MASKAHEAD: UNIFYING KV CACHE SELECTION AND EVICTION IN BLOCK DIFFUSION LANGUAGE MODELS

Anonymous authors
Paper under double-blind review

ABSTRACT

Block diffusion language models decode one block at a time against an exact key-value cache of the prefix, which grows with the prompt and with the model's own long generations. A small cache requires two decisions: which prefix entries the current block should read, and which entries later blocks may need. Selection is temporary. Eviction permanently removes entries from storage. Sparse-read methods cut repeated cache reads but keep the full prefix resident. Eviction reduces storage, but the decision is irreversible: entries removed now cannot serve later blocks. Block diffusion exposes masked positions in upcoming blocks before generation, so a no-write probe can obtain model-derived queries for retention without producing draft tokens. MaskAhead assigns the model's mask queries to the decisions they can inform: current-block queries rank temporary reads, while a no-write probe of upcoming blocks ranks the entries retained until the next eviction. Both decisions use an output-sensitive score that accounts for values. The surviving cache can also be stored at four bits. On MATH-500, Quantized MaskAhead reduces peak KV by 75%, with a controlled accuracy degradation. For long-prompt QA, question-conditioned prefill reduces peak KV by 86.7% on average across HotpotQA, NarrativeQA, and MuSiQue.

1 INTRODUCTION

Diffusion language models (dLLMs) generate text by repeatedly denoising masked positions, replacing the token-by-token autoregressive loop with parallel decoding (Lou et al., 2024; Sahoo et al., 2024; Nie et al., 2026; Ye et al., 2025). This parallelism accelerates generation, but over a long decoding horizon it weakens dependencies among jointly predicted tokens (Wu et al., 2026b; Kang et al., 2026). Current systems therefore use block diffusion, generating blocks in sequence while denoising the tokens within blocks in parallel (Arriola et al., 2025; Wu et al., 2026a; Bie et al., 2025).

Block diffusion restores a causal prefix and with it a key-value (KV) cache: after a block is finalized, later blocks reuse its keys and values without recomputation. As the prompt and response grow, every new block retains more KV entries, and every denoising step reads a longer prefix. Long-context inference therefore pays twice for the same history: in resident cache memory and in repeated cache reads.

Recent methods target the second cost. They use mask-token queries to predict the attention distribution of subsequent denoising steps and read only the most relevant prefix entries (Kwon et al., 2026b;a). This reduces cache reads but retains the full history. Reducing memory instead requires irreversible eviction: deciding which entries future blocks will not need.

Block diffusion makes future queries available before an eviction decision. We introduce MASKAHEAD, which places masks in upcoming blocks and uses the resulting queries to identify which cached tokens future generation will need. The same mechanism unifies both cache decisions: masks in the current block select what to read now, while masks in future blocks determine what to retain for later. Rather than extrapolating future importance from past attention, MASKAHEAD asks the model directly, enabling sparse reads and bounded cache storage within a single framework.

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.

2 Related Work

Sparse reads in block diffusion. An exact prefix cache makes block diffusion practical, but its full contents are read repeatedly during denoising. SparseD (Wang et al., 2026b) exploits attention patterns shared across steps, while FlashBlock (Chen et al., 2026) reuses stable attention outputs outside the active block. MAGE (Kwon et al., 2026b) instead uses attention from the fully masked first step to select one prefix subset for the remaining denoising steps. These methods reduce cache reads across denoising steps but keep the full prefix resident on the GPU, leaving its memory footprint unchanged.

The attention-trace analysis in Figure 1 tests how long a read set remains informative. We use it as a mechanistic diagnostic, not as evidence that a future-mask probe beats alternative query sources.

HERALD (Kwon et al., 2026a) offloads the full cache to host memory and retrieves a selected subset for each block. It uses one representative mask query to reduce the cost of selection. Offloading lowers GPU-resident memory but retains the full cache in host memory and incurs retrieval and transfer costs (Section U). MaskAhead instead physically removes entries, enforcing a storage budget after each eviction. GroupKV (Wang et al., 2026a) extends block-dLLM retrieval with hierarchical group selection and predictive prefetching across cache tiers.

Eviction in diffusion models. Eviction has been studied for full-attention dLLMs, which revisit the entire prompt at every denoising step. Sparse-dLLM (Song et al., 2026) chooses entries using saliency that persists across steps, while MaskKV (Huang et al., 2026) ranks prompt entries by mask-query attention and assigns budgets per layer and head. Block diffusion exposes masks only in the active block and continually appends finalized output to an exact cache. This requires recurrent, forward-looking eviction. We test whether future masks provide a robust look-ahead signal and turn it into a practical strategy.

Estimating what future tokens will need in AR generation. AR eviction methods must make retention decisions without the queries that future tokens will form. StreamingLLM (Xiao et al., 2024) protects attention sinks and recent tokens, while H2O (Zhang et al., 2023) accumulates past attention to preserve heavy hitters. Our eviction policy protects initial and recent positions, and ranks the remaining entries with future-mask attention.

LookaheadKV (Ahn et al., 2026) learns an auxiliary predictor for future queries. It requires additional data and model-specific optimization. Training-free methods instead approximate the missing queries: Expected Attention (Devoto et al., 2025) extrapolates their distribution from past hidden states, while LAQ (Wang et al., 2025) and DapQ (Tian et al., 2026) construct positional surrogates. CAOTE (Goel et al., 2025) addresses a different part of the problem: given attention-based retention weights, it incorporates value vectors to estimate attention-output error. It can be combined with different weight estimators. Its H2O instantiation uses accumulated past attention. Our method supplies the value-aware criterion with the model's masked queries from upcoming blocks. It aligns the evidence with the lifetime of each retention decision and enforces that decision through recurrent physical compaction.

Precision as another storage decision. For diffusion models, the DART NPU study introduces block-adaptive online smoothing for KV quantization and evaluates four-bit cache formats under

Table 1: Capabilities of methods in the experimental comparison. Sparse read selects a temporary attention subset; training-free eviction permanently removes KV entries; quantization stores KV at reduced precision.

Method	Sparse read	Eviction	KV quantization
Dense bf16	×	×	×
HERALD (Kwon et al., 2026a)	✓	×	×
MAGE (Kwon et al., 2026b)	✓	×	×
H2O (Zhang et al., 2023)	×	✓	×
H2O + CAOTE (Zhang et al., 2023; Goel et al., 2025)	×	✓	×
Expected Attention (Devoto et al., 2025)	×	✓	×
DapQ (Tian et al., 2026)	×	✓	×
MaskAhead	✓	✓	×
Q-MaskAhead	✓	✓	✓

blocked inference (Lou et al., 2026). This establishes four-bit KV quantization as prior work in diffusion inference. DART quantizes the full cache representation; it does not combine precision reduction with per-token selection and recurrent eviction. In AR models, KIVI (Liu et al., 2024) and KVQuant (Hooper et al., 2024) reduce cache precision, while mixed-precision schemes jointly allocate entries and bits (Yang et al., 2024; Zhang et al., 2025; 2026). Q-Hitter (Zhang et al., 2024) shows that quantization can alter importance ranking and develops a quantization-aware selector. We test a complementary setting: a physically compacted cache whose four-bit keys and values are used for both ranking and executed attention, together with recurring mask-query-guided eviction.

Table 1 makes the design distinction explicit: existing comparisons cover sparse reads or eviction, while MaskAhead combines them and Q-MaskAhead adds reduced-precision storage.

Contributions.

- We formulate KV selection and eviction in block diffusion as decisions with different horizons. Current-block queries choose what to read, future-block mask queries choose what survives, and physical compaction enforces the configured, sequence-dependent storage budget.
- We introduce MaskAhead, a training-free method that uses current-block queries to select temporary reads and a no-write probe of upcoming blocks to select persistent KV. A value-aware attention-output score ranks entries, and recurrent physical compaction enforces the sequence-dependent storage budget .
- We also introduce Q-MaskAhead, which combines MaskAhead's read selection and recurring eviction with four-bit key and value storage. It performs ranking and attention from the quantized cache, reducing cache use through both fewer retained entries and fewer bits per entry.
- On MATH-500, Q-MaskAhead uses up to 4 $\times$ less peak KV memory than dense bf16. On LongBench, Q-MaskAhead reduces generation-end KV by 20$\times$. On the synthetic NIAH test, Q-MaskAhead solves all 75 placements at a 10% retention budget; on a harder 30-placement pilot it solves 17, versus 19 for dense attention.

Our code is available at `https://anonymous.4open.science/r/MaskAhead`.

3 Problem Setup: Read and Retention Decisions

3.1 Two decisions over one prefix

At block t, let $\mathcal{I}_{t,\ell,h}$ denote the prefix entries stored for layer ℓ and KV head h. The read decision selects at most k_t entries for the remaining denoising steps of this block. Entries outside that temporary read set remain in storage. The retention decision keeps at most C_t entries for later blocks and permanently removes the others. Thus the two budgets act on different objects, even though both rank the same live prefix.

For a query q and retained set S, write $\boldsymbol{o}(q; S)$ for its prefix-attention output. A read set $R_t \subseteq \mathcal{I}_t$ is used only by later denoising steps of block t, so its idealized error is accumulated over those queries. A retention set $S_t \subseteq \mathcal{I}_t$ serves the next E blocks, so its error is accumulated over a different query population:

$$\begin{aligned} \mathcal{L}_{\text{read}}(R_t) &= \sum_{q \in \mathcal{Q}_t^{\text{later}}} \|\boldsymbol{o}(q; \mathcal{I}_t) - \boldsymbol{o}(q; R_t)\|_2, \qquad |R_t| \le k_t, \\ \mathcal{L}_{\text{retain}}(S_t) &= \sum_{j=t}^{t+E-1} \sum_{q \in \mathcal{Q}_j^{\text{realized}}} \|\boldsymbol{o}(q; \mathcal{I}_t) - \boldsymbol{o}(q; S_t)\|_2, \quad |S_t| \le C_t. \end{aligned} \tag{1}$$

Here $\mathcal{Q}_t^{\text{later}}$ contains the later-step queries of the current block, while $\mathcal{Q}_j^{\text{realized}}$ contains queries of a generated block. These sets explain the decision horizons. We do not optimize Equation (1) directly because realized future queries are unavailable when S_t must be chosen.

A full-cache MATH-500 trace shows two different timescales (Figure 1). The fresh-set curves use target-point queries from recorded trajectories, so they are offline coverage references rather than options available at decision time. Within a block, the shared read set retains 97.75% of prefix-attention mass at denoising step 8, versus 98.50% for a fresh, same-budget set ($n = 29$ at that

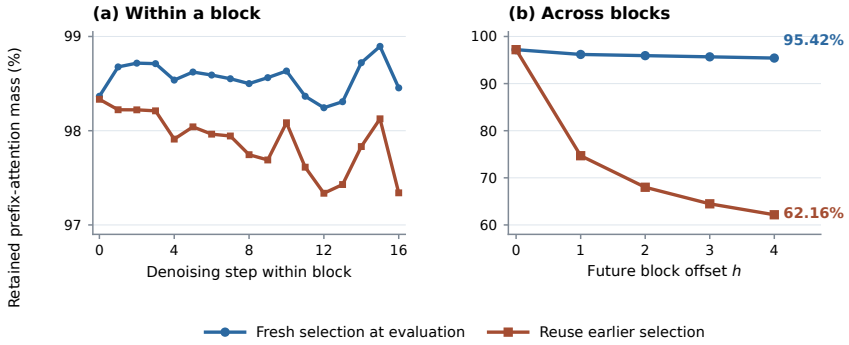

Figure 1: Full-cache MATH-500 trace ($n = 30$ initially). Within-block reuse stays close to step-specific selection; cross-block coverage declines as the prefix grows.

step). Across blocks, its coverage falls to 74.69% after one block and 62.16% after four, while fresh selections retain 96.19% and 95.42%, respectively. The cross-block comparison includes newly appended positions unavailable to the reused set. It measures coverage loss over an expanding prefix, not ranking drift among old entries. These traces motivate within-block reuse and a new read decision for each block. They do not establish the best retention horizon or answer-quality gains; controlled probe ablations and matched-budget task results address those questions in Section 5.2.

3.2 Queries Available Before Commitment

The first step of block t contains B masks. Their queries, $\mathcal{Q}_t^{(1)}$, reveal what the current block requests before any token in it is finalized. Every E blocks, we make a retention decision at this first step. A block-causal probe places masks at the next $M - 1$ block positions and returns query sets $\mathcal{Q}_t^{(2)}, \ldots, \mathcal{Q}_t^{(M)}$ without writing KV or producing output tokens. We set $M = E = 4$, so the probe spans the blocks served by the retention decision. The future masks are conditioned on the known prefix rather than on as-yet-unfinalized intermediate blocks. Their role is to forecast cache use, not to reproduce the eventual queries exactly.

The probe preserves the block-causal view of the prefix: it cannot read finalized tokens from a block that has not been generated. It uses the model's own query projection at each masked position, rather than constructing a query from unrelated past states. Because future blocks are still masked, the forecast can be imperfect. The relevant test is whether it ranks entries used over the eviction interval better than historical or synthetic evidence, not whether every probed query equals a later realized query.

The horizon length follows the lifetime of a retention decision. With $M = 1$, retention reuses only the current block's queries and has no direct evidence from a later block. With $M < E$, some blocks served by the retained cache have no probed queries. With $M > E$, the score includes positions beyond the next checkpoint, when retention can be revised. Setting $M = E$ covers the interval once and then refreshes the forecast as the prefix changes. The ablation in Table G.1 tests this alignment instead of treating a longer probe as automatically better.

4 MaskAhead: Two Decisions, Two Query Horizons

A block needs the prefix in two different ways. It reads a small subset during the current denoising trajectory, then carries a bounded subset forward to blocks that have not been generated. MaskAhead makes these decisions separately. Both use the model's mask queries and the same output-sensitive ranking rule, but each ranks with queries from the horizon it must serve.

A score for attention-output sensitivity. Let $p_{q,i}$ denote the softmax weight of live prefix entry i for query q, normalized over the live prefix, and let $\boldsymbol{o}_q = \sum_i p_{q,i} \boldsymbol{v}_i$. Removing i and renormalizing gives

$$\|\boldsymbol{o}_q - \boldsymbol{o}_{q,-i}\|_2 = \frac{p_{q,i}}{1 - p_{q,i}} \|\boldsymbol{v}_i - \boldsymbol{o}_q\|_2 . \tag{2}$$

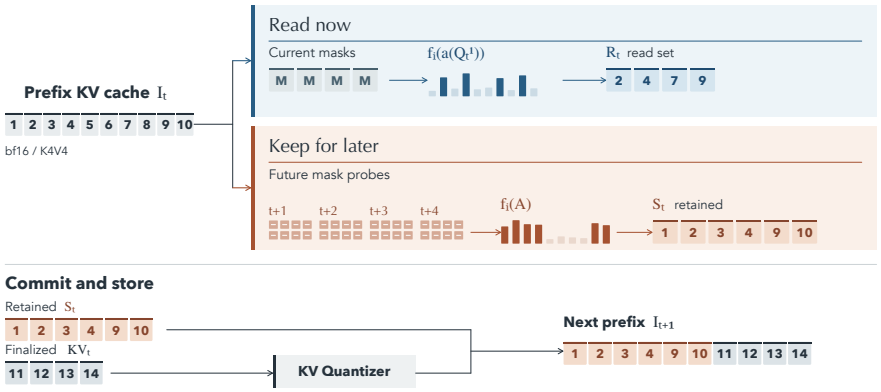

Figure 2: Current-block masks select temporary reads; future masks rank entries retained until the next eviction.

Attention mass alone cannot distinguish two equally attended values with different effects on the output. CAOTE (Goel et al., 2025) uses this output-error principle with attention-based weights for cache eviction. For a population of mask queries $\mathcal{Q}$, we use the pooled proxy

$$a_i(\mathcal{Q}) = \frac{1}{|\mathcal{Q}|}\sum_{q\in\mathcal{Q}} p_{q,i}, \qquad \boldsymbol{c}(\boldsymbol{w}) = \frac{\sum_j w_j \boldsymbol{v}_j}{\sum_j w_j}, \qquad f_i(\boldsymbol{w}) = w_i\|\boldsymbol{v}_i - \boldsymbol{c}(\boldsymbol{w})\|_2. \tag{3}$$

This is a ranking proxy for deleting a set, not the exact joint deletion error. The score specifies how to rank entries once a query population is given. In block diffusion, choosing that population separately for reading and retention is the decisive step.

Current masks select reads; future masks select survivors. At the first denoising step of block t, all B positions are masked. Their queries $\mathcal{Q}_t^{(1)}$ rank the live prefix for the remaining steps. Every E blocks, a no-write probe also places masks in the next $M-1$ block positions and obtains $\mathcal{Q}_t^{(2)}, \ldots, \mathcal{Q}_t^{(M)}$. We choose $M = E = 4$, matching the probe to the interval before the next eviction. The two rankings are

$$s_i^{\text{read}} = f_i\left(\boldsymbol{a}(\mathcal{Q}_t^{(1)})\right), \qquad s_i^{\text{retain}} = f_i\left(\sum_{m=1}^{M} \boldsymbol{a}(\mathcal{Q}_t^{(m)})\right). \tag{4}$$

The first top-k_t set is a temporary read buffer. The second top-C_t set is physically retained, with four initial positions and a 64-position recent window protected. Scores and budgets are per layer and KV head. The first two layers remain dense for read selection. At a checkpoint, the read buffer is gathered before the persistent cache is compacted, so the current block still sees its selected entries. The finalized block then appends its KV with original logical positions. Algorithm 1 makes this order explicit.

Algorithm 1 MaskAhead uses separate query horizons for temporary reads and persistent KV retention.

INPUT: live cache $\mathcal{I}_t$, budgets k_t, C_t, horizon M, interval E.
Run the first all-mask step of block t against $\mathcal{I}_t$; collect $\mathcal{Q}_t^{(1)}$.
$R_t \leftarrow \mathrm{TopK}(s^{\text{read}}, k_t)$; gather R_t into a temporary read buffer.
if t is an eviction checkpoint **then**
    Probe masks in blocks $t+1, \ldots, t+M-1$ without writing persistent KV.
    Rank the live prefix by s^{retain}; protect sinks and recent positions.
    Compact persistent KV to the retained set of budget C_t.
end if
Denoise the remaining steps using R_t; finalize block t and append its KV.

Q-MaskAhead decides from compact storage. Q-MASKAHEAD stores both keys and values at four bits and computes both scores from this representation, without a full-precision shadow cache. Keys use asymmetric per-channel quantization over groups of 32 positions; values use per-token

groups of 32 channels with fp16 metadata. Incomplete key groups remain bf16. The scoring pass unpacks stored values as needed, while complete surviving key groups are copied bit for bit and partial groups are regrouped. Thus quantization changes the cache from which we select and retain, rather than compressing a full-precision decision afterward. Table 5 tests selection and eviction under stored precision; Sections E and Q examine ranking stability and long-prompt quality.

The probe runs once per E blocks. Physical compaction reduces persistent KV, whereas the gathered read buffer lasts only for the current block. Long prompts are prefilled in 512-token chunks and standard retention begins when response generation starts, after the question becomes available. Peak prefill memory and generation-end KV must therefore be reported separately.

5 EXPERIMENTS

We test four consequences of the two-horizon design: quality under a bounded cache, transfer to another block diffusion model, the independent roles of query horizon and score, and the cost of storing decisions at four bits. Our experiments follow the cache through long prefill and serving.

Protocol. Fast-dLLM-v2-7B uses greedy decoding on MATH-500 ($n = 500$), GSM8K ($n = 1319$), and HotpotQA, NarrativeQA, and MuSiQue ($n = 200$ each). Reasoning uses answer accuracy; QA uses token F1. The read budget is $k_t = \min(N_t, \max(128, \lceil 0.05 N_t \rceil))$ and the retention budget is $C_t = \max(256, \lfloor 0.1 N_t^{\text{seen}} \rfloor)$, where N_t is the live prefix and N_t^{seen} includes evicted positions. Long-context QA uses floors of 256 reads and 512 retained entries. Baseline adaptations use the same inference harness, decoding, budgets, four protected initial entries, recent window of 64, and $E = 4$. Selection-only MAGE keeps the full cache. Memory includes method state and is normalized to dense bf16; reasoning uses the run's peak and QA also reports generation-end KV. Hardware and detailed settings are in Section S. DreamReasoner reports MATH-500 ($n = 500$), and LLaDA reports three LongBench tasks ($n = 200$ each). Fast-dLLM quality cells are complete greedy runs; DreamReasoner and LLaDA are supplied point estimates without per-example logs here. Paired Fast-dLLM F1 intervals use 10,000 bootstrap resamples of aligned examples.

5.1 A BOUNDED CACHE OFFERS A SMALLER-MEMORY OPERATING POINT

Tables 2 and 3 report the matched reasoning and LongBench comparisons, separating sparse read selection with a full cache from sparse reads with recurrent physical eviction. The two benchmarks expose different operating regimes (Tables 2 and 3). On reasoning, future-query retention preserves much of the full-cache quality under a restrictive memory budget. On long-context QA, the stronger result is memory reduction at nearly unchanged quality: the HotpotQA score is close to full-cache MAGE and matched-memory DapQ, while the cache occupies about one tenth of dense generation-end KV. These outcomes do not establish uniform superiority. Across the full task and model matrix, the preferred eviction policy changes with the task, and transfer to DreamReasoner is weaker on MATH-500. The practical claim is therefore a new quality-memory operating point, especially under tight retention budgets, rather than a general accuracy gain. Wider budgets weaken this advantage

Table 2: Reasoning quality / peak KV relative to dense bf16. Entries report accuracy (%) / normalized peak KV. DreamReasoner values are supplied point estimates. On GSM8K, DreamReasoner is lossless at our budgets.

		Fast-dLLM-v2-7B		DreamReasoner-8B
Category	Method	MATH-500	GSM8K	MATH-500
Selection	Dense	58.60 / 1.00	86.13 / 1.00	75.20 / 1.00
	HERALD	48.40 / 1.00	78.85 / 1.00	67.40 / 1.00
	MAGE	52.20 / 1.00	85.44 / 1.00	72.80 / 1.00
	H2O	50.80 / 0.62	81.50 / 0.85	66.80 / 0.68
Eviction	H2O+CAOTE	49.60 / 0.61	85.44 / 1.01	67.80 / 0.68
	Expected Attention	55.20 / 2.13	84.91 / 3.03	69.80 / 1.37
	DapQ	51.80 / 0.62	83.85 / 0.86	65.40 / 0.68
Selection & Eviction	MaskAhead, bf16	54.60 / 0.62	84.91 / 0.85	68.00 / 0.68
	Q-MaskAhead	54.00 / 0.25	84.23 / 0.32	67.40 / 0.28

Table 3: LongBench quality / generation-end KV relative to dense bf16. Entries report token F1 / normalized generation-end KV. LLaDA values are supplied point estimates.

		Fast-dLLM-v2-7B			LLaDA 2.0		
Category	Method	HotpotQA	NarrativeQA	MuSiQue	HotpotQA	NarrativeQA	MuSiQue
Selection	Dense	41.31 / 1.000	24.56 / 1.000	17.43 / 1.000	59.82 / 1.000	26.93 / 1.000	40.46 / 1.000
	HERALD	38.02 / 1.000	24.97 / 1.000	17.13 / 1.000	57.17 / 1.000	26.71 / 1.000	37.37 / 1.000
	MAGE	38.14 / 1.000	25.70 / 1.000	16.78 / 1.000	57.17 / 1.000	27.03 / 1.000	39.49 / 1.000
Eviction	H2O	31.91 / 0.130	22.31 / 0.120	13.91 / 0.120	48.32 / 0.130	23.72 / 0.120	32.25 / 0.120
	H2O+CAOTE	31.72 / 0.130	22.00 / 0.116	15.21 / 0.119	48.18 / 0.130	24.53 / 0.120	31.48 / 0.120
	Expected Attention	22.52 / 0.210	21.81 / 0.160	9.28 / 0.170	40.98 / 0.170	21.25 / 0.140	21.21 / 0.150
	DapQ	38.35 / 0.110	24.86 / 0.100	16.74 / 0.100	56.27 / 0.110	27.59 / 0.100	39.04 / 0.100
Selection & Eviction	MaskAhead	38.41 / 0.110	24.94 / 0.100	17.15 / 0.100	55.47 / 0.130	27.59 / 0.100	38.99 / 0.130
	Q-MaskAhead	36.75 / 0.050	23.90 / 0.050	17.11 / 0.050	54.87 / 0.050	27.24 / 0.040	39.17 / 0.050

(Table J.1), consistent with the capacity sweep in Section O and table O.1. The separate HotpotQA retention-floor results appear in Table I.1.

The comparison also separates quality from the ability to fit a cache budget. Expected Attention can score well on reasoning, but its auxiliary state pushes peak KV above dense storage. Conversely, matching quality at a wider budget is not specific to our query source: historical or synthetic estimators can catch up once retention is less restrictive. The capacity sweep localizes the useful regime to genuinely tight caches rather than implying that future masks dominate at every budget (Table J.1 and section O).

The cross-model results support transfer of the mechanism, not a model-independent ranking. MaskAhead is competitive on several Fast-dLLM tasks, while DreamReasoner MATH-500 and some LLaDA task-policy pairs favor other methods. Because DreamReasoner and LLaDA are supplied point estimates, their small differences should be read as task-specific evidence rather than stable wins. The consistent pattern is that cache reduction survives across models, while the quality cost depends on both task and model.

5.2 Which decision needs which evidence?

The factorized comparison in Tables 4 and N.1 prevents attributing the end-to-end result to a single component. Replacing mass with the value-aware score tends to improve the point estimates, but the paired intervals for value/value versus mass/mass include zero on both tasks. We therefore treat the score as a principled output-sensitive ranking rule, while the data do not establish an independent answer-quality gain from value awareness.

The query-source controls provide the sharper mechanistic result: at a fixed score and budget, probing the four-block interval outperforms both current-block-only evidence and DapQ-style pseudo tokens on MATH-500. The random-token control shows that probe length alone does not explain the difference. This supports matching the query population to the retention lifetime; it does not imply that longer probes are always better, since $M = 8$ regresses (Tables G.1 and M.1). Broader horizon and protection controls appear in Sections M and O and table O.2.

Table 4: Independent read and retention scores in bf16. Full-cache rows have no eviction. All evicting rows share the same read and retention budgets within each task: MATH-500 ($n = 500$, $k = 128, C = 256$) and HotpotQA ($n = 200$, $k = 256, C = 512$). MATH peak KV is relative to dense bf16. The score changes without changing the query source or budget.

Read score	Retain score	MATH acc.	Hotpot F1	MATH peak KV
Mass	Full	52.2	38.14	1.000
Value-aware	Full	53.8	37.85	1.000
Mass	Mass	52.0	37.84	0.625
Mass	Value-aware	53.8	38.08	0.624
Value-aware	Mass	53.4	37.91	0.618
Value-aware	Value-aware	54.6	38.41	0.624

Table 5: Measured quality and normalized KV for stored precision on Fast-dLLM-v2-7B. Q-MaskAhead uses four-bit keys and values in ranking and attention.

Selection	KV	Storage	MATH	GSM8K	HotpotQA	MATH peak KV	Hotpot end KV
Mass	bf16	Full	52.2	85.44	38.14	1.000	1.000
Mass	K4V4	Full	52.0	84.61	36.07	0.313	0.313
Value-aware	bf16	Full	53.8	84.91	37.85	1.000	1.000
Value-aware	bf16	Evict	54.6	84.91	38.41	0.624	0.107
Value-aware	K4V4	Evict	54.0	84.23	36.75	0.252	0.050

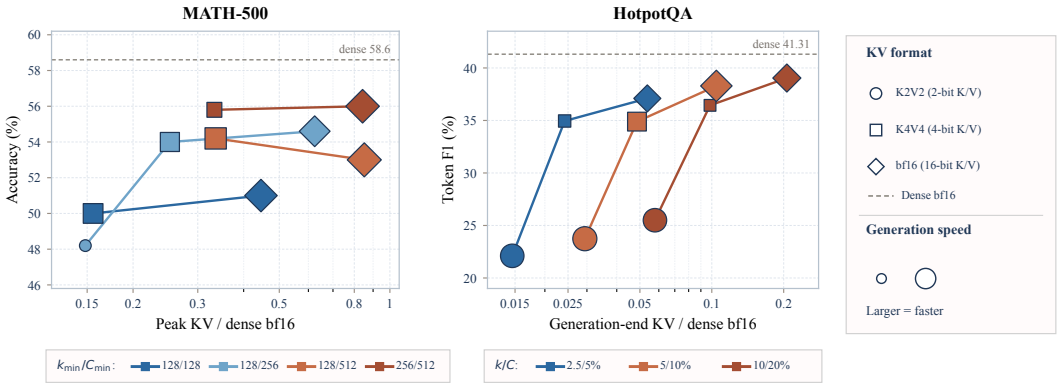

Figure 3: Quality and allocated KV across read-budget, retention-budget, and precision formats.

5.3 Q-MaskAhead trades precision for resident capacity

A packed cache changes both the number of stored entries and their representation. In the selection-only MATH-500 control, K4V4 changes accuracy from 52.2% to 52.0% at 0.313 of dense peak KV. With eviction, K4V4 lowers MATH-500 peak KV from 0.624 to 0.252, while accuracy changes from 54.6% to 54.0%. On HotpotQA, it lowers generation-end KV from 0.107 to 0.050, while F1 changes from 38.41 to 36.75. Rotating and centering keys recovers 2.59 F1 points at the same four-bit budget (paired 95% CI $[0.12, 5.14]$ over 200 examples), leaving a 1.08-point gap to bf16. In the matched ranking diagnostic, bf16 and K4V4 select sets with 97.44% overlap and retain sets with 97.65% overlap. This is near-stable ranking in the measured sample, not lossless answer quality; task-level results and the precision grid are in Sections E and Q and table E.1.

The quality-memory envelope. Figure 3 varies read budget, retention capacity, and stored precision one at a time. On MATH-500, raising the read floor from $k = 128$ to $k = 512$ changes accuracy from 54.6% to 58.4% while peak KV remains near 0.62 of dense bf16. On HotpotQA, increasing the read fraction from 2.5% to 10% at $C = 20\%$ raises F1 from 36.85 to 39.04 without changing the 0.206 generation-end KV. These are distinct ways to spend compute, entries, and bits. Full read and retention sweeps, including settings that do not improve quality monotonically, are in Section P and tables H.1, P.1 and P.2.

5.4 Long prompts: prefill and retrieval are separate decisions

The standard policy waits until the question is known to evict. On HotpotQA it reaches 38.41 F1 with 0.107 generation-end KV, but peaks at 0.999 during prefill. A question-conditioned prefill probe lowers that peak to 0.144 at 37.16 F1. Eviction by fixed 512-token chunks reaches 26.90 F1 at 0.185 peak KV, showing why retention must account for the question rather than only prompt order (Table 6). For retrieval, our primary check is the official passage task ($n = 200$): MaskAhead scores 82.0% at 0.1042 end KV, DapQ scores 82.5% at the same memory, and Q-MaskAhead scores 75.5% at 0.0503 (Table F.1).

Table 6: HotpotQA prefill policies ($n = 200$, bf16). KV includes method state and is normalized to dense bf16; only the standard policy waits until after prefill to evict.

Policy	F1	Peak KV	End KV
Standard MaskAhead	38.41	0.999	0.107
Question probe	37.16	0.144	0.107
Chunk eviction	26.90	0.185	0.107

We also include a synthetic needle-in-a-haystack diagnostic, where a target is placed in a longer context and the response must recover it. Its first setting saturates at 75/75 for dense, MAGE, DapQ, and MaskAhead, so it cannot distinguish them. A harder 30-placement pilot gives 17/30 for MaskAhead and 19/30 for dense. Section R states the tested budgets and outcomes; the official task carries the retrieval comparison.

Batch-size scaling. The batch sweep reports generation-end KV and total batch time (Table 7). Its useful signal is the memory headroom: Q-MaskAhead stores less KV as batch size grows, while measured batch time is similar to MaskAhead at larger tested batches. This is not yet a throughput claim because the sweep uses one seed-42 prompt mixture, excludes unreliable full-device allocation, and marks the largest batches as forecasts. Measurements through $B = 32$ are preliminary, not medians over three mixtures.

Table 7: Generation-end KV cache (GiB) / total batch time (s).

B	Dense bf16	MaskAhead bf16	Q-MaskAhead
1	0.723 / 4.8	∼0.13 / ∼5.7	0.069 / 5.6
2	1.774 / 11.5	∼0.27 / ∼9.3	0.177 / 8.6
4	3.548 / 16.4	∼0.53 / ∼16.5	0.349 / 15.4
8	7.096 / 36.7	1.061 / 31.3	0.711 / 31.5
16	14.191 / 73.4	2.122 / 62.8	1.422 / 63.4
32	28.383 / 147.4	4.243 / 124.0	2.895 / 124.4
64	OOM	8.487 / 242	5.812 / 245

6 Conclusion

MaskAhead turns the query availability of block diffusion into two horizon-matched cache decisions: current-block masks select temporary reads, while a no-write probe of future blocks guides irreversible retention. In the tested restrictive regime, this combination creates a smaller-cache operating point that preserves much of the full-cache quality on reasoning and long-context QA. Four-bit storage extends the memory reduction, but its quality cost is task-dependent and becomes pronounced for standard long-context QA.

The evidence also identifies where the method helps. Matched controls favor future-block masks over current-only and pseudo-token probes, while value-aware scoring has favorable point estimates without statistically established answer-quality gains. At looser retention budgets, competing policies catch up. Thus the contribution is decision-aligned evidence for recurrent eviction, with the clearest benefit when storage is tight, rather than a universal quality improvement.

Limitations. Our evaluation focuses on block diffusion models and the cache budgets, tasks, and hardware considered here. Future-mask probes use the available prefix and cannot condition on tokens finalized in intervening blocks, so their retention signal is an estimate of future cache use. Memory savings also depend on when eviction occurs: standard retention reduces generation-end KV, while question-conditioned prefill is needed to reduce the prompt-time peak. Finally, the quality cost of compression depends on the task and storage format, with four-bit KV offering the strongest memory reduction and larger losses on some long-context QA settings.

AI use statement

Generative AI assisted with prose polishing, repository cleanup, small author-reviewed code changes, and Python scripts for figures. Authors verified the methods, analyses, measurements, and final text.

References

Jinwoo Ahn, Ingyu Seong, Akhil Kedia, Junhan Kim, Hyemi Jang, Kangwook Lee, and Yongkweon Jeon. LookaheadKV: Fast and accurate KV cache eviction by glimpsing into the future without generation. In *International Conference on Learning Representations*, 2026. arXiv:2603.10899.

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.

Tiwei Bie, Maosong Cao, Kun Chen, Lun Du, Mingliang Gong, Zhuochen Gong, Yanmei Gu, Jiaqi Hu, Zenan Huang, Zhenzhong Lan, et al. Llada 2.0: Scaling up diffusion language models to 100b, 2025.

Zhuokun Chen, Jianfei Cai, and Bohan Zhuang. Flashblock: Attention caching for efficient long-context block diffusion, 2026.

Alessio Devoto, Maximilian Jeblick, and Simon Jégou. Expected attention: KV cache compression by estimating attention from future queries distribution, 2025.

Raghavv Goel, Junyoung Park, Mukul Gagrani, Dalton Jones, Matthew Morse, Harper Langston, Mingu Lee, and Chris Lott. CAOTE: KV cache selection for LLMs via attention output error-based token eviction, 2025.

Coleman Hooper, Sehoon Kim, Hiva Mohammadzadeh, Michael W. Mahoney, Yakun Sophia Shao, Kurt Keutzer, and Amir Gholami. Kvquant: Towards 10 million context length llm inference with kv cache quantization. In *Advances in Neural Information Processing Systems*, 2024. arXiv:2401.18079.

Jianuo Huang, Yaojie Zhang, Yicun Yang, Benhao Huang, and Linfeng Zhang. Mask tokens as prophet: Fine-grained cache eviction for efficient dLLM inference. In *Findings of the Association for Computational Linguistics: ACL 2026*, pp. 3456–3479, San Diego, California, United States, 2026. Association for Computational Linguistics. doi: 10.18653/v1/2026.findings-acl.170. URL https://aclanthology.org/2026.findings-acl.170/. arXiv:2510.09309.

Wonjun Kang, Kevin Galim, Seunghyuk Oh, Minjae Lee, Yuchen Zeng, Shuibai Zhang, Coleman Hooper, Yuezhou Hu, Hyung Il Koo, Nam Ik Cho, and Kangwook Lee. Parallelbench: Understanding the trade-offs of parallel decoding in diffusion llms. In *International Conference on Learning Representations*, 2026. arXiv:2510.04767.

Omin Kwon, Doyeon Kim, Jongseok Park, Seung Yul Lee, Ion Stoica, and Jae W. Lee. Herald: High-throughput block diffusion llm serving via cpu-gpu cooperative kv cache retrieval, 2026a.

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, 2026b.

Zirui Liu, Jiayi Yuan, Hongye Jin, Shaochen Zhong, Zhaozhuo Xu, Vladimir Braverman, Beidi Chen, and Xia Hu. Kivi: A tuning-free asymmetric 2bit quantization for kv cache. In *International Conference on Machine Learning*, 2024. arXiv:2402.02750.

Aaron Lou, Chenlin Meng, and Stefano Ermon. Discrete diffusion modeling by estimating the ratios of the data distribution. In *International Conference on Machine Learning*, 2024.

Binglei Lou, Haoran Wu, Yao Lai, Jiayi Nie, Can Xiao, Xuan Guo, Rika Antonova, Robert Mullins, and Aaron Zhao. Beyond gemm-centric npus: Enabling efficient diffusion llm sampling. *arXiv e-prints*, pp. arXiv–2601, 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.

Subham Sekhar Sahoo, Marianne Arriola, Yair Schiff, Aaron Gokaslan, Edgar Marroquin, Justin T. Chiu, Alexander Rush, and Volodymyr Kuleshov. Simple and effective masked diffusion language models. In *Advances in Neural Information Processing Systems*, 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. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 40, pp. 33038–33046, 2026.

Zhenxu Tian, Yi Su, Juntao Li, and Min Zhang. Where matters more than what: Decoding-aligned KV cache compression via position-aware pseudo queries, 2026.

Jinhao Wang, Zhexin Hu, Kangjie Zhou, Xin Zhou, and Fangfang Liu. GroupKV: Hierarchical KV cache management for long-context diffusion LLM inference, 2026a. URL https://arxiv.org/abs/2609.17573.

Yixuan Wang, Shiyu Ji, Yijun Liu, Yuzhuang Xu, Yang Xu, Qingfu Zhu, and Wanxiang Che. Lookahead Q-cache: Achieving more consistent KV cache eviction via pseudo query. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 34158–34174, 2025.

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. arXiv:2509.24014.

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*, volume 2026, pp. 128353–128370, 2026a.

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. arXiv:2505.22618.

Guangxuan Xiao, Yuandong Tian, Beidi Chen, Song Han, and Mike Lewis. Efficient streaming language models with attention sinks. In *International Conference on Learning Representations*, 2024. arXiv:2309.17453.

June Yong Yang, Byeongwook Kim, Jeongin Bae, Beomseok Kwon, Gunho Park, Eunho Yang, Se Jung Kwon, and Dongsoo Lee. No token left behind: Reliable kv cache compression via importance-aware mixed precision quantization, 2024.

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

Jiebin Zhang, Dawei Zhu, Yifan Song, Wenhao Wu, Chuqiao Kuang, Xiaoguang Li, Lifeng Shang, Qun Liu, and Sujian Li. More tokens, lower precision: Towards the optimal token-precision trade-off in kv cache compression. In *Findings of the Association for Computational Linguistics: EMNLP*, 2025. arXiv:2412.12706.

Junkai Zhang, Hang Guo, Luca Benini, and Yawei Li. Rdkv: Rate-distortion bit allocation for joint eviction and quantization of the kv cache, 2026.

Zhenyu Zhang, Ying Sheng, Tianyi Zhou, Tianlong Chen, Lianmin Zheng, Ruisi Cai, Zhao Song, Yuandong Tian, Christopher Ré, Clark Barrett, Zhangyang Wang, and Beidi Chen. H2o: Heavy-hitter oracle for efficient generative inference of large language models. In *Advances in Neural Information Processing Systems*, 2023.

Zhenyu Zhang, Shiwei Liu, Runjin Chen, Bhavya Kailkhura, Beidi Chen, and Zhangyang Wang. Q-hitter: A better token oracle for efficient LLM inference via sparse-quantized KV cache. In *Proceedings of Machine Learning and Systems*, volume 6, pp. 381–394, 2024.

A Traffic and Capacity Accounting

Let T be denoising steps per block, N live prefix entries, and k the temporary read budget. Express probe and compaction work per checkpoint as Q_{probe} and Q_{compact} in equivalent prefix-entry reads. Dense attention reads approximately TN entries per block. MaskAhead reads $N + (T-1)k$ for block attention plus $(Q_{\text{probe}} + Q_{\text{compact}})/E$ amortized over checkpoints. This simplified traffic decreases when

$$(T-1)(N-k) > \frac{Q_{\text{probe}} + Q_{\text{compact}}}{E}. \tag{5}$$

The model omits kernel launches, scheduling, and parallel requests; measured wall-clock time is the systems metric.

For device memory M_{device}, model weights M_{weights}, and non-KV workspace M_{work}, an approximate KV-limited batch is

$$b_{\max} \approx \left\lfloor \frac{M_{\text{device}} - M_{\text{weights}} - M_{\text{work}}}{M_{\text{KV,request}}} \right\rfloor. \tag{6}$$

The request term depends on serving phase. A reduction in generation-end KV need not increase a batch constrained by simultaneous prefill peaks.

B Reproducibility and Ethics

Reproducibility statement

The decision timing, ranking rule, cache representation, and block cycle are specified in Sections 3 and 4 and algorithm 1. Section 5 gives the model, accelerator, datasets, sample counts, decoding rule, and per-task cache budgets. Baselines use the same inference runtime and hardware within each comparison. Each reported cell is a single greedy run, and the tables distinguish peak cache memory from memory at the end of generation.

C Wider-Budget Interpretation

The fixed-budget comparison above isolates the restrictive $k = 128$, $C = 256$ regime. We also repeat complete Fast-dLLM-v2-7B runs with $k_t = \min(N_t, \max(256, \lceil 0.10 N_t \rceil))$ and $C_t = \max(512, \lfloor 0.20 N_t^{\text{seen}} \rfloor)$. All rows in Table J.1 use the same task examples and common inference harness. On MATH-500, CAOTE-style historical retention reaches 57.6%, DapQ and Expected Attention each reach 57.2%, H2O reaches 56.4%, and MaskAhead reaches 56.2%. Expected Attention uses 2.346 times the dense bf16 peak KV on this task because its Gaussian state dominates the retained cache. The advantage of future-mask retention in Table 2 therefore does not extend to this wider reasoning budget. On HotpotQA, DapQ and MaskAhead differ by only 0.06 F1, while both exceed H2O by at least 5.95 F1. On MuSiQue, H2O is strongest among these evicting rows. Q-MaskAhead scores 55.4% on MATH-500 and 36.44 F1 on HotpotQA at the wider budget. GSM8K is less discriminating here: H2O, DapQ, and bf16 MaskAhead each score 85.44%, while Q-MaskAhead scores 86.35% across all 1,319 examples. These are single greedy runs, so small differences are descriptive rather than evidence of a reliable ordering.

D Long-Prompt Retrieval and Prefill Results

Generation-end KV hides a separate full-prompt peak. The standard policy waits until response generation to evict, so HotpotQA reaches 38.41 F1 at $\mathbf{0.107}$ end KV but still reaches 0.999 peak KV during prefill. Early 512-token chunk eviction reduces that peak to 0.185 and lowers F1 to 26.90. A question-conditioned prefill probe preserves more information: 37.16 F1 at 0.144 peak and $\mathbf{0.107}$ end KV. The question-conditioned policy also reaches 24.10 F1 at 0.125 peak KV on NarrativeQA and 15.35 F1 at 0.129 on MuSiQue. It is a distinct operating point for long-prompt serving, while standard MaskAhead primarily reduces memory after prefill.

Figure 4 summarizes the HotpotQA endpoints over 200 examples. These measurements show why generation-end KV alone understates standard prefill memory: its peak is 0.999 of dense bf16 even though the generation-end cache is 0.107. The figure reports endpoint KV ratios, not a chunk-by-chunk memory trace or total device allocation; synchronized lifecycle measurements remain necessary to characterize transient workspace and the location of the peak.

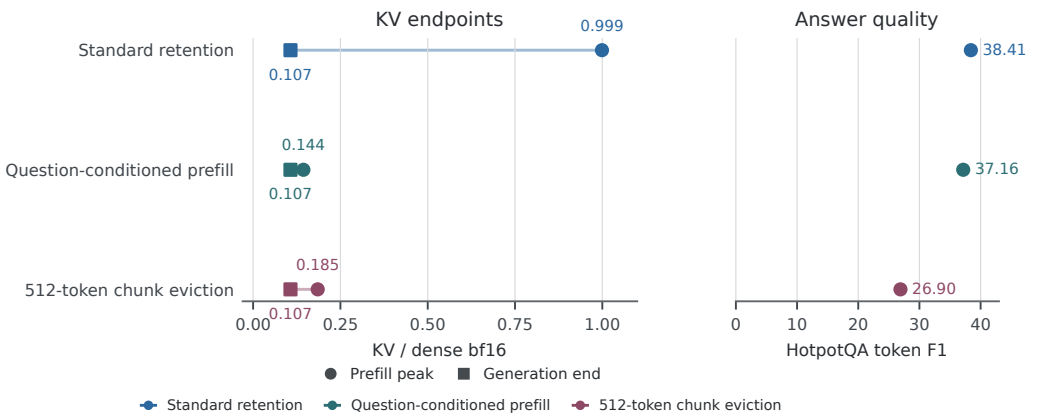

Figure 4: HotpotQA quality with prefill-peak and generation-end KV.

Official passage retrieval gives a more discriminating long-context check than the synthetic needle pilot, on which dense and every tested MASKAHEAD budget solve all held-out cases. On 200 retrieval examples, DapQ scores 82.5% and bf16 MASKAHEAD scores 82.0% at the same 0.1042 end KV. Full-cache MAGE reaches 83.5%. Q-MaskAhead reaches 75.5% at 0.0503 end KV. This task exposes a narrow matched-memory gap to DapQ that the saturated synthetic test cannot reveal.

E REASONING PRECISION GRID

The precision sweep shows that four-bit storage is a favorable point on the tested memory-quality frontier, while reducing key precision further is more damaging than reducing value precision at comparable cache size. The smallest cache is not the best quality-memory choice. Table E.1 quantifies this boundary.

Table E.1: MATH-500 KV precision under the same retention policy. Peak KV is relative to dense bf16.

KV format	Acc.	Peak KV
bf16	**54.6**	0.624
K4V4	54.0	0.252
K4V2	53.4	0.203
K2V4	46.0	0.193
K2V2	48.2	**0.148**

F OFFICIAL PASSAGE RETRIEVAL

At matched generation-end memory, MaskAhead is close to DapQ, while four-bit storage roughly halves the cache again with a measurable accuracy cost. The synthetic needle task saturates for stronger policies, so this official passage benchmark is the more informative retrieval comparison (Table F.1).

Table F.1: Official passage retrieval, $n = 200$. Accuracy is percent. Peak and end KV are relative to dense bf16. Bold and underlining rank compressed methods, excluding dense and selection-only MAGE.

Policy	Acc.	Peak KV	End KV
Dense	83.0	1.0000	1.0000
MAGE	83.5	1.0000	1.0000
H2O	66.0	0.9999	0.1225
H2O+CAOTE	68.0	0.9999	0.1225
Expected Attention	21.5	1.0702	0.1929
DapQ	**82.5**	0.9995	0.1042
MASKAHEAD, bf16	82.0	0.9995	0.1042
Q-MASKAHEAD	75.5	**0.3137**	**0.0503**

G MAIN-BUDGET PROBE CONTROLS

The horizon sweep is non-monotonic: the decision-matched four-block probe performs best, while extending the probe to eight blocks does not help. This supports matching probe horizon to cache lifetime rather than maximizing look-ahead (Table G.1).

Table G.1: MATH-500 probe controls with the same read and retention budgets. Peak KV is relative to dense bf16.

Probe	Acc.	Peak KV
$M = 1$	52.0	**0.625**
$M = 2$	54.4	**0.625**
$M = 4$	**54.6**	0.624
$M = 8$	53.2	0.621
Random masks	53.6	0.625
DapQ-filled	51.2	**0.625**

H FOUR-BIT RETENTION FLOORS

Under K4V4, raising the retention floor from 128 to 256 entries recovers substantial reasoning quality, but larger floors add memory without a corresponding score increase. The four-bit operating point is therefore budget-sensitive (Table H.1).

Table H.1: K4V4 MATH-500 retention floors at fixed $k = 128$. Peak KV is relative to dense bf16.

C floor	Acc.	Peak KV
128	50.0	**0.156**
256	54.0	0.252
512	**54.2**	0.336
1024	54.0	0.338

I HOTPOTQA RETENTION FLOORS

For HotpotQA, the future-query policy maintains the same observed F1 as the retention floor changes from 5% to 10%, making the smaller floor the more memory-efficient measured point. This contrasts with H2O, where increasing the floor improves quality but also raises end-cache size (Table I.1).

Table I.1: Fast-dLLM-v2-7B HotpotQA, $n = 200$, at the same read budget. Memory is generation-end KV relative to dense bf16. Peak prefill KV stays near one.

Policy	C (%)	F1	End KV
H2O	5	30.62	0.085
H2O	25	33.43	0.276
DapQ	10	38.35	0.107
MASKAHEAD	5	**38.41**	**0.061**
MASKAHEAD	10	**38.41**	0.107

J WIDER-BUDGET QUALITY TABLE

The wider budget changes which estimator is most useful: historical CAOTE-style retention leads on MATH-500, while MaskAhead and DapQ remain close on HotpotQA. This confirms that the restrictive-budget result is a regime-specific advantage, not a fixed ordering across capacities (Table J.1).

K RECORDED DECISION-HORIZON COVERAGE

On a full-cache MATH-500 recording, we compare the block-shared read set against a same-budget offline top-k set chosen from each later step's realized queries. The sample starts with 30 examples, with fewer reaching later steps. Table K.1 reports selected denoising steps. The offline set is an attention-coverage reference and was not used to generate answers.

Table J.1: Fast-dLLM-v2-7B at the wider $k = 10\%$, $C = 20\%$ budget, with floors of 256 and 512 entries. MATH-500 reports accuracy over 500 examples. The three QA tasks report token F1 over 200 examples each. Baselines are common-harness adaptations. This table reports quality only because prefill and generation have different KV memory peaks. Bold and underlining mark the best and second-best completed values per task.

Policy	MATH-500	HotpotQA	NarrativeQA	MuSiQue
H2O, bf16	56.4	33.09	24.36	**18.02**
H2O+CAOTE, bf16	**57.6**	34.21	24.23	16.82
DapQ, bf16	<u>57.2</u>	**39.10**	<u>24.43</u>	<u>17.17</u>
Expected Attention, bf16	<u>57.2</u>	24.94	23.79	11.99
MASKAHEAD, bf16	56.2	<u>39.04</u>	**24.97**	16.89
Q-MASKAHEAD	55.4	36.44	24.32	16.92

Table K.1: Realized prefix-attention mass retained by a block-shared read set and a same-budget step-specific offline top-k set on MATH-500. The final column counts examples reaching that denoising step. Steps have different sample sizes, so cross-row ordinal highlighting is not meaningful.

Step	Shared (%)	Offline top-k (%)	Examples
0	98.33	98.37	30
1	98.22	98.68	30
4	97.91	98.54	30
8	97.75	98.50	29
12	97.34	98.24	24
16	97.34	98.45	21

The step-specific set retains slightly more attention mass throughout the recorded trajectory. This is expected because it is selected offline from queries observed at each target step, while the shared set is fixed at block start. The set-overlap decline across blocks (47.2% at offset four) also reflects an expanding candidate prefix. Neither measurement compares answer quality. Quantized-ranking overlap is reported separately in Section Q.

L WIDER-BUDGET RESULT PROVENANCE

In Table J.1 every displayed MATH-500 row has 500 examples and every displayed QA row has 200. The reported budget is determined by the saved configuration: read floor 256, read fraction 10% of live entries, retention floor 512, retention fraction 20% of all seen positions, and eviction every four blocks. H2O, CAOTE-style retention, DapQ, and MASKAHEAD use the same bf16 cache representation except for the separately labeled Q-MaskAhead row.

M PROBE HORIZON AND QUERY SOURCE

The probe is useful only if its queries match the decision interval. We hold k, C, the value-aware score, and the four-block eviction interval fixed while changing the number of probed blocks M. Table M.1 also replaces future masks with random tokens or DapQ-style pseudo tokens at the same positions. MATH-500 scores peak at $M = 4$. Its 54.6% accuracy exceeds $M = 1$ by 2.6 points and pseudo-token filling by 3.4 points. The corresponding HotpotQA differences are smaller. Its highest F1 in this ablation is also the four-block mask probe.

The MATH-500 comparison between masks and DapQ-filled probe blocks has 28 wins and 11 losses for masks in the paired run, with a two-sided sign-test $p = 0.0095$. Filling the probe with random tokens yields 53.6%, so the gain cannot be assigned to probe length alone. On HotpotQA, all six variants fall within 0.55 F1 points on the 32-token output protocol.

N INDEPENDENT READ AND RETENTION DECISIONS

The read ranking and the retention ranking can use attention mass or the value-aware score independently. Table N.1 extends the MATH-500 factorization in the main text to HotpotQA. Among the

Table M.1: Probe ablation with the main read and retention budgets. All rows use Fast-dLLM-v2-7B and the same value-aware ranking rule. MATH-500 has 500 examples and HotpotQA has 200. Bold and underlining rank each task column.

Probe	MATH-500 acc. (%)	HotpotQA F1
$M = 1$	52.0	38.26
$M = 2$	<u>54.4</u>	37.86
$M = 4$	**54.6**	**38.41**
$M = 8$	53.2	<u>38.29</u>
Four blocks, random tokens	53.6	<u>38.29</u>
Four blocks, DapQ tokens	51.2	38.01

evicting policies, value-aware ranking for both decisions reaches the highest observed task score on each benchmark. The full value/value contrast over mass/mass is $+2.6$ MATH-500 points (paired 95% bootstrap CI $[-0.6, 5.8]$) and $+0.568$ HotpotQA F1 (CI $[-0.008, 1.348]$). Both intervals include zero, so the measured ordering is descriptive rather than statistically established. Figure 5 shows all four paired comparisons; bootstrap intervals are formed from matched example IDs and plotted relative to mass/mass. Full-cache selection-only rows are references and are excluded from the ordinal markings.

Table N.1: Read and retention scores at the main budget. "Full" retains the entire cache. The four evicting rows have matched budgets and are ranked within each task column. MATH-500 has 500 examples and HotpotQA has 200.

Read score	Retention score	MATH acc. (%)	Hotpot F1
Mass	Full	52.2	38.14
Value-aware	Full	53.8	37.85
Mass	Mass	52.0	37.84
Mass	Value-aware	<u>53.8</u>	<u>38.08</u>
Value-aware	Mass	53.4	37.91
Value-aware	Value-aware	**54.6**	**38.41**

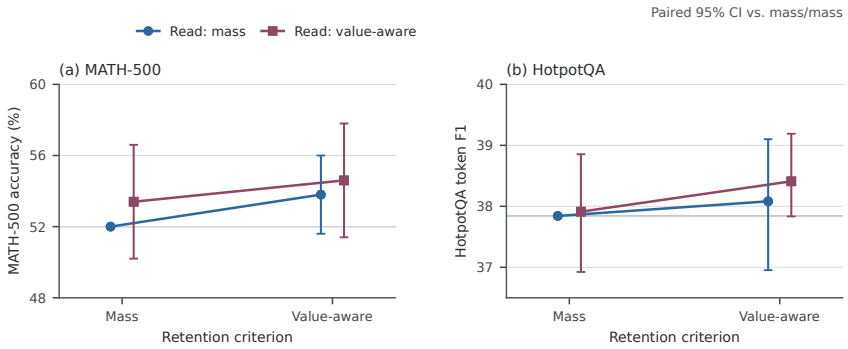

Figure 5: Read and retention score interaction on MATH-500 and HotpotQA.

The two scores use the same value-aware formula with different query populations. The result does not isolate value-awareness from lookahead on its own. The probe controls in Table M.1 make that second comparison while holding the scoring rule fixed.

O Capacity, Recency, and Eviction Timing

A tight cache exposes the difference between historical and future-mask retention. At fixed $k = 128$, Table O.1 compares MaskAhead and H2O across four retention floors on the same 500 MATH-500 examples. The paired gain is 11.6 points at $C = 128$ and 3.8 points at $C = 256$. At $C = 512$ and $C = 1024$, the measured difference is 0.2 points. These are distinct memory operating points because an eviction every four blocks allows temporary growth between decisions.

The capacity sweep defines where the method matters most. At the lowest floor, the 95% paired bootstrap interval for the 11.6-point gain is $[7.8, 15.4]$ points. At $C = 512$ the 0.2-point difference

Table O.1: MATH-500 capacity sweep at fixed $k = 128$, bf16. Peak KV is relative to dense bf16 and includes method state. Bold and underlining compare the two policies at each capacity. The paired difference uses identical 500 example IDs.

C floor	MaskAhead		H2O	
	Acc. (%)	Peak KV	Acc. (%)	Peak KV
128	**51.0**	0.446	39.4	**0.429**
256	**54.6**	0.624	50.8	**0.617**
512	**53.0**	**0.852**	52.8	0.853
1024	**53.8**	**0.982**	53.6	0.984

has interval $[-1.6, 1.8]$ points. The relative advantage is therefore concentrated where retention is irreversible and genuinely restrictive.

Table O.2 tests whether the result depends on protecting sinks and recent entries or on the four-block eviction interval. The default $W = 64$, four sinks, and $E = 4$ scores 54.6% on MATH-500. Evicting every two blocks gives a smaller cache peak but lowers accuracy to 51.8%. Evicting every eight blocks scores 53.8% with a larger peak. The HotpotQA scores vary by at most 0.12 F1 among the completed rows in this table.

Table O.2: Timing and protection ablations at the main read and retention budgets. All completed MATH-500 rows have 500 examples, and completed HotpotQA rows have 200. A dash indicates an unrun cross-task arm. MATH peak KV includes method state. Bold and underlining rank comparable rows within each metric.

Policy	MATH acc. (%)	MATH peak KV	Hotpot F1
Default: sinks 4, $W = 64$, $E = 4$	**54.6**	0.624	**38.41**
No protected sinks	53.0	0.615	38.35
$W = 32$	54.0	0.621	**38.41**
$W = 128$	53.6	0.618	–
$E = 2$	51.8	**0.543**	–
$E = 8$	53.8	0.718	38.29

P Long-Context Budget Series

The main long-context result uses a floor of 256 reads and 512 retained entries. The separate budget series below uses $k = C/2$ as a fraction of seen tokens, with floors of 32 reads and 64 retained entries. This isolates budget scaling without changing the inference harness. Table P.1 compares full 200-example runs on each task. MaskAhead reaches the highest NarrativeQA F1 at all three capacities and the highest HotpotQA F1 at $C = 10\%$. DapQ leads HotpotQA at $C = 5\%$ and $C = 20\%$, while H2O leads MuSiQue at $C = 20\%$. The separate wider-budget table uses larger read and retention floors, so its results should not be pooled with this series.

Increasing the read fraction at fixed $C = 20\%$ changes quality without changing the retained-cache size. On HotpotQA, MaskAhead scores 36.85, 37.89, and 39.04 F1 at read fractions 2.5%, 5%, and 10%. On NarrativeQA the corresponding values are 25.50, 24.96, and 24.78, and on MuSiQue they are 16.74, 17.06, and 16.89. The task-specific ordering shows why a single read fraction cannot be inferred from one QA benchmark. Table P.2 gives the exact values for both storage formats.

Q Ranking Stability under Four-Bit Storage

In a recording-only matched diagnostic, we recomputed the same selection and eviction scores from bf16 and K4V4 representations on 30 MATH-500 and 30 HotpotQA examples. The selected top-k sets overlap by 97.44% on average across 60 examples and 64,896 layer/head events; an example-level bootstrap interval is [97.12, 97.77]%. The retention top-C sets overlap by 97.65% across 48 examples with an eviction event and 9,968 layer/head events, with interval [97.42, 97.90]%. These set overlaps show that packed ranking is mostly stable. They do not imply identical attention outputs or answer quality: the HotpotQA quality change in Section 5.3 remains visible despite high ranking overlap.

Table P.1: LongBench budget series on Fast-dLLM-v2-7B, $n = 200$ per task. C is a fraction of seen tokens and $k = C/2$. Each entry is F1 followed by generation-end KV relative to dense bf16. Bold and underlining rank the three policies separately within each C group and task, with higher F1 and lower KV preferred. Standard prefill peak remains near one.

		HotpotQA		NarrativeQA		MuSiQue	
C (%)	Policy	F1	End KV	F1	End KV	F1	End KV
5	H2O	29.71	0.077	22.28	0.066	14.41	0.069
5	DapQ	**37.31**	**0.054**	24.89	**0.053**	**16.38**	**0.053**
5	MASKAHEAD	37.10	**0.054**	**25.19**	**0.053**	15.99	**0.053**
10	H2O	32.98	0.127	22.84	0.116	14.17	0.119
10	DapQ	38.22	**0.105**	24.81	**0.104**	**17.11**	**0.104**
10	MASKAHEAD	**38.29**	**0.105**	**25.02**	**0.104**	16.74	**0.104**
20	H2O	34.17	0.226	23.59	0.216	**17.19**	0.219
20	DapQ	**39.10**	**0.206**	24.47	**0.206**	17.17	**0.206**
20	MASKAHEAD	39.04	**0.206**	**24.78**	**0.206**	16.89	**0.206**

Table P.2: Read-budget sweep at $C = 20\%$ on 200 examples per task. The retention budget and generation-end KV remain fixed within each storage format. Bold and underlining rank read fractions separately for bf16 and K4V4 in each task column.

KV	k (%)	HotpotQA F1	NarrativeQA F1	MuSiQue F1
bf16	2.5	36.85	**25.50**	16.74
bf16	5	37.89	24.96	**17.06**
bf16	10	**39.04**	24.78	16.89
K4V4	2.5	33.98	24.23	**16.94**
K4V4	5	35.43	23.68	16.83
K4V4	10	**36.44**	**24.39**	16.92

R SYNTHETIC RETRIEVAL STRESS TEST

The five-seed synthetic needle test contains 15 placements per seed. Dense, MAGE, HERALD, DapQ, and MASKAHEAD solve all 75 placements, including MASKAHEAD with $C = 5\%$ and Q-MaskAhead at $C = 10\%$. Thus this test is saturated for the strongest policies and does not support a ranking among them. H2O solves 16/75 and H2O+CAOTE 16/75 at $C = 10\%$; Expected Attention solves 15/75. The capacity choice $C = 10\%$ was made using two separate development seeds, whose 30/30 scores tied across 5, 10, and 20%, with the main-paper budget breaking the tie.

A harder pilot places one needle at five depths in each of 8K, 16K, and 28K contexts for two seeds ($n = 30$ seeded placements). Dense solves 19/30; MASKAHEAD solves 17/30 at both $C = 5\%$ and $C = 20\%$, and Q-MaskAhead at $C = 20\%$ also solves 17/30. H2O at $C = 20\%$ solves 12/30. The pilot is too small and too synthetic to establish a general retrieval advantage, but it rules out interpreting the 75/75 saturated test as a ceiling independent of context difficulty. Official passage retrieval in Table F.1 remains the primary retrieval benchmark.

S HARDWARE AND RESULT PROVENANCE

Fast-dLLM-v2-7B quality and the batch-16 serving pilot were run on one Alibaba PPU. DreamReasoner-8B scores were supplied from a separate B200 evaluation. The corresponding raw DreamReasoner logs were unavailable in the provided archive, so those scores are reported as a separate preliminary check.

T VARIED-PROMPT SERVING CAPACITY

The matched serving study uses one 96-GiB PPU and generates 256 tokens per request without early stopping. It evaluates 8K, 12K, and 16K HotpotQA prompts from pools of 173, 136, and 66 examples; larger batches cycle through the pool. Capacity boundaries at 8K and 12K are resolved within eight requests. At 16K, MaskAhead-Q completed a batch of 280; its OOM boundary is still under test. Throughput and memory are compared at common safe batch sizes of 64, 48, and 32 requests. The figure also reports each policy's largest tested safe batch; the 16K MaskAhead-Q point

is a lower bound, not a measured maximum. Throughput includes prefill, and requests/s is output tokens/s divided by 256. In separate 200-example quality runs, MaskAhead-Q scores 37.16 token F1 versus 38.41 for standard MaskAhead. Plain chunk-eviction prefill scores 26.90 F1 and is not a like-quality serving comparison. The throughput and memory panels provide a complementary long-prompt serving result to the fixed-batch decode pilot in the main text.

U Limits of Representative-Query Offloading

HERALD makes host-side selection practical by replacing the B queries of a fully masked block with one query at its center (Kwon et al., 2026a). This approximation reduces CPU work by a factor proportional to B, but weakens the selected set. HERALD reports that the representative query recovers about 84% of the all-query top-k and enlarges the cache budget by roughly $1.7\times$ to recover the resulting quality loss. The cost is most visible at tight budgets, where ranking errors cannot be absorbed by retaining additional entries.

Table U.1 isolates query population and scanned-key precision in the selection-only harness of the earlier paper version: Fast-dLLM-v2-7B on GSM8K, $n = 200$, $k = 64$, with no eviction. Scores are raw exact match and cannot be compared numerically with the current normalized-answer MATH-500 and GSM8K results. Values and executed attention remain bf16, so key width changes selection alone. At bf16 the center query loses 27.5 exact-match points relative to all $B = 32$ queries; five equispaced queries recover 17 points. At all B queries, four-bit scanned keys lose 0.5 points relative to bf16, while the two-bit row loses 8.5 points. The reported coverage is selected top-k overlap with the dense-attention reference, not answer accuracy or retention quality.

Reading the interaction. The heatmaps summarize an earlier selection-only run (Fast-dLLM-v2-7B, GSM8K, $n = 200$, $k = 64$; no eviction). Four-bit scanned keys preserve the all-query score closely (84.0 versus 84.5 at bf16), while reducing the query population has a much larger effect: one center query scores 57.0. This supports the value of broad query coverage in this harness, but does not establish a four-bit result for recurrent eviction. The cells are descriptive single-run values; the measured query settings are given in Table U.1.

Table U.1: Earlier selection-only GSM8K harness ($n = 200$, $k = 64$, no eviction): raw exact match (dense top-k overlap). Keys are quantized only for ranking; executed attention and values remain bf16. This does not test recurrent eviction.

Step-0 queries	bf16 keys	4-bit	3-bit	2-bit
1 random	50.5 (0.878)	43.5 (0.876)	40.5 (0.874)	40.0 (0.868)
1 center	57.0 (0.889)	48.5 (0.886)	43.0 (0.884)	41.5 (0.875)
5 equispaced	74.0 (0.907)	72.5 (0.906)	71.0 (0.903)	69.5 (0.894)
all $B = 32$	84.5 (0.919)	84.0 (0.916)	78.5 (0.915)	76.0 (0.909)

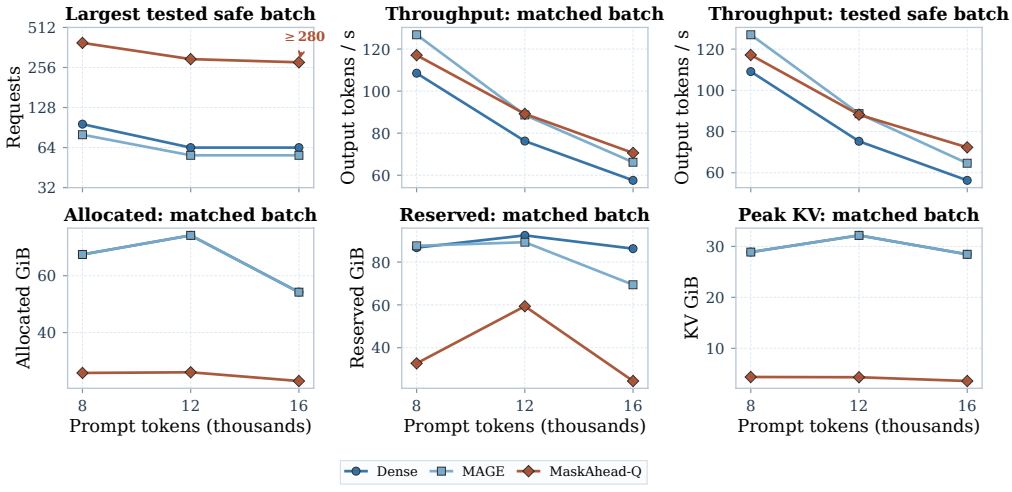

Figure 6: Batch capacity, throughput, and memory for varied HotpotQA requests.

HERALD's latency hiding also depends directly on the number of denoising steps T: one host-side retrieval must finish while the GPU executes the current block. Its reported ablation shows that at $T = 8$ retrieval becomes the bottleneck and HERALD only matches dense GPU execution, whereas larger T provides a longer overlap window and higher throughput (Kwon et al., 2026a). The paper does not evaluate $T < 8$, so a slowdown below that point is a design hypothesis, not a measured result.

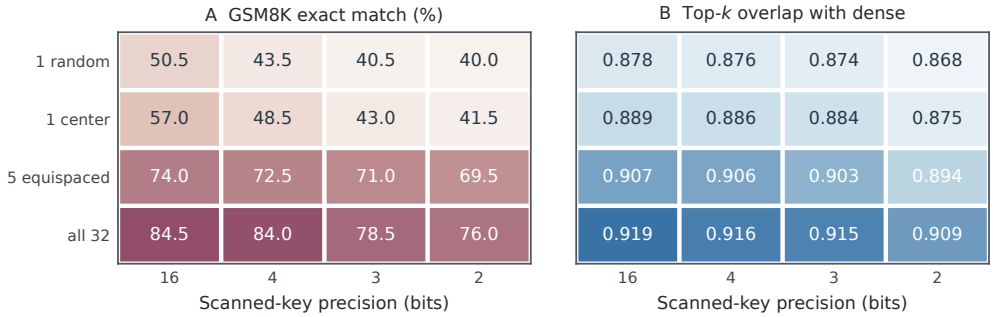

Figure 7: Legacy GSM8K selection quality across query populations and key widths.