and retrieval. RISA obtains 86.0% on GSM8K and 88.8% on MATH-500, compared with dense's 85.0% and 88.2%. Its 50.0% LiveCodeBench accuracy matches dense and exceeds QUEST and MAGE by 2.5 and 7.5 points. On HotpotQA, it is 0.3 F1 below dense but 1.0 and 4.7 above the sparse alternatives.

The ranking is task-dependent: RISA ties QUEST on GSM8K, while MAGE is 0.2 points higher on MATH-500. On AIME25, RISA scores 60.4%, versus 58.3% for dense, 59.4% for QUEST, and 59.2% for MAGE. Overall, RISA remains close to dense while matching or exceeding the reported sparse alternatives on four of the five DreamReasoner tasks.

SDAR-8B-Chat. We use the same history-management design on SDAR with the fixed configuration stated above (Table 2). RISA reaches 89.5% on GSM8K and 66.8% on MATH-500, versus dense's 88.5% and 65.4%, and is 0.2 F1 below dense on HotpotQA. It leads the sparse baselines on all three tasks under identical prompts, benchmark settings, and generation limits throughout the comparison.

With $\rho = 1$, SDAR reads all eligible HOT pages until the cap binds. Its shorter accuracy inputs therefore mainly test admission and COLD recovery, while longer efficiency inputs activate the cap. Transfer therefore tests the same history-management mechanism across workloads whose realized sparse read fractions can differ substantially across task distributions and the context lengths represented in the efficiency study.

5.2 Efficiency Evaluation

5.2.1 Attention and complete-step latency

Our primary metric is *sparse attention latency*: GPU work for attention, selection and indices, normalization/merge, and required KV-cache operations, including attention in full-prefix finalization. Measurements use blocks 10–15 after admission has begun. Recurring step work is combined with per-block work using a common dense-derived amortization horizon of 15.6 steps per block for DreamReasoner and 10.3 for SDAR, covering seed, planning, maintenance, and finalization under one inclusive timing definition shared by both evaluated backbones.

Complete-step latency uses two paired sources per context, measured without profiling. It includes the model, decoder, charged setup, and block boundaries, divided by the number of completed denoising steps. Unlike normalized attention cost, this metric follows each method's realized execution.

Each sparse method is compared with a dense reference measured under the same model, context length, block size, and timing protocol, keeping the execution boundary consistent across the reported speedups and avoiding method-specific timing windows.

Scaling with prefix length. RISA has the lowest measured attention cost among the evaluated implementations at every length on both models (Figure 5). On DreamReasoner, speedup over dense increases from $1.50\times$ at 16K to $2.32\times$, $3.51\times$, and $5.53\times$ at 32K, 64K, and 128K. Over this eightfold

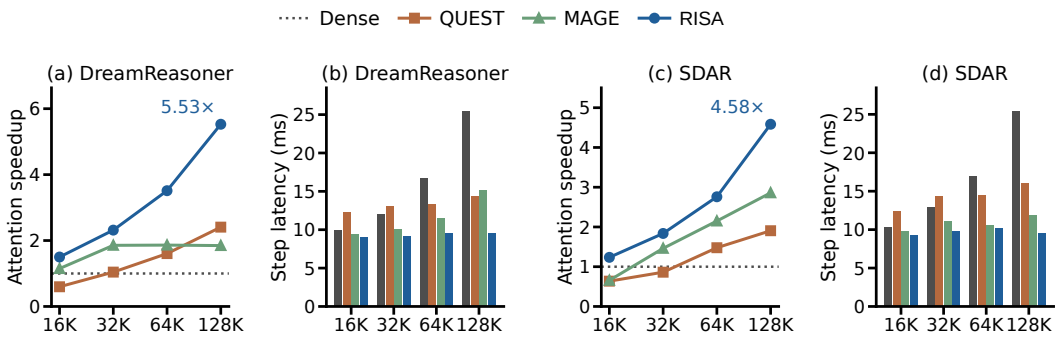


Figure 5. Efficiency across prefix lengths on both models. (a,c) Inclusive sparse attention speedup over the session-paired dense reference; per-block work uses a common model-specific amortization. (b,d) Uninstrumented complete-step latency. All methods use 32-token blocks.

increase in prefix length, dense cost rises from 3.201 to 18.763 ms, whereas RISA rises from 2.135 to 3.394 ms. The cap constrains repeated eligible-HOT reads, but seed, metadata, and finalization work still scale with history. The result is slower cost growth, not constant-time attention. This widening gap reflects the bounded read plan: as the prefix grows, dense attention repeatedly revisits more history, whereas RISA limits the historical support consumed by later denoising steps.

The same trend holds on SDAR: RISA reaches 1.24×, 1.84×, 2.76×, and 4.58× speedup as the prefix grows. This cross-model trend supports the execution design: amortizing selection over repeated within-block queries and bounding the history they collectively read. These long prompts remain in HOT, so the largest gains primarily reflect capped prompt reads. Recovery diagnostics instead evaluate later use of demoted generated history.

Scalability to Blackwell GPUs. We additionally measure amortized attention-path latency for SDAR-8B on an NVIDIA RTX PRO 6000 Blackwell Server Edition at a 64K-token prefix and block size 32. Using SDAR-8B's measured average of 10.30 denoising steps per block and the same inclusive accounting for selection, gathering, KV maintenance, and block finalization, dense requires 10.43 ms per step, compared with 3.48 ms for RISA (2.99× speedup), 7.18 ms for MAGE (1.45×), and 5.39 ms for QUEST (1.94×). The attention-path advantage therefore persists on a newer GPU generation.

From attention savings to model execution. RISA's complete-step latencies on DreamReasoner are 9.013, 9.192, 9.555, and 9.553 ms, yielding paired speedups of 1.10×, 1.31×, 1.76×, and 2.66× over dense. On SDAR the corresponding speedups are 1.11×, 1.33×, 1.67×, and 2.65×. RISA is also faster per step than either sparse baseline at each measured length. Projection, feed-forward, decoding, and synchronization remain outside the sparse-attention savings and therefore determine how much of the attention-path improvement appears in complete-step latency.

5.2.2 Where the attention path spends time

Figure 6 separates Attention, the prefix computation, from Index, importance scoring, and Top-k, selection and plan construction. Merge combines attention partials and observations; Cache covers KV movement and layout maintenance. Index therefore includes the work of deriving relevance scores, not merely accessing an existing index buffer.

MAGE's large Index segment comes from its all-mask observation, not repeated reselection. At each block's seed, the evaluated implementation computes attention output and separately forms full-prefix query–key scores, normalizes them, and averages probabilities over query positions and grouped heads to rank tokens. These operations count as Index; the final token selection counts as Top-k. The scoring work grows with the prefix even when the retained token budget is fixed, so its once-per-block cost remains visible after amortization.

RISA scores page summaries rather than materializing full token-level probabilities for ranking, then reuses the bounded plan. Both methods reuse indices; their score construction differs. This

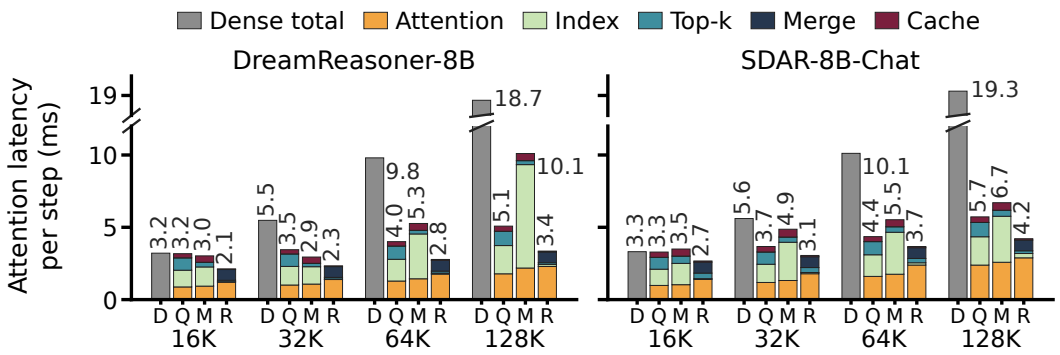


Figure 6. Attention-path components on both models. DreamReasoner-8B is shown on the left and SDAR-8B-Chat on the right. Within each prefix group, D/Q/M/R denote Dense, QUEST, MAGE, and RISA. Colors separate Attention, Index (score computation), Top-k (selection), Merge, and Cache; Dense is an inclusive total. Broken axes preserve the 128K dense bars while emphasizing the smaller costs; labels are rounded, and all components use the same inclusive timing boundary.

explains the contrasting Index costs in the measured implementations without implying that either seed attention or full-prefix finalization becomes constant-time.

QUEST scores each query and attends the resulting union per KV head, so its Index and Top-k components reflect query-wise support construction. The component breakdown is diagnostic; the session-paired totals in Figure 5 remain the primary efficiency measurements.

Memory footprint. Recoverability trades storage for fewer reads. Peak allocation is 29.38 versus dense's 27.62 GiB at 16K and 44.20 versus 39.27 GiB at 64K in a separate memory cohort. Packed KV, metadata, and scratch add storage, so these measurements characterize a read-efficiency tradeoff rather than KV-memory compression in the measured configurations.

Core mechanism ablation. Table 3 isolates recoverable history, block-static plan reuse, and bounded HOT reads. Accuracy uses greedy decoding on an AIME25 subset reserved for this controlled study, while all latency variants use the same 32K implementation and timing boundary; all non-varied settings are fixed. Removing COLD recovery lowers accuracy from 56.7% to 50.0%. Rebuilding support every step raises sparse-attention latency from 2.307 to 3.710 ms and complete-step latency from 9.365 to 11.152 ms, while removing the HOT cap raises sparse-attention latency to 2.754 ms. Together, these variants support recovery, plan reuse, and bounded recurring reads as complementary components of the design. These ablations separate the roles of the three components: recovery preserves access to older history, plan reuse amortizes support construction, and the HOT cap bounds recurring reads.

6 Conclusion

RISA separates preserving historical KV from repeatedly reading it. Denoising observations guide storage priority, while a protected, bounded plan is shared across queries and reused with fresh attention. This block-wise structure uses evolving denoising queries twice: first to observe recent history before prioritization, and then to amortize a bounded support across later steps. Across both evaluated backbones, the method retains comparable task scores while lowering sparse-attention and complete-step latency; the 128K H200 scaling results and the 64K SDAR Blackwell study show that the attention-path advantage persists as context and hardware change.

The evaluation also separates the roles of the main mechanisms. Recoverable COLD storage preserves access to deprioritized history, block-static plans amortize support construction, and the HOT cap bounds recurring reads as the prefix grows. Because exact historical KV remains resident, RISA targets read efficiency rather than KV-cache compression, trading storage for fewer repeated reads.

More broadly, RISA is a runtime optimization for block-wise diffusion: it leaves the backbone, decoder, and stored KV values unchanged while reducing how often historical rows are selected.

AI USE STATEMENT

Generative AI tools were used to edit author-written text for grammar, clarity, and readability. The authors reviewed all AI-assisted edits and take full responsibility for the final content of the paper.

REPRODUCIBILITY STATEMENT

The paper specifies the inference policy, decoding settings, and measurement scope. Static figures and directly editable tables accompany the source. Primary and supplementary studies use the measurement definitions and benchmark protocols documented in the paper and accompanying source.

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*, volume 2025, pp. 50726–50753, 2025.

Jacob Austin, Daniel D Johnson, Jonathan Ho, Daniel Tarlow, and Rianne Van Den Berg. Structured denoising diffusion models in discrete state-spaces. *Advances in neural information processing systems*, 34:17981–17993, 2021.

Yushi Bai, Xin Lv, Jiajie Zhang, Hongchang Lyu, Jiankai Tang, Zhidian Huang, Zhengxiao Du, Xiao Liu, Aohan Zeng, Lei Hou, et al. 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.

Shuang Cheng, Yihan Bian, Dawei Liu, Linfeng Zhang, Qian Yao, Zhongbo Tian, Wenhai Wang, Qipeng Guo, Kai Chen, Biqing Qi, and Bowen Zhou. Sdar: A synergistic diffusion-autoregression paradigm for scalable sequence generation, 2025. URL https://arxiv.org/abs/2510.06303.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. Training verifiers to solve math word problems, 2021. URL https://arxiv.org/abs/2110.14168.

Tri Dao. Flashattention-2: Faster attention with better parallelism and work partitioning. In *International Conference on Learning Representations*, volume 2024, pp. 35549–35562, 2024.

Tri Dao, Dan Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. Flashattention: Fast and memory-efficient exact attention with io-awareness. *Advances in neural information processing systems*, 35: 16344–16359, 2022.

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. Measuring mathematical problem solving with the math dataset, 2021. URL https://arxiv.org/abs/2103.03874.

Naman Jain, Han, Alex Gu, Wen-Ding Li, Fanjia Yan, Tianjun Zhang, Sida Wang, Armando Solar-Lezama, Koushik Sen, and Ion Stoica. Livecodebench: Holistic and contamination free evaluation of large language models for code. In Y. Yue, A. Garg, N. Peng, F. Sha, and R. Yu (eds.), *International Conference on Learning Representations*, volume 2025, pp. 58791–58831, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/file/94074dd5a072d28ff75a76dabed43767-Paper-Conference.pdf.

Omin Kwon, Yeonjae Kim, Doyeon Kim, Minseo Kim, Yeonhong Park, and Jae W Lee. Mage: All-[mask] block already knows where to look in block diffusion llm. *arXiv preprint arXiv:2602.14209*, 2026.

Yuhong Li, Yingbing Huang, Bowen Yang, Bharat Venkitesh, Acyr Locatelli, Hanchen Ye, Tianle Cai, Patrick Lewis, and Deming Chen. Snapkv: Llm knows what you are looking for before generation. *Advances in Neural Information Processing Systems*, 37:22947–22970, 2024.

Zichang Liu, Aditya Desai, Fangshuo Liao, Weitao Wang, Victor Xie, Zhaozhuo Xu, Anastasios Kyrillidis, and Anshumali Shrivastava. Scissorhands: Exploiting the persistence of importance hypothesis for llm kv cache compression at test time. *Advances in Neural Information Processing Systems*, 36:52342–52364, 2023.

Aaron Lou, Chenlin Meng, and Stefano Ermon. Discrete diffusion modeling by estimating the ratios of the data distribution. *arXiv preprint arXiv:2310.16834*, 2023.

Xinyin Ma, Runpeng Yu, Gongfan Fang, and Xinchao Wang. dkv-cache: The cache for diffusion language models. *Advances in Neural Information Processing Systems*, 38:149009–149033, 2026.

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. *Advances in Neural Information Processing Systems*, 38:50608–50646, 2026.

Luka Ribar, Ivan Chelombiev, Luke Hudlass-Galley, Charlie Blake, Carlo Luschi, and Douglas Orr. Sparq attention: Bandwidth-efficient llm inference, 2024. URL https://arxiv.org/abs/2312.04985.

Subham S Sahoo, Marianne Arriola, Yair Schiff, Aaron Gokaslan, Edgar Marroquin, Justin T Chiu, Alexander Rush, and Volodymyr Kuleshov. Simple and effective masked diffusion language models. *Advances in Neural Information Processing Systems*, 37:130136–130184, 2024.

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, 2025. URL https://arxiv.org/abs/2508.02558.

Jiaming Tang, Yilong Zhao, Kan Zhu, Guangxuan Xiao, Baris Kasikci, and Song Han. Quest: Query-aware sparsity for efficient long-context llm inference. *arXiv preprint arXiv:2406.10774*, 2024.

Zeqing Wang, Gongfan Fang, Xinyin Ma, Xingyi Yang, and Xinchao Wang. Sparsed: Sparse attention for diffusion language models. In *International Conference on Learning Representations*, volume 2026, pp. 99320–99337, 2026.

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*, volume 2026, pp. 57027–57051, 2026a.

Zirui Wu, Lin Zheng, Jiacheng Ye, Shansan Gong, Xueliang Zhao, Yansong Feng, Wei Bi, and Lingpeng Kong. Dreamreasoner-8b: Block-size curriculum learning for diffusion reasoning models. *arXiv preprint arXiv:2606.19257*, 2026b.

Haocheng Xi, Harman Singh, Yuezhou Hu, Coleman Hooper, Rishabh Tiwari, Aditya Tomar, Minjae Lee, Wonjun Kang, Michael Mahoney, Chenfeng Xu, et al. Losa: Locality aware sparse attention for block-wise diffusion language models. *arXiv preprint arXiv:2604.12056*, 2026.

Chaojun Xiao, Pengle Zhang, Xu Han, Guangxuan Xiao, Yankai Lin, Zhengyan Zhang, Zhiyuan Liu, and Maosong Sun. Infllm: Training-free long-context extrapolation for llms with an efficient context memory. *Advances in neural information processing systems*, 37:119638–119661, 2024a.

Guangxuan Xiao, Yuandong Tian, Beidi Chen, Song Han, and Mike Lewis. Efficient streaming language models with attention sinks. In *International Conference on Learning Representations*, volume 2024, pp. 21875–21895, 2024b.

Guangxuan Xiao, Jiaming Tang, Jingwei Zuo, Junxian Guo, Shang Yang, Haotian Tang, Yao Fu, and Song Han. Duoattention: Efficient long-context llm inference with retrieval and streaming heads. In *International Conference on Learning Representations*, volume 2025, pp. 37228–37253, 2025.

Table 4. Absolute efficiency results on DreamReasoner-8B. Costs are milliseconds per denoising step. Attention uses one profiled source per context and a common 15.6-step amortization of block work; complete-step costs average two paired sources. Bold marks the fastest sparse method.

Method	16K	32K	64K	128K
Sparse attention latency				
Dense	3.201	5.438	9.829	18.763
QUEST	5.355	5.211	6.123	7.772
MAGE	2.785	2.954	5.280	10.159
RISA	**2.135**	**2.348**	**2.799**	**3.394**
Complete-step latency				
Dense	9.946	12.002	16.759	25.438
QUEST	12.330	13.124	13.314	14.329
MAGE	9.424	10.055	11.513	15.121
RISA	**9.013**	**9.192**	**9.555**	**9.553**

Zhiqiang Xie, Zhangheng Huang, Tingwei Huang, Ziyi Xu, Ruiyang Ma, and Christos Kozyrakis. Hisparse: Scaling sparse-attention decoding with hierarchical kv cache management. *arXiv preprint arXiv:2608.07009*, 2026.

Lijie Yang, Zhihao Zhang, Zhuofu Chen, Zikun Li, and Zhihao Jia. Tidaldecode: Fast and accurate llm decoding with position persistent sparse attention, 2024. URL https://arxiv.org/abs/2410.05076.

Jiacheng Ye, Zhihui Xie, Lin Zheng, Jiahui Gao, Zirui Wu, Xin Jiang, Zhenguo Li, and Lingpeng Kong. Dream 7b: Diffusion large language models, 2025.

Zhenyu Zhang, Ying Sheng, Tianyi Zhou, Tianlong Chen, Lianmin Zheng, Ruisi Cai, Zhao Song, Yuandong Tian, Christopher Ré, Clark Barrett, et al. H2o: Heavy-hitter oracle for efficient generative inference of large language models. *Advances in neural information processing systems*, 36:34661–34710, 2023.

A Additional Efficiency Results

Table 4 records the original absolute measurements underlying Figure 5; these are not reconstructed from rounded labels in the component plot. Attention includes selection, indices, merge, cache operations, and finalization attention. Block work uses the common 15.6-step horizon on DreamReasoner and 10.3-step horizon on SDAR. Complete-step latency instead includes the backbone and decoder and uses each trajectory's actual step count for the measured blocks.

Measurement scope. Kernels are assigned to their serving call sites. Captured kernels inherit same-name stage shares from an eager run of the same requests, while fused work is counted once. Profiles cover blocks 10–15 of a 512-token-cap run after admission has begun; complete-step timing uses the same output cap without profiling. One fixed source per context is used for component attribution and two paired sources are timed for complete-step latency, with the same protocol applied consistently across methods, prefix lengths, and the reported component profiles.

Memory footprint. Exact recovery does not imply KV compression. In the separate memory study, peak allocation rises from dense's 27.62 to 29.38 GiB at 16K and from 39.27 to 44.20 GiB at 64K. Packed copies, metadata, and scratch trade additional storage for fewer repeated reads, consistent with RISA's focus on read efficiency rather than cache compression.

B Block Execution Algorithm

The pseudocode below makes the ordering explicit: the seed forward observes the outgoing grace-window block before admission, the resulting seed queries build one historical read plan if denoising continues, and finalization remains a full-prefix carrier operation.

Algorithm 1: RISA: denoising one block.

Input: Masked block x; KV cache M; statistics A; frozen model F and priority rule π
Output: Committed block x; updated M, A

$(M, a) \leftarrow \mathrm{StageRecent}(M, G)$
$(z, Q, o) \leftarrow \mathrm{SeedForward}(F, x, M, q)$
$A \leftarrow \mathrm{Update}(A, o)$
if $a \neq \varnothing$ **then** $(M, A) \leftarrow \mathrm{Admit}(M, a, A, \pi)$
$x \leftarrow \mathrm{Commit}(x, z, 0)$; $t \leftarrow 1$
if $\mathrm{Continue}(x, t)$ **then**
  foreach *layer ℓ and KV head h* **do**
    $(H, C, R) \leftarrow \mathrm{Pages}(M, \ell, h)$
    $H_p \leftarrow \mathrm{First}_4(H) \cup \mathrm{Last}_8(H)$; $E \leftarrow H \setminus H_p$
    $u \leftarrow \mathrm{Score}(Q_{\ell,h}, E \cup C)$; $k \leftarrow \mathrm{Budget}(|E|)$
    $S_{\ell,h} \leftarrow R \cup \mathrm{rows}\big(H_p \cup \mathrm{Top}_k(u, E) \cup \mathrm{Top}_q(u, C)\big)$
  while $\mathrm{Continue}(x, t)$ **do**
    $(z, o) \leftarrow \mathrm{SparseForward}(F, x, M, S)$
    $A \leftarrow \mathrm{Update}(A, o)$
    $x \leftarrow \mathrm{Commit}(x, z, t)$; $t \leftarrow t + 1$
$M \leftarrow \mathrm{FinalizeKV}(F, x, M)$
return (x, M, A)

C Additional Evaluation Details

Benchmark protocols. DreamReasoner-8B is evaluated on GSM8K, MATH-500, AIME25, LiveCodeBench v6, and HotpotQA; SDAR-8B-Chat is evaluated on GSM8K, MATH-500, and HotpotQA. All methods use identical prompts, tokenization, decoding settings, and generation limits within each model and benchmark. GSM8K and MATH use mathematical-answer scoring (Cobbe et al., 2021; Hendrycks et al., 2021); LiveCodeBench (Jain et al., 2025) uses execution pass@1; HotpotQA uses LongBench token F1 (Bai et al., 2024). For LiveCodeBench, the final fenced Python program must pass the benchmark tests under Python 3.12.13 with a six-second per-test timeout; missing code scores zero. AIME25 uses temperature 0.6 with a 32,768 total-position limit.

Configuration. The same frozen configuration is used across the reported tasks for each model. DreamReasoner uses $G = 8$, $K_{\max} = 512$, $q = 16$, and $\rho = 0.5$; SDAR uses the same history-management settings with $\rho = 1.0$. QUEST and MAGE use their stated 1,024-token budgets under the common carrier and decoder used throughout the main evaluation.

Motivation cohort. Motivation diagnostics use a fixed request cohort and sample every fourth eligible block at five layers and all KV heads. Recovery error is per-query relative Euclidean error against full-domain attention, with a 10^{-6} denominator floor. At quotas 16/32/64, additional HOT candidates are insufficient in 2.1/48.2/84.9% of layer/head states; these mismatches are retained rather than treated as successful matches in the reported summaries.