HashAttention: Semantic Sparsity for Faster Inference

Learning a compact selector for long-context attention

Paper by Aditya Desai, Shuo Yang, Alejandro Cuadron,
Matei Zaharia, Joseph E. Gonzalez, and Ion Stoica

ICML 2025 | arXiv:2412.14468v2
Research discussion | 12 October 2026

Learn 32-bit query/key codes; retrieve tokens; attend to their original KV.

The target is KV traffic during long-context decoding

Research question

How can we identify a small useful set of tokens without first computing every full-dimensional query–key dot product?

- Dense decoding reads a growing KV cache for each new query.
- Important tokens change with the query, even for the same document.
- HashAttention retains the full cache and selects what to read each step.

The 32-bit code is an auxiliary index per token per head.

The reported sparsity factors describe token budgets, not resident KV compression.

Paper, Secs. 1, 4.1, and 6; PDF pp. 1–2, 4–5, 9.

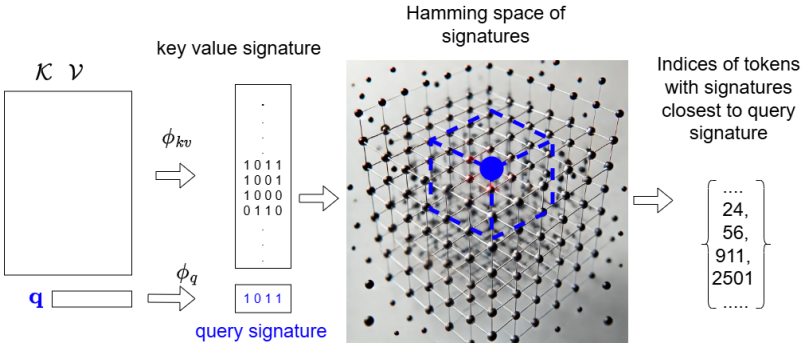
Existing selectors spend their budgets differently

Method	Selection signal	Main tradeoff in this setting
StreamingLLM	Initial and recent tokens	Middle tokens are never retrieved
H2O	Historical attention	Evicted tokens cannot be retrieved later
InfLLM / Quest	Block representatives	A useful token can require an entire block
Double Sparsity	Selected key channels	Channel budget and quantization affect recall
HashAttention	Learned binary codes	Requires model-specific adapter training

Better recall can reduce the expensive attention budget downstream.

Paper, Sec. 2.2 and Appendix A; PDF pp. 3, 12–13.

Learn a retrieval space, then return to the original KV



- Separate networks encode queries and cached keys.
- Nearest signatures provide token indices for sparse attention.
- The general notation includes values; experiments encode keys only.

Paper, Fig. 2 and Sec. 4.1; PDF pp. 4–5. Original author figure.

Only selection is approximate; selected KV stay in full form

Let $z_i = \langle q, k_i \rangle / \sqrt{d}$, with n cached tokens.

$$o = \frac{\sum_{i=1}^{n} e^{z_i} v_i}{\sum_{j=1}^{n} e^{z_j}} \qquad \longrightarrow \qquad \widehat{o} = \frac{\sum_{i \in \mathcal{I}} e^{z_i} v_i}{\sum_{j \in \mathcal{I}} e^{z_j}}$$

Three stages

SCORE: compare compact signatures.
TOPK: select m positions and include fixed windows.
GATHER-ATT: compute and normalize attention on selected KV.

- m is the retrieval budget; $\mathcal{I}$ is the final selected set.
- Normalization changes after selection.

Paper, Sec. 4; PDF p. 4. Denominator index corrected; SDPA scaling made explicit.

A 32-bit signature makes scoring cheap

$$\phi(x) = \mathrm{ReLU}(\mathrm{sign}(\mathcal{F}(x))) \in \{0,1\}^b$$

$$\mathrm{SCORE}(k_i, q) = -\,\mathrm{popcount}(h_i \,\mathrm{XOR}\, h_q)$$

Illustrative 4-bit example

Query: 1011
Key 1011: distance 0
Key 1001: distance 1
Key 0110: distance 3

- 32 bits = 4 bytes per token per head.
- A 128-D FP16 key uses 256 bytes.
- All signatures are still scanned; this is not a sublinear hash-table lookup.

Paper, Eqs. (1)–(2), Sec. 4.1; PDF pp. 4–5. Example and byte comparison are explanatory.

Train the selector locally while freezing the LLM

1. Run the LLM on text chunks; cache keys and collect queries.
2. Label important tokens, for example the true top-64.
3. Train each attention module with weighted binary cross-entropy and Adam.

Make learning feasible

Train with tanh; infer with sign.
At 64K tokens, 64 positives = 0.1%.
$w_+(n) = \alpha + \beta n$ balances labels.

Match the deployment distribution

Generic: concatenated OpenWebText.
Task-specific: HashAttention*.
Training at $\leq$64K did not naturally transfer to 128K.

Paper, Sec. 4.2 and Eq. (4); PDF p. 5.

An isolated-contribution score leads to MIPS

For full-attention weights a_i, consider $s_i = a_i \|v_i\|_2$.

$$\log s_i = \left\langle \frac{q}{\sqrt{d}}, k_i \right\rangle + \log \|v_i\|_2 - \log Z$$

$$\operatorname{rank}(s_i) = \operatorname{rank} \left\langle \left[q/\sqrt{d}, 1 \right], [k_i, \log \|v_i\|_2] \right\rangle$$

- For nonzero values, the ranking reduction is exact.
- The residual argument drops cross-value correlations and preserves the original weights.
- Actual sparse attention renormalizes; the score is a useful proxy rather than a general optimality guarantee.

Paper, Lemmas 4.1–4.2, Appendix B; PDF pp. 5, 13–14. Corrected cosine lift in backup.

Token-level sparsity still needs a careful kernel

- Use paged attention with page size $= 1$.
- Pass selected indices directly; avoid a separate KV gather buffer.
- A token vector can contain enough contiguous bytes for useful memory transactions.

$$T \approx T_{\text{map}} + T_{\text{scan}} + T_{\text{TOPK}} + T_{\text{selected attention}}$$

Budget accounting

The experiments keep the first 128 and most recent 128 tokens.
Example: at 16,384 tokens, $m = 1,024$ plus up to 256 fixed positions means up to 1,280 reads, or 7.81% of the context.

Paper, Sec. 4.4 and evaluation protocol; PDF pp. 5–7. Budget calculation is illustrative.

The protocol models a prepared document plus a short query

Models	Llama-3.1-8B-Instruct; Mistral-7B-Instruct-v0.3
Quality suites	LongBench and RULER
Sparse region	Last 128 prompt tokens for LongBench; last 32 for RULER; all subsequent generation
Dense region	Earlier prompt tokens use full attention
Auxiliary budget	Usually 32 bits per token per attention head
Efficiency	A100; GPT-FAST and FlashDecode; throughput at batch 1

This protocol does not establish sparse prefill quality over the whole document.

Paper, Sec. 5; PDF pp. 6–8.

At 512 retrieved tokens, learned codes compete well

Method	Aux. bits	Score ↑
Full model	–	63.55
Exact top-k oracle	–	63.43
H2O	–	43.47
StreamingLLM	–	33.28
DS	32	61.88
Quest	32	62.36
Quest	64	63.33
HashAttention	**32**	**64.15**

Paper, Table 1 and its caption; PDF p. 6. Scores are averages over six tasks.

Six LongBench categories; Llama-3.1-8B-Instruct.

Table 1 uses task-related training, despite no star in the row name.

25 examples per dataset enter training; MFQA is excluded.

An exact top-k oracle is not an upper bound on task quality.

Generic training preserves most average quality at 16×

Evaluation set	Full	HA-16×	Change
LongBench: 16 non-Chinese tasks	48.78	48.00	−0.78
Without passage-count	51.48	51.08	−0.40
RULER@16K: 13 tasks	90.66	89.53	−1.13
LongBench: 5 Chinese tasks	50.888	48.228	−2.660

- Scores increase with quality; changes are absolute score points.
- English-only adapter training also transfers to Chinese, with a larger loss.
- Aggregate averages hide important task-specific differences.

Paper, Tables 2–3 and 5; PDF pp. 6, 14. Table 5 row order inferred from Sec. 5.1 and Table 4.

Retrieval can survive while aggregation degrades

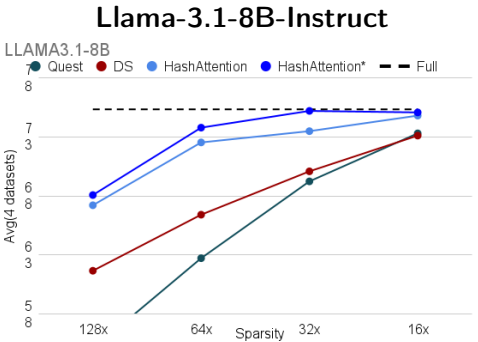
Suite	Task	Full	HA-16×	Change
LongBench	Passage retrieval (en)	100.00	100.00	0.00
LongBench	Passage count	8.29	1.85	−6.44
LongBench	HotpotQA	55.40	51.92	−3.48
LongBench	RepoBench-P	55.31	60.49	+5.18
RULER	Multi-key needle 2	100.00	90.00	−10.00
RULER	Common-word extraction	50.50	62.40	+11.90

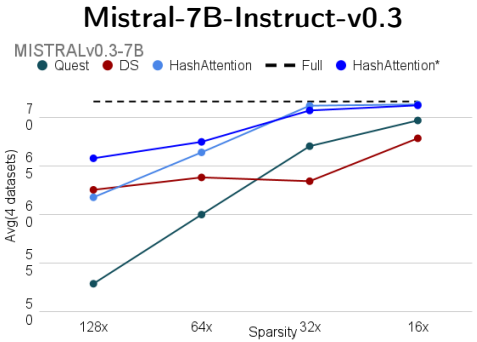
The authors suggest distractor removal as an explanation for gains.

The experiments show the gains; they do not isolate that mechanism.

Paper, Tables 2–3 and Sec. 5.1; PDF pp. 6, 8.

At equal auxiliary memory, fewer tokens can suffice



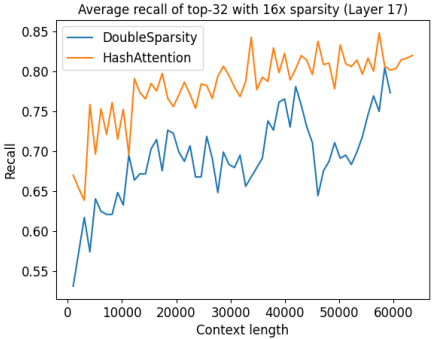


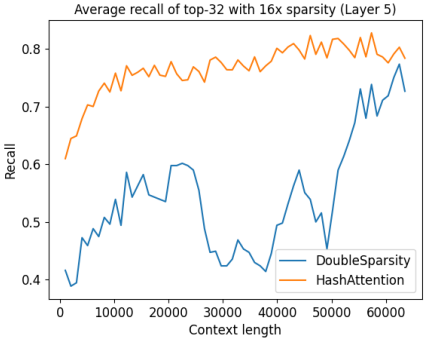
32 bits per token per head; averages over four LongBench tasks.

Llama: generic training approaches full quality at 16×;
task tuning approaches it at 32×. Mistral reaches this region earlier.

Paper, Fig. 3; PDF p. 7. Original figures; vertical-axis digits wrap in the source images.

Higher recall supports the selector-quality explanation





Recall of true top-32 tokens, using a 16× sparse retrieval budget.
Orange: HashAttention. Blue: Double Sparsity.
The two displayed layers favor learned signatures across most lengths.

Paper, Fig. 5; PDF p. 14. Original author panels.

Latency gains emerge after mapping costs are amortized

Framework	Context	Dense (ms)	HA (ms)	Ratio
GPT-FAST	32K	2.03	0.47	4.32×
GPT-FAST	131K	8.09	2.53	3.20×
FlashDecode	32K	0.41	0.37	1.11×
FlashDecode	65K	0.79	0.55	1.44×
FlashDecode	524K	6.24	2.43	2.57×

- A100; 32× sparsity; MLP: $128 \to 128 \to 128 \to 32$.
- Short contexts can lose to the fixed mapping overhead.
- Sequence parallelism makes the FlashDecode baseline stronger.

Values transcribed from Fig. 4(a), PDF p. 8; ratios use rounded labels. Text reports maxima of 4.3× and 2.54×. The figure calls GPT-FAST "GPT-FLASH."

The plotted 3.12× throughput gain occurs near 65K

Context	GPT-FAST (tok/s)	HA-32× (tok/s)	Ratio
32K	7.27	18.28	2.51×
65K	**4.00**	**12.48**	**3.12×**
131K	1.76	4.78	2.72×

- Llama-3.1-8B; A100; batch 1; model generation throughput.
- The paper's prose places the maximum at 32K; the plotted values place it at 65K.
- The authors attribute the later decline in the ratio to TOPK cost.

Paper, Fig. 4(b) and Sec. 5.2; PDF p. 8. Ratios calculated from the figure labels.

Choose speed at a matched quality budget

RULER@32K, Llama-3.1-8B-Instruct, GPT-FAST, A100, batch 1.

Working point	Quality ↑	Change	Throughput ratio
Full model	84.28	–	1.0×
HashAttention, 4× sparse	83.41	−0.87	∼1.3×
HashAttention, 8× sparse	82.56	−1.72	∼1.7×

These joint quality/speed points are more informative for deployment than combining a 16× quality result with a 32× speed result.

Paper, Fig. 1 and Sec. 1; PDF pp. 1–2. Fig. 1 omits DS quantization/dequantization time and the throughput impact of auxiliary-memory size.

Learning the codes matters more than merely hashing

Selector	Bits	Sparsity	Four-task mean ↑
HashAttention	**32**	**16×**	**75.0675**
Random-projection LSH	32	16×	48.8975
Random-projection LSH	256	16×	69.7875
Random-projection LSH	512	16×	68.6350
Random-projection LSH	1024	16×	Incomplete: TriviaQA OOM

- At 32 bits, the average gap is 26.17 score points.
- More random bits do not guarantee a monotonic improvement.
- This compares signatures, not the complete MagicPig algorithm.

Paper, Sec. 5.3 and Table 8; PDF pp. 9, 16.

The useful operating region has concrete conditions

1. **Model and context adaptation**
   Each LLM needs learned mappings; training length and task distribution matter.
2. **Access savings with retained storage**
   Full KV remains on GPU in the reported experiments; CPU offload was not evaluated.
3. **Task-dependent information loss**
   Counting and multi-key retrieval can degrade despite a small average loss.
4. **A remaining selection bottleneck**
   Cheaper scores do not remove the cost of TOPK or attention over selected KV.

Paper, Secs. 4.2, 5.1–5.2, and 6; PDF pp. 5, 6–9.

Better retrieval is a systems resource

Central contribution

Independent learned query/key maps produce compact binary codes
that help preserve attention quality at smaller retrieval budgets.

- Optimize the whole chain: selector quality, auxiliary traffic, and selected attention.
- Read the 16× quality and 32× efficiency results with their own protocols.
- Promising next steps: adaptive budgets, faster discrete-score selection, and offloaded KV.

Desai et al., HashAttention: Semantic Sparsity for Faster Inference.
ICML 2025; arXiv:2412.14468v2, revised 3 June 2025.

Main presentation ends here. Following slides provide derivations and additional results.

Backup: the cosine lift needs a zero query coordinate

Define $x_i = [k_i, \log \|v_i\|_2]$, $u = [q/\sqrt{d}, 1]$,
and choose $M > 0$ with $M \geq \max_i \|x_i\|_2$.

$$P(x_i) = \left[x_i, \sqrt{M^2 - \|x_i\|_2^2} \right], \qquad Q(u) = [u, 0].$$

Then $\|P(x_i)\|_2 = M$ and

$$\cos(P(x_i), Q(u)) = \frac{\langle x_i, u \rangle}{M \|u\|_2},$$

so cosine similarity preserves the inner-product ranking.

Correction to the displayed formula in Sec. 4.3

The paper writes $[q, 1, 1]$. The final 1 adds a key-dependent square-root term and can reverse the ranking. It must be 0 for this lift.

Paper, Sec. 4.3, PDF p. 5. Corrected derivation by the reader; the learned MLP does not implement this explicit lift.

Backup: isolated contribution is not general optimality

Without renormalization, the residual contains cross terms:

$$\left\| \sum_{i \notin \mathcal{I}} a_i v_i \right\|_2^2 = \sum_{i \notin \mathcal{I}} a_i^2 \, \|v_i\|_2^2 + 2 \sum_{\substack{i<j \\ i,j \notin \mathcal{I}}} a_i a_j \langle v_i, v_j \rangle.$$

The appendix drops the second sum. Orthogonal values make this exact.

An illustrative counterexample when correlations matter

$a = (0.45, 0.35, 0.20)$ and scalar values $v = (1, -1, 1)$ give $o = 0.30$.
Keeping the largest contribution, 0.45, leaves error 0.15.
Keeping the smaller contribution, 0.20, leaves error 0.10.

Actual GATHER-ATT also renormalizes, creating a second distinction.

Paper, Appendix B.1, Eqs. (5)–(10), PDF pp. 13–14. Expansion and example are reader analysis.

Backup: RULER task changes have opposite signs

RULER@16K task	Full	HA-16×	Change
Single needle 1 / 2 / 3 (each)	100.00	100.00	0.00
Multi-key needle 2	100.00	90.00	−10.00
Variable tracking	99.20	93.00	−6.20
Common-word extraction	50.50	62.40	+11.90
Frequent-word extraction	90.00	87.33	−2.67
QA 1	87.00	82.00	−5.00
QA 2	54.00	57.00	+3.00

Figure 3 uses a six-task subset: 80.29 vs. 80.12 at 16×.
Table 3 uses all 13 tasks: 89.53 vs. 90.66 at 16×.

Paper, Table 3 and Fig. 3; PDF pp. 6–7. Higher scores are better.

Backup: transfer to Chinese is useful but imperfect

Chinese LongBench task	Full	HA-16×	Change
MultifieldQA-zh	61.36	59.31	−2.05
DuReader	34.51	30.93	−3.58
VCSum	17.14	17.48	+0.34
Passage retrieval-zh	96.86	91.71	−5.15
LSHT	44.57	41.71	−2.86
Mean	50.888	48.228	−2.660

The adapter is trained on English data only.

Retrieval and QA lose more than summarization in this table.

Paper, Table 5 and Sec. 5.1; PDF pp. 14, 8. Row identities inferred from the text and adjacent Table 4.

Backup: more bits help; related pairs become closer

Bit width vs. proxy loss

Bits	Cross-entropy ↓
16	0.243
32	0.212
64	0.204

Evaluation: 250×32K samples.
Main text says top-32; Appendix C.6 says top-64.

Cosine similarity to top-32 keys

Space	Layer 0	Layer 4
Original	−0.1118	−0.1200
Tanh	0.1777	0.4922
Sign	0.1802	0.1792

Related pairs move closer. This statistic alone does not establish equality of the full query/key distributions.

Paper, Tables 7 and 9, Sec. 5.3 and Appendix C.6; PDF pp. 9, 15–16. Cosines rounded to four decimals.

Backup: scoring stays cheap even with wider signatures

Candidates	FP16 dot product	64 bits	128 bits	256 bits	512 bits
262,144	0.130	0.085	0.084	0.087	0.112
1,048,576	0.619	0.087	0.129	0.226	0.429
8,388,608	6.733	0.552	0.960	1.755	3.378
33,554,432	16.610	2.188	3.817	7.005	13.480

- The dot product uses 128-dimensional FP16 vectors.
- Lower is better; the source table does not specify time units.
- This compares score computation, not mapping + TOPK + attention.
- At 1,048,576 candidates, 64-bit scoring is about 7.11× faster.

Paper, Table 6 and Appendix C.3; PDF pp. 14–15. Original numerical values; ratio calculated.