

Mask-Guided KV Cache Eviction in Block Diffusion Language Models



Gleb Molodtsov[†]
BRAIn Lab

Ekaterina Alimaskina
BRAIn Lab
Yandex Research

Evgeny Uskov
Yandex School of Data Analysis

Artur Zagitov
BRAIn Lab

Aleksandr Beznosikov
BRAIn Lab

[†]Correspondence: gleb.molodtsov@gmail.com

Code: <https://github.com/brain-lab-research/MaskAhead>

Block diffusion language models keep a large key-value (KV) cache throughout generation and attend to it at every denoising step, limiting both memory capacity and generation speed. Reducing these costs requires deciding which past tokens to use for denoising the current block (selection) and which to keep in memory for future blocks (eviction). We propose MASKAHEAD, a training-free method that solves both tasks with a single mask-query-based ranking mechanism. Current-block masks guide selection, while probes of upcoming masked blocks guide eviction. Both rank KV entries by their estimated contribution to the attention output. Our quantized variant, Q-MASKAHEAD, computes selection and attention directly from low-bit KV, largely preserving the selected entries. Experiments on Fast-dLLM-v2, DreamReasoner, and LLaDA2.0-mini cover long-generation reasoning, long-prompt question answering, and needle-in-a-haystack retrieval. On long-prompt QA, MASKAHEAD reduces KV memory by 9.5× on average with a 1.2-point mean F1 loss relative to dense inference. Q-MASKAHEAD increases the reduction to 20.1× with a 2.3-point mean F1 loss. In a batch-32 systems profile, MASKAHEAD achieves 1.23× end-to-end and 1.68× decode-stage speedups over dense inference.

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., 2025, 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 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 permits a query probe before future blocks are generated: masked positions expose model-derived queries conditioned on the known prefix. We use these queries as a forecast for eviction, while current-block queries select temporary reads. The forecast cannot include tokens that intermediate blocks will later finalize. MASKAHEAD tests whether this forward-looking evidence helps most where retention is irreversible and storage is tight.

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 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. 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 soft lookahead tokens and adapters to estimate future token importance. 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] ranks entries by attention-output sensitivity, combining attention weights with value vectors. Its H2O instantiation uses accumulated past attention. MASKAHEAD uses value-aware scores with two query horizons: current-block masks for temporary reads and upcoming masked blocks for persistent retention. A no-write probe supplies the future queries, and recurrent compaction enforces the storage budget.

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 blocked inference [Lou et al., 2026]. This establishes four-bit KV quantization as prior work in

Table 1: Native capabilities of the cited methods. 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	✓	✓	✓

* HERALD offloads the full logical cache to host memory rather than physically evicting entries.

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. Our main contributions can be summarized as follows:

- **Two decisions, two query horizons.** We distinguish temporary reads from irreversible retention and identify the query evidence each decision requires.
- **Future masks guide permanent retention.** MASKAHEAD combines current-block read selection with future-mask probes and value-aware eviction, physically enforcing a persistent KV budget.
- **Four-bit storage preserves selection.** We disentangle selection, eviction, and precision. The ranking diagnostic finds 97.44% read-set and 97.65% retention-set overlap between four-bit and bf16 storage, supporting Q-MASKAHEAD’s compact persistent cache without a bf16 shadow copy.
- **Validation across three models and distinct workloads.** Experiments on Fast-dLLM-v2-7B, DreamReasoner-8B, and LLaDA 2.0 cover long-generation reasoning, long-prompt QA, and needle-in-a-haystack (NIAH) passage retrieval. MASKAHEAD achieves the highest F1 among evicting methods in all six main QA settings, including one tie, while reducing KV memory by 9.5× on average.

3 Problem Setup: Read and Retention Decisions

3.1 Two decisions over one prefix

Block diffusion finalizes one block before decoding the next. Let $\mathcal{I}_{t,\ell,h}$ be the live prefix entries at block t for layer ℓ and KV head h . During block t , a read set $R_{t,\ell,h} \subseteq \mathcal{I}_{t,\ell,h}$ supplies prefix attention for the remaining denoising steps. Entries outside $R_{t,\ell,h}$ stay in storage and can be read by a later block. At an eviction checkpoint, a retained set $S_{t,\ell,h} \subseteq \mathcal{I}_{t,\ell,h}$ replaces the persistent cache. An entry outside $S_{t,\ell,h}$ is unavailable until it is recomputed, which the inference procedure does not do. The read budget k_t and retention budget C_t therefore constrain different resources.

For a query q and prefix set A , let $\mathbf{o}(q; A)$ be the prefix-attention output obtained after normalizing attention over A . If Q_t^{later} contains later-step queries of block t and Q_j^{realized} contains the queries eventually realized in block j , the two ideal decisions minimize different errors:

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

The equations describe the decision horizons, not objectives that our implementation optimizes exactly. In particular, Q_j^{realized} is unknown when S_t must be chosen. The reference set $\mathcal{I}_t$ excludes KV appended after the checkpoint, so the retention objective isolates the effect of deleting currently live entries. At block t , the temporary read buffer is gathered before eviction. The retained set must serve later blocks and the first step at the next checkpoint.

The horizons differ in a recorded full-cache MATH-500 trace (Figure 1). At denoising step 8 of the current block, a shared read set covers 97.75% of prefix-attention mass, against 98.50% for a freshly selected set at the same budget ($n = 29$ at that step). One block later, coverage by the old set is 74.69%, and after four blocks it is 62.16%. The fresh-set comparison includes entries appended after the old set was selected. The trace therefore measures loss of coverage as the prefix grows, including newly available positions. It motivates refreshing read selection each block but cannot, by itself, identify the best retention rule.

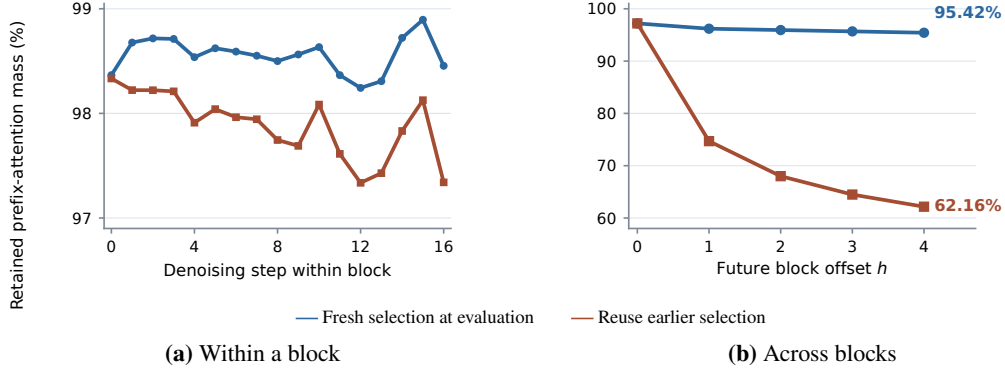


Figure 1: Full-cache MATH-500 trace ($n = 30$ initially). Within-block reuse stays close to step-specific selection; cross-block coverage declines as the prefix grows.

3.2 Queries available before commitment

The first denoising step of block t contains B masks, with queries $Q_t^{(1)}$. These queries are available before any token in the block is finalized and can rank reads for its remaining steps. Retention needs evidence about blocks that have not yet been generated. At each eviction checkpoint, a block-causal, no-write probe places masks at positions in the next $M - 1$ blocks and returns $Q_t^{(2)}, \dots, Q_t^{(M)}$. The probe does not append KV or produce draft tokens. Its future-block queries depend on the known prefix and masks, since the intermediate blocks have not been finalized. They estimate future cache demand rather than reproduce eventual queries.

We set the eviction interval to $E = 4$ blocks and probe $M = 4$ blocks, including the current one and the next three. The retained cache also serves the first step of block $t + 4$, which this probe does not cover. Current-block queries are included as available evidence, although the current block reads from its gathered buffer after eviction. The horizon is therefore an empirical design choice, not an exact enumeration of the queries served by S_t . The fixed-budget controls in Table 4 compare it with shorter and longer probes.

4 MASKAHEAD: Two Decisions, Two Query Horizons

MASKAHEAD builds two rankings from the queries available at block t . The first ranking serves the remaining denoising steps of that block. The second ranks entries kept in persistent KV until the next eviction. This changes the evidence used for each decision while retaining one scoring rule.

A score for attention-output sensitivity. For a query q , let $p_{q,i}$ be the attention weight of live prefix entry i , normalized over the live prefix, and let v_i be its value. Writing $\mathbf{o}_q = \sum_i p_{q,i} v_i$, deletion of a single entry and renormalization give

$$\|\mathbf{o}_q - \mathbf{o}_{q,-i}\|_2 = \frac{p_{q,i}}{1 - p_{q,i}} \|\mathbf{v}_i - \mathbf{o}_q\|_2. \quad (2)$$

Thus entries with the same attention mass can have different effects on the output. The identity applies to one deletion and does not solve the subset-selection problem. For a query population Q , we use

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

The pooled score weights each value’s deviation from the attention centroid by its relevance to the query population. MASKAHEAD computes separate read and retention rankings with this multi-entry removal proxy. The score ablation in Table 5 checks its empirical contribution without treating the single-entry identity as a guarantee about answer accuracy.

Current masks select reads. Future masks guide retention. Let $\mathbf{a}(Q)$ collect the weights $a_i(Q)$ across live entries. At the first step of block t , MASKAHEAD computes

$$s_i^{\text{read}} = f_i(\mathbf{a}(Q_t^{(1)})), \quad s_i^{\text{retain}} = f_i\left(\sum_{m=1}^M \mathbf{a}(Q_t^{(m)})\right). \quad (4)$$

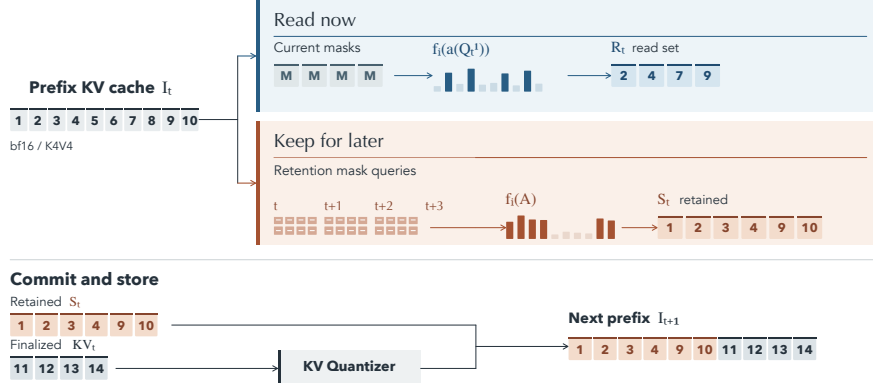


Figure 2: Current-block masks select temporary reads. Current and future mask queries rank entries retained until the next eviction.

The top- k_t read entries are gathered into a temporary buffer. At an eviction checkpoint, four initial positions and a 64-position recent window are protected within the C_t budget. The remaining slots go to the highest-scoring entries. Ranking and budgets are applied per layer and KV head. The first two layers remain dense for read selection. The read buffer is gathered before persistent KV is compacted, so the current block still has its selected entries. After its remaining denoising steps, the finalized block appends KV at its original logical positions. These operations follow the order in Algorithm 1.

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

- 1 **Input:** live cache I_t , budgets k_t , C_t , horizon M , interval E .
- 2 **Run** the first all-mask step of block t against I_t ; collect $Q_t^{(1)}$.
- 3 **Select reads:** $R_t \leftarrow \text{TopK}(s^{\text{read}}, k_t)$; **gather** R_t into a temporary read buffer.
- 4 **if** t is an eviction checkpoint **then**
- 5 **Probe** masks in blocks $t+1, \dots, t+M-1$ without writing persistent KV.
- 6 **Rank** the live prefix by s^{retain} ; protect sinks and recent positions.
- 7 **Compact** persistent KV to the retained set of budget C_t .
- 8 **end if**
- 9 **Denoise** the remaining steps using R_t ; finalize block t and append its KV.

Decisions from four-bit storage. Q-MASKAHEAD stores keys and values at four bits and computes rankings from that stored representation without a bf16 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, and the scoring pass unpacks stored values when needed. After eviction, complete surviving key groups are copied bit for bit and partial groups are regrouped. The four-bit result therefore tests selection, retention, and attention with a compact cache rather than quantizing a preselected bf16 cache.

For long prompts, the retention mechanism can be applied after full prefill or during prompt processing. We compare full prefill, chunked prefill, and question-probe heuristics in Section 5.5.

5 Experiments

We evaluate Fast-dLLM-v2-7B [Wu et al., 2026a], DreamReasoner-8B [Wu et al., 2026c], and LLaDA-2.0-mini [Bie et al., 2025].

Tasks. Long-generation reasoning includes MATH-500 ($n = 500$), containing competition-level mathematics problems, and GSM8K ($n = 1319$), containing grade-school mathematical word problems, both scored by answer accuracy. Long-prompt QA includes HotpotQA, NarrativeQA, and MuSiQue ($n = 200$ each), covering multi-hop factual QA, narrative comprehension, and compositional multi-hop QA, respectively, scored by token F1. Needle-in-a-haystack (NIAH) retrieval tests recovery of a target passage embedded in a long context ($n = 200$), scored by retrieval accuracy.

Compared methods. We compare dense inference, sparse-read baselines, cache-eviction baselines, and our methods:

- Dense: reads and retains the full prefix cache in bf16.
- MAGE [Kwon et al., 2026b]: uses attention from the fully masked first denoising step to select a prefix subset reused across the remaining steps. The full cache stays on the GPU.
- HERALD [Kwon et al., 2026a]: selects prefix entries using a representative mask query and retrieves them from a full cache offloaded to host memory.
- H2O [Zhang et al., 2023]: retains entries with high accumulated historical attention, together with protected initial and recent positions.
- H2O+CAOTE [Zhang et al., 2023, Goel et al., 2025]: augments historical attention scores with value information to estimate the effect of eviction on attention outputs.
- Expected Attention [Devoto et al., 2025]: estimates future attention from a query distribution fitted to past hidden states and retains entries with high expected importance.
- DapQ [Tian et al., 2026]: constructs surrogate future queries from pseudo tokens at future positions and uses them to rank cache entries for retention.
- MASKAHEAD: uses current-block mask queries for temporary read selection and future-block mask queries for persistent retention, with value-aware scoring and physical cache compaction.
- Q-MASKAHEAD: combines MASKAHEAD with 4-bit key/value storage to reduce the memory occupied by retained entries.

Protocol. Fast-dLLM-v2-7B is evaluated with greedy decoding. For reasoning, the read budget is $k_t = \min(N_t, \max(128, \lceil 0.05N_t \rceil))$, and the retention budget is $C_t = \max(256, \lfloor 0.1N_t^{\text{seen}} \rfloor)$. Here N_t is the live prefix and N_t^{seen} counts positions seen before any eviction. Long-context QA uses floors of 256 reads and 512 retained entries. Baseline adaptations share the inference harness, decoding rule, per-task budgets, four protected initial positions, a 64-position recent window, and a four-block eviction interval where applicable.

KV memory includes retained keys and values and method state, normalized within each model and task to dense bf16. For reasoning, we measure the maximum during the run. For QA and retrieval, we measure the resident cache after the response. The prefill experiments report the maximum across prompt processing and generation (Section 5.5). Bold and underlining rank the best and second distinct values, maximizing quality and minimizing KV, within the comparison stated by each table. The separate batch-profile pilot reports absolute GiB and seconds (Section 5.4), rather than task-normalized KV ratios.

5.1 Quality under a bounded cache

Retention quality matters most when the cache cannot preserve many alternative candidates. Accuracy does not increase monotonically with retention capacity at a fixed read budget, so storing more entries alone does not resolve all errors from sparse inference.

The main task matrix separates answer quality from the memory needed to obtain it (Tables 2 and 3). On Fast-dLLM reasoning, MASKAHEAD retains much of the quality of full-cache sparse reads while

Table 2: Reasoning accuracy (%) / KV memory relative to dense bf16.

Category	Method	Fast-dLLM-v2-7B		DreamReasoner-8B
		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
Eviction	H2O	50.80 / 0.62	81.50 / <u>0.85</u>	66.80 / <u>0.68</u>
	H2O+CAOTE	49.60 / <u>0.61</u>	85.44 / 1.01	67.80 / <u>0.68</u>
	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 &	MASKAHEAD	<u>54.60</u> / 0.62	<u>84.91</u> / <u>0.85</u>	<u>68.00</u> / <u>0.68</u>
Eviction	Q-MASKAHEAD	54.00 / 0.25	84.23 / 0.32	67.40 / 0.28

Table 3: LongBench quality (F1) / KV relative to dense bf16.

Category	Method	Fast-dLLM-v2			LLaDA-2.0-mini		
		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 / 0.129	22.31 / 0.116	13.91 / 0.119	48.32 / 0.128	23.72 / 0.115	32.25 / 0.119
	H2O+CAOTE	31.72 / 0.129	22.00 / 0.116	15.21 / 0.119	48.18 / 0.128	24.53 / 0.115	31.48 / 0.119
	Expected Attention	22.52 / 0.209	21.81 / 0.163	9.28 / 0.175	40.98 / 0.173	21.25 / 0.141	21.21 / 0.151
	DapQ	38.35 / 0.107	24.86 / 0.104	16.74 / 0.104	55.47 / 0.107	27.59 / 0.104	39.04 / 0.104
Selection & Eviction	MASKAHEAD	38.41 / 0.107	24.94 / 0.104	17.15 / 0.104	56.27 / 0.107	27.59 / 0.104	39.17 / 0.104
	Q-MASKAHEAD	34.75 / 0.050	23.90 / 0.051	17.11 / 0.048	54.87 / 0.053	27.24 / 0.045	38.99 / 0.052

physically discarding prefix entries. Its advantage over full-cache MAGE on MATH-500 shows that cache capacity alone does not determine accuracy: the query source and read score also change between these policies. Dense inference remains more accurate, and the relative ordering among evicting methods varies across tasks.

Expected Attention illustrates why the retained-token budget is insufficient for comparing memory. Its Gaussian query state adds storage beyond the retained K/V, taking KV memory above the dense cache despite eviction. Its strong MATH-500 accuracy therefore belongs to a different memory operating point. The comparison must account for auxiliary state as well as retained entries, rather than interpret eviction itself as a guarantee of compression.

Long-prompt QA exposes a different pattern: MASKAHEAD and DapQ are close at matched KV memory, while both preserve substantially more quality than historical-attention retention. MASKAHEAD has the highest evicting-policy F1 in all six model/task settings, including one tie. Across these settings, the main result is an average 9.5 $\times$ reduction in KV memory with a mean 1.2-point F1 loss.

Q-MASKAHEAD further reduces the storage cost of surviving entries, giving the smallest KV footprint in every main model/task setting. On reasoning tasks, this additional compression preserves accuracy close to bf16 MASKAHEAD. Across long-prompt QA, it increases the average memory reduction to 20.1 $\times$ with a mean 2.3-point F1 loss relative to dense inference. Thus, quantization extends the memory savings beyond eviction alone, with an additional quality cost that varies across tasks.

5.2 What evidence does each decision need?

Future-mask evidence improves retention without relying on a larger resident cache. The query-source control fixes the score, budgets, and eviction interval, with nearly unchanged MATH-500 KV memory across variants (Table 4). Looking beyond the current block improves reasoning accuracy, but extending the probe further does not give a monotonic benefit. Two and four blocks have similar point estimates, which favors a short lookahead over an unrestricted horizon.

Masks provide better retention evidence than DapQ-filled future positions on MATH-500: the paired mask-versus-DapQ comparison has 28 wins and 11 losses ($p = 0.0095$, two-sided sign test). Unlike pseudo tokens, masks do not commit the probe to an unfinalized continuation. The control compares query sources without isolating the cause of the quality difference. HotpotQA is much less sensitive to query source under its 32-token output protocol, so the reasoning result should not be

Table 4: Probe controls on Fast-dLLM-v2-7B: MATH-500 ($n = 500$) and HotpotQA ($n = 200$). KV is measured on MATH-500 relative to dense bf16. M counts the current and future probed blocks.

Probe	MATH (%)	HotpotQA (F1)	KV
$M = 1$	52.0	38.26	0.625
$M = 2$	54.4	37.86	0.625
$M = 4$	54.6	38.41	0.624
$M = 8$	53.2	<u>38.29</u>	0.621
$M = 4$, random tokens	53.6	<u>38.29</u>	0.625
$M = 4$, DapQ tokens	51.2	38.01	0.625

Table 5: Read and retention scores in bf16 at fixed queries and budgets: MATH-500 ($n = 500$, $k = 128$, $C = 256$) and HotpotQA ($n = 200$, $k = 256$, $C = 512$). Full denotes no eviction. KV is measured on MATH-500 relative to dense bf16.

Read	Retain	MATH (%)	HotpotQA (F1)	KV
Mass	Full	52.2	<u>38.14</u>	1.000
Value-aware	Full	<u>53.8</u>	37.85	1.000
Mass	Mass	52.0	37.84	0.625
Mass	Value-aware	<u>53.8</u>	38.08	0.625
Value-aware	Mass	53.4	37.91	0.618
Value-aware	Value-aware	54.6	38.41	<u>0.624</u>

Table 6: Selection, retention, and stored precision on Fast-dLLM-v2-7B. MATH and GSM8K report accuracy (%), and HotpotQA reports F1. KV memory is relative to dense bf16.

Read score	Format	Retain	MATH	GSM8K	HotpotQA	MATH KV	HotpotQA KV
Mass	bf16	Full	52.2	85.44	<u>38.14</u>	1.000	1.000
Mass	K4V4	Full	52.0	84.61	36.07	<u>0.313</u>	0.313
Value-aware	bf16	Full	53.8	<u>84.91</u>	37.85	1.000	1.000
Value-aware	bf16	Evict	54.6	<u>84.91</u>	38.41	0.624	<u>0.107</u>
Value-aware	K4V4	Evict	<u>54.0</u>	84.23	34.75	0.252	0.050

generalized to short-answer QA. We use four blocks as a short horizon on the same scale as the eviction interval, rather than as a uniquely optimal probe length.

The score factorization distinguishes better retention evidence from a better ranking criterion (Table 5). Changing the read score affects what the current block sees, while changing the retention score affects what later blocks can still access. Using the value-aware score for both decisions gives the highest point estimates among the evicting combinations, but its paired quality intervals against mass scoring include zero on both tasks (Section C.3). The single-entry deletion identity in (2) motivates value-aware scoring by quantifying how removing an entry changes the attention output. The quality experiments provide clearer evidence for using future-mask queries than for replacing attention-mass scoring with value-aware scoring.

5.3 Precision changes the memory and quality budgets

Eviction and quantization reduce different parts of the storage cost: the number of resident entries and the bytes used to represent each entry. Their combination gives the smallest measured KV footprint in the main task matrix (Table 6). On MATH-500, four-bit storage adds a $2.48\times$ KV-memory reduction over bf16 MASKAHEAD at a 0.6-point accuracy cost. The reduction is smaller than the nominal ratio between bf16 and four-bit payloads because the reported KV also includes method state, not only compressed entries.

The quality cost of stored precision is task dependent. Untransformed K4V4 loses more F1 on Fast-dLLM HotpotQA than on the other main QA settings, while the rotated-and-centered key variant recovers much of that gap at unchanged KV memory (Table D.3). This comparison shows that the quantization representation matters within the same storage budget, although it does not separate the effects of rotation and centering. Across the six main QA settings, Q-MASKAHEAD reduces KV memory by $20.1\times$ on average at a mean 2.3-point F1 loss relative to dense inference.

High ranking overlap is insufficient to certify quality preservation. The recording-only diagnostic finds over 97% overlap for both reads and retention, yet the generated answers remain sensitive to precision (Section D.3). Even unchanged sets use perturbed keys and values in attention, and set overlap does not measure the importance of the entries that differ.

The budget sweeps expose a separate quality lever: more reads can improve answers without enlarging the persistent cache (Figure 5). At fixed retention, increasing the MATH-500 read floor brings accuracy close to dense inference while KV memory remains near the same level. HotpotQA also benefits from more reads, but the QA tasks have different read-fraction orderings (Table B.4). Thus, the storage budget need not dictate one read budget for every workload. These within-policy controls distinguish errors caused by limited access to surviving entries from the consequences of deleting entries, without establishing a global frontier over all baselines.

5.4 Batch-profile memory and execution time

The systems pilot separates smaller storage from faster execution (Figure 3). K4V4-only storage substantially reduces resident KV but does not reduce the recorded batch-32 decode time. Quantization alone leaves the full logical prefix available to the attention procedure, whereas MASKAHEAD also reduces the entries gathered for repeated denoising and removes entries from persistent storage.

At batch 32, MASKAHEAD and Q-MASKAHEAD reduce KV memory by 85.1% and 89.9%, respectively. Their recorded decode-time ratios relative to dense bf16 are $1.68\times$ and $1.64\times$, while complete-batch ratios are smaller at $1.23\times$ and $1.20\times$. Most of the difference in reported median times lies in decoding, which is the stage targeted by repeated sparse reads.

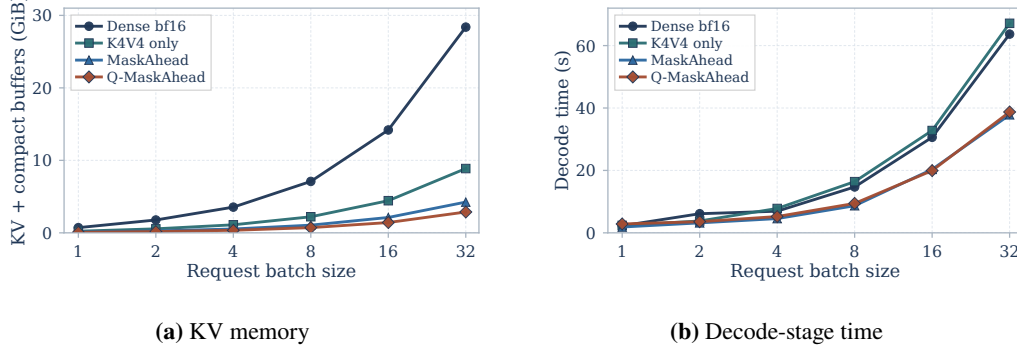


Figure 3: Batch-profile medians over three prompt mixes, restricted to batches 1–32. Memory is measured after batch completion and includes resident KV and compact buffers. Decode timing excludes the other stages of the request. Complete-batch time and the batch-64 records are in Section A.

Each policy processes the same 32 requests within each of three seeded prompt cohorts and generates 256 token positions per request, with EOS stopping disabled. The median end-to-end output-token throughputs are 55.6 tokens/s for dense bf16, 68.2 for MASKAHEAD, and 66.5 for Q-MASKAHEAD. Thus, the complete-batch ratios compare matched fixed-length outputs, not differences in generated length. The profile is separate from the quality experiments and does not measure throughput with natural stopping or at matched quality. Section A reports the complete timings and output-length accounting.

5.5 Retention heuristics for long prefill

The QA comparisons above use full prefill: the model processes the complete prompt before applying retention, once the question is available. This reduces KV memory during decoding. When the prompt is much longer than the response, however, the memory peak occurs during prefill. We therefore apply the same retention framework with three prefill heuristics:

- Full prefill: retain the prompt cache until prefill completes, then compress it for decoding.
- Chunked prefill: process the prompt in 512-token chunks and evict entries between chunks.
- Question probe: use question-conditioned probing to rank entries for early retention during prompt processing.

Question probe preserves substantially more answer quality than chunked prefill while using less KV memory (Table 7). Both apply retention before the prompt cache reaches its full size, but the question provides task-specific evidence for deciding which context to preserve. The comparison changes both ranking and retention timing, so it evaluates the complete heuristics rather than isolating question conditioning.

Table 7: MASKAHEAD prefill heuristics on HotpotQA ($n = 200$, bf16).

Heuristic	F1	KV
Full prefill	38.41	0.999
Chunked prefill	26.90	<u>0.185</u>
Question probe	<u>37.16</u>	0.144

Combining question probe with four-bit storage requires a separate quality check: this combination sharply lowers F1 on all three QA tasks (Table E.1). NIAH retrieval provides a complementary test of retained information: bf16 MASKAHEAD and DapQ remain close to full-cache MAGE after compression, while four-bit storage incurs a larger retrieval loss (Table E.2).

6 Conclusion

MASKAHEAD combines current-block read selection with future-mask-guided retention in block diffusion. It achieves the highest F1 among evicting methods across all six main QA settings, including one tie, with an average $9.5\times$ reduction in KV memory and a 1.2-point mean F1 loss relative to dense inference. Q-MASKAHEAD extends the average reduction to $20.1\times$ with a 2.3-point F1 loss.

Matched controls favor future-block masks over current-only and pseudo-token probes. Value-aware scoring has the highest point estimates in the score factorization, while paired quality intervals include zero. The reasoning comparisons identify the budget dependence: the gap over H2O is largest at the smallest retention floor, while H2O+CAOTE leads at the wider MATH-500 budget.

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, pages 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. In *Forty-third International Conference on Machine Learning*, 2026.
- Alessio Devoto, Maximilian Jeblick, and Simon Jégou. Expected attention: KV cache compression by estimating attention from future queries distribution, 2025.
- Raghav 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*, pages 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, Kevin Lau, Gregor MacDonald, Jiayi Nie, Yao Lai, Can Xiao, Xuan Guo, Jianyi Cheng, Rika Antonova, et al. Npu design for diffusion language model inference. *arXiv preprint arXiv:2601.20706*, 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, 2025.
- 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, pages 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*, pages 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, pages 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.
- 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*, 2026c.
- 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, pages 381–394, 2024.

Appendix

A Batch Profiling and Cost Accounting

The supplementary results follow the two cache decisions in the main text. This section records the systems pilot and its memory metrics. Section B examines read and retention budgets, Section C isolates query and score choices, Section D studies stored precision, and Section E separates prefill from generation.

A.1 Measurement scope and complete batch profile

The displayed profile contains four cache policies over batches 1–64. All plots stop at batch 32 so that timing differences remain visible. The plotted medians aggregate three seeded prompt cohorts with the same requests across policies within each cohort. The results record output-token throughput as generated positions divided by complete-batch wall time. The source JSON retains per-cohort timing, token throughput, request IDs, prompt-token totals, and source/configuration hashes. The fixed-output ratios do not establish throughput with natural stopping, fixed-quality throughput, or a serving-capacity limit.

Generation-end resident KV includes used entries and compact buffers. The figures and tables use this KV metric. The source CSV also preserves a full-run device-allocation field, which is not used for the memory comparison here. Figure 4 shows KV together with complete-batch and decode-stage times over the common successful range. Only successful measurements have numerical coordinates. At batch 64, the supplied summary records OOM for dense bf16 and K4V4-only storage. The completed MASKAHEAD run uses 8.5 GiB of resident KV and takes 242.4 s in total, including 78.6 s for decode. The Q-MASKAHEAD batch-64 run uses 5.8 GiB and takes 245.0 s in total, including 81.3 s for decode.

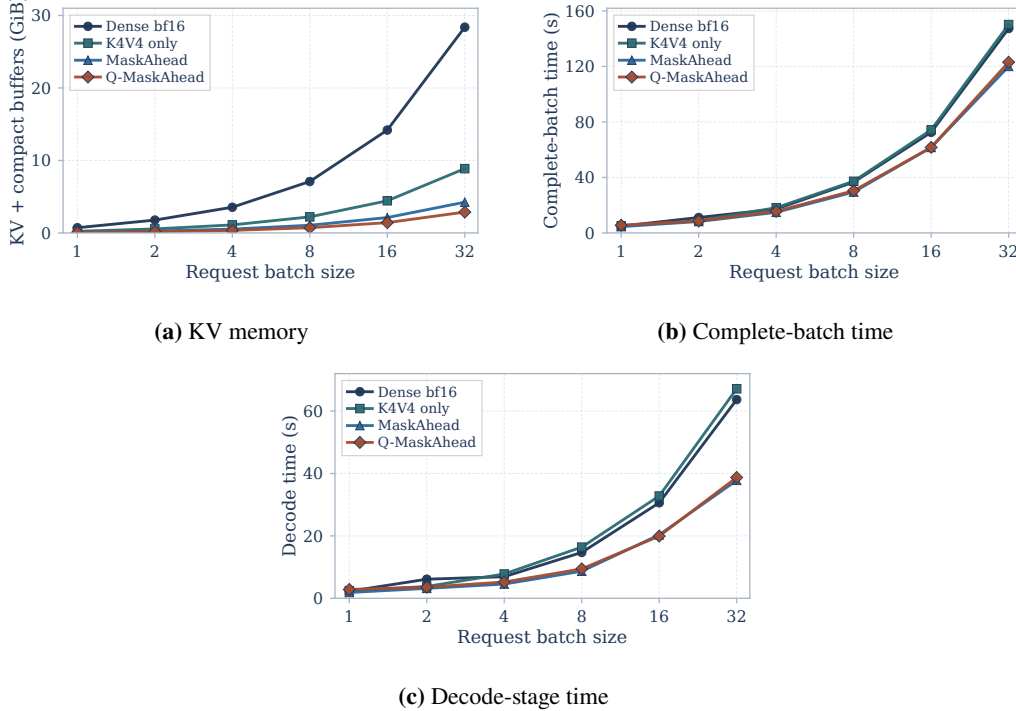


Figure 4: Systems pilot over batches 1–32: resident KV, complete-batch time, and decode time. Times are supplied medians over three prompt mixes. Every plotted setting is successful. Memory is measured after batch completion and includes compact buffers. Batch-64 records are listed separately in Table A.1.

Table A.1: KV and timing profile. Memory is measured after batch completion in GiB, and time is in seconds. Values are rounded to one decimal place. OOM indicates the recorded run status.

Policy	Batch	KV	Wall time	Decode time
Dense bf16	1	0.7	4.9	2.3
Dense bf16	2	1.8	11.1	6.1
Dense bf16	4	3.5	17.2	6.9
Dense bf16	8	7.1	36.6	14.7
Dense bf16	16	14.2	72.6	30.6
Dense bf16	32	28.4	147.4	63.7
Dense bf16	64		OOM	
K4V4 only	1	0.2	5.1	2.4
K4V4 only	2	0.6	8.8	3.8
K4V4 only	4	1.1	18.2	7.8
K4V4 only	8	2.2	37.3	16.4
K4V4 only	16	4.4	74.3	32.8
K4V4 only	32	8.9	150.2	67.1
K4V4 only	64		OOM	
MASKAHEAD	1	0.1	4.5	1.9
MASKAHEAD	2	0.3	8.2	3.2
MASKAHEAD	4	0.5	14.9	4.6
MASKAHEAD	8	1.1	29.5	8.7
MASKAHEAD	16	2.1	61.6	20.3
MASKAHEAD	32	4.2	120.1	37.8
MASKAHEAD	64	8.5	242.4	78.6
Q-MASKAHEAD	1	0.1	5.6	2.8
Q-MASKAHEAD	2	0.2	8.6	3.6
Q-MASKAHEAD	4	0.4	15.6	5.2
Q-MASKAHEAD	8	0.7	30.4	9.4
Q-MASKAHEAD	16	1.4	61.7	19.9
Q-MASKAHEAD	32	2.9	123.1	38.7
Q-MASKAHEAD	64	5.8	245.0	81.3

A.2 Generated length in the quality runs

The batch profile fixes the output length, whereas the quality evaluations use their configured stopping rules. Table A.2 reports mean generated tokens per example from the per-example runtime records for the Fast-dLLM-v2 configurations in the main quality tables. All methods within a dataset are evaluated on the same examples.

Table A.2: Mean generated tokens per example under the quality-evaluation stopping rules. The sample sizes are 500 for MATH-500, 1,319 for GSM8K, and 200 for each QA dataset. Method rows use the configurations reported in the main quality tables.

Method	MATH-500	GSM8K	HotpotQA	NarrativeQA	MuSiQue
Dense bf16	611.8	311.8	8.75	13.69	11.40
HERALD	605.0	310.8	9.25	14.11	11.42
MAGE	624.7	314.2	9.32	13.50	11.28
H2O	662.8	313.8	9.88	14.09	12.15
H2O+CAOTE	676.6	313.9	9.77	13.46	11.61
Expected Attention	652.4	312.3	11.14	13.91	13.25
DapQ	647.3	311.3	9.28	13.63	11.40
MASKAHEAD	617.8	313.6	9.28	13.64	11.36
Q-MASKAHEAD	646.8	313.7	10.04	15.26	11.83

On MATH-500, MASKAHEAD generates 617.8 tokens per example, 1.0% more than dense bf16, while Q-MASKAHEAD generates 646.8, 5.7% more. The corresponding GSM8K increases are 0.6% for both variants. For the short-answer QA tasks, the bf16 variant differs from dense by at most 0.53 token per example. The four-bit variant generates up to 1.6 additional tokens per example on NarrativeQA.

Thus, response length is not identical in the quality runs, especially for four-bit storage, and their latency cannot be inferred from the fixed-length batch profile.

A.3 Amortized attention traffic

The pilot above measures execution time. The following accounting describes the attention-read component rather than predicting throughput. 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, with $(Q_{\text{probe}} + Q_{\text{compact}})/E$ amortized over checkpoints. This simplified traffic decreases when

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

The expression omits value-aware score computation, read-buffer gathering, dense-read exceptions, kernel launches, and request scheduling. Resident-cache reduction therefore does not imply an equal reduction in total traffic or full-run memory allocation.

B Read and Retention Budget Tradeoffs

Across the six model/task settings in Table 3, MASKAHEAD reduces KV memory by 9.5× on average with a mean 1.2-point F1 loss relative to dense inference. Q-MASKAHEAD increases the average reduction to 20.1× with a mean 2.3-point loss. The mean F1 difference between bf16 MASKAHEAD and Q-MASKAHEAD is 1.1 points in favor of bf16. We average within-setting dense-to-method memory ratios and F1 differences.

The read budget controls the prefix gathered for repeated denoising, while the retention budget controls entries available to later blocks. We first vary retention on MATH-500, then compare policies at a wider budget, and finally vary read and retention fractions on long-context QA. These series use different floors, which are stated separately rather than pooled.

B.1 Reasoning capacity sweep

With the 128-entry read floor fixed, Table B.1 compares MASKAHEAD and H2O across four retention floors on the same 500 MATH-500 examples. The accuracy difference is 11.6 points at $C = 128$, 3.8 points at $C = 256$, and 0.2 points at both $C = 512$ and $C = 1024$. KV includes growth between four-block eviction checkpoints. The large difference at the smallest floor is a comparison with H2O, while the broader baseline comparison uses the main $C = 256$ floor.

Table B.1: MATH-500 retention-floor sweep at fixed $k_{\min} = 128$, bf16, on the same 500 examples. KV includes method state and is relative to dense bf16. Bold and underlining compare the two policies within each floor.

C floor	MASKAHEAD		H2O	
	Acc. (%)	KV	Acc. (%)	KV
128	51.0	<u>0.446</u>	<u>39.4</u>	0.429
256	54.6	<u>0.624</u>	<u>50.8</u>	0.617
512	53.0	0.852	<u>52.8</u>	<u>0.853</u>
1024	53.8	0.982	<u>53.6</u>	<u>0.984</u>

B.2 Common-harness comparison at a wider budget

The wider-budget runs use $k_t = \min(N_t, \max(256, \lceil 0.1N_t \rceil))$ and $C_t = \max(512, \lfloor 0.20N_t^{\text{seen}} \rfloor)$, with eviction every four blocks. Every displayed MATH-500 row has 500 examples and each QA row has 200. All policies use the same examples and common inference harness, with bf16 storage except for Q-MASKAHEAD. At this budget, historical CAOTE-style retention reaches 57.6% on MATH-500, MASKAHEAD and Expected Attention reach 57.2%, and DapQ reaches 56.2% (Table B.2). Expected Attention uses 2.346 times dense bf16 KV memory because its Gaussian state dominates the retained cache. MASKAHEAD has the highest F1 on HotpotQA and NarrativeQA, scoring 39.10 and 24.97 compared with 39.04 and 24.43 for DapQ. On MuSiQue, H2O leads at 18.02 F1, followed

by MASKAHEAD at 17.17. On GSM8K, H2O, DapQ, and MASKAHEAD each score 85.44%, while Q-MASKAHEAD scores 86.35% across 1,319 examples. These complete greedy runs locate the budget regime in which retention rules differ.

Table B.2: Fast-dLLM-v2-7B at $k = 10\%$, $C = 20\%$, with read/retention floors of 256/512. MATH-500 uses 500 examples and each QA task uses 200. Bold and underlining rank the evicting policies within each 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	56.2	<u>39.04</u>	<u>24.43</u>	16.89
Expected Attention, bf16	57.2	24.94	23.79	11.99
MASKAHEAD	<u>57.2</u>	39.10	24.97	<u>17.17</u>
Q-MASKAHEAD	55.4	36.44	24.32	16.92

B.3 Read-budget controls at fixed retention

Figure 5 makes the read-budget dependence explicit while holding retention fixed. On MATH-500, increasing the read floor from 128 to 512 raises accuracy from 54.6% to 58.4% while KV memory stays near 0.62 of dense bf16. On HotpotQA, both bf16 and four-bit storage improve as the read fraction increases, while each curve retains its own fixed storage format and retention budget. Retention-floor and precision comparisons are reported separately in Tables B.1 and D.1.

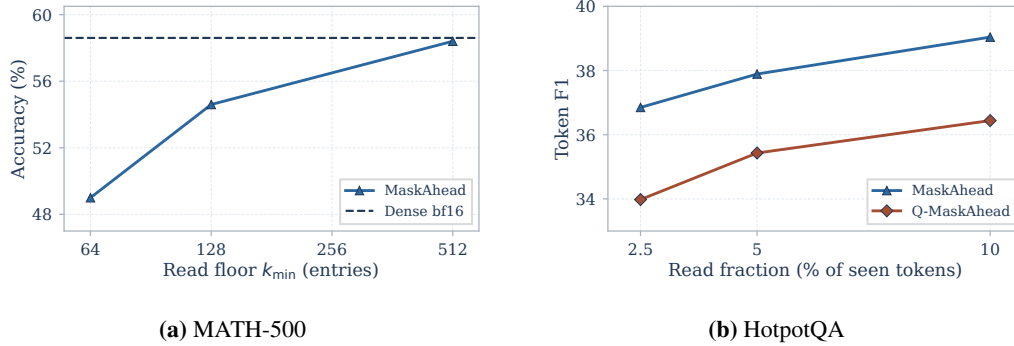


Figure 5: Read-budget controls on Fast-dLLM-v2-7B, with fixed retention: MATH-500 ($n = 500$, $C_{\min} = 256$) and HotpotQA ($n = 200$, $C = 20\%$). Only the read floor or fraction varies within each curve. The dashed line is dense bf16 MATH accuracy.

B.4 Long-context budget and read-fraction series

The long-context budget series uses $k = C/2$, expressed as fractions of seen tokens, with floors of 32 reads and 64 retained entries. These floors differ from the main QA setting of 256 reads and 512 retained entries and from the wider-budget comparison above. MASKAHEAD improves HotpotQA and NarrativeQA over H2O at all three budget fractions while using less KV memory (Table B.3). H2O leads MuSiQue by 0.30 F1 at $C = 20\%$.

Table B.3: LongBench retention-budget comparison of H2O and MASKAHEAD on Fast-dLLM-v2-7B, $n = 200$ per task. C is a fraction of seen tokens and $k = C/2$, with floors of 64 retained entries and 32 reads. KV is relative to dense bf16. These runs use full prefill, whose memory peak remains near dense bf16.

C (%)	Policy	HotpotQA		NarrativeQA		MuSiQue	
		F1	KV	F1	KV	F1	KV
5	H2O	29.71	0.077	22.28	0.066	14.41	0.069
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	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	MASKAHEAD	39.04	0.206	24.78	0.206	<u>16.89</u>	0.206

At fixed $C = 20\%$, increasing reads from 2.5% to 10% raises HotpotQA F1 from 36.85 to 39.04

at the same 0.206 KV memory. NarrativeQA and MuSiQue have different read-fraction orderings (Table B.4), so a single task does not determine the read budget for all QA workloads.

Table B.4: Read-budget sweep at fixed $C = 20\%$, with 200 examples per task. Retained-cache size is fixed within each storage format. Bold and underlining rank read fractions within each task and format.

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

C Query Horizons and Ranking Criteria

This section separates three kinds of evidence: attention-coverage diagnostics, controls over probe queries, and controls over the read/retention score. Coverage uses recorded dense-cache trajectories. The other controls generate complete answers with matched cache budgets.

C.1 Recorded within-block coverage

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

Table C.1: Realized prefix-attention mass covered by a block-shared read set and same-budget step-specific offline top- k . The final column counts examples reaching each step, so rows have different sample sizes.

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 shared set remains close to the offline reference within a block. Across blocks, its overlap with the fresh set falls to 47.2% at offset four, partly because the candidate prefix expands. These diagnostics motivate refreshing reads. They do not compare retention-policy answer quality.

C.2 Probe horizon and query-source controls

We hold the read and retention budgets, value-aware score, and four-block eviction interval fixed while varying the probe horizon M . The same future positions are also filled with random tokens or DapQ-style pseudo tokens. The complete matched control is in Table 4 in the main text. It combines task quality with the corresponding MATH-500 KV memory measurements.

The masks-versus-DapQ comparison has 28 wins and 11 losses on paired MATH-500 examples, giving a two-sided sign-test $p = 0.0095$. Two-block and four-block masks have similar point estimates, 54.4% and 54.6%, while an eight-block probe scores 53.2%. The sweep supports a short lookahead rather than a monotonic benefit from more future positions. On HotpotQA, all six variants lie within 0.55 F1 points under the 32-token output protocol.

C.3 Independent read and retention scores

The factorization in Table 5 changes the read and retention scores independently while keeping query sources and budgets fixed. Value-aware ranking for both decisions has the highest evicting-policy point estimate on MATH-500 and HotpotQA. Relative to attention mass for both decisions, the

differences are +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. Bootstrap samples are formed from aligned example IDs.

The single-entry deletion identity motivates the score, while these intervals leave its answer-quality contribution unresolved. The probe controls above isolate query source with the scoring rule fixed.

C.4 Protected positions and eviction interval

The default policy protects four sinks and a recent window $W = 64$, then evicts every $E = 4$ blocks.

Table C.2: Timing and protection controls on MATH-500 ($n = 500$) at the main budgets. KV includes method state and is relative to dense bf16.

Policy	MATH acc. (%)	MATH KV
Default: sinks 4, $W = 64$, $E = 4$	54.6	0.624
No protected sinks	53.0	<u>0.615</u>
$W = 32$	<u>54.0</u>	0.621
$W = 128$	53.6	0.618
$E = 2$	51.8	0.543
$E = 8$	53.8	0.718

Table C.2 changes these choices at the main budgets. Evicting every two blocks lowers MATH-500 KV memory to 0.543 and accuracy to 51.8%. Evicting every eight blocks raises KV memory to 0.718 and scores 53.8%.

D Four-Bit Storage and Ranking Stability

Stored precision changes both the bytes per entry and the keys/values used for ranking and attention. We examine key/value precision separately, then retention floors under K4V4, a key transform, and ranking overlap against bf16. Early-retention quantization is evaluated with the prefill policies in Section E.2.

D.1 Key/value precision and retention floors

At the main MATH-500 retention policy, K4V4 gives 54.0% accuracy at 0.252 dense-relative KV memory. Reducing key precision to two bits costs more accuracy than reducing value precision at similar cache size (Table D.1). The lowest-memory format is therefore a different accuracy operating point.

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

KV format	Acc.	KV
bf16	54.6	0.624
K4V4	<u>54.0</u>	0.252
K4V2	53.4	0.203
K2V4	48.2	0.193
K2V2	46.0	0.148

Table D.2: K4V4 MATH-500 retention floors at fixed $k = 128$. KV is relative to dense bf16.

C floor	Acc.	KV
128	50.0	0.156
256	<u>54.0</u>	<u>0.252</u>
512	54.2	0.336
1024	<u>54.0</u>	0.338

With $k_{\min} = 128$ fixed, raising the K4V4 retention floor from 128 to 256 improves MATH-500 accuracy from 50.0% to 54.0%. Larger floors add KV memory with little change in the measured accuracy (Table D.2).

D.2 Rotated and centered keys

At the same HotpotQA main budgets and 0.050 KV memory, the rotated-and-centered variant reaches 37.33 F1, compared with 34.75 for untransformed K4V4 (Table D.3). These completed aggregates use 200 examples each. Rotation and centering change together, so the comparison does not isolate their individual effects. Per-example outputs for this pair are not in the supplied bundle and no paired interval is assigned.

Table D.3: HotpotQA four-bit key variants at matched budgets. KV memory is normalized to dense bf16.

Variant	Token F1	KV
Untransformed K4V4	34.75	0.050
Rotated and centered keys	37.33	0.050

D.3 Ranking overlap under stored precision

In a recording-only diagnostic, the same selection and retention scores are recomputed from bf16 and K4V4 on 30 MATH-500 and 30 HotpotQA examples. The selected top- k sets overlap by 97.44% on average over 60 examples and 64,896 layer/head events, with example-level bootstrap interval [97.12, 97.77]%. The retained top- C sets overlap by 97.65% over 48 examples with an eviction event and 9,968 layer/head events, with interval [97.42, 97.90]%. High set overlap measures rank stability. It does not establish equal attention outputs or answer quality, as the HotpotQA K4V4 quality loss shows.

E Prefill Policies and Long-Prompt Retrieval

We compare full prefill, chunked prefill, and question probe, followed by their interaction with four-bit storage and passage retrieval. Unless stated otherwise, the prefill comparisons measure the maximum KV memory over prompt processing and generation.

E.1 Prefill heuristics

Full prefill applies retention after processing the complete prompt. On HotpotQA, it reaches 38.41 F1 with a maximum KV memory of 0.999 relative to dense bf16. Question probe uses question-conditioned evidence for early retention and reaches 37.16 F1 at 0.144 KV memory. Chunked prefill evicts between 512-token chunks and reaches 26.90 F1 at 0.185 KV memory. All three leave the same normalized resident cache of 0.107 after the response, despite their different memory requirements during prompt processing. These heuristics change ranking and retention timing together. Question probe also reaches 24.10 F1 at 0.125 KV memory on NarrativeQA and 15.35 F1 at 0.129 on MuSiQue.

E.2 Precision boundary for early retention

Applying K4V4 at the same early-retention timing lowers KV memory further but also lowers F1 on all three QA tasks (Table E.1). The bf16 prefill result and four-bit generation-time storage are separate operating points. Their combination requires its own quality evaluation.

Table E.1: Question-conditioned prefill with bf16 and K4V4 ($n = 200$ per task), using the same early-retention timing. F1 is token F1; KV is relative to dense bf16.

Task	KV format	F1	KV
HotpotQA	bf16	37.16	0.144
	K4V4	10.79	0.073
NarrativeQA	bf16	24.10	0.125
	K4V4	5.70	0.067
MuSiQue	bf16	15.35	0.129
	K4V4	6.74	0.069

Table E.2: NIAH passage retrieval ($n = 200$). Accuracy is percent; maximum and resident KV are relative to dense bf16. Resident KV is measured after the response. Bold and underlining rank evicting policies.

Policy	Acc.	Max. KV	Resident 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	82.0	<u>0.9995</u>	<u>0.1042</u>
MASKAHEAD	82.5	<u>0.9995</u>	<u>0.1042</u>
Q-MASKAHEAD	75.5	0.3137	0.0503

E.3 NIAH passage retrieval

On 200 NIAH passage-retrieval examples, DapQ scores 82.0% and MASKAHEAD scores 82.5% at the same 0.1042 KV memory. Full-cache MAGE scores 83.5%. Q-MASKAHEAD scores 75.5% at 0.0503 KV memory (Table E.2). This comparison measures retrieval under compressed storage, with full bf16 prefill and its memory peak near dense.