ENTMAXKV: SUPPORT-AWARE DECODING FOR ENTMAX ATTENTION

Anonymous authors
Paper under double-blind review

ABSTRACT

Long-context decoding is increasingly limited by KV-cache memory traffic as each generated token attends over a cache whose size grows linearly with context length. Existing sparse decoding methods reduce this cost by selecting subsets of tokens or pages, but are designed for softmax attention, whose dense tails make any truncation discard nonzero probability mass. In contrast, α-entmax produces exact zeros, turning sparse decoding from dense-tail approximation into support recovery: if the selected candidates contain the entmax support, sparse decoding remains exact. While recent entmax kernels enable efficient training, they do not address the autoregressive decoding bottleneck, where dense inference still streams the full KV cache before sparsity is known. In this work, we introduce EntmaxKV, an entmax-native sparse decoding framework that exploits sparsity before KV pages are loaded. EntmaxKV combines query-aware page scoring, support-aware candidate selection, and sparse entmax attention. We analyze truncation error through the dropped probability mass δ, showing that output error is controlled by δ and vanishes when the entmax support is recovered. We further introduce a Gaussian-aware entmax selector that estimates the entmax threshold from lightweight page statistics, adapting the selected budget to the score distribution. Empirically, EntmaxKV drops less probability mass, retains more support tokens, and achieves lower output error than softmax-based sparse decoding at matched KV budgets. On long-context and language modeling benchmarks, it closely matches full-cache entmax while using a small fraction of the KV cache, achieving up to $2.88\times$ (softmax) and $5.5\times$ (entmax) speedup over full attention baselines as well as over $2\times$ speedup over softmax and entmax top-k at 1M context length.[1]

1 INTRODUCTION

Recent large language models (LLMs) support increasingly long context windows, with some systems extending to hundreds of thousands or even millions of tokens (Meta, 2025; Team, 2025). This reasoning over long documents, code repositories, conversation histories, and retrieval-augmented inputs. However, long contexts come at substantial computational cost, especially during autoregressive decoding: at each generation step, the model must attend over the key-value (KV) cache containing all previous tokens. As a result, long-context decoding is often memory-bound, and KV-cache reads can account for over half of the per-token latency (Tang et al., 2024).

A natural way to reduce this cost is to avoid reading the entire cache. Existing sparse decoding methods select only a subset of tokens, blocks, or pages for attention. Some approaches permanently discard tokens from the KV cache using positional, learned, or attention-based heuristics (Zhang et al., 2023; Xiao et al., 2024; Oren et al., 2024; Li et al., 2024). Others keep the full cache available but select a query-dependent subset at each decoding step, using page-level bounds, clustering, or adaptive thresholds to identify the tokens that matter for the current query (Tang et al., 2024; Lin et al., 2025; Singhania et al., 2024; Nawrot et al., 2024; Zhu et al., 2026). These methods exploit the empirical observation that attention is often concentrated on a small subset of the context. However, most sparse decoding methods are designed around softmax attention. Since softmax assigns strictly positive probability to every token, any truncation of the KV cache necessarily discards nonzero probability mass.

[1] Our code will be released upon acceptance.

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

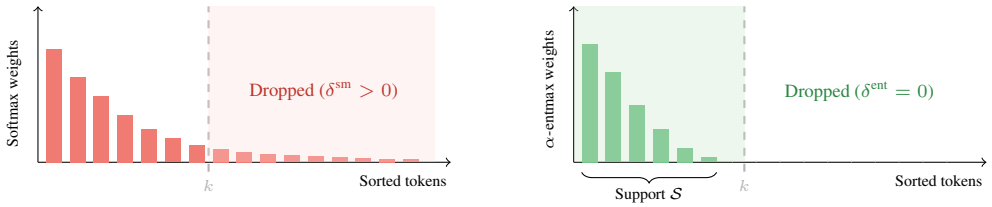

Figure 1: Softmax assigns non-zero probability mass to all tokens, so truncation always discards some mass ($\delta^{\text{sm}} > 0$). In contrast, α-entmax concentrates probability on a sparse support $\mathcal{S}$; when $\mathcal{S}$ falls within the budget, $\delta^{\text{ent}} = 0$. As budget increases, the α-entmax error drops to zero once $k \geq |\mathcal{S}|$, while the softmax error remains positive for any k.

In this work, we revisit sparse decoding through the lens of α-entmax attention (Peters et al., 2019; Blondel et al., 2019). Unlike softmax, α-entmax can produce probability distributions with exact zeros. Therefore, rather than approximating a dense distribution with a truncated one as done by softmax-based approaches, an entmax decoder only needs to recover the support. As shown in Figure 1, if the selected KV pages contain all tokens in the entmax support, sparse decoding is exactly equal to full-cache entmax attention while avoiding the memory traffic associated with irrelevant pages.

Recent work has shown that entmax is both useful for long-context modeling and practical to train at scale (Vasylenko et al., 2026; Gonçalves et al., 2025; 2026). However, these advances do not address the autoregressive decoding bottleneck: full entmax decoding still reads the full KV cache before discovering which tokens receive zero attention weight. We introduce EntmaxKV, an entmax-native sparse decoding framework that exploits sparsity before KV pages are loaded. Concretely, EntmaxKV is a paged selective decoding method for α-entmax attention: it scores KV pages using lightweight metadata, selects candidate pages, and computes entmax attention only over the selected KV entries. Overall, our main contributions are:

- We formulate entmax-native sparse decoding as support recovery: unlike softmax truncation, which always drops nonzero mass, entmax sparse decoding is exact whenever the selected candidates contain the true support.
- We propose a Gaussian-aware entmax selector that estimates the entmax threshold from lightweight page statistics and adapts the candidate budget to the score distribution.
- We show empirically that EntmaxKV retains more support tokens, drops less probability mass, and achieves lower output error than softmax-based sparse decoding while also being faster on long sequences.

2 Background

2.1 Softmax Attention

Let $\boldsymbol{q} \in \mathbb{R}^d$ be the query vector, and let $\{\boldsymbol{k}_j, \boldsymbol{v}_j\}_{j=1}^n$ denote the keys and values stored in the KV cache for the n previous tokens. The single-head attention output is computed as follows:

$$s_j(\boldsymbol{q}) = \frac{\boldsymbol{q}^\top \boldsymbol{k}_j}{\sqrt{d}}, \qquad \boldsymbol{p} = \pi(\boldsymbol{s}(\boldsymbol{q})), \qquad \boldsymbol{o} = \sum_{j=1}^{n} p_j \boldsymbol{v}_j, \tag{1}$$

where $\pi : \mathbb{R}^n \to \triangle_n$ is a probability transformation. Full decoding reads $O(n)$ keys and values per step, making long-context attention memory-bound. Softmax assigns positive probability to every finite score, although attention mass often concentrates on a small subset of tokens (Zhang et al., 2023; Li et al., 2024). Sparse decoding exploits this concentration by reading only selected KV entries.

2.2 The α-entmax Transformation

α-entmax (Peters et al., 2019; Blondel et al., 2019) is a differentiable transformation that generalizes softmax while allowing exactly sparse probability distributions. For an input score vector $\boldsymbol{s} \in \mathbb{R}^n$

and $\alpha > 1$, α-entmax has the form

$$\alpha\text{-entmax}(\boldsymbol{s})_i = [(\alpha - 1)s_i - \tau]_+^{1/(\alpha-1)}, \tag{2}$$

where $[x]_+ = \max\{x, 0\}$ and $\tau \in \mathbb{R}$ is chosen so that the output sums to one. The support of the resulting distribution is

$$\mathcal{S} = \operatorname{supp}(\alpha\text{-entmax}(\boldsymbol{s})) = \{i : (\alpha - 1)s_i > \tau\}. \tag{3}$$

Special cases include softmax as $\alpha \to 1$ and sparsemax at $\alpha = 2$ (Martins & Astudillo, 2016); see Appendix A. Tokens outside $\mathcal{S}$ receive zero probability max and don't contribute to the output, so selecting a superset of $\mathcal{S}$ preserves full-cache entmax attention exactly (Section 3). Section 4 develops page-selection policies that exploit this property before loading KV.

3 Theoretical Analysis

We analyze sparse decoding independently of implementation details such as paging, tiling, or kernel design. Our goal is to identify what controls the approximation error when attention is computed over only a subset of tokens, and why α-entmax differs fundamentally from softmax in this regime. Throughout this section, let $\boldsymbol{p} \in \triangle_n$ denote the full attention distribution over n tokens, let $\boldsymbol{v}_1, \ldots, \boldsymbol{v}_n \in \mathbb{R}^{d_v}$ denote value vectors, and assume $\|\boldsymbol{v}_i\|_2 \le B$ for all i.

3.1 Approximation Error Under Truncation

Consider a selection strategy that keeps a subset $\mathcal{I}_{\text{keep}} \subseteq [n]$ and drops its complement $\mathcal{I}_{\text{drop}} = [n] \setminus \mathcal{I}_{\text{keep}}$. Let

$$\delta = \sum_{i \in \mathcal{I}_{\text{drop}}} p_i \tag{4}$$

denote the dropped probability mass. The truncated and renormalized attention distribution is

$$\tilde{p}_i = \begin{cases} p_i/(1-\delta), & i \in \mathcal{I}_{\text{keep}}, \\ 0, & i \in \mathcal{I}_{\text{drop}}. \end{cases} \tag{5}$$

The full and approximate attention outputs are

$$\boldsymbol{o} = \sum_{i=1}^{n} p_i \boldsymbol{v}_i, \quad \text{and} \quad \tilde{\boldsymbol{o}} = \sum_{i=1}^{n} \tilde{p}_i \boldsymbol{v}_i. \tag{6}$$

Proposition 1 (Approximation error under truncation)**.** *Let $\delta > 0$. If $\|\boldsymbol{v}_i\|_2 \le B$ for all i, then*

$$\|\boldsymbol{o} - \tilde{\boldsymbol{o}}\|_2 \le 2B\delta. \tag{7}$$

Moreover, the constant is tight: equality can be attained with two tokens whose value vectors have norm B and point in opposite directions.

The proof is given in Appendix C. Proposition 1 shows that truncated attention error is controlled by the mass assigned to discarded tokens. The identity of the dropped tokens matters only through their total probability mass. Thus, a sparse decoding method should select candidate tokens that make δ small.

3.2 Exact Sparse Attention with entmax

For softmax, $\delta > 0$ whenever any token is dropped, because every token receives strictly positive probability. Therefore, exact sparse decoding under softmax is only possible when $\mathcal{I}_{\text{keep}} = [n]$. In contrast, α-entmax produces exact zeros, which can lead to full support recovery under truncation.

Proposition 2 (Exact sparse entmax attention)**.** *Let $\boldsymbol{p} = \alpha\text{-entmax}(\boldsymbol{s})$ with support $\mathcal{S}$. If the kept set satisfies $\mathcal{S} \subseteq \mathcal{I}_{keep}$, then $\delta = 0$ and $\tilde{\boldsymbol{o}} = \boldsymbol{o}$ exactly.*

The proof is given in Appendix C.2. This proposition is the key difference between softmax and entmax sparse decoding. For softmax, sparse decoding is always a dense-tail approximation. For entmax, sparse decoding becomes a support-recovery problem. In other words, once the selected candidates contain the support, the sparse output becomes fully exact.

Partial support capture. In practice, a candidate set may not capture the entire entmax support. To separate support recovery from probability-mass recovery, we report the support retention ratio

$$\rho = \frac{|\mathcal{S} \cap \mathcal{I}_{\text{keep}}|}{|\mathcal{S}|}. \tag{8}$$

The ratio ρ measures how many support tokens are kept, whereas δ measures how much probability mass is dropped. We note that Proposition 1 depends on δ, not directly on ρ. Thus, a method can miss some low-mass support tokens and still have small output error.

Furthermore, the advantage of entmax over softmax under the same kept set can be expressed through the dropped mass. Let $\boldsymbol{p}^{\text{sm}} = \text{softmax}(\boldsymbol{s})$ and $\boldsymbol{p}^{\text{ent}} = \alpha\text{-entmax}(\boldsymbol{s})$, and let $\mathcal{S} = \text{supp}(\boldsymbol{p}^{\text{ent}})$ be the entmax support. For the same dropped set $\mathcal{I}_{\text{drop}}$, the difference in dropped mass is

$$\delta^{\text{sm}} - \delta^{\text{ent}} = \underbrace{\sum_{i \in \mathcal{I}_{\text{drop}} \cap \mathcal{S}} \left(p_i^{\text{sm}} - p_i^{\text{ent}}\right)}_{\text{support-miss term}} + \underbrace{\sum_{i \in \mathcal{I}_{\text{drop}} \setminus \mathcal{S}} p_i^{\text{sm}}}_{\text{softmax tail term}}. \tag{9}$$

The second term is always nonnegative and captures probability mass that softmax assigns to tokens outside the entmax support. The first term, however, has no fixed sign and depends on the input score distribution. Thus, entmax's advantage is largest when the candidate set captures the entmax support, in which case the support-miss term vanishes and the dropped entmax mass becomes zero.

4 EntmaxKV

The theory above suggests that sparse decoding with α-entmax should recover the entmax support while avoiding unnecessary KV-cache reads. EntmaxKV implements this principle with a paged decoding abstraction, following the block-based memory layout popularized by PagedAttention and vLLM (Kwon et al., 2023). It scores pages using lightweight metadata, selects candidate pages, and computes α-entmax attention only over the selected KV entries.

4.1 Pipeline Overview

At each decoding step and for each attention head, EntmaxKV receives a query $\boldsymbol{q} \in \mathbb{R}^d$ and a KV cache containing n previous tokens. We partition the cache into pages of size P, yielding $M = \lceil n/P \rceil$ pages. Let $\mathcal{P}_p \subseteq [n]$ denote the token indices in page p. The scaled attention score for token j is $s_j(\boldsymbol{q}) = \boldsymbol{q}^\top \boldsymbol{k}_j / \sqrt{d}$. The selector chooses a set of pages $\mathcal{C}_{\text{page}}(\boldsymbol{q}) \subseteq \{1, \ldots, M\}$ with corresponding token set $\mathcal{C}_{\text{tok}}(\boldsymbol{q}) = \bigcup_{p \in \mathcal{C}_{\text{page}}(\boldsymbol{q})} \mathcal{P}_p$. Entmax attention is then computed over $\mathcal{C}_{\text{tok}}(\boldsymbol{q})$:

$$\tilde{\boldsymbol{p}} = \alpha\text{-entmax}\left(\{s_j(\boldsymbol{q}) : j \in \mathcal{C}_{\text{tok}}(\boldsymbol{q})\}\right), \tag{10}$$

$$\tilde{\boldsymbol{o}} = \sum_{j \in \mathcal{C}_{\text{tok}}(\boldsymbol{q})} \tilde{p}_j \boldsymbol{v}_j. \tag{11}$$

We set $\tilde{p}_j = 0$ for $j \notin \mathcal{C}_{\text{tok}}(\boldsymbol{q})$. If $\mathcal{C}_{\text{tok}}(\boldsymbol{q})$ contains the full-cache entmax support, then Proposition 2 implies that $\tilde{\boldsymbol{o}}$ is exactly equal to the full-cache entmax output.

In terms of efficiency, the dominant cost is loading selected KV pages from memory. In EntmaxKV, we obtain $O\left(|\mathcal{C}_{\text{page}}|Pd\right) = O\left(|\mathcal{C}_{\text{tok}}|d\right)$ instead of $O(nd)$ for dense decoding. Page metadata scoring costs $O(Md) = O((n/P)d)$ and does not load values. Thus, when $|\mathcal{C}_{\text{tok}}| \ll n$, EntmaxKV drastically reduces the dominant KV-cache memory traffic during autoregressive inference.

4.2 Query-Aware Page Scoring

On top of the page-based memory layout, we use query-aware page scoring in the spirit of Quest (Tang et al., 2024). Each page stores lightweight metadata that can be evaluated before loading the full keys and values. Concretely, for each page p, we store coordinate-wise minimum and maximum key vectors,

$$\boldsymbol{k}_{\min}^{(p)} = \min_{j \in \mathcal{P}_p} \boldsymbol{k}_j, \qquad \boldsymbol{k}_{\max}^{(p)} = \max_{j \in \mathcal{P}_p} \boldsymbol{k}_j, \tag{12}$$

where the min and max are taken element-wise. Given a query $\boldsymbol{q}$, the maximum possible score of any token in page p is upper-bounded by

$$\bar{s}^{(p)}_{\text{box}}(\boldsymbol{q}) = \frac{1}{\sqrt{d}} \sum_{i=1}^{d} \max\left(q_i k^{(p)}_{\min,i}, q_i k^{(p)}_{\max,i}\right). \tag{13}$$

For the Gaussian-aware entmax selector (presented next), we additionally store first and second coordinate-wise moments, along with their associated standard deviation:

$$\boldsymbol{k}^{(p)}_{\text{avg}} = \frac{1}{|\mathcal{P}_p|} \sum_{j \in \mathcal{P}_p} \boldsymbol{k}_j, \qquad \boldsymbol{m}^{(p)}_2 = \frac{1}{|\mathcal{P}_p|} \sum_{j \in \mathcal{P}_p} \boldsymbol{k}_j \odot \boldsymbol{k}_j, \qquad \boldsymbol{k}^{(p)}_{\text{std}} = \sqrt{\boldsymbol{m}^{(p)}_2 - \boldsymbol{k}^{(p)}_{\text{avg}} \odot \boldsymbol{k}^{(p)}_{\text{avg}}}. \tag{14}$$

4.3 Candidate Selection Policies

Given the page scores $\{\bar{s}^{(p)}_{\text{box}}(\boldsymbol{q})\}_{p=1}^{M}$, EntmaxKV selects a subset of pages $\mathcal{C}_{\text{page}}(\boldsymbol{q}) \subseteq \{1, \ldots, M\}$. We propose two different policies for page selection.

Top-k page selection. The fixed-budget policy selects the k pages with largest page scores:

$$\mathcal{C}^{\text{top-}k}_{\text{page}}(\boldsymbol{q}) = \text{TopK}_k\left(\left\{\bar{s}^{(p)}_{\text{box}}(\boldsymbol{q})\right\}_{p=1}^{M}\right). \tag{15}$$

Here TopK_k returns the indices of the k largest entries in its input. The token budget is at most kP.

Gaussian-aware entmax selection. Top-k and top-p are generic page-selection rules. Entmax gives a more specific target: select pages likely to contain tokens above the entmax threshold. For page p, we approximate the distribution of token scores inside the page by a scalar Gaussian random variable $S^{(p)} \sim \mathcal{N}(\mu_p(\boldsymbol{q}), \sigma^2_p(\boldsymbol{q}))$, where

$$\mu_p(\boldsymbol{q}) = \frac{\boldsymbol{q}^\top \boldsymbol{k}^{(p)}_{\text{avg}}}{\sqrt{d}}, \qquad \sigma^2_p(\boldsymbol{q}) = \frac{1}{d} \sum_{i=1}^{d} q_i^2 \left(k^{(p)}_{\text{std},i}\right)^2. \tag{16}$$

Here $S^{(p)}$ should be interpreted as the score of a randomly chosen token from page p. This diagonal approximation ignores cross-coordinate covariance, but it can be computed using only page metadata. Recall that entmax probabilities have the form

$$g_\alpha(s; \tau) = [(\alpha - 1)s - \tau]_+^{1/(\alpha-1)}, \tag{17}$$

where the exact threshold τ is chosen so that $\sum_i g_\alpha(s_i; \tau) = 1$. Computing this sum exactly would require visiting every token score. Instead, we replace the empirical score distribution within each page by its Gaussian approximation. Under this model, the expected contribution of one token in page p at threshold τ is $\mathbb{E}_{S^{(p)}}[g_\alpha(S^{(p)}; \tau)]$. Summing over the expected contribution of whole pages gives the approximate normalization equation

$$\sum_{p=1}^{M} |\mathcal{P}_p| \, \mathbb{E}_{S^{(p)}}\left[g_\alpha(S^{(p)}; \hat{\tau})\right] = 1. \tag{18}$$

Solving Eq. (18) gives an estimate $\hat{\tau}$ of the full entmax threshold using only page-level moments. For $\alpha \in \{4/3, 3/2, 2\}$, the expectation has a closed form in terms of truncated Gaussian moments, as shown in Appendix D.[2] Each root-finding iteration costs constant time per page, since each page contributes only through μ_p and σ_p. Given $\hat{\tau}$, we next estimate whether a page may contain at least one token above the threshold. Under the Gaussian model, the probability that all scores in page p are below a value b is approximately

$$\Pr\left[\max_{j \in \mathcal{P}_p} s_j \le b\right] \approx \Phi\left(\frac{b - \mu_p}{\sigma_p}\right)^{|\mathcal{P}_p|}. \tag{19}$$

[2]For these values, $\beta = 1/(\alpha - 1)$ is an integer: $\beta = 1$ for $\alpha = 2$, $\beta = 2$ for $\alpha = 3/2$, and $\beta = 3$ for $\alpha = 4/3$. The expectation then reduces to truncated Gaussian moments of first, second, and third order, respectively.

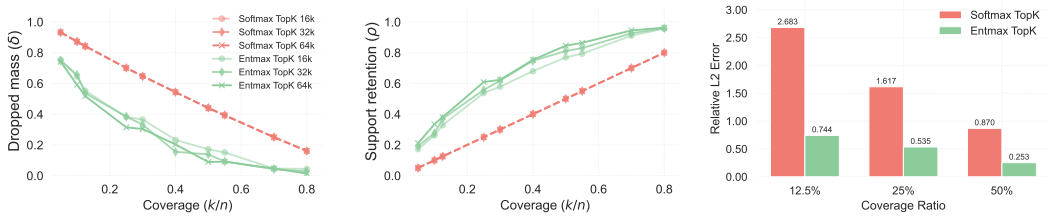

Figure 2: Attention approximation quality in terms of the dropped probability mass δ (left), support retention ρ (middle), and relative output error $\|\boldsymbol{o} - \tilde{\boldsymbol{o}}\|_2 / \|\boldsymbol{o}\|_2$ (right). Coverage ratio denotes the fraction of the KV cache used in the attention computation.

Therefore, the value

$$\bar{s}^{(p)}_{\text{Gauss}} = \mu_p + \sigma_p \Phi^{-1}\left(q^{1/|\mathcal{P}_p|}_{\text{page}}\right) \tag{20}$$

is a q_{page}-confidence estimate of the maximum score in page p. The exponent $1/|\mathcal{P}_p|$ appears because the confidence is for the maximum over the entire page, not for a single token. Finally, we select pages whose estimated maximum is above the estimated entmax threshold:

$$\mathcal{C}^{\text{Gauss}}_{\text{page}}(\boldsymbol{q}) = \left\{ p : (\alpha - 1)\bar{s}^{(p)}_{\text{Gauss}} > \hat{\tau} - \Delta \right\}. \tag{21}$$

Here, $\Delta \geq 0$ is a safety margin hyperparameter. Larger Δ selects more pages and reduces false negatives, whereas lower values make the estimation more precise. We note that unlike the deterministic box bound in Eq. (13), $\bar{s}^{(p)}_{\text{Gauss}}$ is not a worst-case upper bound; instead, it is a model-based estimate that aligns page selection with the entmax support condition $(\alpha - 1)s_i > \tau$.

4.4 Triton Kernel

Our Triton kernels operate directly on a paged KV cache, reading selected pages through their indices. The top-k variant uses Quest-style metadata scoring (Tang et al., 2024), selects pages, and computes entmax over their tokens using histogram-based threshold initialization and Halley refinement, as done by AdaSplash-2 (Gonçalves et al., 2026). The Gaussian variant computes score moments per head and batch element, solves Eq. (18), and selects pages using Eq. (21). It passes the selected indices and estimated threshold to the decode kernel, which performs one additional Halley refinement on the selected scores before computing the output. Both selectors operate on page metadata before reading selected KV. More details about the Triton implementation and our sparse decoding kernels can be found in Appendix F.

5 Experiments

Efficiency benchmarks are executed on synthetic and real data while the the task-based benchmarks use the 1B-parameter and 32K context length softmax and entmax NAPE checkpoints from AdaSplash-2 (Gonçalves et al., 2026). We provide details about the experimental setup in Appendix E.1. Moreover, we emphasize that every time we refer to softmax top-k, it means exactly Quest Tang et al. (2024) ported to Triton and our model architecture.[3]

5.1 Approximation Error Analysis

Figure 2 compares sparse softmax and sparse entmax at matched coverage ratios. Entmax-native selection consistently drops less probability mass and retains a larger percentage of the support, with the largest gains appearing in the most aggressive budget regimes. This behavior is reflected in the output error: sparse entmax produces substantially smaller relative error than sparse softmax at the same coverage. These results agree with Proposition 1, which shows that truncation error is

[3]Porting Quest to Triton and adapting it to our model architecture ensures a fair comparison. Otherwise, our Triton implementation would be orders of magnitude faster than Quest's original implementation.



Figure 3: Passkey retrieval accuracy across KV-cache lengths (x) and token budgets (y). The left panel shows softmax top-k sparse decoding; the right part shows entmax top-k and Gaussian-aware variants.

controlled by the dropped mass δ. They also support the support-recovery interpretation of sparse entmax decoding: once the selected pages contain the entmax support, the approximation error disappears. This advantage also holds for real activations from the model, as shown in Appendix E.2 where we can see that the Gaussian-Aware method shows near perfect accuracy with high sparsity and an adaptive budget. Table 3 in App. E.2.2 shows the comparison between softmax and entmax top-k, with entmax consistently dropping less probability mass, retaining a higher percentage of the support, and achieving lower relative error than softmax top-k.

5.2 Passkey Retrieval

Figure 3 shows a clear separation between softmax and entmax under small sparse budgets. Softmax sparse attention degrades rapidly as the sequence length grows, even when the selected token budget is increased. In contrast, entmax-based sparse decoding maintains perfect retrieval across substantially longer contexts, in some cases using less than 0.1% of the KV cache. These results support the central hypothesis of the paper: because entmax attention has finite support, sparse decoding can remain exact or near-exact when the selected pages cover that support. Moreover, both entmax variants begin to degrade only at around 200k context tokens, roughly 6x the model's training context length. By contrast, the softmax approaches degrade even at in-domain lengths of 12k–20k tokens and typically require substantially larger budgets to recover.

5.3 Language Modeling Perplexity

Figure 4 shows that entmax sparse decoding remains close to full-cache entmax even at aggressive budgets while softmax sparse decoding incurs a larger relative degradation. Notably, entmax with a 1024-token budget achieves lower absolute perplexity than softmax with substantially larger budgets. This suggests that entmax sparsity not only improves approximation quality at the attention level, but also translates into stronger language-modeling behavior under sparse decoding.

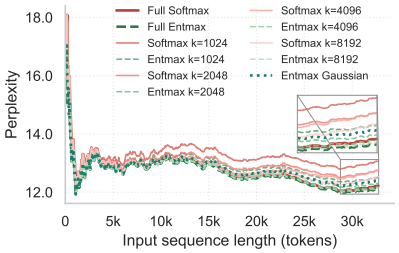

Figure 4: Perplexity on PG19.

5.4 RULER Benchmarks

Table 1 shows that entmax outperforms softmax across context lengths, and that the gap is especially pronounced under sparse decoding. At 16k and 32k softmax degrades sharply while entmax with the same budget remains substantially closer to its dense performance. This indicates that the support-recovery behavior observed in the approximation metrics carries over to downstream long-context tasks. Both entmax variants closely match the baseline performance, moreover, the Gaussian variant is able to avoid the CWE/FWE degradation inherent to top-k approaches shown in MagicPIG (Chen et al., 2024). Twilight Lin et al. (2025) combines a coarse top-k selection with top-p selection and 4-bit K cache quantization for greater efficiency but we found that it was unable to prevent softmax

Table 1: RULER scores across context lengths (n). We use 1024 tokens fox fixed budget variants; Gaussian-aware selection uses a variable budget. Bold and underline indicate the best and second-best average within each context length. Softmax top-k refers to Quest (Tang et al., 2024).

n	Method	mk1	mk2	mk3	mq	mv	niah1	niah2	niah3	cwe	fwe	vt	qa1	qa2	avg
4k	Softmax	84.4	39.8	10.0	20.4	16.3	100.0	100.0	100.0	28.0	30.3	14.3	22.6	37.3	46.4
4k	Softmax (top-k)	79.6	37.8	5.2	35.2	22.0	100.0	100.0	99.8	10.8	24.6	9.5	22.4	38.8	45.1
4k	Softmax (Twilight)	76.4	36.2	6.2	37.2	24.8	100.0	100.0	88.4	10.7	25.0	21.1	23.0	38.0	45.1
4k	Softmax (SnapKV)	9.0	0.6	1.8	5.9	5.6	100.0	10.4	2.2	57.5	58.2	27.84	26.0	26.67	25.5
4k	Entmax	81.8	13.0	29.6	56.3	60.6	100.0	100.0	76.4	48.5	58.1	37.7	29.8	37.4	**56.1**
4k	Entmax (top-k)	81.4	21.4	22.6	60.1	63.4	100.0	100.0	75.6	40.4	57.9	36.8	28.4	36.4	<u>55.7</u>
4k	Entmax (Gaus.)	82.2	12.8	28.6	56.1	62.4	100.0	99.8	75.4	40.4	56.3	37.1	29.8	36.6	55.2
8k	Softmax	76.0	35.0	11.2	27.0	26.2	100.0	100.0	98.8	16.4	23.2	10.4	20.8	20.7	43.5
8k	Softmax (top-k)	70.2	33.0	1.4	29.3	25.0	100.0	93.6	86.4	5.5	23.9	7.0	19.6	15.3	39.3
8k	Softmax (Twilight)	51.0	11.0	0.0	19.8	21.1	89.4	83.8	12.4	3.6	10.1	3.3	17.8	15.4	26.0
8k	Softmax (SnapKV)	25.6	3.6	0.0	15.85	14.25	96.8	31.6	3.2	14.54	24.07	0.96	17.4	13.0	20.1
8k	Entmax	73.6	6.2	19.2	63.5	32.8	100.0	100.0	82.0	34.3	47.1	39.2	28.6	25.8	**50.2**
8k	Entmax (top-k)	72.2	18.2	5.8	61.5	39.7	100.0	100.0	83.2	15.0	40.5	38.5	27.0	25.0	48.2
8k	Entmax (Gaus.)	73.0	4.0	15.6	60.7	32.4	100.0	100.0	82.0	32.1	47.1	27.0	26.7	39.5	<u>49.3</u>
16k	Softmax	82.6	28.8	10.4	22.6	25.8	100.0	100.0	98.8	15.0	23.9	9.7	19.4	20.8	42.9
16k	Softmax (top-k)	54.4	4.0	0.0	4.3	11.2	99.0	88.2	48.6	2.5	9.2	2.1	19.0	16.2	27.6
16k	Softmax (Twilight)	11.6	0.8	0.0	2.2	2.3	34.6	27.0	0.0	1.2	2.1	1.0	10.0	8.7	7.8
16k	Softmax (SnapKV)	17.2	1.8	0.2	8.1	7.3	76.4	18.9	1.1	8.3	13.7	1.2	13.8	11.5	13.8
16k	Entmax	77.8	1.2	9.4	37.8	22.4	100.0	100.0	71.4	23.0	49.3	35.0	27.6	27.7	**44.8**
16k	Entmax (top-k)	73.4	8.4	1.2	33.0	20.6	100.0	98.8	71.2	4.4	42.4	33.4	23.6	25.6	41.2
16k	Entmax (Gaus.)	75.4	2.6	6.2	37.9	21.7	100.0	99.8	72.8	23.5	48.0	34.32	26.8	27.0	<u>44.3</u>
32k	Softmax	56.6	5.4	5.0	21.4	25.2	100.0	84.4	96.0	7.1	3.26	6.5	20.4	20.4	34.7
32k	Softmax (top-k)	25.0	0.6	0.0	3.9	5.5	40.4	12.8	1.0	0.9	0.3	0.2	12.4	11.2	8.78
32k	Softmax (Twilight)	2.4	0.2	0.0	1.5	2.1	13.8	2.0	0.0	0.5	0.1	2.3	6.8	6.4	2.9
32k	Softmax (SnapKV)	5.8	0.4	0.0	2.7	3.1	27.2	4.6	0.2	2.1	3.5	0.4	8.2	7.0	5.0
32k	Entmax	59.4	2.4	4.6	26.1	16.1	100.0	90.6	71.2	11.6	42.1	27.8	25.4	23.1	<u>38.5</u>
32k	Entmax (top-k)	69.8	4.8	0.0	30.3	16.9	100.0	95.0	75.4	1.8	38.0	31.4	25.8	19.2	36.2
32k	Entmax (Gaus.)	70.8	1.8	0.6	33.5	17.5	100.0	98.4	76.4	12.6	45.9	31.2	25.8	20.9	**41.2**

top-k's degradation in our experiments. We also included a comparison with SnapKVLi et al. (2024), even though it is a KV cache cache compression method, not a KV cache selection method, so it incurs a higher risk of losing information.

5.5 Efficiency Benchmarks

Fig. 5 reports end-to-end attention latency normalized by full-cache softmax implemented in FlashDecoding (Dao et al., 2023). Full entmax becomes increasingly expensive with context length, taking twice as much as FlashAttention in Triton at 1M tokens. Entmax top-k closes the gap with softmax top-k (Quest) by avoiding full-cache KV reads and leveraging and optimized paged kernel. The Gaussian-aware entmax selector scales best among the entmax variants. Specifically, while its distributional selection stage adds overhead at shorter lengths, it avoids the explicit top-k bottleneck and grows slower than the other methods with sequence length. At 1M tokens, it runs in 3.6ms, achieving a $2.88\times$ speedup over full-cache softmax FlashDecoding and a $5.5\times$ speedup over the full entmax reference. Overall, EntmaxKV shifts decoding cost from full-cache memory traffic to lightweight page selection and sparse paged entmax computation while twice as fast as both top-k methods.

6 Related Work

KV-cache compression and sparse decoding. Several methods reduce KV-cache cost by evicting or compressing tokens deemed unimportant. H2O evicts tokens based on cumulative attention scores (Zhang et al., 2023); StreamingLLM preserves attention sinks and a sliding window (Xiao et al., 2024); TOVA removes tokens according to current-query attention behavior (Oren et al., 2024); and SnapKV compresses prompts by selecting important KV states (Li et al., 2024). These methods reduce memory footprint, but risk losing information needed by future queries. A complementary line of work keeps the full cache available and selects a query-dependent subset at decoding time. Quest

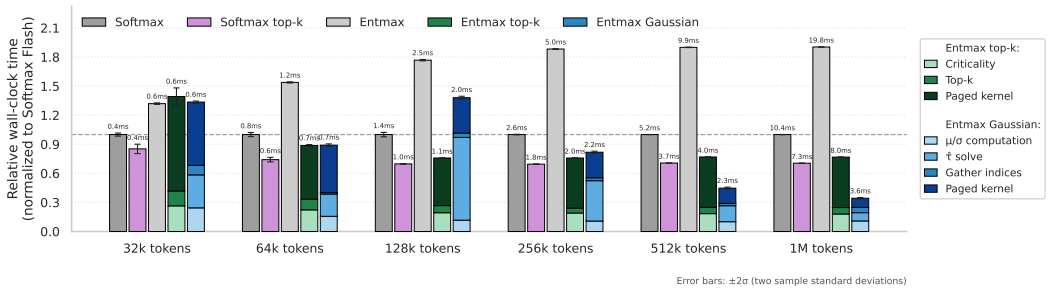

Figure 5: Wall-clock time speedup relative to softmax (FlashDecoding) at different KV-cache lengths. Top-k methods use 25% of the total length. We also report the actual wall-clock time (in ms) on top of each bar for each method.

uses page-level min/max key bounds to identify critical pages without scanning the full cache (Tang et al., 2024). Twilight extends query-aware selection by first forming a top-k candidate pool and then applying top-p filtering within that pool (Lin et al., 2025). Other methods use clustering, dynamic sparse patterns, or compression-based approximations to reduce attention computation during inference (Singhania et al., 2024; Chen et al., 2024; Nawrot et al., 2024; Zhu et al., 2026; Devoto et al., 2024). Our work belongs to this query-aware family, but differs in that the selection objective is tailored to α-entmax support recovery rather than softmax tail approximation.

Sparse probability mappings and entmax attention. Sparsemax and α-entmax are alternatives to softmax that can assign exactly zero probability to low-scoring entries (Martins & Astudillo, 2016; Peters et al., 2019; Blondel et al., 2019). Adaptively sparse transformers learn or tune the sparsity parameter across heads, allowing attention distributions to interpolate between dense and sparse regimes (Correia et al., 2019). ASentmax introduces input-dependent learnable temperatures and argues that sparse attention improves length generalization by mitigating attention dispersion, representational collapse, and over-squashing (Vasylenko et al., 2026).

Efficient entmax kernels. AdaSplash introduced GPU kernels for α-entmax attention (Gonçalves et al., 2025). AdaSplash-2 further accelerates entmax training through a histogram-based initialization and efficient block skipping, showing strong long-context results on benchmarks such as RULER and HELMET (Gonçalves et al., 2026). Our work is orthogonal. While AdaSplash kernels make sparse attention practical during training, decoding still reads the full KV cache before sparsity is known. EntmaxKV exploits entmax sparsity during autoregressive decoding by selecting KV pages before they are loaded, reducing KV-cache memory movement while preserving exactness whenever the support is captured.

7 Conclusion

We introduced EntmaxKV, an entmax-native sparse decoding framework for long-context inference, combining query-aware page scoring, support-aware candidate selection, and sparse entmax attention for autoregressive decoding. We showed that truncation error is controlled by the dropped probability mass δ, and that this error vanishes when the entmax support is recovered. We also introduced a Gaussian-aware entmax selector that estimates the entmax threshold from page-level statistics, allowing the candidate budget to adapt to the score distribution. Empirically, EntmaxKV retains more support tokens, drops less probability mass, and achieves lower output error than softmax-based sparse decoding at matched KV budgets, while improving decoding efficiency over full entmax and dense attention baselines.

AI use statement

In this work, we used generative AI tools to assist in writing the proofs and fix notation issues. We also used generative AI tools for summarizing and analyzing existing literature, searching for information, discover new literature besides the manual investigation, and edit the writing and polishing the paper

for readability. We used generative AI tools to write and edit software code. All uses of AI tools were accompanied by a careful review process. In particular, we manually checked the proofs and made them more accessible, and LLM-generated code was verified and tested for correctness. We take responsibility for the final content of this work, including any text, claims, or artifacts produced with the aid of generative AI.

Reproducibility statement

Our code will be published upon acceptance. All implementation and training details can be found in Appendix F and E.

References

Mathieu Blondel, Andre Martins, and Vlad Niculae. Learning classifiers with fenchel-young losses: Generalized entropies, margins, and algorithms. In Kamalika Chaudhuri and Masashi Sugiyama (eds.), *Proceedings of the Twenty-Second International Conference on Artificial Intelligence and Statistics*, volume 89 of *Proceedings of Machine Learning Research*, pp. 606–615. PMLR, 16–18 Apr 2019. URL https://proceedings.mlr.press/v89/blondel19a.html.

Zhuoming Chen, Ranajoy Sadhukhan, Zihao Ye, Yang Zhou, Jianyu Zhang, Niklas Nolte, Yuandong Tian, Matthijs Douze, Leon Bottou, Zhihao Jia, et al. Magicpig: Lsh sampling for efficient llm generation. *arXiv preprint arXiv:2410.16179*, 2024.

Gonçalo M. Correia, Vlad Niculae, and André F. T. Martins. Adaptively sparse transformers. In Kentaro Inui, Jing Jiang, Vincent Ng, and Xiaojun Wan (eds.), *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pp. 2174–2184, Hong Kong, China, November 2019. Association for Computational Linguistics. doi: 10.18653/v1/D19-1223. URL https://aclanthology.org/D19-1223/.

Tri Dao, Daniel Haziza, Francisco Massa, and Grigory Sizov. Flash-decoding for long-context inference. *CRFM Blog*, 10 2023. URL https://crfm.stanford.edu/2023/10/12/flashdecoding.html. Accessed: 2025-12-01.

Alessio Devoto, Yu Zhao, Simone Scardapane, and Pasquale Minervini. A simple and effective l_2 norm-based strategy for KV cache compression. In Yaser Al-Onaizan, Mohit Bansal, and Yun-Nung Chen (eds.), *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pp. 18476–18499, Miami, Florida, USA, November 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.emnlp-main.1027. URL https://aclanthology.org/2024.emnlp-main.1027/.

Nuno Gonçalves, Marcos V Treviso, and Andre Martins. Adasplash: Adaptive sparse flash attention. In *Forty-second International Conference on Machine Learning*, 2025. URL https://openreview.net/forum?id=OWIPDWhUcO.

Nuno Gonçalves, Hugo Pitorro, Vlad Niculae, Edoardo Ponti, Lei Li, Andre Martins, and Marcos Vinicius Treviso. Adasplash-2: Faster differentiable sparse attention. In *Forty-third International Conference on Machine Learning*, 2026. URL https://openreview.net/forum?id=7qpvff2gWI.

Cheng-Ping Hsieh, Simeng Sun, Samuel Kriman, Shantanu Acharya, Dima Rekesh, Fei Jia, Yang Zhang, and Boris Ginsburg. Ruler: What's the real context size of your long-context language models?, 2024. URL https://arxiv.org/abs/2404.06654.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the 29th Symposium on Operating Systems Principles*, pp. 611–626, 2023.

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

Chaofan Lin, Jiaming Tang, Shuo Yang, Hanshuo Wang, Tian Tang, Boyu Tian, Ion Stoica, Song Han, and Mingyu Gao. Twilight: Adaptive attention sparsity with hierarchical top-p pruning. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025. URL https://openreview.net/forum?id=Ve693NkzcU.

Andre Martins and Ramon Astudillo. From softmax to sparsemax: A sparse model of attention and multi-label classification. In Maria Florina Balcan and Kilian Q. Weinberger (eds.), *International Conference on Machine Learning (ICML)*, volume 48 of *Proceedings of Machine Learning Research*, pp. 1614–1623, New York, New York, USA, 20–22 Jun 2016. PMLR. URL http://proceedings.mlr.press/v48/martins16.html.

AI Meta. The llama 4 herd: The beginning of a new era of natively multimodal ai innovation. *https://ai. meta. com/blog/llama-4-multimodal-intelligence/, checked on*, 4(7):2025, 2025.

Piotr Nawrot, Adrian Łańcucki, Marcin Chochowski, David Tarjan, and Edoardo Ponti. Dynamic memory compression: Retrofitting LLMs for accelerated inference. In *Forty-first International Conference on Machine Learning*, 2024. URL https://openreview.net/forum?id=tDRYrAkOB7.

Matanel Oren, Michael Hassid, Nir Yarden, Yossi Adi, and Roy Schwartz. Transformers are multi-state RNNs. In Yaser Al-Onaizan, Mohit Bansal, and Yun-Nung Chen (eds.), *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pp. 18724–18741, Miami, Florida, USA, November 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.emnlp-main.1043. URL https://aclanthology.org/2024.emnlp-main.1043/.

Ben Peters, Vlad Niculae, and André F. T. Martins. Sparse sequence-to-sequence models. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pp. 1504–1519, Florence, Italy, July 2019. Association for Computational Linguistics. doi: 10.18653/v1/P19-1146. URL https://www.aclweb.org/anthology/P19-1146.

Jack W. Rae, Anna Potapenko, Siddhant M. Jayakumar, Chloe Hillier, and Timothy P. Lillicrap. Compressive transformers for long-range sequence modelling. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=SylKikSYDH.

Prajwal Singhania, Siddharth Singh, Shwai He, Soheil Feizi, and Abhinav Bhatele. Loki: Low-rank keys for efficient sparse attention. In A. Globerson, L. Mackey, D. Belgrave, A. Fan, U. Paquet, J. Tomczak, and C. Zhang (eds.), *Advances in Neural Information Processing Systems*, volume 37, pp. 16692–16723. Curran Associates, Inc., 2024. doi: 10.52202/079017-0532. URL https://proceedings.neurips.cc/paper_files/paper/2024/file/1e027da6bec9ceb2ec37951ceeccae93-Paper-Conference.pdf.

Jiaming Tang, Yilong Zhao, Kan Zhu, Guangxuan Xiao, Baris Kasikci, and Song Han. QUEST: Query-aware sparsity for efficient long-context LLM inference. In *Forty-first International Conference on Machine Learning*, 2024. URL https://openreview.net/forum?id=KzACYw0MTV.

The Kimi Team. Kimi linear: An expressive, efficient attention architecture. *arXiv preprint arXiv:2510.26692*, 2025.

Pavlo Vasylenko, Hugo Pitorro, Andre Martins, and Marcos Vinicius Treviso. Long-context generalization with sparse attention. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=PsB6Lynznk.

Guangxuan Xiao, Yuandong Tian, Beidi Chen, Song Han, and Mike Lewis. Efficient streaming language models with attention sinks. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=NG7sS51zVF.

Zhenyu Zhang, Ying Sheng, Tianyi Zhou, Tianlong Chen, Lianmin Zheng, Ruisi Cai, Zhao Song, Yuandong Tian, Christopher Re, Clark Barrett, Zhangyang Wang, and Beidi Chen. H2o: Heavy-hitter oracle for efficient generative inference of large language models. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023. URL https://openreview.net/forum?id=RkRrPp7GKO.

Kan Zhu, Tian Tang, Qinyu Xu, Zhan Jin, Yile Gu, Zhichen Zeng, Rohan Kadekodi, Liangyu Zhao, Ang Li, Arvind Krishnamurthy, and Baris Kasikci. Tactic: Adaptive sparse attention with clustering and distribution fitting for long-context LLMs. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=tJod11fK1A.

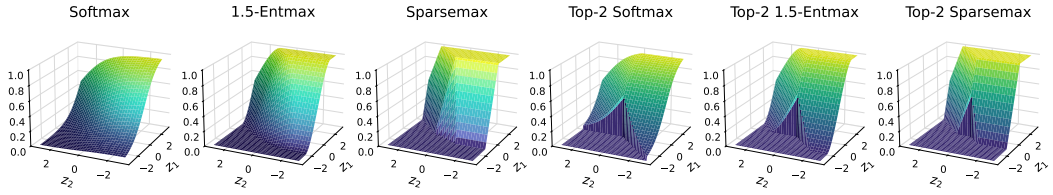

Figure 6: Visualization of softmax and α-entmax for different values of α, alongside top-k variants with $k = 2$. Each panel shows how p_0 varies for the input $\boldsymbol{z} = [0, z_1, z_2]$.

A THE α-ENTMAX TRANSFORMATION

For $\alpha > 1$, the α-entmax transformation (Peters et al., 2019) maps scores $\boldsymbol{s} \in \mathbb{R}^n$ to a probability distribution $\boldsymbol{p} \in \triangle_n$ of the form

$$p_i = [(\alpha - 1)s_i - \tau]_+^{1/(\alpha-1)}, \tag{22}$$

where $\tau \in \mathbb{R}$ is chosen so that

$$\sum_{i=1}^{n} p_i = 1. \tag{23}$$

The notation $[x]_+ = \max\{x, 0\}$ is applied element-wise. The support of $\boldsymbol{p}$ is

$$\begin{aligned} \mathcal{S} = \operatorname{supp}(\boldsymbol{p}) &= \{i : p_i > 0\} \\ &= \{i : (\alpha - 1)s_i > \tau\}. \end{aligned} \tag{24}$$

This threshold characterization is central to our method since a token contributes to entmax attention if and only if its scaled score is above the threshold. Special cases include sparsemax at $\alpha = 2$ (Martins & Astudillo, 2016) and softmax in the limit $\alpha \to 1$ (Peters et al., 2019; Blondel et al., 2019). For $\alpha > 1$, the transformation can produce exact zeros, unlike softmax. α controls the propensity to sparsity, with larger values of α generally encouraging sparser distributions. Figure 6 illustrates α-entmax for $\alpha \in \{1.0, 1.5, 2.0\}$ and top-k variants ($k = 2$) for a 3D input.

B ADDITIONAL DETAILS FOR ENTMAXKV

This appendix provides additional details for the page scoring and Gaussian selection rules used in Section 4.

B.1 DETERMINISTIC PAGE SCORE BOUND

Recall that page p stores coordinate-wise key minima and maxima:

$$k_{\min,i}^{(p)} = \min_{j \in \mathcal{P}_p} k_{j,i}, \qquad k_{\max,i}^{(p)} = \max_{j \in \mathcal{P}_p} k_{j,i}. \tag{25}$$

For any token $j \in \mathcal{P}_p$,

$$k_{j,i} \in \left[k_{\min,i}^{(p)}, k_{\max,i}^{(p)}\right]. \tag{26}$$

Proposition 3 (Page score upper bound). *For any query $\boldsymbol{q}$ and token $j \in \mathcal{P}_p$,*

$$s_j(\boldsymbol{q}) = \frac{\boldsymbol{q}^\top \boldsymbol{k}_j}{\sqrt{d}} \le \bar{s}_{\text{box}}^{(p)}(\boldsymbol{q}), \tag{27}$$

where

$$\bar{s}_{\text{box}}^{(p)}(\boldsymbol{q}) = \frac{1}{\sqrt{d}} \sum_{i=1}^{d} \max\left(q_i k_{\min,i}^{(p)}, q_i k_{\max,i}^{(p)}\right). \tag{28}$$

Proof. For each coordinate i, the scalar $k_{j,i}$ lies between the coordinate-wise minimum and maximum of the page. Therefore,

$$q_i k_{j,i} \le \max\left(q_i k_{\min,i}^{(p)}, q_i k_{\max,i}^{(p)}\right). \tag{29}$$

If $q_i \ge 0$, the maximum is attained at $k_{\max,i}^{(p)}$; if $q_i < 0$, it is attained at $k_{\min,i}^{(p)}$. Summing over coordinates gives

$$\begin{aligned} \boldsymbol{q}^\top \boldsymbol{k}_j &= \sum_{i=1}^{d} q_i k_{j,i} \\ &\le \sum_{i=1}^{d} \max\left(q_i k_{\min,i}^{(p)}, q_i k_{\max,i}^{(p)}\right). \end{aligned} \tag{30}$$

Dividing by $\sqrt{d}$ proves the result. □

Importantly, this bound is conservative because it maximizes each coordinate independently over an axis-aligned box. While the maximizing vector may not correspond to an actual key, the bound is cheap and safe to compute.

B.2 Support-Superset Selection from Page Bounds

Proposition 4 (No false negatives from deterministic page bounds)**.** *Let $\boldsymbol{p} = \alpha\text{-entmax}(\boldsymbol{s})$ with threshold τ, and suppose $\hat{\tau} \le \tau$. Define*

$$\mathcal{C}_{\text{page}} = \left\{p : (\alpha-1)\bar{s}_{\text{box}}^{(p)} > \hat{\tau}\right\}. \tag{31}$$

Then every page containing an entmax support token is selected:

$$\text{supp}(\boldsymbol{p}) \subseteq \bigcup_{p \in \mathcal{C}_{\text{page}}} \mathcal{P}_p. \tag{32}$$

Proof. Let $i \in \text{supp}(\boldsymbol{p})$, and let $p(i)$ be the page containing token i. Since i is in the entmax support,

$$(\alpha-1)s_i > \tau. \tag{33}$$

Because $\hat{\tau} \le \tau$,

$$(\alpha-1)s_i > \hat{\tau}. \tag{34}$$

By Proposition 3,

$$s_i \le \bar{s}_{\text{box}}^{(p(i))}. \tag{35}$$

Therefore,

$$\begin{aligned} (\alpha-1)\bar{s}_{\text{box}}^{(p(i))} &\ge (\alpha-1)s_i \\ &> \hat{\tau}. \end{aligned} \tag{36}$$

Thus page $p(i)$ is selected. Since this holds for every support token, all support tokens lie in selected pages. □

This proposition gives an ideal deterministic picture: if page maxima are upper-bounded and the threshold estimate is conservative, page selection cannot miss support tokens. The Gaussian selector trades this worst-case guarantee for adaptivity.

C Approximation Error Under Truncation

This appendix proves the truncation results used in Section 3.

C.1 Proof of Proposition 1

Proof. Let $\delta > 0$, and recall that $\delta = \sum_{i \in \mathcal{I}_{\text{drop}}} p_i$ and

$$\tilde{p}_i = \begin{cases} p_i/(1-\delta), & i \in \mathcal{I}_{\text{keep}}, \\ 0, & i \in \mathcal{I}_{\text{drop}}. \end{cases} \tag{37}$$

We first compute the L^1 distance between $\boldsymbol{p}$ and $\tilde{\boldsymbol{p}}$. On dropped indices,

$$\sum_{i \in \mathcal{I}_{\text{drop}}} |p_i - \tilde{p}_i| = \sum_{i \in \mathcal{I}_{\text{drop}}} p_i = \delta. \tag{38}$$

On kept indices,

$$\begin{aligned} \sum_{i \in \mathcal{I}_{\text{keep}}} |\tilde{p}_i - p_i| &= \sum_{i \in \mathcal{I}_{\text{keep}}} \left(\frac{p_i}{1-\delta} - p_i \right) \\ &= \left(\frac{1}{1-\delta} - 1 \right) \sum_{i \in \mathcal{I}_{\text{keep}}} p_i \\ &= \frac{\delta}{1-\delta}(1-\delta) \\ &= \delta. \end{aligned} \tag{39}$$

Therefore,

$$\|\boldsymbol{p} - \tilde{\boldsymbol{p}}\|_1 = 2\delta. \tag{40}$$

Now,

$$\begin{aligned} \|\boldsymbol{o} - \tilde{\boldsymbol{o}}\|_2 &= \left\| \sum_{i=1}^{n} (p_i - \tilde{p}_i) \boldsymbol{v}_i \right\|_2 \\ &\leq \sum_{i=1}^{n} |p_i - \tilde{p}_i| \|\boldsymbol{v}_i\|_2 \\ &\leq B \|\boldsymbol{p} - \tilde{\boldsymbol{p}}\|_1 \\ &= 2B\delta. \end{aligned} \tag{41}$$

To see tightness, consider two tokens with probabilities $p_1 = 1 - \delta$ and $p_2 = \delta$, and keep only token the first token. Let $\boldsymbol{v}_1 = B\boldsymbol{u}$ and $\boldsymbol{v}_2 = -B\boldsymbol{u}$ for a unit vector $\boldsymbol{u}$. Then

$$\boldsymbol{o} = (1-\delta)B\boldsymbol{u} - \delta B\boldsymbol{u} = (1-2\delta)B\boldsymbol{u}, \qquad \tilde{\boldsymbol{o}} = B\boldsymbol{u}. \tag{42}$$

Thus

$$\|\boldsymbol{o} - \tilde{\boldsymbol{o}}\|_2 = 2B\delta. \tag{43}$$

□

C.2 Proof of Proposition 2

Proof. If $\mathcal{S} \subseteq \mathcal{I}_{\text{keep}}$, then every dropped index has zero entmax probability:

$$\begin{aligned} i \in \mathcal{I}_{\text{drop}} &\implies i \notin \mathcal{S} \\ &\implies p_i = 0. \end{aligned} \tag{44}$$

Therefore,

$$\delta = \sum_{i \in \mathcal{I}_{\text{drop}}} p_i = 0. \tag{45}$$

Since $\delta = 0$, the renormalized distribution satisfies $\tilde{\boldsymbol{p}} = \boldsymbol{p}$. Hence

$$\tilde{\boldsymbol{o}} = \sum_{i=1}^{n} \tilde{p}_i \boldsymbol{v}_i = \sum_{i=1}^{n} p_i \boldsymbol{v}_i = \boldsymbol{o}. \tag{46}$$

□

D Closed-Form Distributional Entmax Thresholds for Special α

Recall from Section 4.3 that the Gaussian-aware entmax selector estimates the entmax threshold by replacing the empirical sum over token scores with a page-wise Gaussian approximation. The exact entmax normalization condition is

$$\sum_{i=1}^{n} [(\alpha - 1)s_i - \tau]_+^{\beta} = 1, \quad \text{with} \quad \beta = \frac{1}{\alpha - 1}. \tag{47}$$

For each page p, we model token scores by a scalar random variable

$$S^{(p)} \sim \mathcal{N}\left(\mu_p, \sigma_p^2\right). \tag{48}$$

Under this model, the normalization condition is approximated by

$$\sum_{p=1}^{M} |\mathcal{P}_p| \, \mathbb{E}_{S^{(p)}}\left[g_\alpha(S^{(p)}; \tau)\right] = 1, \tag{49}$$

where

$$g_\alpha(s; \tau) = [(\alpha - 1)s - \tau]_+^{\beta}. \tag{50}$$

Thus, each page contributes its expected unnormalized entmax mass, multiplied by the number of tokens in the page. Let $a = \alpha - 1$. For a generic Gaussian score $S \sim \mathcal{N}(\mu, \sigma^2)$, define

$$Y = aS - \tau. \tag{51}$$

Then

$$Y \sim \mathcal{N}\left(\mu_Y, \sigma_Y^2\right), \quad \mu_Y = a\mu - \tau, \quad \sigma_Y = a\sigma. \tag{52}$$

Since

$$g_\alpha(S; \tau) = (Y)_+^{\beta}, \tag{53}$$

we need the truncated Gaussian moment

$$\mathbb{E}\left[g_\alpha(S; \tau)\right] = \mathbb{E}\left[(Y)_+^{\beta}\right] = \mathbb{E}\left[Y^{\beta} \mathbf{1}\{Y > 0\}\right]. \tag{54}$$

Let

$$t = \frac{\mu_Y}{\sigma_Y}, \tag{55}$$

and let ϕ and Φ denote the standard normal pdf and cdf. When $\beta \in \mathbb{N}$, this expectation is a truncated Gaussian moment with a closed form. We use the cases $\beta = 1, 2, 3$, corresponding to $\alpha = 2, 3/2, 4/3$.[4]

Case $\alpha = 2$ ($\beta = 1$). Here $a = 1$ and

$$g_2(s; \tau) = (s - \tau)_+. \tag{56}$$

The first truncated Gaussian moment is

$$\mathbb{E}\left[Y \mathbf{1}\{Y > 0\}\right] = \mu_Y \Phi(t) + \sigma_Y \phi(t). \tag{57}$$

Therefore,

$$\mathbb{E}\left[g_2(S; \tau)\right] = (\mu - \tau)\Phi\left(\frac{\mu - \tau}{\sigma}\right) + \sigma\phi\left(\frac{\mu - \tau}{\sigma}\right). \tag{58}$$

[4]These values are convenient because $\beta = 1/(\alpha - 1)$ is an integer. For non-integer β, the same distributional threshold equation remains valid, but the expectation generally requires numerical evaluation.

Case $\alpha = 3/2$ ($\beta = 2$). Here $a = 1/2$ and

$$g_{3/2}(s;\tau) = \left(\frac{1}{2}s - \tau\right)_+^2. \tag{59}$$

The second truncated Gaussian moment is

$$\mathbb{E}\left[Y^2\mathbf{1}\{Y > 0\}\right] = \left(\mu_Y^2 + \sigma_Y^2\right)\Phi(t) + \mu_Y\sigma_Y\phi(t). \tag{60}$$

Thus,

$$\mathbb{E}\left[g_{3/2}(S;\tau)\right] = \left(\mu_Y^2 + \sigma_Y^2\right)\Phi\left(\frac{\mu_Y}{\sigma_Y}\right) + \mu_Y\sigma_Y\phi\left(\frac{\mu_Y}{\sigma_Y}\right), \tag{61}$$

$$\text{with } \mu_Y = \frac{1}{2}\mu - \tau, \text{ and } \sigma_Y = \frac{1}{2}\sigma. \tag{62}$$

Case $\alpha = 4/3$ ($\beta = 3$). Here $a = 1/3$ and

$$g_{4/3}(s;\tau) = \left(\frac{1}{3}s - \tau\right)_+^3. \tag{63}$$

The third truncated Gaussian moment is

$$\mathbb{E}\left[Y^3\mathbf{1}\{Y > 0\}\right] = \left(\mu_Y^3 + 3\mu_Y\sigma_Y^2\right)\Phi(t) + \left(\mu_Y^2\sigma_Y + 2\sigma_Y^3\right)\phi(t). \tag{64}$$

Therefore,

$$\mathbb{E}\left[g_{4/3}(S;\tau)\right] = \left(\mu_Y^3 + 3\mu_Y\sigma_Y^2\right)\Phi\left(\frac{\mu_Y}{\sigma_Y}\right) + \left(\mu_Y^2\sigma_Y + 2\sigma_Y^3\right)\phi\left(\frac{\mu_Y}{\sigma_Y}\right), \tag{65}$$

$$\text{with } \mu_Y = \frac{1}{3}\mu - \tau, \text{ and } \sigma_Y = \frac{1}{3}\sigma. \tag{66}$$

Solving for the distributional threshold. For each page p, we evaluate the appropriate closed form using $\mu = \mu_p$ and $\sigma = \sigma_p$, then substitute the result into Eq. (49). The estimated threshold $\hat{\tau}$ is the solution of the scalar equation

$$\sum_{p=1}^{M} |\mathcal{P}_p|\, \mathbb{E}_{S^{(p)}}\left[g_\alpha(S^{(p)};\hat{\tau})\right] = 1. \tag{67}$$

We solve this equation with a small fixed number of Newton or Halley iterations over τ. Each iteration has constant cost per page because each page contributes only through μ_p, σ_p, and evaluations of ϕ and Φ. Importantly, this step does not require loading the full KV cache. For generic $\alpha > 1$ with non-integer β, the same distributional equation can still be used, but the expectation $\mathbb{E}[g_\alpha(S;\tau)]$ no longer reduces to a low-order polynomial truncated Gaussian moment. In that case, the expectation can be evaluated numerically.

E Experiments

E.1 Setup

Hardware and models. Experiments run on NVIDIA H200 GPUs. Model-based evaluations use the 1B-parameter softmax and entmax NAPE checkpoints from AdaSplash-2 (Gonçalves et al., 2026), trained with 32k contexts and ALiBi bias. The checkpoints use grouped-query attention with 24 query heads and 12 KV heads; the entmax checkpoint uses $\alpha = 1.5$.

Decoding efficiency. We benchmark the attention computation using 24 heads, with a head dimension of 32 and a batch size of 8. Timing includes page scoring, selection, selected KV reads, and attention computation, but excludes prefill and other model components. We compare full-cache softmax FlashDecoding (Dao et al., 2023), full-cache entmax, and sparse variants. The full-cache entmax reference materializes the score matrix and applies AdaSplash-2's `triton_entmax_v2` kernel. Softmax and entmax top-k share our Triton implementation of Quest's page-scoring and selection pipeline (Tang et al., 2024). After 50 warm-up iterations, we report median wall-clock latency over 50 repeated iterations, synchronized with `torch.cuda.synchronize()` before and after each measurement.

Table 2: Gaussian-aware selection on real model activations. Coverage, dropped mass δ, support-token recall ρ, and relative output error R are reported as mean $\pm$ standard deviation.

KV length	Coverage	$\delta \downarrow$	$\rho \uparrow$	$R \downarrow$
16k	0.365 ± 0.106	0.0003 ± 0.0011	0.9991 ± 0.0021	0.0019 ± 0.0068
32k	0.269 ± 0.057	0.0021 ± 0.0039	0.9984 ± 0.0022	0.0093 ± 0.0146

Approximation quality. We measure dropped probability mass δ, support-token recall $\rho = |\mathcal{S} \cap \mathcal{I}_{\text{keep}}|/|\mathcal{S}|$, and relative output error $R = \|\boldsymbol{o} - \tilde{\boldsymbol{o}}\|_2/\|\boldsymbol{o}\|_2$ against the corresponding full-cache attention output. Here ρ measures support cardinality, not probability mass. Coverage is the fraction of KV entries selected for attention. Fixed-budget comparisons use matched coverage; Gaussian-aware selection determines its own coverage. The synthetic top-k diagnostics use standard-normal Q/K/V. Section E.2.1 evaluates Gaussian-aware selection on captured model activations.

Downstream benchmarks. We evaluate passkey retrieval across context lengths and passkey positions, including randomized positions, and report the percentage of correct answers. RULER (Hsieh et al., 2024) evaluates retrieval, aggregation, tracking, and question answering at 4k–32k contexts. Quest, (Softmax Top-k), Entmax- Top-k, Twilight and SnapKV use a 1024-token budget; Gaussian-aware selection uses a variable budget. PG19 (Rae et al., 2020) perplexity is evaluated on test sequences up to 32k tokens under full-cache and sparse decoding, with fixed budgets of 1024, 2048, 4096, and 8192 tokens.

E.2 Quality metrics using real activations

We evaluate our methods using Q/K/V activations captured from the entmax and softmax checkpoints on six RULER tasks: `niah_multikey_1`, `niah_multikey_2`, `niah_multivalue`, `niah_single_1`, `niah_single_2`, and `vt` and report quality metrics, the Gaussian Aware uses an adaptive budget while the Top-k variants use matched budgets.

E.2.1 Gaussian Entmax

Table 2 reports means and standard deviations over capture/task/layer/step records, comparing sparse outputs with full-cache entmax on the same activations. The selector determines its page budget based on its τ estimate, so coverage reports the selected fraction.

Across these records, the selector retains an average of 99.88% of support tokens at 32.2% coverage, with 0.52% relative output error. Mean coverage decreases from 36.5% at 16k to 26.9% at 32k, while mean relative error increases from 0.19% to 0.93%. The minimum observed support-token recall is 98.48%, and the maximum relative output error is 6.35%. These results demonstrate high empirical support recall at comparatively low coverages.

E.2.2 Entmax top-k and softmax top-k (Quest) under real activations

Table 3 shows quality metrics such as support recall, percentage of probability mass dropped and relative error collected from real activations and compared with a full kv cache execution both for softmax and α-entmax. We can see that across all coverage levels α-entmax drops less probability mass, retains a larger percentage of the support and shows a lower relative error.

F Implementation details

Our kernel algorithm to compute paged candidates in entmax top-k attention is shown in Alg. 1. The gaussian variant uses a small modification to the τ root-finding algorithm of AdaSplash (Gonçalves et al., 2025); we present its kernel in Alg. 2.

Table 3: Quality metrics for entmax and softmax top-k attention on real activations. Values are mean $\pm$ sample standard deviation. R denotes relative output error. Results are aggregated over the tasks available for both models.

KV length	Method	Coverage	$\delta \downarrow$	$\rho \uparrow$	$R \downarrow$
16k	Entmax top-k	10%	0.1737 ± 0.1868	0.7684 ± 0.1784	0.0916 ± 0.0861
16k	Softmax top-k	10%	0.2428 ± 0.1676	0.1 ± 0.0000	0.1738 ± 0.0819
16k	Entmax top-k	12.5%	0.1646 ± 0.1820	0.7823 ± 0.1752	0.0824 ± 0.0805
16k	Softmax top-k	12.5%	0.2315 ± 0.1641	0.125 ± 0.0000	0.1590 ± 0.0756
16k	Entmax top-k	25%	0.1289 ± 0.1563	0.8354 ± 0.1556	0.0549 ± 0.0618
16k	Softmax top-k	25%	0.1854 ± 0.1442	0.25 ± 0.0000	0.1111 ± 0.0560
16k	Entmax top-k	50%	0.0760 ± 0.1004	0.9056 ± 0.1071	0.0273 ± 0.0365
16k	Softmax top-k	50%	0.1128 ± 0.0984	0.5 ± 0.0000	0.0593 ± 0.0353
32k	Entmax top-k	10%	0.1756 ± 0.1812	0.7687 ± 0.1739	0.0925 ± 0.0980
32k	Softmax top-k	10%	0.2350 ± 0.1689	0.1 ± 0.0000	0.1582 ± 0.0692
32k	Entmax top-k	12.5%	0.1643 ± 0.1772	0.7857 ± 0.1723	0.0823 ± 0.0913
32k	Softmax top-k	12.5%	0.2230 ± 0.1661	0.125 ± 0.0000	0.1432 ± 0.0662
32k	Entmax top-k	25%	0.1241 ± 0.1531	0.8445 ± 0.1570	0.0522 ± 0.0684
32k	Softmax top-k	25%	0.1744 ± 0.1477	0.25 ± 0.0000	0.0991 ± 0.0571
32k	Entmax top-k	50%	0.0718 ± 0.0969	0.9115 ± 0.1094	0.0253 ± 0.0396
32k	Softmax top-k	50%	0.1042 ± 0.1016	0.5 ± 0.0000	0.0519 ± 0.0356

Algorithm 1: Paged candidate attention for Entmax Top-k

Input: q_h, K, V, T, L, a_h; selected logical pages S; candidate-token count N; $\alpha \in \{3/2, 2\}$; refinement count $R \geq 1$.

Output: Attention output o_h.

1. Partition the selected page slots into disjoint sets $\mathcal{S}_1, \ldots, \mathcal{S}_P$.
2. **Materialize scores.** In parallel over partitions, visit each slot s and token offset $t \in \{0, \ldots, B-1\}$. Set $\ell = S_s$ and $j = \ell B + t$. For $\ell \geq 0$ and $j < L$, resolve $b = T_\ell$ and store

$$Z_{s,t} \leftarrow (\alpha - 1)\left(\frac{q_h^\top K_{b,t,g(h)}}{\sqrt{d}} + a_h(j - L + 1)\right).$$

   Store $Z_{s,t} \leftarrow -\infty$ for invalid tokens. Reduce partition maxima to obtain

$$m \leftarrow \max_{s,t} Z_{s,t}.$$

   This is the only pass over the selected candidate set.
3. **Initialize the threshold.** Build and reduce score histograms over $[m-1, m]$. Use the histogram estimate to initialize τ, clipped to

$$\tau_{\text{lo}} = m - 1, \qquad \tau_{\text{hi}} = m - N^{1-\alpha}.$$

4. **Refine the threshold.** For $r = 1, \ldots, R$:
   (a) In parallel over partitions, read cached scores and form

$$x_{s,t} \leftarrow [Z_{s,t} - \tau]_+.$$

   On the first iteration, store the page mask

$$C_s \leftarrow \left(\max_t Z_{s,t} > m - 1\right).$$

   (b) Reduce the normalization residual and its derivatives:

$$(f, f', f'') \leftarrow \begin{cases} \left(\sum x_{s,t}^2 - 1,\ -2\sum x_{s,t},\ 2\,|\{(s,t) : Z_{s,t} > \tau\}|\right), & \alpha = 3/2, \\ \left(\sum x_{s,t} - 1,\ -|\{(s,t) : Z_{s,t} > \tau\}|,\ 0\right), & \alpha = 2. \end{cases}$$

   (c) If $f > 0$, set $\tau_{\text{lo}} \leftarrow \tau$. If $f < 0$, set $\tau_{\text{hi}} \leftarrow \tau$.
   (d) Form the Halley proposal

$$\tau_{\text{H}} \leftarrow \tau - \frac{2ff'}{2(f')^2 - ff''}.$$

   Accept it if it lies within the updated bracket, allowing numerical tolerance $\varepsilon = 10^{-6}$; otherwise set $\tau \leftarrow (\tau_{\text{lo}} + \tau_{\text{hi}})/2$.
5. **Accumulate values.** In parallel over partitions, initialize $o_p \leftarrow 0$. For each slot $s \in \mathcal{S}_p$ with C_s true, read its cached scores and identify tokens with $Z_{s,t} > \tau$. For these tokens only, resolve $b = T_{S_s}$, load the value, and add

$$o_p \leftarrow o_p + [Z_{s,t} - \tau]_+^{1/(\alpha-1)} V_{b,t,g(h)}.$$

6. **Return** $o_h \leftarrow \sum_{p=1}^{P} o_p$.

Algorithm 2: Gaussian decode

Input: Query q_h; paged caches K, V; block table T; page key statistics $\mathcal{M}$; sequence length L; ALiBi slope a_h; selection parameters θ; number of τ refinement iterations $R \geq 0$.

Output: Attention output o_h.

1. Compute page and global score statistics:

$$(\mu_{\text{page}}, \sigma_{\text{page}}, \mu, \sigma, N_{\text{hist}}, P_{\text{hist}}) \leftarrow \text{GaussianScoreStats}(q_h, \mathcal{M}, T, L, a_h).$$

2. Estimate the threshold $\widehat{\tau}$ as described in D:
3. Select and pack logical candidate pages estimated to have tokens above the support, and the last page:

$$(S, N) \leftarrow \text{SelectAndPackPages}(\mu_{\text{page}}, \sigma_{\text{page}}, \widehat{\tau}, \sigma, L, B, \theta).$$

   Here N counts valid candidate tokens.
4. Materialize candidate scores and partial maxima:

$$(Z, m_{\text{partial}}) \leftarrow \text{ScoreSelectedPages}(q_h, K, T, S, N, L, a_h).$$

   The kernel partitions the compact candidate-token range across parallel splits and stores scores in FP32.
5. **Initialize the threshold bracket.** If $R > 0$, reduce the partial maxima to obtain $m = \max_j z_j$ over the selected tokens. For $\alpha = 3/2$, the exact candidate-set threshold satisfies

$$\tau^\star \in [m - 1, m].$$

   This bracket contains the root of $f(\tau) = \sum_j [z_j - \tau]_+^2 - 1$: the lower endpoint gives mass at least one, while the upper endpoint gives zero mass.
   Call the bracket-initialization kernel to set

$$\tau_{\text{lo}} \leftarrow m - 1, \qquad \tau_{\text{hi}} \leftarrow m, \qquad \tau \leftarrow \min\{\tau_{\text{hi}}, \max\{\tau_{\text{lo}}, \widehat{\tau}\}\}.$$

   The safeguard state $\mathcal{B}$ stores these bounds and the previous iterate and residual, initialized to τ and 0. During correction, a positive residual raises the lower bound; a negative residual lowers the upper bound. If $R = 0$, use $\tau \leftarrow \widehat{\tau}$ without bracketing.
6. For $r = 1, \ldots, R$:
   (a) $\mathcal{A} \leftarrow \text{AccumulateThresholdStatistics}(Z, N, \tau)$.
   (b) $(\tau, \mathcal{B}) \leftarrow \text{SafeguardedThresholdUpdate}(\mathcal{A}, \tau, \mathcal{B}, r)$.
7. **Accumulate selected values.** Launch SelectedValueAccumulation over the same candidate-token splits used for scoring. Each split p initializes a vector accumulator $o_p \leftarrow 0$ and a scalar mass $M_p \leftarrow 0$, then processes its cached scores in tiles.
   For each valid compact token index i, compute

$$w_i \leftarrow [Z_i - \tau]_+^2, \qquad M_p \leftarrow M_p + w_i.$$

   If every weight in a tile is zero, skip its value-cache loads. Otherwise, map each valid compact index to its selected page and offset:

$$s = \lfloor i/B \rfloor, \qquad t = i \bmod B, \qquad \ell = S_s, \qquad b = T_\ell.$$

   Load the corresponding values and accumulate

$$o_p \leftarrow o_p + \sum_{i \text{ in the tile}} w_i V_{T_{S_{\lfloor i/B \rfloor}}, \, i \bmod B, \, g(h)}.$$

   Only indices $i < N$ participate, excluding padding and invalid tail tokens. Store o_p and M_p in O_{partial} and M_{partial} for the final reduction.
8. $(o_h, M) \leftarrow \text{ReduceOutput}(O_{\text{partial}}, M_{\text{partial}})$.
9. **Return** o_h.