OPENATTENTION: ONE PERCENT IS ENOUGH WITH TAIL PROXIES

Anonymous authors
Paper under double-blind review

ABSTRACT

Sparse attention reads only a small subset of the key-value cache, and how to treat the unread tail remains a key challenge. Discarding the tail biases the output, and sampling it as previous approaches do fails at reads of 5% and below, where the tail likely holds heavy hitters the selector missed. Our key observation is that the simplest possible proxy, one token that folds the entire tail into its count-aware mean, already yields lower output error than a sample at every read behind a practical selector. We further show in closed form that at low reads sampling nearly doubles the weight error of discarding while a single proxy stays below it. Building on these findings, we explore approximating the dense attention distribution with a few proxies that target the tail. We propose OpenAttention, a pipeline of clustering, ErrorGate and top-k selection followed by a count-aware mean proxy. To implement clustering and selection both efficiently and effectively, we design training-free Query-Geometry Clustering (QGC) and a Query-Geometry Selector (QGS), which compress the keys while respecting the query directions. Additionally, unlike prior methods that rank clusters by centroid alone, ErrorGate allocates the exact read by the estimated weight error of each cluster. On RULER-32K HARD with Llama-3.1-8B, OpenAttention retains 99% of dense accuracy at a 1% read. It also achieves state-of-the-art accuracy across models, context lengths and datasets. Our customized attention kernel, OpenAttention.flash, runs the decode attention stage 10× faster than dense attention at a 1% read.

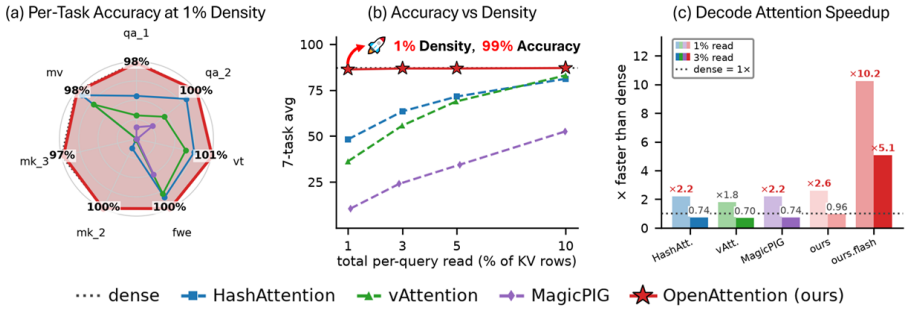


Figure 1: OpenAttention against sparse attention baselines on RULER-32K HARD with Llama-3.1-8B. (a) Per-task accuracy at a 1% read. (b) Mean accuracy against the total per-query read. (c) Decode attention-stage speedup over dense attention at batch 32 and 32K context.

1 INTRODUCTION

Long-context decoding is bound by the memory traffic of the key-value (KV) cache, and sparse attention reduces this cost by computing attention over only a subset of the cache. Existing top-k approaches (Singhania et al., 2024; Yang et al., 2024; Desai et al., 2025) explore frameworks that focus solely on predicting the top-k scores accurately and discard the remaining tail, as shown in Figure 2(a). However, omitting the tail information fundamentally prevents these methods from approximating the dense attention distribution. This limitation becomes especially critical in aggregation tasks (e.g., FWE) that depend on the entire distribution of attention weights. To address this

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

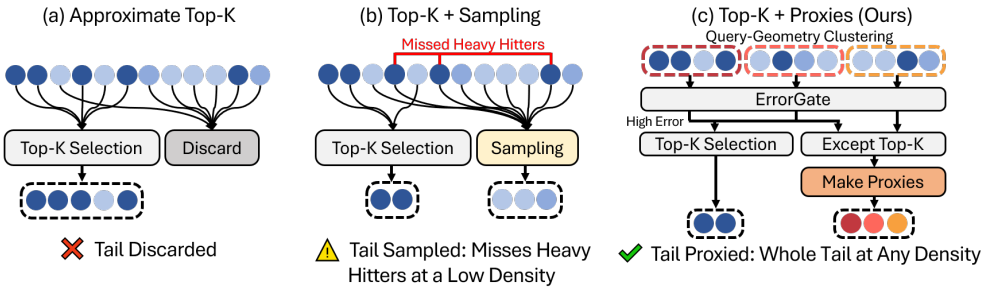


Figure 2: Three treatments of the tail for sparse attention. (a) Approximate top-k relies on top-k accuracy and ignores the tail. (b) Top-k with sampling misses heavy hitters at low density. (c) Ours clusters and represents the tail by proxies built with a method that minimizes the weight error.

problem, as shown in Figure 2 (b), MagicPIG (Chen et al., 2025) estimates the output by locality-sensitive hashing (LSH) based importance sampling, and vAttention (Desai et al., 2026) selects heavy hitters and then approximates the residual tail by a random sample sized to a relative error bound. Both methods presume that the softmax output of the tail can be estimated from a small sample. However, the estimation error of sampling grows sharply as the density decreases.

Our key finding, as shown in Figure 3, is that a random sample complements the tail well at high density but yields a higher error than top-k alone at low density. However, folding the same tail into a single proxy with a simple count-aware mean keeps the error low in every setting we examine. This holds across different attention distributions and densities as well as behind both HashAttention and the oracle top-k selector. Based on this empirical finding, we measure the statistics of the tails left by HashAttention and the oracle top-k. Using these quantities, we show in closed form that a single proxy yields a lower attention weight error than sampling. Consequently, the tail should be approximated by well-constructed proxies rather than by samples, and this work explores how to build and exploit such proxies effectively.

As shown in Figure 2(c), we propose OpenAttention to approximate the tail distribution rather than truncate or sample the tail. OpenAttention constructs tail proxies through a framework that consists of clustering, ErrorGate, top-k selection and a count-aware mean. The effectiveness of this framework depends on how well it minimizes the weight error, which is governed by the mean and variance of each cluster. Two considerations are central to achieving this goal. First, the clustering and the selector must run both effectively and efficiently to form clusters with small variance. Since this variance is measured in attention scores, the keys alone are insufficient and the query geometry must also be taken into account. We therefore compress the key tokens with the query eigenvectors and perform both clustering and top-k selection on the compressed keys. Second, clusters should be selected by their error, which then determines the top-k budget. Unlike prior methods that select clusters only by the mass of their centroid or mean, ErrorGate efficiently approximates the weight error from a mean token and an approximate variance token and uses it to allocate the budget.

With these methods, OpenAttention preserves 99% of dense accuracy at a 1% read, scoring 86.30 against 87.04 for full attention and 48.2 for the best baseline on RULER-32K HARD (Hsieh et al., 2024) with Llama-3.1-8B (Grattafiori et al., 2024). It also achieves state-of-the-art accuracy on DeepSeek-R1-Distill-Llama-8B (DeepSeek-AI, 2025) and Mistral-7B (Mistral AI, 2024) and across context lengths from 16K to 96K. Moreover, unlike prior sampling-based methods (Chen et al., 2025; Desai et al., 2026), OpenAttention targets practical latency gains. Our Triton-based (Tillet et al., 2019) customized kernel, OpenAttention.flash, processes the exact tokens and proxies in one online softmax pass and runs the decode attention stage $10\times$ faster than dense attention at a 1% read.

Our contributions are the following.

- We propose OpenAttention, which approximates the full softmax distribution rather than truncating or sampling the tail. It represents the tail by proxies built through clustering, ErrorGate, top-k selection and a count-aware mean.
- We design QGC and QGS, which compress the key tokens under the query geometry so that clustering and top-k selection follow the attention scores while remaining efficient.

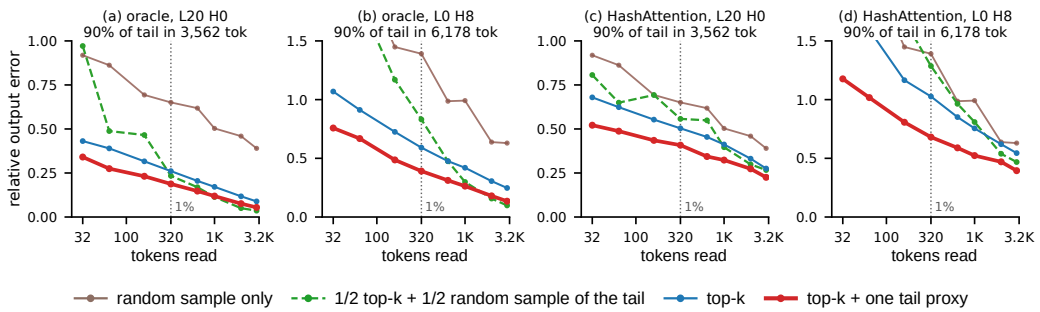


Figure 3: Relative attention-output error versus total tokens read for four tail treatments on two RULER-32K HARD attention heads, using oracle top-k (a,b) or HashAttention (c,d).

- We introduce ErrorGate, which selects clusters and allocates the budget based on an approximated weight error rather than the mass of each cluster alone.
- OpenAttention preserves 99% of dense accuracy at a 1% read on RULER-32K HARD with Llama-3.1-8B. It also achieves state-of-the-art accuracy across different models and context lengths from 16K to 96K.
- We customise a Triton-based FlashAttention-2 (Dao, 2024) decode kernel into OpenAttention.flash, which runs the estimator as one online softmax pass. It accelerates the decode attention stage by 5–10$\times$ over dense attention at 3% and 1% reads.

2 Key Observation on One Simple Proxy

In this section, we examine how random sampling and a single simple proxy treat the tail. First, as shown in Figure 3, we analyze the empirical finding on the relative output error. Second, as shown in Figure 4, we compare the closed-form weight errors of both treatments as functions of the tail statistics under a given top-k selector.

2.1 Relative Output Error Across Densities.

As shown in Figure 3, we measure the relative output error of random sampling and our proxy at different densities. We evaluate two decode heads with different tail distributions, in which 90% of the tail mass lies in 3,562 and 6,178 of the 32K tokens, respectively. Each head is evaluated behind two top-k selectors, namely HashAttention and the oracle top-k. The sampling method splits the read equally between top-k and a random sample of the tail, while the proxy method replaces the entire tail with a single proxy. The sampling method improves on its top-k selector at high density but performs worse than the selector alone at low density. This indicates that a small sample cannot approximate the tail distribution as more tokens are left in the tail. In contrast, the single proxy reduces the error even in this simple form. Behind HashAttention it yields the lowest error at every density, and behind the oracle top-k it does so at low density. Based on this empirical finding, we analyze the two treatments mathematically in the following subsection.

2.2 Attention Weights and Tail Approximation Error

Let $w_i = e^{z_i}$ with $z_i = d^{-1/2} q^\top k_i$. An exact set X leaves tail $T = [n] \setminus X$, $N = |T|$, and $Z_X = \sum_{i \in X} w_i$. For a uniform sample $B \subseteq T$ of size m, let $\eta = N/m$ and $g_T = e^{\bar{z}_T}$. Following the deterministic and randomized SDPA formulations of Desai et al. (2026), discarding, sampling and one proxy assign

$$\hat{a}_i^{\mathrm{d}} = \frac{\mathbf{1}_X(i) w_i}{Z_X}, \qquad \hat{a}_i^{\mathrm{s}} = \frac{[\mathbf{1}_X(i) + \eta \mathbf{1}_B(i)] w_i}{\hat{Z}_{\mathrm{s}}}, \qquad \hat{a}_i^{\mathrm{p}} = \frac{\mathbf{1}_X(i) w_i + \mathbf{1}_T(i) g_T}{\hat{Z}_{\mathrm{p}}}, \tag{1}$$

where $\hat{Z}_{\mathrm{s}} = Z_X + \eta \sum_{j \in B} w_j$ and $\hat{Z}_{\mathrm{p}} = Z_X + N g_T$. The proxy is equivalently one virtual token with logit $\bar{z}_T + \log N$ and mean value $\bar{v}_T$.

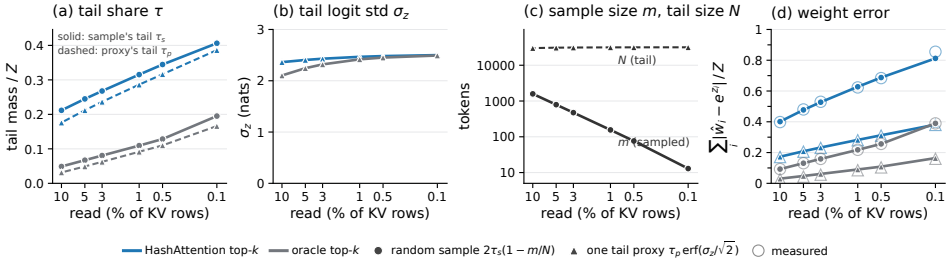


Figure 4: Weight-discrepancy comparison behind HashAttention and oracle top-k. Panels show the tail share, logit spread, sample/tail sizes, and the predicted and measured discrepancies of Eq. (2).

For the implied unnormalised weights define $\mathcal{E} = Z^{-1}\sum_i |\hat{w}_i - w_i|$, $Z = \sum_i w_i$, and $\tau = \sum_{i\in T} w_i/Z$. Then

$$\mathcal{E}_{\mathrm{d}} = \tau, \qquad \mathbb{E}[\mathcal{E}_{\mathrm{s}}] = 2\tau\left(1 - \frac{m}{N}\right), \qquad \mathcal{E}_{\mathrm{p}}^{\mathrm{G}} = \tau \operatorname{erf}\left(\frac{\sigma_z}{\sqrt{2}}\right). \tag{2}$$

The first two are exact for a fixed tail (the second in expectation); the last is a Gaussian-population prediction. Under the fixed unnormalised-weight convention above, sampling approaches twice the discard discrepancy when $m \ll N$; this is not a factor-of-two claim for normalized attention or output error.

Figure 4 shows, however, that the proxy and discard are much closer in $\mathcal{E}$ than their output errors in Figure 3. Thus $\mathcal{E}$ diagnoses unnormalised weight distortion but does not by itself explain sampling's density-dependent output failure. That dependence is captured by the sampling-normalizer uncertainty, which grows as the sample shrinks; missed high-weight tokens can also perturb the value direction (App. A.2). The exact output identity in App. A explains why similar weight discrepancy need not imply similar output error.

To connect this observation to the method, let $\mathcal{P}$ be a partition of the unread tokens into nonempty residual clusters, with sizes n_b and mean logits $\bar{z}_b$. Their exact discrepancy decomposes as

$$\mathcal{E} = \frac{1}{Z}\sum_{b\in\mathcal{P}} \hat{M}_b D_b, \qquad \hat{M}_b = n_b e^{\bar{z}_b}, \qquad D_b = \frac{1}{n_b}\sum_{i\in b} |e^{z_i - \bar{z}_b} - 1|. \tag{3}$$

Each contribution $\varepsilon_b = \hat{M}_b D_b$ is proxy mass times weight spread; it vanishes for equal logits and is removed by reading the entire cluster exactly. Under the Gaussian model, D_b is estimated by $D_{\mathrm{G}}(\sigma_b) \approx 0.8\sigma_b$ at small within-cluster logit standard deviation σ_b. Writing $o = \sum_i a_i v_i$, $\hat{o} = \sum_i \hat{a}_i v_i$ and $\hat{Z} = \sum_i \hat{w}_i > 0$, we also have $\|\hat{o} - o\|_2 \le (Z/\hat{Z})\mathcal{E}\max_i \|v_i - o\|_2$, without Gaussian or weight–value independence assumptions (App. A). Section 3 develops this multi-cluster construction using QGC grouping, ErrorGate discrepancy-based allocation and QGS selection within clusters.

3 OpenAttention: Approximate Tail Distribution by Proxies

3.1 Overall Process of OpenAttention

As shown in Figure 5 (a), OpenAttention clusters the cache once per request into C clusters under the query geometry (QGC). Each cluster caches its size n_b, a *mean token* $(\bar{k}_b, \bar{v}_b)$ and a *variance token* $\operatorname{diag}\operatorname{Var}_b(k)$. For each query, two small matrix products give every $\bar{z}_b$ and σ_b^2, from which ErrorGate estimates the error each proxy would leave. The exact budget E is split in proportion to this estimate, and each cluster reads its highest-scoring tokens under the QGS. The remaining members of each cluster then take the role of the tail T in $\hat{a}_i^{\mathrm{p}}$ of Eq. (1). Each cluster thus enters the softmax as one virtual token with the mean value and the logit $\bar{z}_b + \log n_b$ of its remaining members. The exact set X now includes the initial sink, local-window and selector top-k tokens as well as the E allocated reads. The residual-proxy estimator in Eq. (8) merges all exact tokens and proxies in one online softmax. The initial exact set is small in the deployed 1% configuration, and ErrorGate allocates the expansion budget rather than eliminating the initial selection stage.

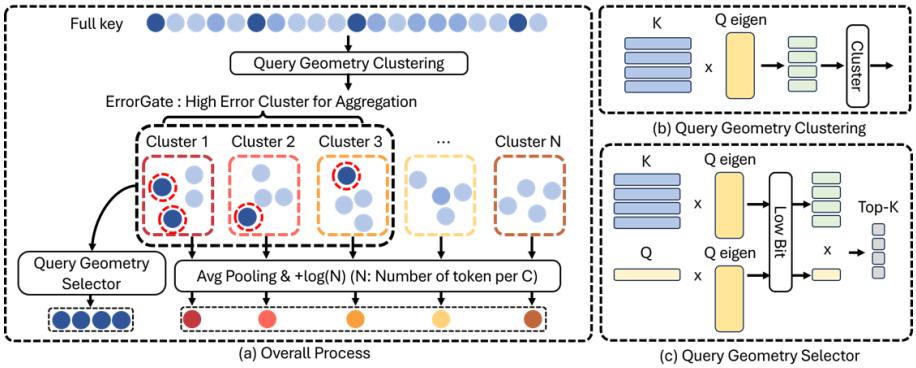


Figure 5: Overview of OpenAttention. (a) QGC clusters the cache once per request. ErrorGate reads the top tokens of high-error clusters via the QGS, and the rest of each cluster becomes one count-aware mean logit proxy. (b) QGC clusters the keys projected onto the query eigenvectors. (c) QGS scores the top-k with low-bit projections of the keys and query.

3.2 Query Geometry Clustering and Selection

Both QGC and QGS use the top-r eigenvectors of the queries to account for the query geometry when clustering and selecting key tokens. The queries used to obtain these eigenvectors combine pre-sampled query tokens ($\bar{M}_{\text{st}}$) with the query tokens of the current question chunk ($\bar{M}_q$) as $M = \frac{1}{2}\bar{M}_{\text{st}} + \frac{1}{2}\bar{M}_q$, and the top-$r$ eigenpairs (U_r, Λ_r) of M are used. These eigenpairs are computed once at the prefill stage. Clustering is then performed once after prefill, while selection reuses them at every decoding step. Therefore, as shown in Figure 6, we conduct an experiment to verify how well the M obtained at prefill is preserved across decoding steps. The median absolute cosine similarity between a decode query and the leading eigenvector of M is 0.82, which confirms that the query geometry is well preserved.

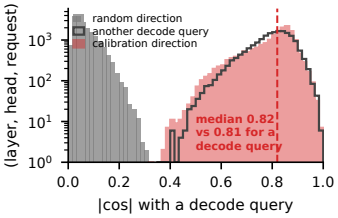


Figure 6: $|\cos|$ of a decode query with the leading eigenvector of M, another decode query and a random direction.

3.2.1 Query Geometry Clustering

Rather than grouping tokens by their keys alone, QGC aims to group the tokens whose *attention scores* will be close, taking the geometry of the queries into account. Since $\mathbb{E}_q(z_i - z_j)^2 = s^2(k_i - k_j)^\top M(k_i - k_j)$, minimizing the within-cluster logit variance reduces to k-means under the metric M. Therefore, as shown in Figure 5(b), QGC projects the keys onto the query eigenvectors and performs clustering in this projected space. With $K \in \mathbb{R}^{n\times d}$ the keys of a head and (U_r, Λ_r) the top-r eigenpairs of M, r=16,

$$\Phi = K\,U_r\Lambda_r^{1/2} \in \mathbb{R}^{n\times r}, \qquad a_i = \arg\min_b \|\phi_i - \mu_b\|^2, \qquad \mu_b = \frac{1}{n_b}\sum_{i:\,a_i=b}\phi_i, \tag{4}$$

where $\phi_i^\top$ denotes the i-th row of Φ. Under this projection, the Euclidean distance between two projected keys matches their expected logit distance within the top-r subspace. A few Lloyd iterations of the assignment and centroid steps in Eq. (4) then yield the C clusters. Each cluster then caches its count n_b, a mean token $(\bar{k}_b, \bar{v}_b)$ of its original keys and values and a variance token $\operatorname{diag}\operatorname{Var}_b(k)$ of per-dimension key variances.

Table 1: RULER-32K HARD mean accuracy by clustering method at a 1% read (C=64).

clustering	dim	avg
token position	–	84.37
k-means on keys only	128	85.42
QGC (ours)	**16**	**86.30**

Validation of QGC. To validate the role of query geometry, we compare QGC with clustering by token position and with clustering on uncompressed keys alone (Hu et al., 2026; Chen et al., 2026). As shown in Table 1, QGC is the most effective of the three while clustering in only 16 dimensions.

3.2.2 QUERY GEOMETRY SELECTOR

As shown in Figure 5(c), QGS uses the eigenvectors of the same M at rank r=32. Because selection runs at every decoding step, each key is coded once in low bits as its projection onto the query subspace to reduce latency. A query projected onto the same subspace then scores these codes as

$$c_i = \mathrm{Q}_8\big(U_r^\top k_i\big), \qquad \tilde{z}_i = (U_r^\top q)^\top c_i, \quad (5)$$

where Q_8 is int8 quantization with one scale per head and coordinate. The score $\tilde{z}_i \approx q^\top U_r U_r^\top k_i$ is the logit restricted to the query subspace, and its top-k set $\mathcal{T} = \text{top-}k(\tilde{z})$ is read exactly. One U is shared by all query heads of a KV head, and new keys are coded on arrival at $8r$ bits per token.

Table 2: Top-32 logit recall at $1/16$ and $1/32$ reads and selection time per layer on one H100.

selector	16×	32×	ms
SparQ	0.96	0.94	0.29
Loki	0.84	0.75	0.18
Quest	0.77	0.66	0.33
Double Sparsity	0.97	0.95	0.50
Squeezed Attention	0.92	0.86	0.23
HashAttention	0.74	0.63	0.24
QGS (ours)	**0.97**	**0.94**	0.22

Top-k accuracy. We compare QGS with six published top-k selectors on the same decode rows using their released configurations. We keep the first and last 128 tokens at a $1/16$ read and 64 each at a $1/32$ read. The evaluation covers all layers and heads over the first eight decode steps of 21 samples. As shown in Table 2, QGS achieves high top-k recall at a low selection time, giving a favorable balance between accuracy and cost.

3.3 ERRORGATE

Estimating cluster discrepancy. Equation (3) gives the exact contribution $\varepsilon_b = \hat{M}_b D_b$, $\hat{M}_b = n_b e^{\bar{z}_b}$. A Gaussian model replaces D_b by $D_{\mathrm{G}}(\sigma_b) = e^{\sigma_b^2/2}(2\Phi(\sigma_b) - 1)$; diagonal key moments approximate the exact variance $\sigma_b^2 = s^2 q^\top \mathrm{Cov}_b(k) q$ (Corollary 2). The deployed score is

$$\begin{aligned} \hat{\varepsilon}_b &= n_b e^{\bar{z}_b}\, e^{\hat{\sigma}_b^2/2}\big(2\Phi(\hat{\sigma}_b) - 1\big) = \hat{M}_b D_{\mathrm{G}}(\hat{\sigma}_b), \\ \bar{z}_b &= s\, q^\top \bar{k}_b, \qquad \hat{\sigma}_b^2 = s^2 \textstyle\sum_d q_d^2 \,\mathrm{Var}_b(k_d). \end{aligned} \quad (6)$$

It uses cached mean and diagonal variance tokens and one error function per cluster. The signed mean enters through $e^{\bar{z}_b}$, and $D_{\mathrm{G}}(\hat{\sigma}_b) \approx 0.8\hat{\sigma}_b$ at small dispersion. Thus $\sum_b \hat{\varepsilon}_b$ estimates $Z\mathcal{E}$ rather than equalling it. The score is neither an exact finite-cluster discrepancy nor a certified output bound; no independence assumption between weights and values is used. The signed mass factor is retained: for fixed cluster size and dispersion, $e^{\bar{z}_b}$ distinguishes a cluster the query aligns with from one it looks away from, whereas a statistic based on $\bar{z}_b^2$ does not. As shown in Table 3, ranking clusters by the estimated error of Eq. (6) yields 86.30 at a 1% read, whereas ranking by the centroid mass as in prior methods (Chen et al., 2026; Hu et al., 2026) yields 85.99. This shows that accounting for within-cluster dispersion selects clusters more effectively than centroid mass alone.

Table 3: ErrorGate ranking ablation at a 1% read on RULER-32K HARD.

ranking statistic	avg
mean logit (mass)	85.99
variance	84.81
estimated error (ours)	**86.30**

Validity of the diagonal estimate. The mean and variance tokens cost 1.5 KV rows per cluster per query, charged to the read. On the C=64 QGC diagnostic, diagonal and full-variance model scores agree at Spearman 0.92 (Fig. 7) and recover the eight largest exact-discrepancy clusters at similar rates (91%, 97%). These measurements assess the diagonal substitution empirically; they are not a general variance inequality.

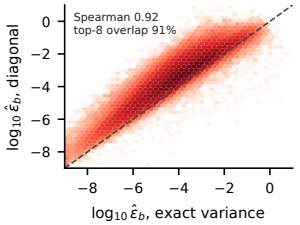


Figure 7: ErrorGate estimates from diagonal and full within-cluster logit variance on 137,216 decode rows.

Hierarchical budget. For a fixed number of complete clusters, removing the largest exact ε_b terms maximally reduces $\mathcal{E}$. Deployment instead uses estimated scores and a token budget. Motivated by the measured concentration of discrepancy (App. A.1), we allocate

$$q_b = \min\left(n_b, \lambda \frac{\hat{\varepsilon}_b}{\max_{b'} \hat{\varepsilon}_{b'}}\right), \qquad \sum_b q_b = E, \quad (7)$$

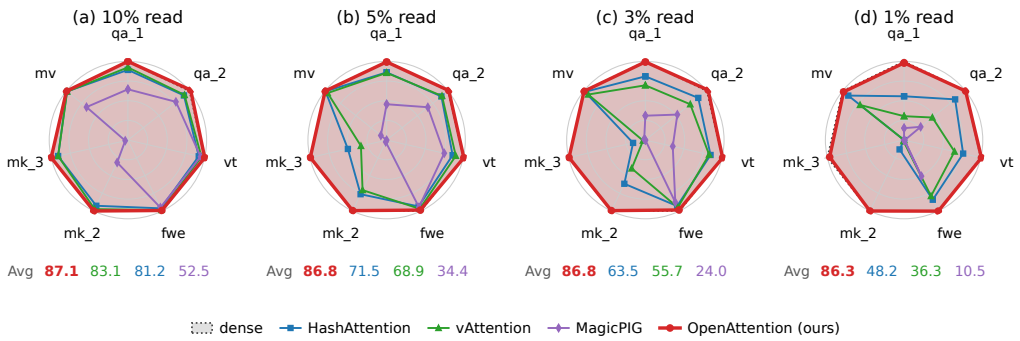


Figure 8: RULER-32K HARD with Llama-3.1-8B-Instruct at matched total per-query reads of 10%, 5%, 3%, and 1%, including the proxy stream. Each panel shows per-task accuracy relative to dense (grey, 100%), with mean accuracy below. Dense has a mean of 87.04 on RULER hard. MagicPIG realises reads of 10.0%, 5.1%, 2.9%, and 1.1%. Per-task results are in App. E.

with λ set by water filling. Leftover tokens after flooring go to the highest-ranked clusters with capacity so that the budget E is spent exactly. Each cluster then reads its q_b highest QGS-scored tokens and recomputes its residual proxy statistics. This proportional allocation is a heuristic rather than an optimal rule. Scores estimate discrepancy before allocation; Proposition 1 applies after residual recomputation. Allocation details are in App. A.1.

3.4 OpenAttention.flash: OpenAttention on FlashAttention-2.

Because a proxy has the same form as a token, we customise a Triton-based FlashAttention-2 decode kernel into OpenAttention.flash (Algorithm 1 in App. C), which processes the exact tokens and the proxies in one online softmax pass. Each program gathers, in tiles of 64, the selected tokens, the tokens read from expanded clusters and the $(\bar{k}_b, \bar{v}_b)$ of the clusters with $+\log n_b$ on the logit. Pooled sums are cached per cluster, so removing the tokens a query reads costs one subtraction per token the kernel already loads. The work per step is fixed, with no data dependent loop bound and no host synchronisation.

4 Experiments

Setup. We use Llama-3.1-8B-Instruct (bf16, 32 query and 8 KV heads) on RULER-32K HARD, the seven tasks that HashAttention and vAttention identify as challenging for sparse attention (qa_1, qa_2, vt, fwe, niah_multikey_2/3, niah_multivalue), with 200 samples per task. Following HashAttention and vAttention, the context is prefilled densely, sparse attention applies to the question chunk and generation, and the baselines share a sink and a local window of 0.1% each. The baselines run in one framework on the HashAttention selector with the public signature weights, vAttention under its authors' per budget protocol and MagicPIG with (K, L) realising the target read (App. D, F). OpenAttention uses the QGS, QGC, and Eq. (6) with the hierarchical budget. Sink and local remain at their few-token floor (0.01% each). *Budget accounting.* We charge every stream a query reads in KV rows per query head, including the mean and variance tokens of all C clusters. The index scan is excluded from the budget for every method. The transfer experiments on other models, datasets and context lengths use the conventional 3% split (App. D).

4.1 Comparison with Other Methods across Read Densities

As shown in Figure 8, we compare per-task accuracy on RULER-32K HARD at densities from 10% to 1%. HashAttention discards the tail while MagicPIG and vAttention rely on sampling, and all three degrade sharply as the density decreases. vAttention slightly improves on HashAttention at a 10% density but falls below it at lower densities even though both share the same selector. In contrast, OpenAttention preserves accuracy close to dense attention at every density. This demonstrates that approximating the tail with proxies is more effective than sampling it.

Table 4: Ablation at a 5% read on RULER-32K HARD (dense 87.04).

proxies	ErrorGate	QGS	avg
×	×	×	67.35
✓	×	×	77.91
✓	✓	×	85.90
✓	×	✓	86.73
✓	✓	✓	**86.82**

Table 5: Mean accuracy on RULER-32K HARD at a 3% read on three models.

method	Llama-8B	DeepSeek-8B	Mistral-7B
dense	87.04	73.46	65.46
HashAttention	63.48	59.85	51.24
vAttention	57.15	49.26	46.14
OpenAttention	**86.59**	**73.49**	**64.83**

4.2 Ablation Study

Table 4 compares the effects of proxies, ErrorGate, and QGS under the same 5% total-read budget. The configuration without any of the three components corresponds to vAttention on the HashAttention selector. With HashAttention as the selector, using proxies instead of residual sampling raises mean accuracy on RULER-32K HARD from 67.35 to 77.91. Adding ErrorGate alone to the proxy variant yields 85.90, while adding QGS alone yields 86.73. Enabling both yields 86.82. Table 3 holds the deployed 1% configuration fixed and varies only the ErrorGate ranking statistic. Ranking by the estimated error of Eq. (6) yields 86.30 mean accuracy on RULER-32K HARD, compared with 85.99 for mean logit and 84.81 for variance alone.

4.3 Evaluation on various models

To evaluate OpenAttention on models other than Llama-3.1-8B-Instruct, we run the same comparison on DeepSeek-R1-Distill-Llama-8B and Mistral-7B-Instruct-v0.3. All methods are evaluated at a 3% density. As shown in Table 5, OpenAttention stays within 0.7 points of dense on all three models while the baselines lose up to 29.9 points. These results show that OpenAttention effectively preserves accuracy across different models.

4.4 Orthogonality to KV Cache Compression

Sparse attention is closely related to KV cache compression but can be applied orthogonally to it. We therefore apply OpenAttention on top of three representative compressors, PyramidKV (Cai et al., 2025), SnapKV (Li et al., 2024b) and StreamingLLM (Xiao et al., 2024b). OpenAttention then reads 10% of the original context from the entries each compressor retains. As shown in Table 6, adding OpenAttention changes the mean accuracy by only -0.14 points on average. KV cache compression saves memory by permanently evicting keys but loses accuracy under a small budget, whereas sparse attention keeps every entry accessible and preserves accuracy.

4.5 Context length

Table 7 compares mean accuracy at a matched 3% total read as the context grows from 16K to 96K, using 100 samples per task. Dense accuracy falls from 87.14 to 78.01 over this range while OpenAttention changes from 87.49 to 78.10. The difference between them remains below 0.7 points at every length, and OpenAttention is numerically higher at 16K and 96K. At 96K HashAttention scores 27.99 while vAttention scores 40.20 and MagicPIG scores 33.19. These results show that OpenAttention maintains accuracy close to dense attention across context lengths even as the other sparse methods fall substantially behind. Per-task scores and actual read rates are reported in App. I.

5 Related work

Top-k selectors. Quest (Tang et al., 2024) and InfLLM (Xiao et al., 2024a) score tokens at a coarse granularity through page-level bounds and block representatives. Learning-to-hash (Sun et al., 2022) and HashAttention (Desai et al., 2025) are hashing-based methods that map queries and keys to compact codes and retrieve tokens by code similarity. Loki (Singhania et al., 2024) and Squeezed Attention (Hooper et al., 2025) rely on the keys alone and score tokens in a PCA subspace of the keys or through offline key clusters. PQCache (Zhang et al., 2025) applies product quantization to the

Table 6: KV cache compression with sparse reading on RULER-32K HARD. Each compressor retains 2048 entries.

compressor	alone	+ OpenAttention
dense attention	87.04	–
StreamingLLM	28.00	28.28
SnapKV	69.98	69.49
PyramidKV	70.34	70.14

Table 7: Mean accuracy on RULER-32K HARD at a 3% read as the context grows. OpenAttention scales only C with the length.

method	16K	32K	64K	96K
dense	87.14	87.75	85.49	78.01
MagicPIG	46.91	51.38	45.73	33.19
HashAttention	67.41	63.87	49.65	27.99
vAttention	52.69	55.63	54.79	40.20
OpenAttention (ours)	87.49	87.38	84.82	78.10

keys and retrieves tokens by approximate inner product search. In contrast, RetrievalAttention (Liu et al., 2025a), SparQ (Ribar et al., 2024) and Double Sparsity (Yang et al., 2024) exploit query information to process the keys. RetrievalAttention builds a key index that accounts for the query distribution, while SparQ and Double Sparsity score the keys on channels chosen from the query at each step or by offline calibration. QGS follows this line of work. However, it differs in that it projects both the keys and the query onto the leading eigenvectors of the query second moment M. The same eigenvectors also define the metric of QGC, so selection and clustering share one query geometry. Although accurate top-k prediction is essential, top-k selection alone cannot account for the distribution of the tail tokens.

Sampling the tail. Rather than discarding the tail, MagicPIG (Chen et al., 2025) estimates the attention output by importance sampling, drawing keys through LSH collisions and weighting them by their collision probabilities. vAttention (Desai et al., 2026) first selects the top-k tokens and then approximates the remaining tail with a random sample. Its sample size is set from a central limit theorem bound so that the estimate meets a user-specified error tolerance. SANTA (Lee et al., 2026) samples value rows after the softmax. However, at a low read these samples can miss the heavy hitters a practical selector leaves in the tail, so we instead summarize every tail token with proxies. SparQ (Ribar et al., 2024) also folds the unread tokens into one mean value vector but weights it by the leftover approximate score mass rather than a count-aware mean logit.

Clustering and centroid proxies. Clustering has been used for both KV cache compression and sparse attention. For compression, CentroidKV (Hu et al., 2026) merges each cluster of similar keys into a single centroid, while ClusterKV (Liu et al., 2025b) keeps every token and recalls whole clusters by the attention weight of their centroids. For sparse attention, Squeezed Attention (Hooper et al., 2025) clusters the keys of a fixed context offline and loads the keys whose centroids match the query. Double-P (Ni et al., 2026) estimates top-p attention mass at the cluster level from size-weighted centroids and refines it at the token level only when needed. However, these methods cluster by key similarity alone, which limits how far they can reduce the variance of attention scores within each cluster. In addition, their cluster selection relies on centroid mass, which ignores the error of representing a cluster by its centroid. In contrast, QGC clusters the keys under the query geometry, and ErrorGate selects clusters and allocates the budget by the estimated weight error.

6 Conclusion and Limitations

We propose OpenAttention, which approximates the information of the tail with proxies to address the loss of tail information in sparse attention. OpenAttention approximates the tail logits effectively through a framework of clustering, ErrorGate, top-k selection and a count-aware mean. To optimize each stage, it exploits the query geometry and minimizes the estimated weight error. As a result, OpenAttention preserves 99% of dense accuracy at a 1% read and achieves state-of-the-art accuracy across benchmarks and models.

Despite its effectiveness, ErrorGate approximates the within-cluster variance with its diagonal terms for efficiency. This approximation agrees closely with the full-variance estimate at a Spearman correlation of 0.92 but cannot recover the exact error. We believe that reducing this proxy error within the OpenAttention framework is a promising direction for future work.

AI USE STATEMENT

In this work, we used generative AI tools to edit the text of the paper for clarity and readability, to translate parts of the authors' draft into English, to help search for and identify related literature, to help write and format its LaTeX source, including tables, captions and references, and to review the manuscript for consistency between the numbers reported in the text, tables and appendices. We have not used generative AI tools to propose or revise research hypotheses, to develop the method, its theoretical analysis or conceptual framework, to formulate the mathematical claims, to provide key ingredients of their proofs or to write the proofs, to design the experiments or provide feedback on the methodology or experiments, to implement the method or the baselines, to generate, clean or reformat datasets, including the synthetic 64K and 96K RULER sets, or to interpret the results. Qualitative or thematic analysis is not applicable to this work. The authors checked every AI-suggested edit, including translations, against our experiment logs and source code, and read every AI-suggested reference before citing it. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

ETHICS STATEMENT

This work does not involve human subjects or personal data. All models and benchmarks we use are publicly available and are used under their licenses, and the 64K and 96K RULER sets we generate are synthetic. Like any approximate attention method, ours can change model outputs, so its accuracy should be validated on the target task before deployment.

REPRODUCIBILITY STATEMENT

The method is defined in §3. QGC is given in Eq. (4), the QGS in Eq. (5), the ErrorGate score in Eq. (6) and the budget allocation in Eq. (7). The read accounting and the evaluation protocol are given in §4. The proofs and their assumptions are in App. A, A.1 and A.2. The deployed hyperparameters and the per-budget grid are in App. D, and per-task results at every budget are in App. E. The baseline configurations are in App. F, F.1 and F.2. The kernel is described in App. C and Algorithm 1, and the generation of the 64K and 96K RULER sets in App. I. All models and benchmarks are publicly available, and all accuracy numbers use greedy decoding with fixed prompts. Code will be released with the final version.

REFERENCES

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

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

Zefan Cai, Yichi Zhang, Bofei Gao, Yuliang Liu, Yucheng Li, Tianyu Liu, Keming Lu, Wayne Xiong, Yue Dong, Junjie Hu, and Wen Xiao. PyramidKV: Dynamic KV cache compression based on pyramidal information funneling. In *Conference on Language Modeling (COLM)*, 2025. arXiv:2406.02069.

Yaoqi Chen, Jinkai Zhang, Baotong Lu, Qianxi Zhang, Chengruidong Zhang, Jing Liu, Jingjia Luo, Di Liu, Huiqiang Jiang, Qi Chen, Bailu Ding, Xiao Yan, Jiawei Jiang, Chen Chen, Mingxing Zhang, Cheng Li, Yuqing Yang, Fan Yang, and Mao Yang. RetroInfer: A vector storage engine for scalable long-context LLM inference. *Proceedings of the VLDB Endowment*, 19(5):1016–1031, 2026. arXiv:2505.02922.

Zhuoming Chen, Ranajoy Sadhukhan, Zihao Ye, Yang Zhou, Jianyu Zhang, Niklas Nolte, Yuandong Tian, Matthijs Douze, Léon Bottou, Zhihao Jia, and Beidi Chen. MagicPIG: LSH sampling for efficient LLM generation. In *International Conference on Learning Representations (ICLR)*, 2025. arXiv:2410.16179.

Tri Dao. Flashattention-2: Faster attention with better parallelism and work partitioning. In *International Conference on Learning Representations (ICLR)*, 2024.

DeepSeek-AI. DeepSeek-R1: Incentivizing reasoning capability in LLMs via reinforcement learning, 2025. URL https://arxiv.org/abs/2501.12948.

Aditya Desai, Shuo Yang, Alejandro Cuadron, Matei Zaharia, Joseph E. Gonzalez, and Ion Stoica. HashAttention: Semantic sparsity for faster inference. In *International Conference on Machine Learning (ICML)*, 2025. arXiv:2412.14468.

Aditya Desai, Kumar Krishna Agrawal, Shuo Yang, Alejandro Cuadron, Luis Gaspar Schroeder, Matei Zaharia, Joseph E. Gonzalez, and Ion Stoica. vAttention: Verified sparse attention via sampling. In *International Conference on Learning Representations (ICLR)*, 2026. arXiv:2510.05688.

Aaron Grattafiori et al. The Llama 3 herd of models, 2024. URL https://arxiv.org/abs/2407.21783.

Coleman Hooper, Sehoon Kim, Hiva Mohammadzadeh, Monishwaran Maheswaran, Sebastian Zhao, June Paik, Michael W. Mahoney, Kurt Keutzer, and Amir Gholami. Squeezed attention: Accelerating long context length LLM inference. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 32631–32652, 2025. doi: 10.18653/v1/2025.acl-long.1568. arXiv:2411.09688.

Cheng-Ping Hsieh, Simeng Sun, Samuel Kriman, Shantanu Acharya, Dima Rekesh, Fei Jia, and Boris Ginsburg. RULER: What's the real context size of your long-context language models? In *Conference on Language Modeling (COLM)*, 2024. arXiv:2404.06654.

Jie Hu, Shengnan Wang, Yutong He, Ping Gong, Jiawei Yi, Juncheng Zhang, Youhui Bai, Renhai Chen, Gong Zhang, Cheng Li, and Kun Yuan. CentroidKV: Efficient long-context LLM inference via KV cache clustering. *Transactions on Machine Learning Research*, 2026. arXiv:2506.11418.

Kyle Lee, Corentin Delacour, Kevin Callahan-Coray, Kyle Jiang, Can Yaras, Samet Oymak, Tathagata Srimani, and Kerem Y. Camsari. Stochastic sparse attention for memory-bound inference. In *International Conference on Machine Learning (ICML)*, 2026. arXiv:2605.01910.

Jiaqi Li, Mengmeng Wang, Zilong Zheng, and Muhan Zhang. LooGLE: Can long-context language models understand long contexts? In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 16304–16333, 2024a. doi: 10.18653/v1/2024.acl-long.859. URL https://aclanthology.org/2024.acl-long.859/.

Yuhong Li, Yingbing Huang, Bowen Yang, Bharat Venkitesh, Acyr Locatelli, Hanchen Ye, Tianle Cai, Patrick Lewis, and Deming Chen. SnapKV: LLM knows what you are looking for before generation. In *Advances in Neural Information Processing Systems*, 2024b.

Chin-Yew Lin. ROUGE: A package for automatic evaluation of summaries. In *Text Summarization Branches Out*, pp. 74–81, 2004. URL https://aclanthology.org/W04-1013/.

Di Liu, Meng Chen, Baotong Lu, Huiqiang Jiang, Zhenhua Han, Qianxi Zhang, Qi Chen, Chengruidong Zhang, Bailu Ding, Kai Zhang, Chen Chen, Fan Yang, Yuqing Yang, and Lili Qiu. RetrievalAttention: Accelerating long-context LLM inference via vector retrieval. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2025a. arXiv:2409.10516.

Guangda Liu, Chengwei Li, Jieru Zhao, Chenqi Zhang, and Minyi Guo. ClusterKV: Manipulating LLM KV cache in semantic space for recallable compression. In *Proceedings of the 62nd ACM/IEEE Design Automation Conference (DAC)*, 2025b. doi: 10.1109/DAC63849.2025.11132479. arXiv:2412.03213.

Mistral AI. Mistral-7B-Instruct-v0.3. Hugging Face model card, 2024. URL https://huggingface.co/mistralai/Mistral-7B-Instruct-v0.3.

Wentao Ni, Kangqi Zhang, Zhongming Yu, Oren Nelson, Mingu Lee, Hong Cai, Fatih Porikli, Jongryool Kim, Zhijian Liu, and Jishen Zhao. Double-P: Hierarchical top-p sparse attention for long-context LLMs. *arXiv preprint arXiv:2602.05191*, 2026.

Luka Ribar, Ivan Chelombiev, Luke Hudlass-Galley, Charlie Blake, Carlo Luschi, and Douglas Orr. SparQ attention: Bandwidth-efficient LLM inference. In *International Conference on Machine Learning (ICML)*, 2024. arXiv:2312.04985.

Prajwal Singhania, Siddharth Singh, Shwai He, Soheil Feizi, and Abhinav Bhatele. Loki: Low-rank keys for efficient sparse attention. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2024. arXiv:2406.02542.

Zhiqing Sun, Yiming Yang, and Shinjae Yoo. Sparse attention with learning-to-hash. In *International Conference on Learning Representations*, 2022.

Jiaming Tang, Yilong Zhao, Kan Zhu, Guangxuan Xiao, Baris Kasikci, and Song Han. Quest: Query-aware sparsity for efficient long-context LLM inference. In *International Conference on Machine Learning (ICML)*, 2024.

Philippe Tillet, Hsiang-Tsung Kung, and David D. Cox. Triton: An intermediate language and compiler for tiled neural network computations. In *Proceedings of the 3rd ACM SIGPLAN International Workshop on Machine Learning and Programming Languages (MAPL)*, pp. 10–19, 2019. doi: 10.1145/3315508.3329973.

Chaojun Xiao, Pengle Zhang, Xu Han, Guangxuan Xiao, Yankai Lin, Zhengyan Zhang, Zhiyuan Liu, and Maosong Sun. InfLLM: Training-free long-context extrapolation for LLMs with an efficient context memory. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2024a. arXiv:2402.04617.

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

Shuo Yang, Ying Sheng, Joseph E. Gonzalez, Ion Stoica, and Lianmin Zheng. Post-training sparse attention with double sparsity. *arXiv preprint arXiv:2408.07092*, 2024.

Zhilin Yang, Peng Qi, Saizheng Zhang, Yoshua Bengio, William Cohen, Ruslan Salakhutdinov, and Christopher D. Manning. HotpotQA: A dataset for diverse, explainable multi-hop question answering. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pp. 2369–2380, 2018. doi: 10.18653/v1/D18-1259. URL https://aclanthology.org/D18-1259/.

Hailin Zhang, Xiaodong Ji, Yilin Chen, Fangcheng Fu, Xupeng Miao, Xiaonan Nie, Weipeng Chen, and Bin Cui. PQCache: Product quantization-based KVCache for long context LLM inference. *Proceedings of the ACM on Management of Data*, 3(3):201:1–201:30, 2025. doi: 10.1145/3725338. arXiv:2407.12820.

A Error bound of cluster aggregation and the ranking rule

This appendix supplies the detailed setting and derivations for §2.2, and analyzes the residual-cluster estimator of §3. Exact finite-cache identities are distinguished from Gaussian population estimates and allocation heuristics.

Detailed setting and tail estimators. For $q \in \mathbb{R}^d$ and $(K, V) \in \mathbb{R}^{n\times d}$, define $z_i = sq^\top k_i$, $s = d^{-1/2}$, $w_i = e^{z_i}$, $Z = \sum_i w_i$, $a_i = w_i/Z$ and $o = \sum_i a_i v_i$. A nonempty exact set X leaves $T = [n] \setminus X$, with $Z_X = \sum_{i\in X} w_i$ and $S_T = \sum_{i\in T} w_i$. Writing $\mu_X = Z_X^{-1} \sum_{i\in X} w_i v_i$, a tail estimator has the form

$$\hat{o} = \frac{Z_X \mu_X + \hat{S}\hat{\mu}}{Z_X + \hat{S}},$$

where $\hat{S}$ and $\hat{\mu}$ represent the estimated tail mass and value mean. Discarding uses $\hat{S} = 0$; uniform residual sampling uses the inverse-probability weights of Eq. (1); and one proxy uses $\hat{S} = Ne^{\bar{z}_T}$ and $\hat{\mu} = \bar{v}_T$, where $N = |T|$ and both means are arithmetic means over T. Its mass equals S_T only when the tail logits are equal. For the implied unnormalised token weights $\hat{w}_i$, define $\hat{Z} = \sum_i \hat{w}_i$, $\hat{a}_i = \hat{w}_i/\hat{Z}$ and $\mathcal{E} = Z^{-1}\sum_i |\hat{w}_i - w_i|$. The normalization of the unnormalised weights is fixed by the numerators in Eq. (1); $\mathcal{E}$ is not $\|\hat{a} - a\|_1$. Accordingly, $\mathcal{E}$ is tied to this numerator convention and is not invariant to a common rescaling of the implied weights, whereas $\hat{a}$ is.

Residual proxy estimator. For the multi-cluster estimator, X includes sink, local-window, selector top-k and allocated exact reads. Let $\mathcal{P}$ contain only the nonempty residual clusters. For $b \in \mathcal{P}$, n_b, $\bar{k}_b$ and $\bar{v}_b$ are the count and arithmetic means *after* exact-read tokens are removed. Set $\bar{z}_b = sq^\top \bar{k}_b$ and $\hat{M}_b = n_b e^{\bar{z}_b}$. The residual-proxy estimator is

$$\hat{o} = \frac{\sum_{i\in X} w_i v_i + \sum_{b\in\mathcal{P}} \hat{M}_b \bar{v}_b}{\sum_{i\in X} w_i + \sum_{b\in\mathcal{P}} \hat{M}_b}, \qquad \hat{M}_b = n_b e^{\bar{z}_b}. \tag{8}$$

Write $\delta_i = z_i - \bar{z}_b$ for $i \in b$, so $\sum_{i\in b} \delta_i = 0$, and define

$$\sigma_b^2 = \frac{1}{n_b}\sum_{i\in b} \delta_i^2 = s^2 q^\top \mathrm{Cov}_b(k) q, \qquad \hat{a}_b^{\mathrm{virt}} = \frac{\hat{M}_b}{\hat{Z}}, \qquad V_b = \max_{i\in b} \|v_i - o\|_2.$$

Also let $\Delta_b^+ = \max_{i\in b} \delta_i$ and $\Delta_b^- = -\min_{i\in b} \delta_i$. The diagonal estimate in Eq. (6) approximates σ_b^2. The virtual-token weight $\hat{a}_b^{\mathrm{virt}}$ is the sum of the implied original-token weights $\hat{a}_i = e^{\bar{z}_b}/\hat{Z}$ for $i \in b$.

Exact clusterwise discrepancy. Exact-read tokens have $\hat{w}_i = w_i$ and contribute no discrepancy; for $i \in b$, the proxy implies $\hat{w}_i = e^{\bar{z}_b}$. Factoring $e^{\bar{z}_b}$ within each cluster gives

$$\mathcal{E} = \frac{1}{Z}\sum_{b\in\mathcal{P}} \hat{M}_b D_b, \qquad D_b = \frac{1}{n_b}\sum_{i\in b} |e^{\delta_i} - 1|, \qquad \varepsilon_b = \hat{M}_b D_b,$$

which proves Eq. (3). This is exact for any finite residual partition, without a Gaussian assumption. Each term is nonnegative and vanishes for equal logits. Reading a whole cluster exactly removes its term. Partial reads change the residual mean and require recomputing the discrepancy.

Lemma 1 (exact error of the estimator)**.**

$$\hat{Z}(\hat{o} - o) = -\sum_{b\in\mathcal{P}}\sum_{i\in b} \big(e^{z_i} - e^{\bar{z}_b}\big)(v_i - o). \tag{9}$$

Proof. Let $N_X = \sum_{i\in X} e^{z_i} v_i$ and $Z_X = \sum_{i\in X} e^{z_i}$. Then $\hat{Z}(\hat{o}-o) = N_X - oZ_X + \sum_b \hat{M}_b(\bar{v}_b - o)$. Since o is the exact output, $\sum_{\text{all } i} e^{z_i}(v_i - o) = 0$, hence $N_X - oZ_X = -\sum_b \sum_{i\in b} e^{z_i}(v_i - o)$. Finally $\hat{M}_b(\bar{v}_b - o) = \sum_{i\in b} e^{\bar{z}_b}(v_i - o)$, and the two sums combine into Eq. (9). □

Relation to normalized attention and output. The normalized-attention bridge of §2.2 follows without a model assumption. For any nonnegative implied weights with $\hat{Z} > 0$, set $\tilde{a}_i = \hat{w}_i/Z = (\hat{Z}/Z)\hat{a}_i$. Then $\|\tilde{a} - a\|_1 = \mathcal{E}$ and $\|\tilde{a} - \hat{a}\|_1 = |\hat{Z} - Z|/Z = r_Z$. The triangle and reverse triangle inequalities give

$$\mathcal{E} - r_Z \le \|\hat{a} - a\|_1 \le \mathcal{E} + r_Z \le 2\mathcal{E},$$

where $r_Z \le \mathcal{E}$. Since $\sum_i (\hat{a}_i - a_i) = 0$, for any reference vector c and $R(c) = \max_i \|v_i - c\|_2$,

$$\hat{o} - o = \sum_i (\hat{a}_i - a_i)(v_i - c), \qquad \|\hat{o} - o\|_2 \le R(c)\|\hat{a} - a\|_1 \le 2R(c)\mathcal{E}.$$

More generally, for any nonnegative implied weights with $\hat{Z} > 0$,

$$\hat{Z}(\hat{o} - o) = \sum_i (\hat{w}_i - w_i)(v_i - o),$$

since $\sum_i w_i(v_i - o) = 0$. Therefore

$$\|\hat{o} - o\|_2 \le \hat{Z}^{-1} \sum_i |\hat{w}_i - w_i| \, \|v_i - o\|_2 \le \frac{Z}{\hat{Z}} \mathcal{E} R(o).$$

Since $|\hat{Z} - Z| \le Z\mathcal{E}$, for $\mathcal{E} < 1$ we have $Z/\hat{Z} \le (1 - \mathcal{E})^{-1}$.

The identity in Lemma 1 also has a covariance/Jensen form. Writing $S_b = \sum_{i \in b} e^{z_i}$ and defining covariance with the uniform measure on cluster members,

$$\hat{Z}(\hat{o} - o) = -\sum_{b \in \mathcal{P}} \Big[n_b \operatorname{Cov}_{i \in b}(e^{z_i}, v_i) + (S_b - \hat{M}_b)(\bar{v}_b - o) \Big].$$

For a single tail T, let $\tilde{v}_T = S_T^{-1} \sum_{i \in T} w_i v_i$ and $\hat{M}_T = N e^{\bar{z}_T}$. The same identity becomes

$$\hat{Z}_{\mathrm{p}}(\hat{o}_{\mathrm{p}} - o) = -S_T(\tilde{v}_T - \bar{v}_T) - (S_T - \hat{M}_T)(\bar{v}_T - o),$$

whereas discarding gives

$$Z_X(\hat{o}_{\mathrm{d}} - o) = -S_T(\tilde{v}_T - o).$$

Thus similar tokenwise weight discrepancy need not imply similar output error: proxying retains the tail mean-value contribution, leaving only its value-direction and mass mismatches.

For each residual cluster define $G_b = \log(n_b^{-1} \sum_{i \in b} e^{z_i - \bar{z}_b})$. Mean-logit proxying gives $S_b = \hat{M}_b e^{G_b} \ge \hat{M}_b$ by Jensen's inequality, hence the exact signed normalizer deficit is

$$Z - \hat{Z} = \sum_{b \in \mathcal{P}} (S_b - \hat{M}_b) = \sum_{b \in \mathcal{P}} \hat{M}_b (e^{G_b} - 1) \ge 0.$$

This deficit is bounded by the exact absolute discrepancy $Z\mathcal{E} = \sum_b \varepsilon_b$ of Eq. (3) and interprets the signed Jensen alternative below; the deployed ErrorGate estimates the absolute discrepancy. The covariance/Jensen decomposition also supports the value-aware alternative in App. A.1. The value-free discrepancy needs no small-covariance assumption: Eq. (9) and the triangle inequality give Proposition 1 directly. Exact-read tokens contribute nothing, and reading a whole cluster removes its nonnegative discrepancy term; partial reads require recomputing the residual statistics.

Proposition 1 (deterministic upper bound)**.** *With* $D_b = n_b^{-1} \sum_{i \in b} |e^{\delta_i} - 1|$,

$$\begin{aligned} \|\hat{o} - o\|_2 &\le \sum_{b \in \mathcal{P}} \hat{a}_b^{\mathrm{virt}} V_b D_b \\ &\le \sum_{b \in \mathcal{P}} \hat{a}_b^{\mathrm{virt}} V_b \big(e^{\Delta_b^+} - e^{-\Delta_b^-} \big) \le \sum_{b \in \mathcal{P}} \hat{a}_b^{\mathrm{virt}} V_b e^{\Delta_b^+}. \end{aligned} \tag{10}$$

Proof. Apply the triangle inequality to Eq. (9) and bound $\|v_i - o\|_2 \le V_b$:

$$\|\hat{o} - o\|_2 \le \hat{Z}^{-1} \sum_b V_b e^{\bar{z}_b} \sum_{i \in b} |e^{\delta_i} - 1| = \sum_b \hat{a}_b^{\mathrm{virt}} V_b D_b.$$

On $[-\Delta_b^-, \Delta_b^+]$, $|e^\delta - 1| \le \max(e^{\Delta_b^+} - 1, 1 - e^{-\Delta_b^-}) \le e^{\Delta_b^+} - e^{-\Delta_b^-}$, giving the remaining inequalities. □

The bound factorises per cluster into proxy mass ($\hat{a}_b^{\mathrm{virt}} \propto n_b e^{\bar{z}_b}$, set by the mean logit), value radius V_b, and dispersion of the true weights around the proxy weight (D_b, set by the logit spread). Both the mean and the spread enter as factors of the same term.

Corollary 1 (small dispersion under a moment condition)**.** *If the within-cluster deviations satisfy* $\frac{1}{n_b} \sum_{i \in b} e^{\lambda \delta_i} \le e^{\lambda^2 \sigma_b^2 / 2}$ *for* $\lambda \in \{1, 2\}$*, then* $D_b \le \sqrt{e^{2\sigma_b^2} - 1}$ *and*

$$\begin{aligned} \|\hat{o} - o\|_2 &\le \sum_{b \in \mathcal{P}} \hat{a}_b^{\mathrm{virt}} V_b \sqrt{e^{2\sigma_b^2} - 1} \approx \sqrt{2} \sum_{b \in \mathcal{P}} \hat{a}_b^{\mathrm{virt}} V_b \sigma_b \quad (\sigma_b \ll 1), \\ 0 &\le \frac{S_b}{\hat{M}_b} - 1 \le e^{\sigma_b^2/2} - 1. \end{aligned} \tag{11}$$

Proof. Cauchy–Schwarz gives $D_b \le (\frac{1}{n_b}\sum_i (e^{\delta_i}-1)^2)^{1/2} = (\overline{e^{2\delta}} - 2\overline{e^{\delta}} + 1)^{1/2}$. Jensen gives $\overline{e^{\delta}} \ge e^{\bar{\delta}} = 1$, and the moment condition gives $\overline{e^{2\delta}} \le e^{2\sigma_b^2}$. The mass-deficit statement uses the same condition at $\lambda = 1$. □

This moment condition is an additional tail-shape assumption; it is not implied by the empirical variance alone and can fail for clusters with rare positive-logit outliers. Under the stated moment condition, the output bound is first order in small σ_b, whereas the bound on missing cluster mass is second order. Under the full query metric, the expected dispersion is $\mathbb{E}_q[\sigma_b^2] = s^2 \operatorname{tr}(\operatorname{Cov}_b(K)\mathbb{E}[qq^\top])$. Minimising $\sum_b n_b \mathbb{E}_q[\sigma_b^2]$ is the k-means objective under $M = \mathbb{E}[qq^\top]$ (§3.2.1); it therefore minimises a token-count-weighted dispersion surrogate, not the mass- and value-weighted output bound itself.

Corollary 2 (Gaussian population estimate of dispersion)**.** *Model centered within-cluster deviations by $\delta \sim \mathcal{N}(0, \sigma_b^2)$. For this population,*

$$\begin{aligned} \mathbb{E}[e^{\delta}] - 1 &= e^{\sigma_b^2/2} - 1, \\ D_{\mathrm{G}}(\sigma_b) := \mathbb{E}|e^{\delta} - 1| &= e^{\sigma_b^2/2}\big(2\Phi(\sigma_b) - 1\big) \approx \sqrt{2/\pi}\,\sigma_b \quad (\sigma_b \ll 1), \end{aligned} \tag{12}$$

where Φ is the standard normal c.d.f. Finite-cluster D_b and $S_b/\hat{M}_b - 1$ need not equal these population expressions. The exact, Gaussian-model and deployed diagonal quantities are respectively

$$\varepsilon_b = \hat{M}_b D_b, \qquad \varepsilon_b^{\mathrm{G}} = \hat{M}_b D_{\mathrm{G}}(\sigma_b), \qquad \hat{\varepsilon}_b = \hat{M}_b D_{\mathrm{G}}(\hat{\sigma}_b),$$

where $\sigma_b^2 = s^2 q^\top \operatorname{Cov}_b(k) q$ and $\hat{\sigma}_b^2 = s^2 \sum_d q_d^2 \operatorname{Var}_b(k_d)$. The last expression is Eq. (6). Thus $Z\mathcal{E} = \sum_b \varepsilon_b$ exactly, while $\sum_b \hat{\varepsilon}_b$ estimates that quantity before allocated reads. No assumption about independence of weights and values is used.

Proof. For $\delta \sim \mathcal{N}(0, \sigma^2)$, the moment generating function gives $\mathbb{E}e^{\delta} = e^{\sigma^2/2}$. Also $\mathbb{E}|e^{\delta} - 1| = 2\mathbb{E}(e^{\delta} - 1)^+ - \mathbb{E}(e^{\delta} - 1)$ and $\mathbb{E}[e^{\delta}\mathbf{1}\{\delta > 0\}] = e^{\sigma^2/2}\Phi(\sigma)$, giving Eq. (12); its small-σ form is $2\varphi(0)\sigma$. The identities for ε_b and $Z\mathcal{E}$ follow from Eq. (3); the diagonal substitution defines the deployed score estimate. □

Signed Jensen-gap alternative. For a residual proxy cluster the exact normalizer deficit is $\Delta_b = S_b - \hat{M}_b = \hat{M}_b(e^{G_b} - 1)$, and its Gaussian-diagonal estimate is $\hat{\Delta}_b = \hat{M}_b(e^{\hat{\sigma}_b^2/2} - 1)$. This signed mass-deficit quantity is useful for interpretation and ranking ablations, but is distinct from the absolute discrepancy estimate deployed in Eq. (6). The latter is first order in small σ_b, while the modeled deficit is second order.

From cluster discrepancy to a token budget. For a fixed partition and a budget of m complete clusters, reading the m clusters with largest exact ε_b removes the largest nonnegative terms from $Z\mathcal{E}$. For partial reads, ErrorGate uses the estimated score and the proportional quota of Eq. (7),

$$q_b = \min\left(n_b, \lambda \frac{\hat{\varepsilon}_b}{\max_{b'} \hat{\varepsilon}_{b'}}\right), \qquad \sum_b q_b = E,$$

with λ set by water filling and integer allocation. The QGS chooses the q_b highest-scoring tokens in each cluster, and proxy statistics are recomputed over unread members (App. A.1). This partial-token allocation is a deployment design choice motivated by the measured concentration of discrepancy; the complete-cluster argument does not prove it optimal.

Corollary 3 (alternative maximum-based ranking heuristic)**.** *Under the sub-Gaussian maximum heuristic $\Delta_b^+ \lesssim \sigma_b\sqrt{2\log n_b}$, the last form of Eq. (10) becomes*

$$\|\hat{o} - o\|_2 \lesssim \frac{1}{\hat{Z}} \sum_{b \in \mathcal{P}} V_b n_b \exp\big(\bar{z}_b + c_b \sigma_b\big), \qquad c_b = \sqrt{2\log n_b}. \tag{13}$$

Large numerator contributions motivate a ranking heuristic, but removing their largest terms does not in general minimise this normalised bound: $\hat{Z}$ also changes with the exact set. Dropping the unavailable V_b, treating n_b as comparable across clusters and fixing c gives the alternative score $\bar{z}_b + c\sigma_b$, distinct from Eq. (6).

Both proxy mass $n_b e^{\bar{z}_b}$ and within-cluster dispersion matter: a cluster with $\sigma_b = 0$ is represented exactly by its proxy, while a cluster with tiny proxy mass contributes little at a fixed dispersion. The constant c in the maximum heuristic is not a calibrated tail bound, and heavy-tailed clusters can degrade its coverage. The maximum-based heuristic and the deployed absolute score use different functions of the same cached moments.

A.1 ErrorGate: the estimate and the hierarchical budget

The estimate. Corollary 2 distinguishes the finite-cluster discrepancy $\varepsilon_b = \hat{M}_b D_b$ from the Gaussian model $\varepsilon_b^{\mathrm{G}} = \hat{M}_b D_{\mathrm{G}}(\sigma_b)$ and the diagonal estimate $\hat{\varepsilon}_b = \hat{M}_b D_{\mathrm{G}}(\hat{\sigma}_b)$ used in Eq. (6). Since $2\Phi(\hat{\sigma}_b) - 1 = \operatorname{erf}(\hat{\sigma}_b/\sqrt{2})$, the deployed score is

$$\hat{\varepsilon}_b = \hat{M}_b e^{\hat{\sigma}_b^2/2} \operatorname{erf}(\hat{\sigma}_b/\sqrt{2}), \qquad \hat{\sigma}_b^2 = s^2 \sum_d q_d^2 \mathrm{Var}_b(k_d).$$

Its units are unnormalised mass, and the estimates add to an estimate of $Z\mathcal{E}$, not to its exact value. The rule uses a few elementwise operations and one error function on a $(B, H, 1, C)$ tensor. The factor $e^{\bar{z}_b}$ retains the sign-sensitive ordering of the mean logit; squaring the mean would erase this distinction.

Alternative estimates. The signed form $\hat{\Delta}_b = \hat{M}_b(e^{\hat{\sigma}_b^2/2} - 1)$ estimates the Jensen mass deficit, in which positive and negative weight deviations cancel. It is second order in small dispersion, whereas $\hat{\varepsilon}_b$ is first order. With the same split and residual statistics, the signed alternative scores 86.28 at a 1% read against 86.30 for Eq. (6). A value-aware alternative estimates output error instead. Define the exact weighted cluster mean $\tilde{v}_b = \sum_{i \in b} e^{z_i} v_i / S_b$ and the residual

$$r_b = \sum_{i \in b} (e^{z_i} - e^{\bar{z}_b})(v_i - o) = (S_b - \hat{M}_b)(\tilde{v}_b - o) + \hat{M}_b(\tilde{v}_b - \bar{v}_b).$$

Then Lemma 1 gives $\hat{Z}(\hat{o} - o) = -\sum_b r_b$. Cauchy–Schwarz gives $\|\tilde{v}_b - \bar{v}_b\|_2 \le \mathrm{CV}_b(e^z)\nu_b$, where $\nu_b^2 = \operatorname{tr} \mathrm{Cov}_b(v)$. Under the Gaussian model the population coefficient of variation is $\sqrt{e^{\sigma_b^2} - 1}$. Substituting the diagonal variance and a partial exact-token output $\hat{o}_X$ yields the alternative heuristic

$$\hat{\varepsilon}_b^{\mathrm{out}} = n_b e^{\bar{z}_b} \Big[(e^{\hat{\sigma}_b^2/2} - 1) \|\bar{v}_b - \hat{o}_X\|_2 + \sqrt{e^{\hat{\sigma}_b^2} - 1}\, \nu_b \Big].$$

This score also replaces the mass-channel value offset by $\bar{v}_b - \hat{o}_X$; it is not the exact residual or a certified bound. It needs the cached value spread and the partial output, and scores 86.04 with the same split. The three variants are within one standard error; the deployed score keeps its estimate value-free and uses only the mean and diagonal variance tokens. This comparison is not a small-covariance assumption in Proposition 1.

Why retain the uncorrected proxy. For a constant weight u, $\sum_{i \in b} |w_i - u|$ is minimized by a median of the weights. Under $z = \bar{z}_b + \delta$ with $\delta \sim \mathcal{N}(0, \sigma_b^2)$, the lognormal population median is $e^{\bar{z}_b}$, the mean-logit proxy weight. We use this uncorrected weight as a model-based design choice, not a finite-cluster or output-error guarantee.

Diagonal moments and discrepancy concentration. The mean and variance tokens cost 1.5 KV rows per cluster per query, both charged to the read. In the C=64 QGC diagnostic of App. B.3, diagonal and full-variance model estimates agree at Spearman 0.92 and recover the eight clusters with the largest measured exact discrepancy at similar rates (91%, 97%). These measurements assess the diagonal substitution, not a general variance inequality.

In the discrepancy-concentration diagnostic, clusters with at least 32 of the exact top-1% tokens are 4% of clusters and carry 69% of the exact discrepancy; the 84% of clusters without such tokens carry 12%. This concentration motivates, but does not prove optimal, the proportional quota of §3.3.

Statistics of the remaining members. Before quota allocation, the initially selected sink, local and QGS top-k tokens are subtracted from the cached pooled count, first moment and second moment. Thus n_b, $\bar{z}_b$ and $\hat{\sigma}_b^2$ used by ErrorGate are statistics of the candidate residual members, not of

the original cluster. After cluster b reads q_b additional tokens, these tokens are subtracted again to form the final proxy statistics, so no token is counted twice. Lemma 1 and Proposition 1 apply to this final residual state, whereas Eq. (6) scores the pre-allocation residual.

Hierarchical budget. Given the estimates, water filling uses $q_b = \min(n_b, \lambda w_b)$ with $w_b = \hat{\varepsilon}_b / \max_{b'} \hat{\varepsilon}_{b'}$ and λ chosen to meet the token budget E. After flooring, the remainder goes one token at a time to the highest-ranked clusters with capacity, so $\sum_b q_b = E$ when the budget is feasible. Within a cluster, the q_b tokens are its highest under the QGS score of Eq. (5). A uniform cap t, used in the earlier comparison, spends the same E on approximately E/t clusters in equal parts; the proportional split instead gives most reads to the few clusters with the largest estimates. At 1%, it reaches about 27 clusters per row with the largest quota near 60 tokens. Its total budget is fixed per row. Neither complete-cluster optimality nor the concentration diagnostic proves the proportional partial-token allocation optimal.

A.2 Closed forms of the sampling and proxy errors

This subsection derives Eq. (2), gives the exact finite-tail proxy discrepancy, and records the protocol behind Figure 4.

Setup. Fix a query and a nonempty exact set X, with nonempty tail $T = [n] \setminus X$, $N = |T|$, $Z_X = \sum_{i \in X} w_i$, $S_T = \sum_{i \in T} w_i$ and $\tau = S_T/Z$, where $Z = Z_X + S_T$. All expectations below condition on this fixed row and tail. Exact-read tokens contribute nothing to $Z\mathcal{E}$.

Discarding and random sampling. Discarding sets $\hat{w}_j = 0$ on T, hence $\mathcal{E}_{\mathrm{d}} = S_T/Z = \tau$. For a uniformly chosen subset $B \subseteq T$ of size m without replacement, set $\eta = N/m$. The sampling weights are $\hat{w}_j = \eta w_j$ on B and zero on $T \setminus B$, giving

$$Z\mathcal{E}_{\mathrm{s}} = \sum_{j \in B} (\eta - 1) w_j + \sum_{j \in T \setminus B} w_j.$$

Each indicator $I_j = \mathbf{1}_B(j)$ has expectation m/N, so

$$\mathbb{E}[Z\mathcal{E}_{\mathrm{s}}] = \big[(\eta - 1)m/N + (1 - m/N)\big] S_T = 2(1 - m/N) S_T.$$

Dividing by the fixed Z gives Eq. (2). The spread of tail weights affects the variance of this discrepancy, not its expectation at fixed S_T, Z and m/N.

Normalizer uncertainty and missed tokens. The sampling normalizer $\hat{Z}_{\mathrm{s}} = Z_X + \eta \sum_{j \in B} w_j$ is unbiased for Z. For $N > 1$, let $\bar{w} = S_T/N$, $\sigma_w^2 = N^{-1} \sum_{j \in T} (w_j - \bar{w})^2$ and $\mathrm{CV}_T = \sigma_w / \bar{w}$. Sampling without replacement gives

$$\frac{\sqrt{\mathbb{E}[(\hat{Z}_{\mathrm{s}} - Z)^2]}}{Z} = \tau \, \frac{\mathrm{CV}_T}{\sqrt{m}} \sqrt{\frac{N - m}{N - 1}}.$$

Equivalently, with $N_{\mathrm{eff}} = S_T^2 / \sum_{j \in T} w_j^2$, $\mathrm{CV}_T^2 = N/N_{\mathrm{eff}} - 1$. Thus the uncertainty grows both as the sample shrinks and as the residual mass concentrates on fewer effective tokens. This measures uncertainty in total mass, not its tokenwise allocation. At fixed tail statistics it grows as the sample shrinks. A given unread token is included with probability m/N; missing a large-weight token can affect both the sampled mass and value direction. At a 1% read behind HashAttention, the reported inclusion probability of the heaviest unread token is 0.5% in the 137,216-row diagnostic (App. B.2). Matched-read experiments also change the tail, so they are not fixed-tail sweeps in m alone.

One tail proxy. Let $g_T = e^{\bar{z}_T}$ and $\bar{v}_T = N^{-1} \sum_{i \in T} v_i$. Equation (1) assigns $g_T / \hat{Z}_{\mathrm{p}}$ to each tail token, equivalently one virtual token of weight $N g_T / \hat{Z}_{\mathrm{p}}$ and value $\bar{v}_T$:

$$\sum_{i \in T} \hat{a}_i^{\mathrm{p}} v_i = \frac{N g_T}{Z_X + N g_T} \bar{v}_T.$$

Define the finite-tail quantities

$$G_T = \log\left(\frac{1}{N}\sum_{j\in T} e^{z_j - \bar{z}_T}\right), \qquad D_T = \frac{1}{N}\sum_{j\in T} |e^{z_j-\bar{z}_T} - 1|.$$

Since $Z\mathcal{E}_{\mathrm{p}} = N g_T D_T$ and $N g_T = S_T e^{-G_T}$, the exact proxy discrepancy is

$$\mathcal{E}_{\mathrm{p}} = \tau e^{-G_T} D_T. \tag{14}$$

Jensen's inequality gives $G_T \geq 0$, and the relative normalizer deficit is exactly $(Z - \hat{Z}_{\mathrm{p}})/Z = \tau(1 - e^{-G_T})$. The proxy has no sampling variance. Under the Gaussian population model, $G_{\mathrm{G}} = \sigma_z^2/2$ and $D_{\mathrm{G}}(\sigma_z) = e^{\sigma_z^2/2}(2\Phi(\sigma_z) - 1)$ (Corollary 2), hence

$$\mathcal{E}_{\mathrm{p}}^{\mathrm{G}} = \tau e^{-\sigma_z^2/2} D_{\mathrm{G}}(\sigma_z) = \tau\,\mathrm{erf}(\sigma_z/\sqrt{2}).$$

This yields the Gaussian prediction in Eq. (2), not a guarantee for every finite tail. Empirical G_T and D_T need not equal the population expressions.

Protocol. The rows are those of App. B.3, 137,216 decode rows of Llama-3.1-8B-Instruct over the RULER-32K HARD tasks, three requests each, every layer and head and the first eight generated tokens. For each total read $\rho \in \{0.1, 0.5, 1, 3, 5, 10\}\%$, each decode row uses its valid cache length n and reserves $n_{\mathrm{base}} = 6$ sink and local tokens. The remaining budget $\mathrm{free} = \rho n - n_{\mathrm{base}}$ is split 50/50: the sampling treatment reads the selector's top half exactly and draws the other half uniformly from the residual tail.

The m values reported in Table 8 are the mean realized sample-mask counts over decode rows. Each treatment therefore has its own residual tail and tail mass τ. Per selector and read, τ_{s} and τ_{p} are row-averaged tail-mass statistics, while the reported m/N is an aggregate sample-to-tail ratio rather than the mean of the rowwise ratios. The quantity $\sigma_z = \sqrt{\langle \sigma_{z,i}^2 \rangle}$ is computed over the proxy tails. The reported predictions are plug-in values evaluated on these aggregate statistics. The measured columns instead average discrepancies computed directly from the raw decode rows, with one draw per row for sampling. The discard measurement follows a tail-selection path that is not rowwise identical to that underlying its plug-in prediction. Thus the prediction/measurement pairs are descriptive comparisons rather than rowwise-matched validation.

Table 8: Inputs and outputs of Fig. 4. $\mathcal{E}$ is the exact-normalizer-scaled unnormalised weight discrepancy. Predictions (f) are plug-in values from aggregate tail statistics, whereas measurements (m) are averages computed from raw decode rows; they are therefore not rowwise-paired. The proxy prediction additionally uses the Gaussian population model.

selector	read	τ_{s}	τ_{p}	σ_z	m	$\mathcal{E}_{\mathrm{s}}$ (f / m)	$\mathcal{E}_{\mathrm{p}}$ (f / m)	$\mathcal{E}_{\mathrm{d}}$ (f / m)
HashAttention	10%	0.212	0.176	2.36	1585	0.401 / 0.398	0.173 / 0.171	0.176 / 0.176
	5%	0.245	0.212	2.41	791	0.477 / 0.480	0.208 / 0.206	0.212 / 0.212
	3%	0.268	0.236	2.43	473	0.527 / 0.530	0.233 / 0.230	0.236 / 0.236
	1%	0.315	0.286	2.47	156	0.628 / 0.623	0.282 / 0.280	0.286 / 0.286
	0.5%	0.345	0.316	2.48	76	0.687 / 0.684	0.312 / 0.310	0.316 / 0.316
	0.1%	0.406	0.386	2.50	13	0.812 / 0.855	0.381 / 0.380	0.386 / 0.385
oracle	10%	0.049	0.032	2.10	1585	0.093 / 0.093	0.030 / 0.027	0.032 / 0.032
	5%	0.067	0.049	2.24	791	0.131 / 0.131	0.048 / 0.043	0.049 / 0.049
	3%	0.081	0.062	2.32	473	0.159 / 0.159	0.061 / 0.056	0.062 / 0.062
	1%	0.109	0.092	2.42	156	0.218 / 0.218	0.090 / 0.085	0.092 / 0.091
	0.5%	0.128	0.110	2.45	76	0.256 / 0.256	0.108 / 0.104	0.110 / 0.110
	0.1%	0.195	0.166	2.49	13	0.390 / 0.390	0.164 / 0.160	0.166 / 0.165

What the weight error does and does not say. The discrepancy $\mathcal{E}$ counts absolute differences of implied unnormalised weights and is not the normalized attention distance $\|\hat{a} - a\|_1$; Appendix A

gives their relation. The prediction/measurement pairs in Table 8 are descriptive rather than a calibration test: the predictions use aggregate tail statistics, whereas the measurements are computed rowwise, and the discard columns additionally use slightly different tail-selection conditions. In particular, agreement of the proxy column does not validate the Gaussian tail model. The metric also ignores values, so its ordering need not match output error; Figure 3 reports the corresponding output comparison on a separate row set. In this subsection m is the sample size; elsewhere in the appendix m counts expanded clusters.

B Attention Quality and Diagnostics

B.1 Quality and Error Across Read Densities

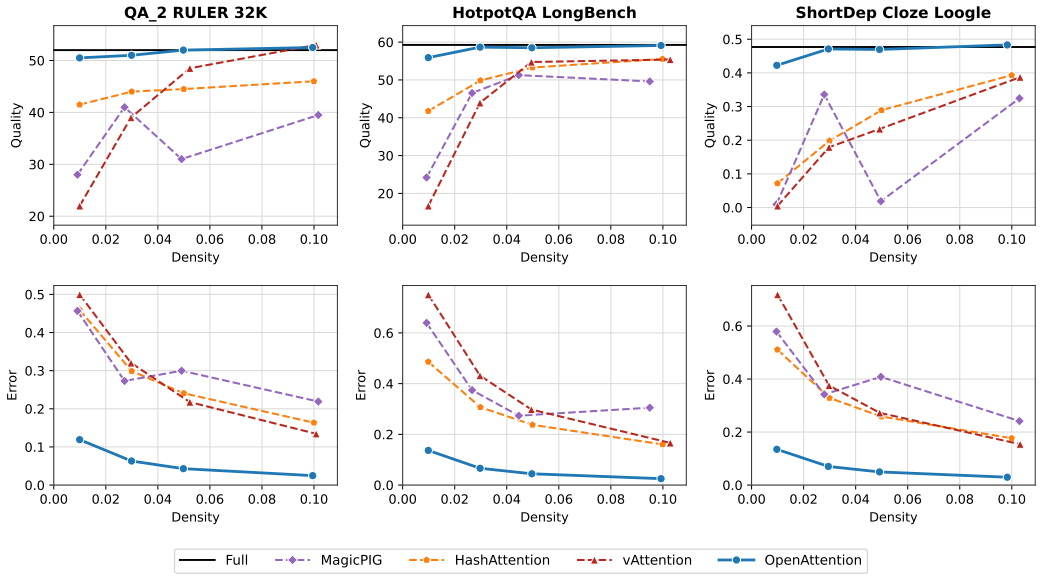


Figure 9: Quality and error versus read density on representative RULER-32K, LongBench, and LooGLE tasks.

Figure 9 plots task quality and error as the read density increases from 1% to 10% for RULER-32K QA_2, LongBench (Bai et al., 2024) HotpotQA (Yang et al., 2018), and LooGLE (Li et al., 2024a) ShortDep Cloze. Across these three tasks, OpenAttention stays close to dense attention quality while showing the lowest error among the sparse methods at the plotted densities. The separation is clearest at low density, where the baselines' errors rise and their task quality falls. These curves complement the fixed-budget results in the main text by showing how both measures change with the read budget.

B.2 Protocol of the Decode-Row Diagnostic

Figure 10 is one pass of `run_v37_tail_diag.py` over the RULER-32K HARD tasks, three requests each, recording every layer, every head and the first eight generated tokens, which gives 137,216 rows. Dense output is returned at every call, so the generation trajectory is the model's own. Within a row the two indices, the five reads and the four treatments all see the same cache and the same query, and each treatment is charged the same *total* read ρn. Sink and local hold the deployed few-token floor (0.01% each), and the remainder is spent in one of four ways. Treatment (i) spends it on the selector top-k only. Treatment (ii) spends half on the top-k and half on a random sample of m residual tokens, rescaled by N/m and merged into the same softmax, which is vAttention's fixed 50/50 split (App. F.2). Treatment (iii) reads C clusters at 1.5 rows each and the rest on the top-k, with every cluster a proxy. Treatment (iv) uses C clusters, a $\sim$9-token top-k and E tokens expanded by Eq. (6) with the uniform per-cluster cap t of earlier versions; the deployed quotas of App. A.1 change the 1% mean by less than one standard error. The clusters are rebuilt

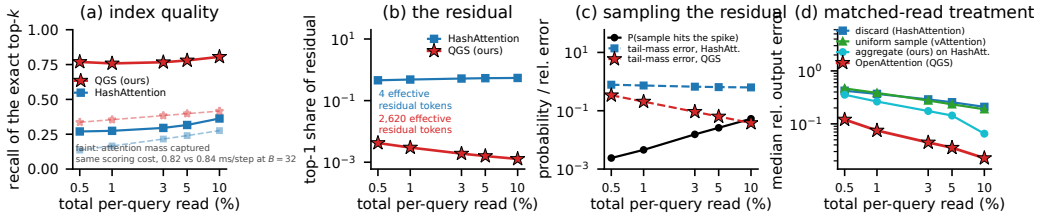


Figure 10: One pass over 137,216 RULER-32K HARD decode rows (Llama-3.1-8B, all layers and heads of 21 requests) against the *total* read. (a) Recall of the exact top-k by the HashAttention signature and by the deployed QGS (64 vs. 32 bits/token/head at the same per-step cost, 0.84 vs. 0.82 ms at B=32), with the captured mass in faint lines. (b) Median share of the unread mass held by its largest token, annotated with the residual's participation ratio. (c) Probability that vAttention's sample contains that token, and the relative error of the tail mass it estimates. (d) Median relative output error of four treatments of the same residual at the *same* read, which discard it, sample it, aggregate it into clusters, or apply the deployed OpenAttention. The protocol is given in App. B.2.

inside the diagnostic as at deployment (keys aligned as $\phi_i = \Lambda_r^{1/2} U_r^\top k_i$ with r=16 under the shrunk M, four Lloyd iterations in bf16, once per request, new keys joining the nearest centroid) and the codes and cluster assignments of tokens generated during decoding are extended incrementally, as at deployment. The QGS is the deployed one (r=32, int8, one U per KV head).

B.3 Protocol of the selector and aggregation diagnostics

Fig. 7 comes from one pass of `run_v49_cluster_diag.py` over the RULER-32K HARD tasks, three requests each, every layer and head and the first eight generated tokens (137,216 rows), with dense output returned so the trajectory is the model's own, as in App. B.2. *Partitions.* The tail is everything but sink, local and the QGS top-1%, and every cluster is aggregated, so the error isolates the partition. Key-only k-means runs the same four Lloyd iterations as ours on the centred raw keys, one partition per KV head, and adjacency cuts the cache into C equal runs. The reported dispersion is the within-cluster standard deviation of the exact logits, averaged over clusters with weights equal to their true attention mass. *Clusters.* Fig. 7 and the error shares by top-k content reported in App. A.1 use the deployed 1% partition (C=64) with only the ∼9-token QGS top-k exact, on every fourth head (2.2M clusters). The exact variance is the within-cluster variance of the true logits, and the true error of a cluster is its weight error $\hat{M}_b D_b / Z = \sum_{i \in b} |e^{z_i} - e^{\bar{z}_b}| / Z$ (Eq. (3)); the partition error is the same sum over all clusters with every cluster a proxy; on these rows the attention weighted within cluster logit std at C=64 is 1.83 for positional clusters, 1.67 for k-means on the keys alone and 1.05 for QGC, and the weight error follows (0.041, 0.040, 0.030), so QGC at C=64 beats key only clusters at C=1024. Over all rows, the eight clusters with the largest $\hat{\varepsilon}_b$ hold 84% of the aggregation error and 95% of the exact top-1% tokens left in the tail, and $\hat{\varepsilon}_b$ orders the clusters by their true error at Spearman 0.92.

C OpenAttention.flash: OpenAttention on FlashAttention-2

Algorithm 1 is the decode attention step of §3.4. A FlashAttention-2 decode kernel keeps, per program, a running maximum, a running normaliser and an output accumulator, and folds one tile of keys and values at a time into them; OpenAttention.flash keeps that loop, written in Triton, and changes only what the tiles hold. Phases 1 and 2 gather the exact tokens, the QGS top-k and the tokens ErrorGate's quotas read from the expanded clusters, from scattered rows of the cache straight into SRAM, where a gather in PyTorch writes the gathered tile to DRAM and reads it back. Phase 3 streams the clusters as virtual tokens, the mean key in place of a key, the mean value in place of a value and the occupancy of the unread members as $+\log n'_b$ on the logit, so the tail costs one more tile loop rather than a second attention pass. The statistics of the unread members are prepared before the launch by subtracting the tokens the query reads from the cached pooled sums (App. A.1), and a cluster read whole carries $\log 0 = -\infty$, so the kernel needs no membership test. The index

lists are padded to a fixed length with masked slots, so the work per step is fixed, with no data dependent loop bound and no host synchronisation.

Algorithm 1 OpenAttention.flash: one decode step of the residual-proxy estimator as a FlashAttention-2 online softmax pass. One program per (sequence, query head); the $G{=}4$ query heads of a KV head read the same cache; $T{=}64$ is the tile size and $s = d^{-1/2}$.

Input: query q; cache K, V of the KV head; the index list $\mathcal{S}$ of the selected tokens (the QGS top-k) and $\mathcal{X}$ of the tokens the expanded clusters read (the ErrorGate quotas q_b of §3.3); per cluster the statistics of its *unread* members, $(\bar{k}'_b, \bar{v}'_b, n'_b)$, obtained from the cached pooled sums by subtracting the tokens in $\mathcal{S} \cup \mathcal{X}$, with $n'_b{=}0$ for a cluster read whole

Output: $\hat{o}$ of the residual-proxy estimator

$m \leftarrow -\infty$, $L \leftarrow 0$, $o \leftarrow \mathbf{0}$ ▷ running maximum, normaliser and accumulator, fp32
for each tile of T indices i in $\mathcal{S}$, then in $\mathcal{X}$ **do** ▷ phases 1 and 2: exact tokens
  gather the rows k_i, v_i from the cache into SRAM, padding slots masked
  $z_i \leftarrow s\, q^\top k_i$
  $(m, L, o) \leftarrow \textsc{Update}(m, L, o, \{z_i\}, \{v_i\})$
end for
for each tile of T clusters b **do** ▷ phase 3: proxies as virtual tokens
  load $\bar{k}'_b$, $\bar{v}'_b$ and $\log n'_b$ ▷ $\log 0 = -\infty$ drops a cluster read whole
  $\ell_b \leftarrow s\, q^\top \bar{k}'_b + \log n'_b$
  $(m, L, o) \leftarrow \textsc{Update}(m, L, o, \{\ell_b\}, \{\bar{v}'_b\})$
end for
return o/L

procedure $\textsc{Update}(m, L, o, \{z_j\}, \{v_j\})$ ▷ the FlashAttention-2 online softmax step
  $m' \leftarrow \max(m, \max_j z_j)$; $\alpha \leftarrow e^{m-m'}$; $p_j \leftarrow e^{z_j - m'}$
  $o \leftarrow \alpha\, o + \sum_j p_j v_j$; $L \leftarrow \alpha L + \sum_j p_j$; $m \leftarrow m'$
  return (m, L, o)
end procedure

Implementation and correctness. The kernel runs one program per (sequence, query head), with $G{=}4$ query heads per KV head, tiles of 64 tokens / 64 clusters, 4 warps and fp32 accumulators. Against an fp32 reference the median relative error is 1.6×10^{-3} and the maximum 2.1×10^{-3} over batch-head counts 4–64, including masked expanded clusters. End to end on RULER-32K HARD at 5%, the kernel scores 85.45 against 85.35 for PyTorch, with per-task differences in $[-0.2, +0.5]$ of mixed sign. At batch 32 the effective bandwidth is 3,634 GB/s for dense SDPA reading 4.3 GB and 1,750 GB/s for the kernel reading 1.09 GB. Scattered (cluster) expansion versus contiguous expansion costs $+4$–6% kernel time, so clusters need not be physically re-packed. Figure 11 sweeps batch and context for the 5% + 256-token configuration ($C{\approx}970$ clusters stored once per KV head, without the variance stream; with the per-query-head clusters of the accuracy pipeline the same run reads 34% of the bytes, 41% with the variances). The unfused implementation is $2.7\times$ slower than dense at batch 32, whereas OpenAttention.flash is $1.9\times$ faster. Each cluster caches n_b, $\bar{k}_b$, $\operatorname{diag} \operatorname{Var}_b(k)$ and $\bar{v}_b$, 385 bf16 scalars, so at the deployed $C{=}64$ the statistics of the whole 32 layer model take 12 MiB when stored once per KV head, 0.3% of the 4.0 GiB KV cache they summarise.

What the index scan costs, and why it is excluded. The scan that turns a compressed cache into a ranking, and the top-k selection over it, is common to every method in Table 9: OpenAttention, HashAttention, vAttention and MagicPIG all scan a compressed cache of the same length and take a top-k over it, so excluding it is what makes the sparse rows comparable with one another. It is not what makes them comparable with dense attention. Its cost is method-specific and is not measured in Table 9, so a decode step that charges it gains less than the attention-stage figure. We report the attention stage because that is the part this paper changes, and the $10\times$ refers to this stage only. Bringing the selection itself under the budget is open work.

Table 9: Decode attention step per layer (ms, H100 NVL, 32K, bf16) at 1% and 3% reads with no ErrorGate expansion (E=0), as in Fig. 1(c). The vAttention/MagicPIG rows are lower bounds charging only their gathers and estimator arithmetic; selection, ErrorGate scoring and residual-statistics updates are excluded for every method. The three models of §4.3 share these attention shapes. The speedup reported here is therefore of the attention stage and not of the decode step as a whole (see below).

method	custom kernel	1% read B=1	1% read B=32	3% read B=1	3% read B=32
dense attention (fused SDPA)	–	0.06	1.20	0.06	1.22
HashAttention	×	0.12	0.54	0.11	1.65
vAttention (lower bound)	×	0.37	0.67	0.35	1.75
MagicPIG (lower bound)	×	0.13	0.55	0.11	1.65
OpenAttention, unfused gather	×	0.29	0.46	0.27	1.26
OpenAttention.flash	✓	**0.05**	**0.12**	**0.07**	**0.24**

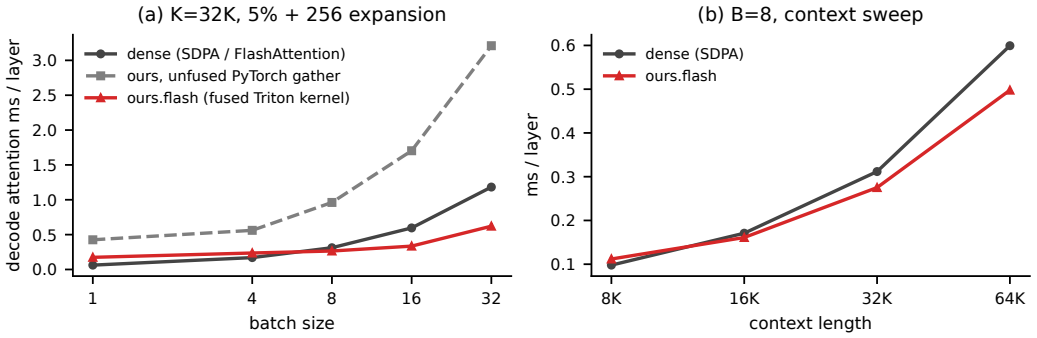


Figure 11: Decode attention time per layer on one H100 NVL (Llama-3.1-8B shapes, bf16) for fused dense SDPA and for OpenAttention with the unfused PyTorch gather path and with OpenAttention.flash, the fused Triton kernel. (a) Batch sweep at 32K; (b) context sweep at batch 8.

D Budget split and deployed hyper-parameters

D.1 Choosing (C, E, t) per budget

The grid below was run with the uniform per-cluster cap t and the bound route ranking $\bar{z}_b + 3\sigma_b$ of earlier versions; the deployed ErrorGate replaces the cap by the error proportional quotas of App. A.1 and keeps the (C, E) the grid selected. Within a budget, C, E and the cap t trade against one another at a fixed ρ ($E + 1.5C$ is what the accounting leaves once sink, local and the ∼9-token top-k are paid), so the triple is chosen per budget on the small grid of Table 10. The free parameter is really the *share of the read the proxy stream takes*, and the grid moves it while keeping the selector, the partition, the ranking rule and the cap rule fixed. The share that wins grows as the read shrinks, because the sparser the read, the more of it is worth spending on knowing where the mass is rather than on reading tokens.

E Results at every budget

Table 11 gives Figure 8 per task. Table 12 gives the per-task results underlying the 5% component ablation in Table 4. OpenAttention scores 87.08, 86.82, 86.83 and 86.30 at 10, 5, 3 and 1%, within 0.25 points of dense down to 3% and within one point at 1%, while every baseline degrades steeply; vAttention trails plain HashAttention by 2.7, 7.7 and 11.9 points at 5, 3 and 1% and overtakes it only at 10%, and MagicPIG, calibrated to the same reads, degrades fastest (App. F.1).

Table 10: Per-budget grid over the split between the proxy stream and the expansion (RULER-32K HARD, 200 samples per task; "proxy" is $1.5C/\rho n$, the share of the read the cluster statistics take). Everything but (C, E, t) is held fixed. The chosen row is the one Figure 8 reports.

ρ	C	E	t	proxy	avg	qa_1	qa_2	vt	fwe	mk_2	mk_3	mv
10%	512	2493	32	23%	86.59	78.0	49.5	95.2	92.7	95.0	99.5	96.2
10%	256	2877	32	12%	86.85	78.0	50.5	95.0	94.2	94.5	99.5	96.2
5%	256	1238	32	23%	86.51	76.5	51.0	94.9	93.2	94.5	99.5	96.0
5%	192	1335	32	18%	86.98	79.0	50.5	95.1	93.5	95.0	99.5	96.2
5%	128	1431	32	12%	86.83	78.5	50.0	95.2	94.3	95.0	99.5	95.2
10%	384	2685	32	18%	86.60	77.5	50.0	95.1	93.0	95.0	99.5	96.1
3%	256	583	32	39%	86.69	78.0	50.5	95.4	92.7	95.0	99.5	95.8
3%	192	679	16	29%	86.63	79.0	49.0	93.5	94.2	95.5	99.5	95.8
3%	192	679	32	29%	86.80	78.5	50.5	94.8	93.3	95.0	99.5	96.0
3%	128	775	32	20%	86.60	77.5	49.5	95.3	93.5	95.5	99.5	95.4

Table 11: RULER-32K HARD, Llama-3.1-8B-Instruct, 200 samples per task, at a matched *total* per-query read from 10% to 1% ("read" counts the exact rows plus, for OpenAttention, the proxy stream of 1.5 rows per cluster, and for vAttention, whose sample is adaptive, the measured read); the averages are Figure 8. Baselines use the HashAttention selector.

budget	method	read	avg	qa_1	qa_2	vt	fwe	mk_2	mk_3	mv
100%	dense	100%	87.04	79.0	50.0	95.2	93.3	95.0	100.0	96.8
10%	MagicPIG	10.0%	52.52	51.0	39.0	90.2	88.8	30.0	3.5	65.1
	HashAttention	10.0%	81.23	70.5	45.5	87.9	89.8	88.0	91.0	95.9
	vAttention	10.0%	83.08	73.0	46.0	91.3	92.2	92.5	91.0	95.6
	OpenAttention (ours)	10.0%	**87.08**	79.0	51.0	95.3	92.7	95.0	100.0	96.6
5%	MagicPIG	5.1%	34.42	36.0	33.5	71.5	88.3	2.5	0.0	9.1
	HashAttention	5.0%	71.55	68.0	44.5	81.9	87.3	72.5	50.5	96.1
	vAttention	5.0%	68.86	67.5	45.0	85.8	90.5	67.0	33.5	92.8
	OpenAttention (ours)	5.0%	**86.82**	78.5	50.5	95.8	92.2	94.5	99.5	96.8
3%	MagicPIG	2.9%	24.01	24.5	26.0	34.0	81.8	0.0	0.0	1.8
	HashAttention	3.0%	63.48	64.0	43.0	80.9	86.8	58.5	16.0	95.1
	vAttention	3.0%	55.74	55.0	36.5	79.1	88.7	38.0	3.0	89.9
	OpenAttention (ours)	3.0%	**86.83**	78.5	51.0	95.9	92.0	94.5	99.5	96.4
1%	MagicPIG	1.1%	10.53	12.0	13.5	0.7	47.5	0.0	0.0	0.0
	HashAttention	1.0%	48.22	44.0	41.5	73.3	78.5	12.5	0.0	87.8
	vAttention	1.0%	36.29	24.0	23.0	62.6	73.7	1.5	0.0	69.2
	OpenAttention (ours)	1.0%	**86.30**	77.5	50.0	96.0	93.5	95.0	97.0	95.1

F BASELINE CONFIGURATIONS

The baselines of §4 run inside one framework on the HashAttention selector with the public signature weights. This appendix gives the MagicPIG and vAttention configurations that realise each target read under our accounting.

F.1 MAGICPIG CONFIGURATION

We use the MagicPIG masker of the vAttention authors' framework (random signed projections, key centring, the Neyshabur–Srebro maximum inner product search (MIPS) transform, a match if any of the L tables collides, importance weights $1/p_i$ from the collision probability). Here K is the number of hash bits per table and L is the number of tables. Its realised density depends on the hash configuration, on the query–key geometry and on the context length, so we keep K fixed within

Table 12: Per-task accuracy for the component ablation of Table 4 at a matched 5% total read on RULER-32K HARD (200 requests per task).

proxies	ErrorGate	QGS	avg	qa_1	qa_2	vt	fwe	mk_2	mk_3	mv
×	×	×	67.35	62.0	45.5	84.0	91.7	63.0	31.5	93.8
✓	×	×	77.91	74.5	45.5	91.8	92.8	82.5	63.0	95.2
✓	✓	×	85.90	78.5	49.0	95.6	92.2	94.0	96.0	96.0
✓	×	✓	86.73	78.0	51.0	95.6	92.0	94.5	99.5	96.5
✓	✓	✓	**86.82**	78.5	50.5	95.8	92.2	94.5	99.5	96.8

each sweep and calibrate L to the target read on the benchmark itself, accepting a configuration only when its measured read matches the budget.

The MagicPIG points in Figure 1(a,b), Figure 8, and Table 11 use $K{=}9$, with $L{=}8, 25, 46, 100$ at the 1, 3, 5, and 10% targets, respectively. Their measured reads are 1.09, 2.88, 5.13, and 9.97%. These 32K runs evaluate 200 requests per task; the 1% run used two RTX 4090 GPUs with the model split across them, and the other three used one H100 NVL. The context-length comparison in Tables 7 and 18 is a separate $K{=}10$ sweep at a 3% target read. At 16K, 32K, 64K, and 96K, it uses $L{=}210, 300, 300, 250$ and measures reads of 2.99, 3.20, 3.19, and 3.02%, respectively, with 100 requests per task.

Note that MagicPIG's own evaluation processes the question with *dense* attention and applies sampling only to generation, whereas the protocol shared by all methods here applies sparse attention to the question chunk as well; under this protocol, and without the dense layers 0 and 16 of the released code, MagicPIG is known to degrade (Desai et al., 2026, App. C, Table 10).

F.2 vATTENTION CONFIGURATION PROTOCOL

vAttention's accuracy at a given read depends on four parameters, namely the shares of the budget given to the heavy-hitter selector (f_t) and to the base sample (f_b), and the tolerances (ϵ, δ) that size the adaptive sample of each head from the base sample's variance estimate. In the released implementation the adaptive budget scales as $(\Phi^{-1}(1-\delta)/\epsilon)^2$, so only that ratio matters and we sweep $\epsilon{=}\delta$. The vAttention paper (App. A.2, Table 3) chooses the parameters per target sparsity by a grid search on a few examples, keeping the configuration with the smallest local attention error among those that meet the target. We follow the same protocol under our accounting. For each target read we probe nominal budgets $\{0.4, 0.6, 0.8, 1.0\}\times$target, sampling shares $\{0.25, 0.5, 0.75\}$ and $\epsilon{=}\delta \in \{0.025, 0.05, 0.1, 0.2, 0.3\}$ (plus, at 10%, the points of the paper's own 10% grid that fit a 10% read) on four requests (two of qa_1, two of niah_multikey_2), logging the realised read and the relative output error against dense attention at every layer call, and run the configuration with the lowest error among those whose realised read is within 5% of the target on the full benchmark (Table 13). Tightening $\epsilon{=}\delta$ below 0.2 makes the adaptive budget overshoot the target at every nominal budget (at a 3% target, $\epsilon{=}\delta{=}0.05$ realises 5.7–14%), so at 0.5–5% the protocol selects the loosest tolerance with the full nominal budget and a 50/50 split, which is the paper's default setting except for $\epsilon{=}\delta{=}0.3$ instead of 0.25, and at 10% it selects the paper's own grid point $f_t{=}7.5\%$, $f_b{=}2.5\%$. Table 14 compares the protocol-chosen runs with the fixed $\epsilon{=}\delta{=}0.25$, 50/50 configuration on RULER-32K HARD; the vAttention rows of the other two models (Table 5) use the fixed configuration. The vAttention paper reports 86.6 for vAttention(HashAttention) at "10% sparsity" on the same tasks against 81.9 for HashAttention alone; under a strictly matched read the protocol recovers part of that gain (Table 14), and what remains is attributable to their search including tolerances down to $\epsilon{=}\delta{=}0.025$, whose adaptive budgets exceed a 10% read under our accounting (Table 13), and to their fixed 128-token sink and local windows.

G QUESTION-FREE TASKS

This appendix expands Table 15 of the main text with all OpenAttention variants on the question-free LongBench summarisation tasks (Table 16).

Table 13: vAttention parameter probe per target read (4 requests). For each $\epsilon{=}\delta$ the table lists the lowest-error configuration that meets the target, its realised read and its output error against dense attention; the chosen configuration is bold.

configuration	realised read	output error
target read 1%: 14 of 40 configurations within $1.05\times$ the target		
$\epsilon{=}\delta{=}0.3$, nominal 1.0%, sampling share 0.50 **(chosen)**	1.00%	0.470
$\epsilon{=}\delta{=}0.2$, nominal 0.8%, sampling share 0.50	0.90%	0.579
$\epsilon{=}\delta{=}0.1$, nominal 0.4%, sampling share 0.25	0.83%	0.628
$\epsilon{=}\delta{=}0.05$, nominal 0.4%, sampling share 0.25 (none fits: smallest read)	3.16%	0.489
$\epsilon{=}\delta{=}0.025$, nominal 0.4%, sampling share 0.25 (none fits: smallest read)	9.32%	0.383
target read 3%: 25 of 60 configurations within $1.05\times$ the target		
$\epsilon{=}\delta{=}0.3$, nominal 3.0%, sampling share 0.50 **(chosen)**	3.00%	0.339
$\epsilon{=}\delta{=}0.2$, nominal 2.4%, sampling share 0.50	2.67%	0.360
$\epsilon{=}\delta{=}0.1$, nominal 1.8%, sampling share 0.25	3.00%	0.369
$\epsilon{=}\delta{=}0.05$, nominal 1.2%, sampling share 0.25 (none fits: smallest read)	5.74%	0.329
$\epsilon{=}\delta{=}0.025$, nominal 1.2%, sampling share 0.25 (none fits: smallest read)	13.84%	0.246
target read 5%: 26 of 60 configurations within $1.05\times$ the target		
$\epsilon{=}\delta{=}0.3$, nominal 5.0%, sampling share 0.50 **(chosen)**	4.97%	0.275
$\epsilon{=}\delta{=}0.2$, nominal 5.0%, sampling share 0.25	5.23%	0.278
$\epsilon{=}\delta{=}0.1$, nominal 3.0%, sampling share 0.25	4.75%	0.299
$\epsilon{=}\delta{=}0.05$, nominal 2.0%, sampling share 0.25 (none fits: smallest read)	7.32%	0.270
$\epsilon{=}\delta{=}0.025$, nominal 2.0%, sampling share 0.25 (none fits: smallest read)	16.33%	0.208
target read 10%: 32 of 75 configurations within $1.05\times$ the target		
$\epsilon{=}\delta{=}0.3$, nominal 10.2%, sampling share 0.25 **(chosen)**	10.03%	0.204
$\epsilon{=}\delta{=}0.2$, nominal 10.0%, sampling share 0.25	10.44%	0.205
$\epsilon{=}\delta{=}0.1$, nominal 6.0%, sampling share 0.50	9.71%	0.220
$\epsilon{=}\delta{=}0.05$, nominal 4.0%, sampling share 0.25	10.45%	0.208
$\epsilon{=}\delta{=}0.025$, nominal 4.0%, sampling share 0.25 (none fits: smallest read)	20.12%	0.170

Table 14: vAttention on RULER-32K HARD (mean accuracy), comparing the fixed $\epsilon{=}\delta{=}0.25$, 50/50 configuration with the protocol-chosen one of Table 13, with the measured read of the latter.

target read	fixed ($\epsilon{=}\delta{=}0.25$, 50/50)	protocol-chosen	measured read
1%	37.36	36.29	1.0%
3%	57.15	55.74	3.0%
5%	68.70	68.86	5.0%
10%	82.26	83.08	10.0%

LongBench-v2. As a non-synthetic check we also run LongBench-v2 (multiple choice, 0-shot prompt, accuracy) on its first 60 requests at 3%, contexts truncated to 32000 tokens as everywhere else; the request count is small and the numbers are indicative only.

method (3% total read)	LongBench-v2 0-shot accuracy
dense	30.00
HashAttention	23.30
OpenAttention (QGS)	33.30

H Composing with KV-cache compression

The three eviction compressors of Table 6 follow the release of the PyramidKV paper (`KVCache-Factory`), which also ships the StreamingLLM and SnapKV baselines that paper compares against, and whose RULER driver fixes their hyper-parameters. The budget b is an *abso-*

Table 15: Beyond RULER at a 3% read, on LongBench summarisation (no question chunk; ROUGE-L (Lin, 2004), 30 requests/task) and LongBench-v2 (Bai et al., 2025) (0-shot accuracy, 60 requests). Absolute scores are low for dense attention too, a property of metric and benchmark, not of sparsity; OpenAttention reads no question tokens. App. G gives details.

method (3% read)	gov_rep.	multi_news	LB-v2
dense	36.08	26.61	30.00
HashAttention	35.34	22.09	23.30
vAttention	31.05	16.33	–
OpenAttention (static only)	35.53	26.80	**33.30**

Table 16: LongBench summarisation tasks (no question chunk), 3% read, 30 requests per task, ROUGE-L. "static metric only" uses no question at all (selector and partition from the calibrated M; the row of Table 15); the last row treats the instruction tokens as the question.

method	gov_report (ROUGE-L)	multi_news (ROUGE-L)
dense	36.08	26.61
HashAttention	35.34	22.09
vAttention	31.05	16.33
OpenAttention (HashAttention selector, static-metric partition)	35.95	26.99
OpenAttention (static metric only: selector and partition)	35.53	26.80
OpenAttention (static + instruction tokens as question)	34.68	26.93

lute average cache size per layer rather than a fraction of the context, scores are max-pooled with kernel 7, and the retained window is the last α=8 tokens for SnapKV and PyramidKV and $b-4$ for StreamingLLM, whose remaining four slots are the attention sinks of Xiao et al. (2024b). Every prefix key is scored by the attention that retained window pays to it, averaged over the query heads of its KV group, and the top of that score survives. StreamingLLM is the exception, since it consults no score and keeps tokens by position alone. PyramidKV's arithmetic allocation gives layer 0 a prefix budget of $2(b-\alpha)-(b-\alpha)/\beta$ and the last layer $(b-\alpha)/\beta$ with β=20, integer-divided steps in between, so the mean over layers is b; at b=2048 that is 3,986 entries in the first layer and 111 in the last. The prompt is encoded with *full* attention and the compression applies to what is *stored*, so only generation reads the compressed cache. That detail is not cosmetic, because biasing the selection call itself, which would encode the question chunk against the keys it has just chosen, costs 9.5 points on qa_1 alone, even when the whole resident set is then read.

Both rows of each pair call the same selector, so a pair differs only in what the decode step reads. The difference from a baseline run is where the surviving set enters. Rather than adding the kept rows to the read mask, we hide the evicted rows from the attention itself, so they are invisible to the selector's scores, absent from the tail the partition clusters, and absent from the statistics of every cluster. Two consequences are worth stating. First, a cluster can lose most of its members to the eviction, and the accounting still charges its 1.5 rows, so the read we charge is an upper bound on the read we take. Second, the partition is still built over all keys, not only the resident ones, so the k-means objective is slightly the wrong one, which costs accuracy rather than buying it.

Table 17: Table 6 per task (RULER-32K HARD, Llama-3.1-8B-Instruct, 200 samples per task).

method	avg	qa_1	qa_2	vt	fwe	mk_2	mk_3	mv
dense attention	**87.04**	79.0	50.0	95.2	93.3	95.0	100.0	96.8
StreamingLLM (2048)	**28.00**	50.5	37.5	5.3	88.3	5.0	5.0	4.4
+ OpenAttention at 10%	**28.28**	52.0	37.5	5.4	88.7	5.0	5.0	4.4
SnapKV (2048)	**69.98**	77.5	49.0	88.6	82.2	93.5	3.0	96.1
+ OpenAttention at 10%	**69.49**	76.0	48.0	87.7	83.5	93.5	2.5	95.2
PyramidKV (2048)	**70.34**	78.0	49.5	90.6	82.5	93.5	2.0	96.2
+ OpenAttention at 10%	**70.14**	76.5	49.5	89.5	83.3	94.0	2.5	95.6

I CONTEXT-LENGTH SWEEP

Table 7 of §4.5 uses 100 samples per task at every length. The 16K and 32K sets are the public releases of the HashAttention and vAttention evaluation (`xAlg-AI/att-hub-ruler-16k` and `att-hub-ruler-32k`). The 32K column rescores the first 100 samples of existing runs of 200 samples with the same metric. For HashAttention and vAttention these are the runs of Table 11, for OpenAttention it is the run of Table 5, and for MagicPIG it is the $K{=}10$ run described below. The 64K and 96K sets are generated with the RULER scripts (Hsieh et al., 2024) at maximum sequence lengths 65,536 and 98,304 tokens with the Llama-3.1 tokenizer, the base template and the same seven tasks (200 samples generated, the first 100 run), and the answer prefix is handed to the model exactly as in the public sets. Contexts are truncated to $L - 768$ tokens at every length. All methods use the code paths of Table 11, namely HashAttention with the released 64K signature weights, vAttention with the protocol configuration of App. F.2 for a 3% read ($\epsilon{=}\delta{=}0.3$, 50/50 split, measured read reported), and OpenAttention with the QGS, the static+question partition and $C = 64 \cdot L/32768$ clusters with $E{=}0$, so that the proxy stream stays at 0.29% of the read and a cluster holds about 500 tokens at every length. MagicPIG is recalibrated per length, because the realised density of a fixed (K, L) shifts with the context (its $L{=}300$ configuration measures 3.20% at 32K but 3.81% at 16K and 3.41% at 96K), so the rows use $K{=}10$ with $L{=}210$, 300, 300 and 250 at 16K, 32K, 64K and 96K, whose measured reads are 2.99%, 3.20%, 3.19% and 3.02% (App. F.1, whereas the density sweep of Table 11 uses $K{=}9$). Table 18 lists the per-task scores and the reads.

Table 18: Table 7 per task, with the total per-query read (measured for vAttention, MagicPIG and OpenAttention).

context	method	read	avg	qa_1	qa_2	vt	fwe	mk_2	mk_3	mv
16K	dense	100%	87.14	86.0	51.0	93.2	91.0	95.0	98.0	95.8
	MagicPIG	3.0%	46.91	49.0	32.0	83.2	79.7	14.0	0.0	70.5
	HashAttention	3.0%	67.41	84.0	53.0	78.4	83.0	62.0	20.0	91.5
	vAttention	3.0%	52.69	60.0	41.0	73.4	83.7	21.0	1.0	88.8
	OpenAttention (ours)	3.0%	87.49	86.0	54.0	94.0	89.7	95.0	98.0	95.8
32K	dense	100%	87.75	83.0	50.0	95.2	93.3	96.0	100.0	96.8
	MagicPIG	3.2%	51.38	46.0	44.0	83.0	81.7	16.0	1.0	88.0
	HashAttention	3.0%	63.87	67.0	48.0	80.4	87.7	54.0	16.0	94.0
	vAttention	3.0%	55.63	58.0	38.0	77.6	89.3	34.0	3.0	89.5
	OpenAttention (ours)	3.0%	87.38	83.0	51.0	96.0	91.7	95.0	99.0	96.0
64K	dense	100%	85.49	81.0	49.0	90.6	88.3	97.0	98.0	94.5
	MagicPIG	3.2%	45.73	41.0	34.0	73.6	77.0	12.0	0.0	82.5
	HashAttention	3.0%	49.65	64.0	36.0	60.6	86.7	15.0	0.0	85.2
	vAttention	3.0%	54.79	67.0	37.0	83.0	85.0	19.0	3.0	89.5
	OpenAttention (ours)	3.0%	84.82	81.0	51.0	90.8	81.7	97.0	98.0	94.2
96K	dense	100%	78.01	78.0	47.0	61.0	90.3	84.0	89.0	96.8
	MagicPIG	3.0%	33.19	31.0	23.0	57.6	57.0	3.0	0.0	60.8
	HashAttention	3.0%	27.99	14.0	19.0	23.6	74.3	0.0	0.0	65.0
	vAttention	3.0%	40.20	34.0	26.0	50.2	85.7	6.0	0.0	79.5
	OpenAttention (ours)	3.0%	78.10	79.0	47.0	62.8	87.7	84.0	89.0	97.2