TOKEN-GATED SPARSE ATTENTION

Anonymous authors
Paper under double-blind review

ABSTRACT

Attention-based architectures struggle in long-context decode settings due to the need to load and process the full set of attention keys and values for the sequence (the KV cache) for each newly generated token. Sparse attention approaches alleviate this bottleneck by computing attention over a subset of the KV cache. Prior sparse attention approaches generally impose restrictive structure on the sparsity pattern, such as targeting a fixed amount of sparsity within a given attention head or at any given context length. We propose *token-gated sparse attention (TGSA)*, a new variant of sparse attention that allows each token within a given attention head to choose which elements of the KV cache it reads, resulting in a dynamic and non-uniform allocation of sparsity across heads, layers, and contexts. TGSA introduces an auxiliary set of small *gating heads*, which control whether or not a given element of the regular attention KV cache must be loaded. Computing the gating heads before regular attention allows TGSA to skip chunks of the (much larger) standard attention KV cache load. To evaluate TGSA, we pretrain models at 600M, 1.7B and 4B scales and find that its textual quality is comparable to regular attention and post-trained sparse attention baselines, and that TGSA leads to up to $3.5\times$ faster decoding than regular attention at 64K context length.

1 INTRODUCTION

The need for LLMs to operate over long, autonomous sessions has increased the demand for improving LLMs' long-context performance. One particular challenge for attention-based architectures is generation performance: since the entire key–value (KV) cache of a sequence must be read on every decode step, Transformers exhibit $O(T)$ latency in sequence length T. This has led to a slew of work that tries to efficiently support long-context generation. One class of approaches exploits the empirical sparsity of attention distributions to reduce memory movement without quality degradation and with minimal need for training (Xiao et al., 2023; Tang et al., 2024; Zhang et al., 2023, *inter alia*). Another class focuses on architectural changes that lead to reduced memory movement when serving attention (Ainslie et al., 2023; Jiang et al., 2023), or outright replacing attention with mechanisms that incur sub-linear latency per token (Katharopoulos et al., 2020; Schlag et al., 2021; Gu & Dao, 2023; Yang et al., 2024b, *inter alia*). More recently, there has been growing interest in a middle ground of those two directions: imposing sparse attention during pretraining to enable efficient generation (Yuan et al., 2025; DeepSeek-AI, 2025; Lu et al., 2025; Lai et al., 2026). These architectures introduce heads that allow tokens to vote on which sections of the sequence to attend to. Importantly, these heads are usually designed to select contiguous chunks of the sequence, and each head is given a fixed budget of chunks to select (e.g., top-k selection).

However, the sparse attention literature has long established that sparsity is highly heterogeneous. For example, Cai et al. (2024) find it beneficial to allocate different degrees of sparsity to different layers, and Jiang et al. (2024) show that sparsity patterns across heads are diverse. To this end we propose *token-gated sparse attention* (TGSA), a mechanism that allows every token within an attention head to choose an arbitrary subset of tokens to attend to, enabling heterogeneous sparsity across heads and across tokens within a given head. Specifically, TGSA adds a small (i.e., low-dimensional) binary gate to every query group that is trained to be as sparse as possible without degrading downstream textual quality, and optimized jointly with the rest of the LLM during pretraining. Like other sparse attention methods, TGSA is amenable to a two-tier decoding strategy where a (small) gate key (GK) cache is used to determine which tokens in the (much larger) KV cache to skip, reducing overall memory movement per token.

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.

To evaluate TGSA, we pretrain LLMs at 600M, 1.7B, and 4B scales at a 50 tokens/parameter budget. We find that TGSA yields models with high (i.e., 80%+) sparsity, with minor cost to performance (up to 1.78% degradation in downstream benchmarks and up to 4.97% degradation in long-context benchmarks). We then integrate TGSA into vLLM (Kwon et al., 2023) and find that, despite its heterogeneity, the sparsity can still be exploited to obtain $1.6\times$ the decode throughput of dense attention at 16K context length, increasing to $3.5\times$ at 64K context. Finally, we conduct an analysis of the gating patterns learnt by TGSA and find that the sparsity patterns learnt by gates vary across heads, layers, and contexts, suggesting that TGSA learns to allocate its sparsity budget non-uniformly to maximize performance.

To summarize, in this paper we:

1. Introduce token-gated sparse attention (TGSA), a mechanism for sparse attention that introduces small-dimensional binary gates that allow each token to select an arbitrary, variable-length subset of tokens to attend to in each step.
2. Pretrain TGSA models at 600M, 1.7B, and 4B scales and show that downstream benchmark performance is generally comparable to dense baselines.
3. Implement TGSA generation kernels in vLLM and show that TGSA increases vLLM decode throughput by $1.6\times$ at 16K context length and $3.5\times$ at 64K context length.
4. Show that the learned gating patterns coincide with established sparse attention heuristics, such as sliding window attention and making early and later layers less sparse.

Our hope is that TGSA highlights the viability of training LLMs with arbitrarily sparse attention patterns that can still yield practical throughput improvements during inference.

2 TOKEN-GATED SPARSE ATTENTION

In this section we detail the architecture (§2.1), training methodology (§2.2), and generation procedure (§2.3) for TGSA.

2.1 TGSA ARCHITECTURE

Consider a single attention head:

$$p_{ij} = \frac{1}{Z_j} \exp\left(\frac{1}{\sqrt{d_h}} \langle \boldsymbol{k}_i, \boldsymbol{q}_j \rangle\right); \qquad \text{(Attention distribution)} \tag{1}$$

$$\boldsymbol{o}_j = \sum_{i \le j} p_{ij} \boldsymbol{v}_i, \qquad \text{(Attention output)} \tag{2}$$

where $\boldsymbol{k}_i, \boldsymbol{q}_j, \boldsymbol{v}_i \in \mathbb{R}^{d_h}$ are the d_h-dimensional attention keys, queries and values, and $Z_j = \sum_{i \le j} \exp(\frac{1}{\sqrt{d_h}} \langle \boldsymbol{k}_i, \boldsymbol{q}_j \rangle)$ is the partition function of the attention distribution. We use $\langle \cdot, \cdot \rangle$ to denote an inner (dot) product. For autoregressive inference, all previous keys and values must be loaded to on-chip memory for each token generated, which dominates runtime as the context becomes longer.

The core idea behind TGSA (illustrated in Fig. 1) is to introduce an additional, small (e.g., 16-dimensional) gating head that sits next to the regular attention head. We define the TGSA gating head as carrying its own d_g-dimensional keys and queries $\widehat{\boldsymbol{k}}_i, \widehat{\boldsymbol{q}}_j \in \mathbb{R}^{d_g}$, and producing a *gate logit*:

$$g_{ij} = \frac{a}{\sqrt{d_g}} \left\langle \widehat{\boldsymbol{k}}_i, \widehat{\boldsymbol{q}}_j \right\rangle + b, \qquad \text{(Gate logit)} \tag{3}$$

where $a, b \in \mathbb{R}$ are learnable per-head parameters. A gating function f maps this logit into $[0, 1]$, which is multiplied with the (unnormalized) attention weight:

$$\widehat{p}_{ij} = \frac{1}{\widehat{Z}_j} \exp\left(\frac{1}{\sqrt{d_h}} \langle \boldsymbol{k}_i, \boldsymbol{q}_j \rangle\right) f(g_{ij}) \qquad \text{(Gated attention distribution)} \tag{4}$$

$$\widehat{\boldsymbol{o}}_j = \sum_{i \le j} \widehat{p}_{ij} \boldsymbol{v}_i \qquad \text{(Gated attention output)} \tag{5}$$

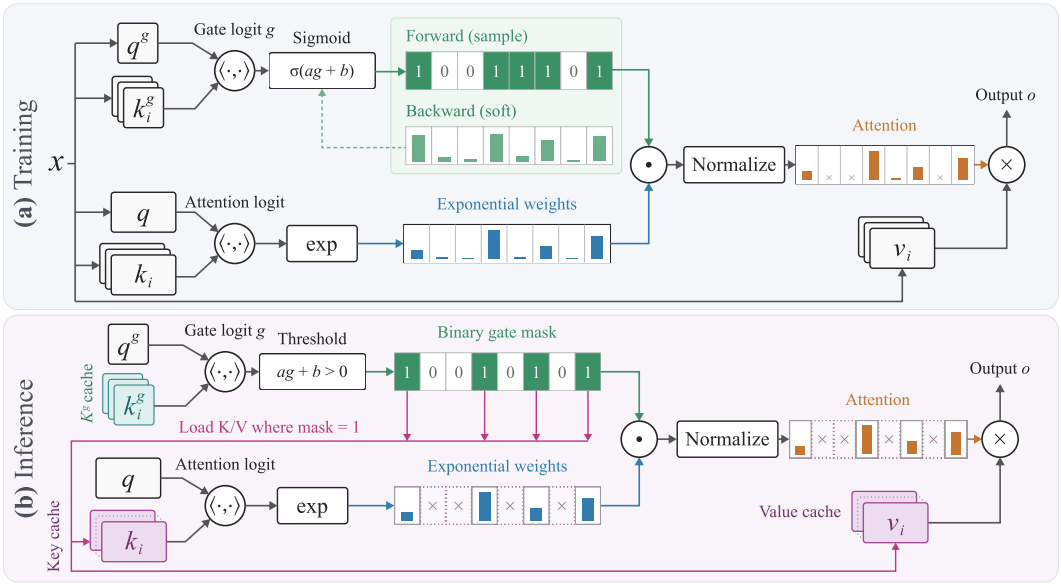


Figure 1: Overview of TGSA. **(a)** During training, we sample gates under sigmoid probabilities, and use the sigmoid gradients during the backward pass. **(b)** During inference, we threshold based on the gate value to determine the specific KV entries to load.

where $\widehat{Z}_j = \sum_{i \le j} \exp\left(\frac{1}{\sqrt{d_h}} \langle \boldsymbol{k}_i, \boldsymbol{q}_j \rangle\right) f(g_{ij})$ is the partition function of the gated attention distribution. In our case, we are especially interested in gating functions with binary outputs $f^{(\text{bin})}(g_{ij}) \in \{0, 1\}$, since this means we can compute attention only over retained positions $\mathcal{S}_j = \{i \le j : f^{(\text{bin})}(g_{ij}) > 0\}$:

$$\widehat{Z}_j^{(\text{bin})} = \sum_{i \in \mathcal{S}_j} \exp(\frac{1}{\sqrt{d_h}} \langle \boldsymbol{k}_i, \boldsymbol{q}_j \rangle) \qquad \text{(Sparse partition function)} \tag{6}$$

$$\widehat{\boldsymbol{o}}_j^{(\text{bin})} = \sum_{i \in \mathcal{S}_j} \widehat{p}_{ij}^{(\text{bin})} \boldsymbol{v}_i \qquad \text{(Sparse attention output)} \tag{7}$$

Importantly, this means that at inference time only the $|\mathcal{S}_j|$ kept entries of the KV cache have to be read,[1] which will enable throughput improvements despite the addition of a (small) gate key (GK) cache (§2.3). While many binary gating functions are possible, in practice during inference we threshold on the value of a sigmoid, i.e., $f^{(\text{bin})}(g_{ij}) = \mathbf{1}\left[\sigma(g_{ij}) > \frac{1}{2}\right] = \mathbf{1}[g_{ij} > 0]$.

2.2 Training

To train a model with TGSA, we use three core techniques: the *straight-through estimator* to propagate the gradients through the discontinuous gating function, a *sparsity loss* to encourage the model to learn sparse attention patterns, and an *attention-matching loss* to encourage the model to learn a sparse attention pattern that minimizes the discrepancy between the gated and ungated attention distributions. In App. D and App. E, we study how to set the strength of the sparsity and the attention-matching objectives, finding that they allow us to smoothly vary the final models' sparsity.

Straight-through estimator. The indicator function $\mathbf{1}[\cdot]$ in $f^{(\text{bin})}$ is not differentiable, which means that to train the model above we must *estimate* gradients. For this, we use a straight-through estimator (STE) for the gradient; in particular, during the forward pass, we use a stochastic gating function $f^{(\text{sto})}$ that samples the binarized gate values from $\text{Bernoulli}(\sigma(g_{ij}))$, while the backward pass instead uses a sigmoid's derivative (i.e., $f^{(\sigma)}(g) = \sigma(g)$).

[1] In practice, we tie the gating heads within a KV group to ensure that every one of its attention heads loads precisely the same set of keys and values.

Sparsity loss. STE-based training does not immediately yield a sparse model since there is no actual incentive to be sparse, e.g., the model can in principle drive all gate logits $g_{ij} \to \infty$ to get a model that is fully dense and behaves identically to regular attention. Inspired by load-balancing losses from the mixture-of-experts literature (Shazeer et al., 2017; Fedus et al., 2022), we introduce a sparsity loss that penalizes open gates. Let the average soft gate openness across the model be:

$$\rho = \frac{1}{|\mathcal{C}|} \sum_{(i,j)\in\mathcal{C}} \sigma(g_{ij}), \tag{8}$$

where $\mathcal{C}$ is the set of valid (causal) query–key pairs across all heads and layers. We then define the sparsity loss as the KL divergence from a Bernoulli with target openness μ to a Bernoulli with probability ρ:

$$\mathcal{L}_{\text{sparsity}}(\mu) = D_{\text{KL}}(\text{Bernoulli}(\mu) \,\|\, \text{Bernoulli}(\rho)). \tag{9}$$

Note that since ρ is computed by averaging sparsity globally across the entire model, this allows the model to trade off additional sparsity in certain layers to make other layers denser. Indeed, in §3.4.2 we find that the model uses this to make early and late layers less sparse than middle layers.

In preliminary experiments, we found that keeping the sparsity target μ static throughout training led to instability (viz., $\mu \approx 0$ destabilized early training) or suboptimal sparsity (viz., low but non-zero μ left room for more sparsity). We thus train using a sparsity schedule: μ remains 1 (i.e., no sparsity) for the first 10% of training, linearly decays to 0 over the next 20% of training, then stays 0.

Attention-matching loss. We found that the sparsity loss was insufficient (alongside the language modeling objective and target sparsity schedule) to obtain stable training. Following similar strategies adopted by recent sparse attention architectures like MSA (Lai et al., 2026) and DSA (DeepSeek-AI, 2025), we introduce an auxiliary objective that incentivizes the model to minimize the KL divergence between the gated and ungated attention distributions. Specifically, if we write out the KL from the (hard-)gated attention distribution (Eq. (4)) to the ungated attention distribution (Eq. (1)), we have

$$D_{\text{KL}}(\widehat{p}^{(\text{bin})}_{\cdot j} \,\|\, p_{\cdot j}) = \sum_{i\le j} \widehat{p}^{(\text{bin})}_{ij} \log \frac{\exp(\frac{1}{\sqrt{d_h}}\langle \boldsymbol{k}_i, \boldsymbol{q}_j\rangle)/\widehat{Z}^{(\text{bin})}_j}{\exp(\frac{1}{\sqrt{d_h}}\langle \boldsymbol{k}_i, \boldsymbol{q}_j\rangle)/Z_j} = \log Z_j - \log \widehat{Z}^{(\text{bin})}_j, \tag{10}$$

which is zero exactly when the gate keeps every position, and grows as the gate discards attention mass. The attention-matching loss averages this quantity over all queries (and heads and layers):

$$\mathcal{L}_{\text{KL}} = \frac{1}{|\mathcal{Q}|} \sum_{j\in\mathcal{Q}} \left(\log Z_j - \log \widehat{Z}^{(\text{bin})}_j\right). \tag{11}$$

Once again, we cannot take the derivative of the gated partition function $\widehat{Z}^{(\text{bin})}_j$ since the gating is binary, so during the backward pass we apply an STE-style estimator: We differentiate the soft partition function[2] $\widehat{Z}^{(\sigma)}_j$ in place of $\widehat{Z}^{(\text{bin})}_j$.[3]

Overall training objective. The two auxiliary losses above act in opposite directions: one tries to maximize sparsity, while the other tries to minimize it. We define our overall training objective as:

$$\mathcal{L} = \mathcal{L}_{\text{LM}} + \lambda_z \mathcal{L}_z + \overbrace{\lambda_{\text{strength}}}^{\text{auxiliary loss weight}} \Big(\underbrace{\lambda_{\text{sparse}}\, \mathcal{L}_{\text{sparsity}}\big(\mu(\tau)\big)}_{\text{closes gates}} + \underbrace{\mathcal{L}_{\text{KL}}}_{\text{keeps attention mass}} \Big), \tag{12}$$

where $\mathcal{L}_{\text{LM}}$ is the usual language modeling cross-entropy and $\mathcal{L}_z = (\log Z^{\text{vocab}})^2$ is the output z-loss (Zoph et al., 2022; Chowdhery et al., 2022). As we show in App. D, this parameterization is convenient in that (in theory) λ_{sparse} can be seen as controlling the overall sparsity of the final model, whereas $\lambda_{\text{strength}}$ trades off the strength of the sparsifying losses against the language modeling loss; in App. E we validate these findings empirically.

[2] This is the partition function that arises from using $f^{(\sigma)}$ instead of $f^{(\text{bin})}$ or $f^{(\text{sto})}$ in eq. (4).

[3] For this loss, we treat the attention scores as constants. This is similar to the gradient detaching of the auxiliary loss employed by MSA (Lai et al., 2026).

Training kernels. We implement training using modified FlashAttention (Dao et al., 2022; Dao, 2023) kernels in Triton. The forward kernel samples the gates on the fly and computes the hard-gated output, soft gate values, and the ungated, hard-gated, and soft-gated log partition functions. The backward kernels additionally fuse the auxiliary losses' contributions into the gate gradients.

2.3 INFERENCE

The whole premise of TGSA is that realized sparsity should translate to improvements in generation throughput. To achieve this, we also write custom generation kernels for Blackwell GPUs.

Generation. For the decode kernel, we build on the FlashInfer (Ye et al., 2025) XQA kernel. Specifically, the vanilla XQA kernel is a FlashDecode-style kernel (Dao et al., 2023) that consumes KV pages, computes partial attention results, and then aggregates them. For the gated kernel, we leverage the same consumer structure, but introduce a new producer kernel responsible for figuring out which gates are open. The producer kernel partitions the gate key cache and produces a set of page–token tuples that are written to a separate buffer. The consumer kernel is adjusted so that instead of reading pages directly, it instead reads the contents of this buffer, issuing HBM loads for the appropriate KV cache tokens.

Prefill kernel. For the prefill kernel, we build on the FlashAttention-4 (Zadouri et al., 2026) forward kernel. Importantly, due to the unstructured nature of sparsity (and the compute-bound characteristics of prefill), this kernel does not attempt to exploit sparsity—it just tries to perform masking as efficiently as possible, without an unreasonable performance hit relative to regular dense attention. An important modification made to the prefill kernel was to reduce the size of the key tiles to free up some space in TMEM for the gate scores.

3 EVALUATION

Our evaluation is split into three parts. First, we pretrain models and evaluate their textual quality on language modeling and long-context benchmarks, to ascertain that we are able to train reasonable models that have the sparsity properties we are after (§3.2). We then explore whether this sparsity translates into practical throughput improvements during generative inference (§3.3). Finally, we conduct some general analysis of TGSA in the context of continued pretraining (§3.4.1) and an interpretability study of what the gates learn (§3.4.2).

3.1 EXPERIMENTAL SETUP

We pretrain LLMs from scratch with the Qwen3 architecture at 0.6B, 1.7B and 4B scales where attention heads are replaced with TGSA. Our training pipeline consists of two stages: a pretraining stage (at roughly 50 tokens/parameter) followed by a long-context extension (LCE) stage.

Pretraining. For our main pretraining experiments we consider an *iso-data* setting, where we fix a token budget for each scale, viz., 50 tokens/parameter with respect to the TGSA model. This is motivated by the observation that (i) training is increasingly constrained by the availability of high-quality data (Kim et al., 2025), and (ii) inference (during both reinforcement learning and serving) is a core bottleneck in modern workloads, so model providers are often willing to spend more pretraining compute to enable faster inference. We also consider an iso-FLOP setting at 600M scale (Fig. 2d).[4] Architectural details and budgets for each setting are shown in Tab. 2 in App. A.[5]

We pretrain on Dolma-3 (Team Olmo et al., 2025), using the Llama-3.2 tokenizer (Meta, 2024) and pack batches into sequences of length 8K with cross-document masking. For our learning rate (LR), we use a $2{,}000$-step linear warmup followed by cosine decay to 10% of the peak LR. We evaluate the quality of the final models on Paloma (Magnusson et al., 2023), reporting macro-averaged PPL across domains. We also report the theoretical KV cache memory movement that would be incurred by a model generating these samples: while a standard GQA model would reload the entire KV

[4] Refer to App. B for details on how FLOPs are calculated.

[5] A detailed comparison of budgets across scales is shown in App. C.

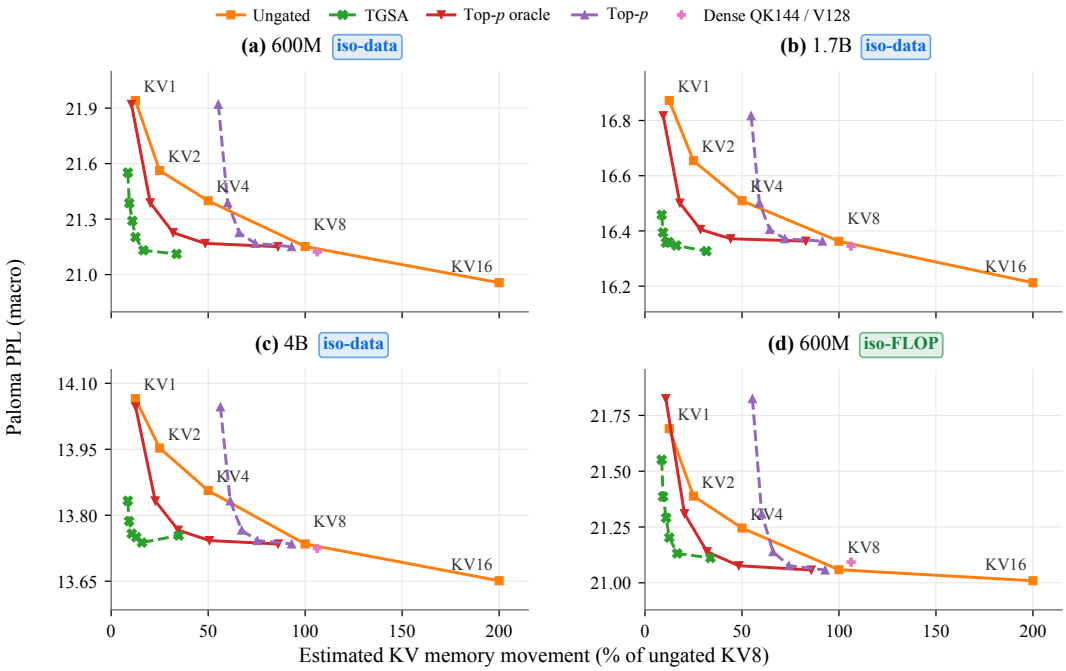


Figure 2: Quality–memory movement trade-off curves for **(a–c)** iso-data training at 600M, 1.7B, and 4B, and **(d)** iso-FLOP training at 600M. KV memory movement is reported relative to each setting's ungated KV8 baseline; lower and to the left is better.

cache on every step, TGSA loads the entire GK cache but only a subset of the KV cache. Since decode is generally memory-bandwidth-bound, this serves as a rough proxy for the expected decode latency decrease (practical throughput gains are evaluated in §3.3). In addition to perplexity, we also evaluate models on the lm-eval-harness benchmark (Gao et al., 2024a).

Long-context extension and evaluation. Our primary efficiency benefit is expected in the long-context setting, since the KV cache size is proportional to sequence length. To evaluate how well TGSA scales to long-context settings we perform long-context extension (LCE) on our models. We do a single YaRN (Peng et al., 2023) 8K $\rightarrow$ 64K iteration, training for 10% of the tokens that a model used during pretraining on a 50% Longmino and 50% Dolma-3 mix (Team Olmo et al., 2025). For LCE, we warm up the LR for 5% of the LCE optimizer steps to the final LR of the pretraining run (i.e., 10% of the peak pretraining LR), and then do cosine decay to 10% of that learning rate (i.e., 1% of the peak pretraining LR). We evaluate long-context capabilities on 12 recall tasks from HELMET (Yen et al., 2025):[6] eleven RULER (Hsieh et al., 2024) tasks and a JSON key–value lookup task. We use greedy decoding, and evaluate at 8K, 16K, 32K and 64K context lengths.

General hyperparameters. We train using AdamW ($\beta_1 = 0.9$, $\beta_2 = 0.95$, weight decay 0.1), gradient clipping at 1.0, with a $\lambda_z = 1 \times 10^{-4}$ z-loss penalty. For the gating heads, we use RMS-normalization on $\widehat{\boldsymbol{q}}, \widehat{\boldsymbol{k}}$ and use a separate RoPE embedding ($\theta = 5 \times 10^5$). We also initialize $a = 1$ and $b = 4$ in eq. (3), so that attention is (almost) fully dense at the start of training. We sweep over λ_{sparse} in our experiments (since these vary sparsity, and in turn final PPL), but set $\lambda_{\text{strength}} = 0.1$ as we found that to work well in practice (App. E). For all TGSA runs, we use GQA with 8 KV groups.

Baselines. We compare against two classes of sparse attention baselines, which aim to cover architectural choices and post-training approaches. First, we consider grouped-query attention (GQA) (Ainslie et al., 2023), which is a ubiquitous architectural choice to reduce memory movement during inference by sharing the KV cache across multiple attention heads. We evaluate GQA models with 1 (i.e., multi-query attention; Shazeer, 2019), 2, 4, 8 and 16 KV groups. Second, we report top-p attention, as this is a fairly reliable post-hoc (training-free) sparsity method (Lin et al., 2025; Zhu et al., 2025; Desai et al., 2026). We apply top-p attention to our (ungated) baseline model

[6] Based on the updated data release.

with 8 KV groups. In particular, we consider both a regular top-p baseline (that loads the whole set of keys but only a subset of values) and also an unrealistic top-p *oracle* baseline (that assumes access to a free oracle that tells the model exactly which subset of keys and values to load at every timestep to accumulate the top-p probability mass, avoiding the complete key load) that is applied post-hoc to the trained model. The oracle can be seen as a rough upper bound for the performance of post-trained sparsity methods. For each of these, we sweep the threshold $p \in \{80\%, 90\%, 95\%, 98\%, 99.9\%\}$.

3.2 Textual quality evaluation

The iso-data pretraining results are shown in Fig. 2a–c. Across the three scales we consider, TGSA achieves the best textual quality–sparsity trade-off. In particular, TGSA outperforms the oracle top-p model, which suggests that TGSA (via the auxiliary losses) is adapting its attention distributions in a way that makes them more amenable to sparsification. TGSA's downstream performance on lm-eval-harness (App. G) is comparable to dense baselines, with at most 1.78% degradation relative to the ungated KV8 baseline. This suggests that the memory movement gains are not at the expense of downstream task performance. Even in the iso-FLOP setting (Fig. 2d), while the 600M TGSA models underperform the baselines on the textual quality axis, they are still Pareto-optimal when accounting for the memory movement reduction.

Interestingly, in the iso-data setting, we also find that some TGSA models attain better perplexity than their ungated counterpart (i.e., ungated, KV8) at 600M and 1.7B scales. Two possible reasons for this are (i) token-gated sparse attention benefits from additional parameters, and (ii) token-gated sparse attention has the effect of increasing the effective dimensionality of the key–query inner product by the gate dimension (i.e., 16). We therefore ran an ablation where we took the ungated KV8 model but increased the query and key dimension by 16. We find that this only partially bridges the PPL gap, suggesting that TGSA may have some beneficial, implicit inductive biases, at least for smaller models in the iso-data setting.

Model		Context length 8K	16K	32K	64K
600M	Ungated	0.530	0.503	0.435	0.375
	TGSA $\lambda_{\text{sparse}} = 1$	0.509	0.490	0.444	0.372
	TGSA $\lambda_{\text{sparse}} = 0.2$	0.574	0.511	0.470	0.362
1.7B	Ungated	0.651	0.580	0.481	0.454
	TGSA $\lambda_{\text{sparse}} = 1$	0.716	0.628	0.543	0.476
	TGSA $\lambda_{\text{sparse}} = 0.2$	0.724	0.660	0.559	0.465
4B	Ungated	0.847	0.783	0.693	0.593
	TGSA $\lambda_{\text{sparse}} = 1$	0.838	0.791	0.669	0.563

Table 1: HELMET macro-average recall for post-LCE models. App. I has per-task breakdowns.

The post-LCE results are shown in Tab. 1 (per-task breakdown can be found in App. I). We see that, on average, TGSA continues to perform comparably to fully dense models even after LCE. Fig. 7 (in App. H) also plots the difference in mean negative log-probability between TGSA and the ungated baseline on 64K Longmino documents, showing that there is no major TGSA degradation in long contexts compared to an ungated model.

3.3 Inference efficiency

Decode kernel. To evaluate the decode speedup due to TGSA, we take the 4B, post-LCE models (with hyperparameters $\lambda_{\text{sparse}} = 1$, $\lambda_{\text{strength}} = 0.1$ for TGSA), and we measure the decode throughput when using vLLM to serve a fixed set of 24 documents from Longmino.[7] We conduct 3 re-runs and take the mean; for the baselines, in addition to the batch size 24 variant, we also consider batch size 25 variants, since eliminating the gates allows us to fit an extra sequence in HBM. The decode results are shown in Fig. 3a. We can see that TGSA underperforms compared to ungated baselines at low context lengths,[8] but beyond 6K tokens TGSA outperforms dense attention, achieving $3.5\times$ the throughput of Nvidia's proprietary TRTLLM-GEN kernel at 64K sequence length.

[7] We pin the commit `8f269a9` from September 4, 2026; we opted for this commit since it implements the trace-replay functionality that we rely on. We use 24 documents since that is the maximum that can be served concurrently using a B300 at the default GPU memory utilization target, i.e., 92%.

[8] This is likely due to kernel launch overheads, since we launch an additional kernel per decode step.

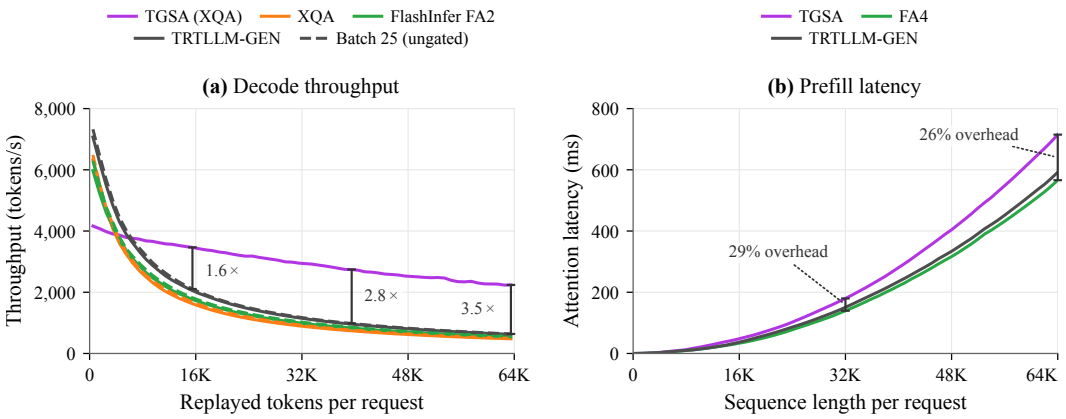


Figure 3: Generation performance of TGSA and dense baselines on a B300 GPU. **(a)** Decode throughput using vLLM. Solid lines correspond to batch size 24, whereas dashed lines correspond to the batch size 25 baselines. **(b)** Attention latency during prefill.

Prefill kernel. To evaluate the prefill kernel, we benchmark the kernel runtime using randomly sampled tensors.[9] We benchmark at progressively longer context lengths from 128 to 64K tokens, and report the median.[10] We consider two dense baselines: FA4 (i.e., the original ungated kernel we based our prefill kernel on), and the closed-source TRTLLM-GEN prefill kernel by Nvidia. The results are shown in Fig. 3b. Our prefill kernel is consistently slower than the best prefill kernel at each configuration, though this difference is at most 30% beyond 32K tokens.

3.4 Analysis

3.4.1 Is pretraining necessary?

The current TGSA recipe applies sparsity throughout pretraining as opposed to being applied during a secondary stage, e.g., by taking a well-trained dense model and doing continued pretraining (CPT) like in DSA. We show how TGSA fares in a CPT setting in Fig. 4, training for an additional 10% of tokens. As baselines, we consider training the KV8 ungated (●) and TGSA (●) models on additional data. We also consider starting from our ungated KV8 model, and adapting it into TGSA (●, varying the target sparsity over $\{60\%, 90\%, 100\%\}$) or DSA[11] (●, varying indexer top-k selection over $\{128, 256, 512\}$).

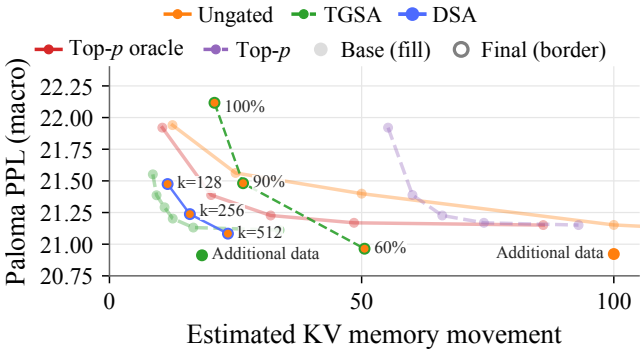


Figure 4: CPT results at 600M scale, with previous pretraining results shown translucently.

While training TGSA and the ungated baseline for extra tokens leads to (mild) improvements in the Pareto frontier (0.219 and 0.230 PPL reduction, respectively, at comparable KV movement), starting from an ungated model and trying to use CPT to *change* it to a gated model substantially underperforms using TGSA for the entirety of (pre)training. If one would insist on starting from a

[9] Sampled from a Gaussian distribution, viz., $\mathcal{N}(0, 1)$. Note that using random inputs here is acceptable, since the prefill kernel does not skip computation based on gate values.

[10] We collect 20 samples at each sequence length and report the median. At small sequence lengths (i.e., $\leq$ 8K) where instrumentation overhead is non-trivial in relation to the kernel, we time 10 kernel invocations and take the average as the sample. We use batch size 24 on a single B300 GPU as in the decode experiments.

[11] To better match our setting, we use a separate indexer per KV group, with 2 indexer query heads of dimension 16 sharing one indexer key. We warm up the indexer for 10% of CPT steps.

dense model and introducing sparsity via CPT, then we find that using a DSA-style recipe is much more effective, though this (mildly) underperforms pretraining with TGSA at the sparser end of the scale (e.g., at roughly matched memory movement, DSA obtains 21.475 PPL at 11.53% memory movement whereas a pretrained TGSA model obtains 21.291 PPL at 10.93% memory movement).

3.4.2 Are gates interpretable?

We took the 4B gated pre- and post-LCE checkpoints and computed the gate statistics for the first 8K tokens of 128 Dolma-3 (Team Olmo et al., 2025) documents. We find that different heads and layers exhibit different aggregate sparsity (Fig. 5), with the very first and last layers naturally converging towards *lower* sparsity. This is consistent with prior sparse-attention designs that retain full attention in early or later layers (Tang et al., 2024; Zarch et al., 2025; Lin et al., 2025; Lu et al., 2025). We also see that LCE leads to reduced sparsity relative to the pretraining phase (from 92.97% to 90.99%); we hypothesize that pretraining at longer sequence lengths, or more care with gate positional embeddings, may be necessary to achieve higher sparsity in long-context settings.

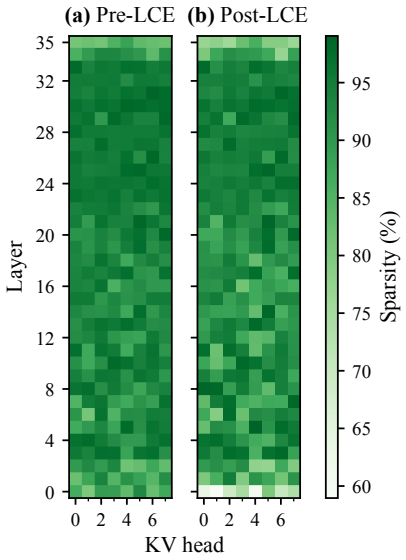


Figure 5: Average gate sparsity across 128 documents of length 8K, **(a)** before and **(b)** after LCE.

Looking at individual heads, we see that the (inter-document) variance of their sparsity can also vary. We selected three heads that exhibited the minimum (Fig. 11a), median (Fig. 11b), and maximum (Fig. 11c) inter-document variability, and inspected those gates' masks on the same four documents (Fig. 12 in App. J). We see that the minimal variance head exhibits a predominantly SWA-like pattern across documents, the median variance head exhibits vertical banding (i.e., consistently kept tokens), and the variable head exhibits document-specific gating behavior, illustrating how gate sparsity is context-dependent.

4 Related work

The two core ingredients in TGSA are the choice of gating heads at an architectural level and the pretraining recipe to induce sparsity and stabilize training. Other work[12] has also explored these components in isolation, and obtained encouraging results. SeerAttention (Gao et al., 2024b) pools keys and queries into contiguous chunks, and learns linear projections that use these pooled representations to gate chunks and skip computation during prefill. DashAttention (Huang et al., 2026) proposes an end-to-end differentiable mechanism for learnable, heterogeneous sparse attention using an α-entmax distribution over chunks, which induces exact zeros that allow chunks to be skipped; in contrast, TGSA uses token-level binary gates trained with a straight-through estimator. DSA (DeepSeek-AI, 2025) is perhaps one of the most well-known successful examples of token-level sparsity (imbued via CPT), though their approach uses top-k selection, precluding variable-length dynamic sparsity. MSA (Lai et al., 2026) recently proposed a pretraining recipe with a block-level top-k sparse selection mechanism; most importantly, we found that several techniques in their recipe (e.g., short dense warmup, KL-based loss with partially detached gradients) were also important for stable training in TGSA. Overall, our hope is that TGSA can motivate further work into methods for dynamic, heterogeneous sparse attention.

5 Conclusion

We propose token-gated sparse attention, a sparse attention architecture that allows an LLM to learn what to attend to, providing it the flexibility to allocate different degrees of sparsity to different heads, layers, and tokens. We find that TGSA offers comparable performance to dense baselines at 600M, 1.7B and 4B scales, but can increase decode throughput by up to $3.5\times$.

[12] We highlight especially relevant work in this section. Refer to App. F for additional related work.

AI STATEMENT

In this work, we used generative AI tools for formulating, reasoning about, proving, and writing mathematical claims, aiding in the design of experiments, aiding with data analysis, and implementing the main method, kernels, and their associated evaluation scripts. We have not used generative AI tools to develop or modify datasets, generate hypotheses or assist with translation. Furthermore, we used AI to help with formatting (including tables, figures, citations, and LaTeX), initial drafting of the paper, general proofreading and editing, helping with literature review, and searching for information. We have reviewed all AI-assisted work: code was checked for correctness and every AI-based edit to the paper was verified. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

REFERENCES

Joshua Ainslie, James Lee-Thorp, Michiel de Jong, Yury Zemlyanskiy, Federico Lebrón, and Sumit Sanghai. GQA: Training generalized multi-query transformer models from multi-head checkpoints. *arXiv preprint arXiv:2305.13245*, 2023.

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. *arXiv preprint arXiv:2406.02069*, 2024. URL https://arxiv.org/abs/2406.02069.

Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, Maarten Bosma, Gaurav Mishra, Adam Roberts, Paul Barham, Hyung Won Chung, Charles Sutton, Sebastian Gehrmann, et al. PaLM: Scaling language modeling with Pathways. *arXiv preprint arXiv:2204.02311*, 2022.

Tri Dao. FlashAttention-2: Faster attention with better parallelism and work partitioning. *arXiv preprint arXiv:2307.08691*, 2023.

Tri Dao, Daniel Y. Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. FlashAttention: Fast and memory-efficient exact attention with IO-awareness. *arXiv preprint arXiv:2205.14135*, 2022. URL https://arxiv.org/abs/2205.14135.

Tri Dao, Daniel Haziza, Francisco Massa, and Grigory Sizov. Flash-Decoding for long-context inference. Stanford CRFM, October 2023. URL https://crfm.stanford.edu/2023/10/12/flashdecoding.html.

DeepSeek-AI. DeepSeek-V2: A strong, economical, and efficient mixture-of-experts language model. *arXiv preprint arXiv:2405.04434*, 2024. URL https://arxiv.org/abs/2405.04434.

DeepSeek-AI. DeepSeek-V3.2: Pushing the frontier of open large language models. *arXiv preprint arXiv:2512.02556*, 2025.

DeepSeek-AI. DeepSeek-V4: Towards highly efficient million-token context intelligence. *arXiv preprint arXiv:2606.19348*, 2026. URL https://arxiv.org/abs/2606.19348.

Aditya Desai, Kumar Krishna Agrawal, Shuo Yang, Alejandro Cuadron, Luis Gaspar Schroeder, Matei Zaharia, Joseph E. Gonzalez, and Ion Stoica. vAttention: Verified sparse attention. In *International Conference on Learning Representations*, 2026. URL https://arxiv.org/abs/2510.05688.

William Fedus, Barret Zoph, and Noam Shazeer. Switch Transformers: Scaling to trillion parameter models with simple and efficient sparsity. *Journal of Machine Learning Research*, 23(120):1–39, 2022. URL https://jmlr.org/papers/v23/21-0998.html.

Leo Gao, Jonathan Tow, Baber Abbasi, Stella Biderman, Sid Black, Anthony DiPofi, Charles Foster, Laurence Golding, Jeffrey Hsu, Alain Le Noac'h, Haonan Li, Kyle McDonell, Niklas Muennighoff, Chris Ociepa, Jason Phang, Laria Reynolds, Hailey Schoelkopf, Aviya Skowron, Lintang Sutawika, Eric Tang, Anish Thite, Ben Wang, Kevin Wang, and Andy Zou. The Language Model Evaluation Harness, July 2024a. URL https://zenodo.org/records/12608602.

Yizhao Gao, Zhichen Zeng, Dayou Du, Shijie Cao, Peiyuan Zhou, Jiaxing Qi, Junjie Lai, Hayden Kwok-Hay So, Ting Cao, Fan Yang, and Mao Yang. SeerAttention: Learning intrinsic sparse attention in your LLMs. *arXiv preprint arXiv:2410.13276*, 2024b. URL https://arxiv.org/abs/2410.13276.

Gemma Team. Gemma 4 technical report. *arXiv preprint arXiv:2607.02770*, 2026. URL https://arxiv.org/abs/2607.02770.

Albert Gu and Tri Dao. Mamba: Linear-time sequence modeling with selective state spaces. *arXiv preprint arXiv:2312.00752*, 2023.

Albert Gu, Karan Goel, and Christopher Ré. Efficiently modeling long sequences with structured state spaces. *arXiv preprint arXiv:2111.00396*, 2021. URL https://arxiv.org/abs/2111.00396.

Jordan Hoffmann, Sebastian Borgeaud, Arthur Mensch, Elena Buchatskaya, Trevor Cai, Eliza Rutherford, Diego de Las Casas, Lisa Anne Hendricks, Johannes Welbl, Aidan Clark, Tom Hennigan, Eric Noland, Katie Millican, George van den Driessche, Bogdan Damoc, Aurelia Guy, Simon Osindero, Karen Simonyan, Erich Elsen, Jack W. Rae, Oriol Vinyals, and Laurent Sifre. Training compute-optimal large language models. *arXiv preprint arXiv:2203.15556*, 2022.

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? *arXiv preprint arXiv:2404.06654*, 2024.

Yuxiang Huang, Nuno M. T. Gonçalves, Federico Alvetreti, Lei Li, Xu Han, Edoardo M. Ponti, André F. T. Martins, and Marcos V. Treviso. DashAttention: Differentiable and adaptive sparse hierarchical attention. *arXiv preprint arXiv:2605.18753*, 2026. URL https://arxiv.org/abs/2605.18753.

Albert Q. Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, Lélio Renard Lavaud, Marie-Anne Lachaux, Pierre Stock, Teven Le Scao, Thibaut Lavril, Thomas Wang, Timothée Lacroix, and William El Sayed. Mistral 7B. *arXiv preprint arXiv:2310.06825*, 2023. URL https://arxiv.org/abs/2310.06825.

Huiqiang Jiang, Yucheng Li, Chengruidong Zhang, Qianhui Wu, Xufang Luo, Surin Ahn, Zhenhua Han, Amir H. Abdi, Dongsheng Li, Chin-Yew Lin, Yuqing Yang, and Lili Qiu. MInference 1.0: Accelerating pre-filling for long-context LLMs via dynamic sparse attention. In *Advances in Neural Information Processing Systems*, volume 37, 2024. URL https://papers.nips.cc/paper_files/paper/2024/hash/5dfbe6f5671e82c76841ba687a8a9ecb-Abstract-Conference.html.

Angelos Katharopoulos, Apoorv Vyas, Nikolaos Pappas, and François Fleuret. Transformers are RNNs: Fast autoregressive transformers with linear attention. *arXiv preprint arXiv:2006.16236*, 2020. URL https://arxiv.org/abs/2006.16236.

Konwoo Kim, Suhas Kotha, Percy Liang, and Tatsunori Hashimoto. Pre-training under infinite compute. *arXiv preprint arXiv:2509.14786*, 2025. URL https://arxiv.org/abs/2509.14786.

Kimi Team et al. Kimi Linear: An expressive, efficient attention architecture. *arXiv preprint arXiv:2510.26692*, 2025. URL https://arxiv.org/abs/2510.26692.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with PagedAttention. *arXiv preprint arXiv:2309.06180*, 2023.

Xunhao Lai, Weiqi Xu, Yufeng Yang, Qiaorui Chen, Yang Xu, Lunbin Zeng, Xiaolong Li, Haohai Sun, Haichao Zhu, Vito Zhang, Jinkai Hu, Jiayao Li, Rui Gao, Zekun Li, Songquan Zhu, Jingkai Zhou, and Pengyu Zhao. MiniMax sparse attention. *arXiv preprint arXiv:2606.13392*, 2026.

Wenhao Li, Yuxin Zhang, Gen Luo, Haiyuan Wan, Ziyang Gong, Fei Chao, and Rongrong Ji. Spotlight Attention: Towards efficient LLM generation via non-linear hashing-based KV cache retrieval. *arXiv preprint arXiv:2508.19740*, 2025. URL https://arxiv.org/abs/2508.19740.

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. *arXiv preprint arXiv:2502.02770*, 2025. URL https://arxiv.org/abs/2502.02770.

Enzhe Lu, Zhejun Jiang, Jingyuan Liu, Yulun Du, Tao Jiang, Chao Hong, Shaowei Liu, Weiran He, Enming Yuan, Yuzhi Wang, Zhiqi Huang, Huan Yuan, Suting Xu, Xinran Xu, Guokun Lai, Yanru Chen, Huabin Zheng, Junjie Yan, Jianlin Su, Yuxin Wu, Neo Y. Zhang, Zhilin Yang, Xinyu Zhou, Mingxing Zhang, and Jiezhong Qiu. MoBA: Mixture of block attention for long-context LLMs. *arXiv preprint arXiv:2502.13189*, 2025. URL https://arxiv.org/abs/2502.13189.

Ian Magnusson, Akshita Bhagia, Valentin Hofmann, Luca Soldaini, Ananya Harsh Jha, Oyvind Tafjord, Dustin Schwenk, Evan Pete Walsh, Yanai Elazar, Kyle Lo, Dirk Groeneveld, Iz Beltagy, Hannaneh Hajishirzi, Noah A. Smith, Kyle Richardson, and Jesse Dodge. Paloma: A benchmark for evaluating language model fit. *arXiv preprint arXiv:2312.10523*, 2023. URL https://arxiv.org/abs/2312.10523.

Meta. Llama 3.2 model card. https://github.com/meta-llama/llama-models/blob/main/models/llama3_2/MODEL_CARD.md, 2024.

OpenAI. gpt-oss-120b & gpt-oss-20b model card. *arXiv preprint arXiv:2508.10925*, 2025. URL https://arxiv.org/abs/2508.10925.

Bowen Peng, Jeffrey Quesnelle, Honglu Fan, and Enrico Shippole. YaRN: Efficient context window extension of large language models. *arXiv preprint arXiv:2309.00071*, 2023.

Imanol Schlag, Kazuki Irie, and Jürgen Schmidhuber. Linear transformers are secretly fast weight programmers. In *Proceedings of ICML*, Proceedings of Machine Learning Research, 2021.

Noam Shazeer. Fast transformer decoding: One write-head is all you need. *arXiv preprint arXiv:1911.02150*, 2019. URL https://arxiv.org/abs/1911.02150.

Noam Shazeer, Azalia Mirhoseini, Krzysztof Maziarz, Andy Davis, Quoc Le, Geoffrey Hinton, and Jeff Dean. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. *arXiv preprint arXiv:1701.06538*, 2017.

Prajwal Singhania, Siddharth Singh, Shwai He, Soheil Feizi, and Abhinav Bhatele. Loki: Low-rank keys for efficient sparse attention. *arXiv preprint arXiv:2406.02542*, 2024. URL https://arxiv.org/abs/2406.02542.

Jiaming Tang, Yilong Zhao, Kan Zhu, Guangxuan Xiao, Baris Kasikci, and Song Han. Quest: Query-aware sparsity for efficient long-context LLM inference. *arXiv preprint arXiv:2406.10774*, 2024.

Team Olmo, Allyson Ettinger, Amanda Bertsch, Bailey Kuehl, et al. Olmo 3. *arXiv preprint arXiv:2512.13961*, 2025.

Guangxuan Xiao, Yuandong Tian, Beidi Chen, Song Han, and Mike Lewis. Efficient streaming language models with attention sinks. *arXiv preprint arXiv:2309.17453*, 2023. URL https://arxiv.org/abs/2309.17453.

Guangxuan Xiao, Jiaming Tang, Jingwei Zuo, Junxian Guo, Shang Yang, Haotian Tang, Yao Fu, and Song Han. DuoAttention: Efficient long-context LLM inference with retrieval and streaming heads. *arXiv preprint arXiv:2410.10819*, 2024. URL https://arxiv.org/abs/2410.10819.

Dongjie Yang, Xiaodong Han, Yan Gao, Yao Hu, Shilin Zhang, and Hai Zhao. PyramidInfer: Pyramid KV cache compression for high-throughput LLM inference. *arXiv preprint arXiv:2405.12532*, 2024a. URL https://arxiv.org/abs/2405.12532.

Songlin Yang, Jan Kautz, and Ali Hatamizadeh. Gated Delta Networks: Improving Mamba2 with delta rule. *arXiv preprint arXiv:2412.06464*, 2024b.

Zihao Ye, Lequn Chen, Ruihang Lai, Wuwei Lin, Yineng Zhang, Stephanie Wang, Tianqi Chen, Baris Kasikci, Vinod Grover, Arvind Krishnamurthy, and Luis Ceze. FlashInfer: Efficient and customizable attention engine for LLM inference serving. *arXiv preprint arXiv:2501.01005*, 2025. URL https://arxiv.org/abs/2501.01005.

Howard Yen, Tianyu Gao, Minmin Hou, Ke Ding, Daniel Fleischer, Peter Izsak, Moshe Wasserblat, and Danqi Chen. HELMET: How to evaluate long-context language models effectively and thoroughly. In *International Conference on Learning Representations (ICLR)*, 2025.

Jingyang Yuan, Huazuo Gao, Damai Dai, Junyu Luo, Liang Zhao, Zhengyan Zhang, Zhenda Xie, Y. X. Wei, Lean Wang, Zhiping Xiao, Yuqing Wang, Chong Ruan, Ming Zhang, Wenfeng Liang, and Wangding Zeng. Native sparse attention: Hardware-aligned and natively trainable sparse attention. *arXiv preprint arXiv:2502.11089*, 2025. URL https://arxiv.org/abs/2502.11089.

Ted Zadouri, Markus Hoehnerbach, Jay Shah, Timmy Liu, Vijay Thakkar, and Tri Dao. FlashAttention-4: Algorithm and kernel pipelining co-design for asymmetric hardware scaling. *arXiv preprint arXiv:2603.05451*, 2026. URL https://arxiv.org/abs/2603.05451.

Hossein Entezari Zarch, Lei Gao, Chaoyi Jiang, and Murali Annavaram. DELTA: Dynamic layer-aware token attention for efficient long-context reasoning. *arXiv preprint arXiv:2510.09883*, 2025. URL https://arxiv.org/abs/2510.09883.

Zhenyu Zhang, Ying Sheng, Tianyi Zhou, Tianlong Chen, Lianmin Zheng, Ruisi Cai, Zhao Song, Yuandong Tian, Christopher Ré, Clark Barrett, Zhangyang Wang, and Beidi Chen. H_2O: Heavy-hitter oracle for efficient generative inference of large language models. *arXiv preprint arXiv:2306.14048*, 2023.

Kan Zhu, Tian Tang, Qinyu Xu, 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. *arXiv preprint arXiv:2502.12216*, 2025. URL https://arxiv.org/abs/2502.12216.

Barret Zoph, Irwan Bello, Sameer Kumar, Nan Du, Yanping Huang, Jeff Dean, Noam Shazeer, and William Fedus. ST-MoE: Designing stable and transferable sparse expert models. *arXiv preprint arXiv:2202.08906*, 2022.

A ARCHITECTURAL AND PRETRAINING DETAILS

	0.6B	1.7B	4B
Layers	28	28	36
Model dim	1,024	2,048	2,560
FFN dim	3,072	6,144	9,728
Query heads	16	16	32
KV heads n_{kv}	8	8	8
Head dim d_h	128	128	128
Gate head dim d_g	16	16	16
Tied embeddings	yes	yes	yes
Parameters (ungated / gated)	572M / 579M	1.67B / 1.69B	3.96B / 3.99B
Global batch (sequences / tokens)	64 / 0.52M	128 / 1.05M	256 / 2.10M
Peak learning rate	1×10^{-3}	1×10^{-3}	1×10^{-3}
Optimizer steps (reference)	55.2k	80.4k	95.0k
Iso-FLOP budget			
Training FLOPs	1.50×10^{20}	–	–
Tokens (ungated / gated)	30.8B / 29.0B	–	–
Tokens/parameter (ungated / gated)	53.8 / 50.0	–	–
Iso-data budget			
Tokens (all configurations)	29.0B	84.3B	199B
Training FLOPs (ungated / gated)	1.41×10^{20} / 1.50×10^{20}	9.68×10^{20} / 9.98×10^{20}	5.48×10^{21} / 5.63×10^{21}
Tokens/parameter (ungated / gated)	50.6 / 50.0	50.4 / 50.0	50.4 / 50.0

Table 2: Architectural and pretraining details for the models in our experiments.

B FLOP ACCOUNTING

To estimate FLOP budgets for a model with N parameters, we price every training step at $6N$ FLOPs per token, plus an additional term that is proportional to the sequence length squared, S^2. We do this since TGSA's gating incurs *quadratic* overhead on both forward and the backward passes, and thus excluding this quadratic attention term would unfairly favor our method. To derive a FLOP budget, we compute the *expected* training FLOPs of a training step by first pre-computing a histogram of effective document lengths, i.e., the sequence lengths we observe after cross-document masking on each batch, and then using this to effectively compute the expectation.

C DATA AND FLOP BUDGETS

Configuration	0.6B	1.7B	4B
Iso-FLOP budget			
Dense GQA, $n_{\text{kv}} = 1$	63.0 (113%)	–	–
Dense GQA, $n_{\text{kv}} = 2$	61.6 (112%)	–	–
Dense GQA, $n_{\text{kv}} = 4$	58.8 (110%)	–	–
Dense GQA, $n_{\text{kv}} = 8$	53.8 (106%)	–	–
Dense GQA, $n_{\text{kv}} = 16$	45.4 (99%)	–	–
TGSA (ours)	50.0 (100%)	–	–
Iso-data budget			
Dense GQA, $n_{\text{kv}} = 1$	55.6	53.8	52.4
Dense GQA, $n_{\text{kv}} = 2$	54.8	53.2	52.2
Dense GQA, $n_{\text{kv}} = 4$	53.4	52.2	51.6
Dense GQA, $n_{\text{kv}} = 8$	50.6	50.4	50.4
Dense GQA, $n_{\text{kv}} = 16$	46.0	47.2	48.0
TGSA (ours)	50.0	50.0	50.0

Table 3: Tokens per parameter used for pretraining in each configuration we consider for Fig. 2. Note that all TGSA methods get the same training data irrespective of auxiliary loss settings. In the iso-FLOP section, we add in parentheses the effective tokens as a fraction of the tokens that the TGSA model uses for training.

D AUXILIARY LOSS DERIVATION

Let x_{ij} be the gate probabilities, i.e., $x_{ij} = \sigma(g_{ij})$, and recall that $\rho = \frac{1}{|\mathcal{C}|}\sum_{(i,j)\in\mathcal{C}} x_{ij}$ (cf. eq. (8)) is the effective (soft) openness, globally averaged across the model. Let m_j be the retained probability mass under (soft) gating, i.e.,

$$m_j \coloneqq \sum_{i:(i,j)\in\mathcal{C}} p_{ij}x_{ij} = \frac{\widehat{Z}_j^{(\sigma)}}{Z_j}. \tag{13}$$

Then the effect of the auxiliary loss, applied to an arbitrary gate, is

$$\begin{aligned}
\mathcal{R}_\mu(x_{ij}) &= \lambda_{\text{strength}}\left[\lambda_{\text{sparse}}\,\mathcal{L}_{\text{sparsity}}(\mu) + \mathcal{L}_{\text{KL}}\right] && (14)\\
&= \lambda_{\text{strength}}\left[\lambda_{\text{sparse}}\,\mathcal{L}_{\text{sparsity}}(\mu) + \frac{1}{|\mathcal{Q}|}\sum_{j\in\mathcal{Q}}\left(\log Z_j - \log \widehat{Z}_j^{(\sigma)}\right)\right] && (15)\\
&= \lambda_{\text{strength}}\left[\lambda_{\text{sparse}}\left(\mu\log\frac{\mu}{\rho} + (1-\mu)\log\frac{1-\mu}{1-\rho}\right) - \frac{1}{|\mathcal{Q}|}\sum_{j\in\mathcal{Q}}\log m_j\right], && (16)
\end{aligned}$$

where in the second equation we expand $\mathcal{L}_{\text{KL}}$ using the soft partition function $\widehat{Z}_j^{(\sigma)}$ in place of $\widehat{Z}_j^{(\text{bin})}$, as explained in §2.2.

Taking the derivative of $\mathcal{R}_\mu(x_{ij})$ with respect to x_{ij}, we have

$$\begin{aligned}
\frac{\partial\mathcal{R}_\mu(x_{ij})}{\partial x_{ij}} &= \lambda_{\text{strength}}\left[\frac{\lambda_{\text{sparse}}}{|\mathcal{C}|}\left(-\frac{\mu}{\rho} + \frac{1-\mu}{1-\rho}\right) - \frac{p_{ij}}{|\mathcal{Q}|m_j}\right] && (17)\\
&= \lambda_{\text{strength}}\Bigg[\underbrace{\frac{\lambda_{\text{sparse}}}{|\mathcal{C}|}\frac{\rho-\mu}{\rho(1-\rho)}}_{\text{(A)}} - \underbrace{\frac{p_{ij}}{|\mathcal{Q}|m_j}}_{\text{(B)}}\Bigg]. && (18)
\end{aligned}$$

Inspecting this gradient, we see a few things:

- Under gradient descent, a gate will be nudged towards being closed if (A) > (B), and vice-versa. Hence, we can interpret (A) as a *closedness* term, whereas (B) is an *openness* term.
- Term (B) is high if the (ungated) attention probability is large relative to the current query's kept mass. This means that the model is incentivized to allocate gate mass to tokens with high underlying attention probability.
- If we assume $\mu = 0$ in (A) (which is the case for the majority of training), we can rewrite (A) $= \frac{\lambda_{\text{sparse}}}{|\mathcal{C}|(1-\rho)}$. Effectively, this means that (i) as $\rho \to 1$ (i.e., as gates become open), (A) pushes them to become closed again, and (ii) increasing λ_{sparse} incentivizes closedness.
- The target openness μ can be interpreted as reducing (A), and indeed when $\rho < \mu$, it causes (A) to compound with (B) to drive gates to be open.
- The $\lambda_{\text{strength}}$ hyperparameter does not control the sign of the auxiliary loss gradient, i.e., it does not affect the direction that the loss nudges the gates, but it does control how aggressively they are moved in that direction.

E EFFECT OF LOSS HYPERPARAMETERS

The analysis in App. D suggests that final sparsity should (mostly) depend on λ_{sparse}, whereas $\lambda_{\text{strength}}$ should control how aggressively we are optimizing towards the sparsity objective as opposed to the regular language modeling objective. To verify this empirically, we trained a suite of 600M models,

varying λ_{sparse} and $\lambda_{\text{strength}}$. Due to the large number of models, we train for only 20 tokens/parameter (i.e., Chinchilla-optimally, Hoffmann et al., 2022), and with a slightly higher peak learning rate of 2×10^{-3}. Overall, we see in Fig. 6 that beyond very low $\lambda_{\text{strength}}$ settings, where the auxiliary loss is likely too weak to act, λ_{sparse} is mostly responsible for controlling the sparsity of the final model, with more noticeable PPL degradation emerging beyond 90% sparsity, at $\lambda_{\text{sparse}} = 3$.

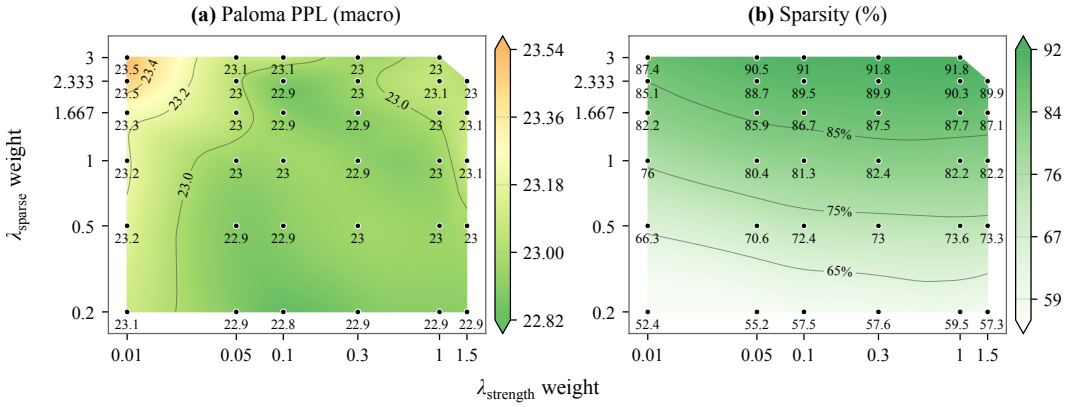


Figure 6: Results of $(\lambda_{\text{strength}}, \lambda_{\text{sparse}})$ sweep at 600M scale. Points are measured values, whereas contours are interpolated from the measurements. Note these experiments train for 20 tokens/parameter, and use a slightly larger learning rate. **(a)** Final Paloma (Magnusson et al., 2023) (macro) PPL of each run. **(b)** Observed sparsity during training.

F Additional Related Work

Sparse attention. The main motivation behind sparse attention is that the Transformer architecture itself, as an inductive bias, leads to empirically sparse attention distributions in practice. This means that the KV cache loads of (a long tail of) low-probability tokens can be skipped with negligible effect on generation quality. Early work in this area focused on training-free approaches (e.g., Tang et al., 2024; Zhang et al., 2023; Xiao et al., 2023) but recent work has also benefited to varying degrees from calibration or additional training (Xiao et al., 2024; Singhania et al., 2024; Li et al., 2025; Huang et al., 2026; Lu et al., 2025). There has also been work exploring how to allocate memory movement budget across both sequence position (Lin et al., 2025; Zhu et al., 2025) and layers (Cai et al., 2024; Yang et al., 2024a), finding that dynamic budgets can outperform static, uniform budgets.

There has been growing interest in architectures that are *pretrained* with sparsity, reasoning that better performance should be possible if sparsity is present throughout pretraining. NSA (Yuan et al., 2025) combines SWA, attention over MLP-compressed blocks, and exact attention over tokens in top-k blocks selected using compressed attention scores. MoBA (Lu et al., 2025) uses a similar block-based selection mechanism, but uses a dot product between the query and the mean-pooled key representations in a block. DeepSeek-V3.2 (DeepSeek-AI, 2025) introduced DeepSeek sparse attention (DSA), which takes an existing dense attention model, introduces additional *indexer* heads, and trains them to approximate the attention distribution via continued pretraining; during inference the indexer heads are used to determine a top-k subset of tokens to perform attention on. MiniMax Sparse Attention (MSA; Lai et al., 2026) similarly uses an indexer to select blocks to attend to, though unlike DSA, the sparsity mechanism is present throughout pretraining. DeepSeek-V4 (DeepSeek-AI, 2026) introduces compressed sparse attention (CSA), which applies the lightning indexer on 4-token-wise compressed representations and has the compressed representations partake in attention.

Other memory-efficient architectures. Perhaps the most widely-adopted technique for memory movement reduction during inference is grouped-query attention (GQA; Ainslie et al., 2023), which shares the same KV entries across multiple attention heads; multi-query attention (MQA; Shazeer, 2019) goes further and has all heads share the same keys and values. Multi-head latent attention (MLA; DeepSeek-AI, 2024) can be thought of as going further by using a shared, low-

dimensional representation for both keys and values. More radical approaches seek to change the asymptotic complexity of generating a token. Linear attention (Katharopoulos et al., 2020; Schlag et al., 2021) and state space models (Gu et al., 2021; Gu & Dao, 2023) replace the attention operation with a sequence-mixing operation that admits a recurrent form, allowing $O(1)$ decode latency. Sliding window attention (SWA; Jiang et al., 2023) constrains attention to a fixed history of tokens. In general, these $O(1)$ layers sacrifice quality (e.g., long-context capability) for inference speed-ups; hybrid models (e.g., Kimi Team et al., 2025; OpenAI, 2025; Gemma Team, 2026) interleave $O(1)$ and $O(T)$ layers to achieve a better trade-off between the two.

G Downstream benchmark scores

Tab. 4 shows lm-eval-harness (Gao et al., 2024a) results for each of the iso-data models we trained. Note that we only report results for pre-LCE checkpoints here.

Size	Model	ARC-c	ARC-e	BoolQ	COPA	HSwag	LAMB	OBQA	PIQA	RACE	SciQ	Wino	Avg
600M	Dense KV1	28.2	55.6	57.3	73.0	44.5	47.0	31.8	69.0	31.7	81.4	54.9	52.2
	Dense KV2	29.2	58.1	49.1	73.0	45.2	48.2	33.2	69.4	31.4	85.0	53.9	52.3
	Dense KV4	29.4	56.0	55.9	69.0	45.7	48.1	32.2	69.5	31.0	83.2	54.1	52.2
	Dense KV8	28.7	55.4	58.1	70.0	46.5	47.7	31.0	68.9	32.2	84.0	54.6	52.5
	Dense KV16	28.3	57.1	56.2	71.0	46.7	48.5	33.6	68.9	32.3	84.6	55.2	52.9
	Wide QK	29.2	57.4	53.4	69.0	46.2	49.6	33.0	68.9	31.8	85.9	54.5	52.6
	TGSA $\lambda_{\text{sparse}} = 0.2$	27.6	58.3	48.5	71.0	46.6	48.7	31.8	69.2	31.2	84.1	55.7	52.1
	TGSA $\lambda_{\text{sparse}} = 1$	28.8	56.9	57.8	70.0	45.8	49.8	32.2	68.0	32.5	85.9	56.2	53.1
	TGSA $\lambda_{\text{sparse}} = 2$	28.8	57.3	55.3	68.0	46.1	48.4	30.8	68.8	31.5	84.4	54.7	52.2
	TGSA $\lambda_{\text{sparse}} = 3$	29.4	58.2	55.9	72.0	46.5	49.6	33.6	69.4	31.7	85.2	55.2	53.3
	TGSA $\lambda_{\text{sparse}} = 5$	28.0	56.8	56.1	72.0	46.3	49.1	31.4	68.3	32.0	82.7	54.1	52.5
	TGSA $\lambda_{\text{sparse}} = 7$	28.3	56.6	54.6	69.0	45.9	48.7	30.6	68.0	31.4	81.0	55.2	51.8
1.7B	Dense KV1	36.0	66.7	59.0	78.0	58.2	59.6	37.0	72.8	35.4	89.9	60.9	59.4
	Dense KV2	36.4	66.2	64.1	75.0	59.0	59.2	37.6	72.3	35.8	88.5	61.8	59.6
	Dense KV4	37.3	66.0	63.1	78.0	59.5	60.8	36.0	73.0	36.0	89.7	58.5	59.8
	Dense KV8	38.5	67.6	61.7	75.0	60.3	60.7	37.2	74.3	36.8	87.8	61.9	60.2
	Dense KV16	38.1	66.8	61.0	80.0	60.8	61.2	37.4	73.0	37.2	89.3	60.6	60.5
	Wide QK	39.6	67.6	62.7	82.0	60.1	60.8	36.2	73.3	37.1	89.8	59.7	60.8
	TGSA $\lambda_{\text{sparse}} = 0.2$	38.4	69.6	63.1	79.0	60.3	61.0	35.6	72.6	36.3	89.3	61.4	60.6
	TGSA $\lambda_{\text{sparse}} = 1$	37.1	67.9	65.0	73.0	60.2	62.3	37.4	72.3	36.6	88.8	60.9	60.1
	TGSA $\lambda_{\text{sparse}} = 2$	39.5	69.0	58.7	81.0	60.1	61.6	38.0	73.1	37.2	91.2	62.3	61.1
	TGSA $\lambda_{\text{sparse}} = 3$	37.3	68.3	64.1	77.0	60.1	61.7	36.4	73.2	37.2	89.1	61.1	60.5
	TGSA $\lambda_{\text{sparse}} = 5$	37.1	65.3	58.8	79.0	60.2	61.2	36.4	73.0	36.6	89.3	60.9	59.8
	TGSA $\lambda_{\text{sparse}} = 7$	37.8	67.5	62.6	81.0	60.1	60.7	37.4	72.4	36.8	88.1	61.9	60.6
4B	Dense KV1	43.6	69.9	69.7	85.0	68.6	67.5	42.4	77.1	39.8	91.2	66.1	65.5
	Dense KV2	44.5	72.1	70.3	81.0	68.3	68.2	40.8	76.3	39.9	89.6	65.1	65.1
	Dense KV4	45.2	72.9	69.7	84.0	68.9	67.6	40.2	77.2	41.2	90.2	65.8	65.7
	Dense KV8	46.6	74.5	72.0	84.0	69.0	69.1	40.2	77.3	41.4	92.2	66.5	66.6
	Dense KV16	46.4	72.8	73.1	83.0	69.6	68.9	40.6	77.2	41.4	91.4	68.2	66.6
	Wide QK	45.3	72.2	71.7	86.0	69.0	69.1	41.2	77.4	40.7	91.6	64.8	66.3
	TGSA $\lambda_{\text{sparse}} = 0.2$	46.4	73.5	74.2	86.0	69.6	68.3	40.4	77.7	40.4	93.1	65.2	66.8
	TGSA $\lambda_{\text{sparse}} = 1$	45.1	69.7	68.5	81.0	69.2	69.2	39.8	77.4	40.3	92.7	66.8	65.4
	TGSA $\lambda_{\text{sparse}} = 2$	46.7	75.3	73.8	83.0	69.0	68.8	41.8	77.5	40.9	91.9	65.5	66.7
	TGSA $\lambda_{\text{sparse}} = 3$	44.4	71.3	70.9	86.0	69.0	68.7	42.2	76.9	41.3	91.7	67.0	66.3
	TGSA $\lambda_{\text{sparse}} = 5$	47.1	74.7	70.7	81.0	69.4	68.8	40.6	77.4	40.9	91.5	66.5	66.2
	TGSA $\lambda_{\text{sparse}} = 7$	46.5	73.4	71.6	87.0	69.4	68.4	40.2	77.4	40.0	91.4	65.3	66.4

Table 4: Per-task scores (%) on the lm-eval-harness benchmark, for the models in Fig. 2.

H LONG-CONTEXT LANGUAGE MODELING

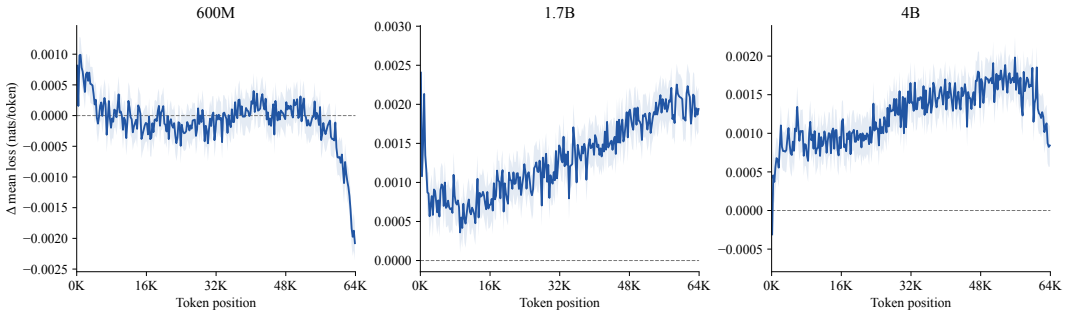


Figure 7: Difference in mean NLL (TGSA minus the ungated model), as a function of token position. The averages are taken using 64K-token documents from Longmino, with the shaded area showing the 95% confidence interval around the mean. A positive value indicates that the ungated model is better than the TGSA model at language modeling, though in practice we note that the scales are small enough that NLL degradation is fairly modest.

I LONG-CONTEXT RECALL, PER FAMILY

Figs. 8 to 10 show the per-task breakdown for the HELMET benchmark reported in Tab. 1.

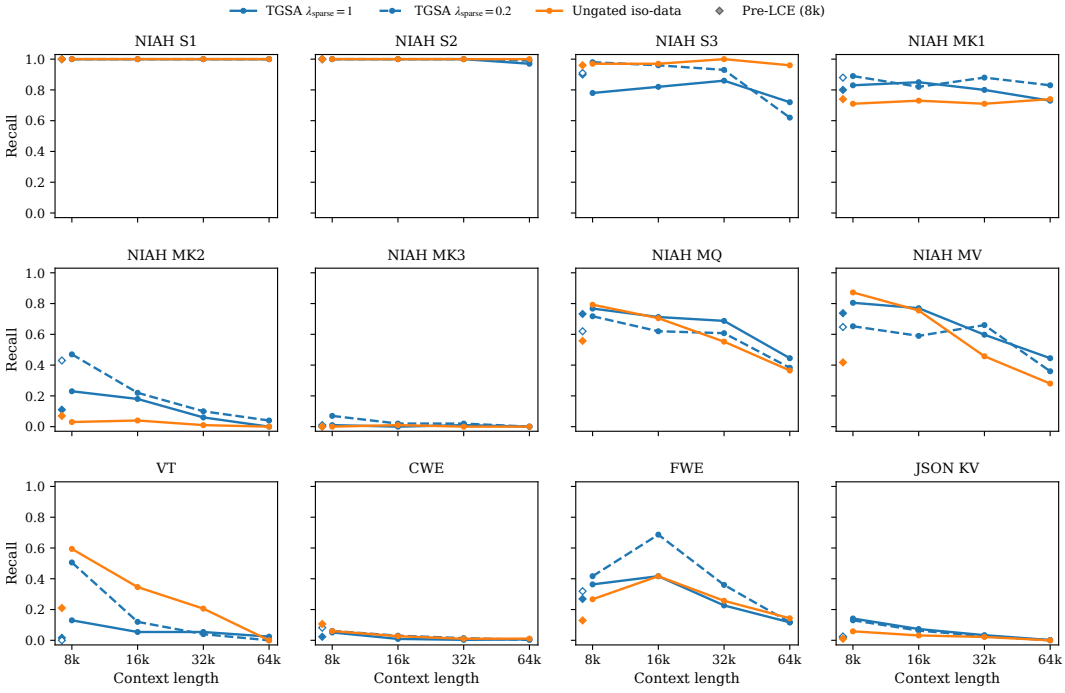


Figure 8: HELMET recall at 600M for TGSA ($\lambda_{\text{sparse}} = 0.2$ and $\lambda_{\text{sparse}} = 1$) and the ungated iso-data baseline. Lines show post-LCE checkpoints; diamonds show pre-LCE recall at 8K.

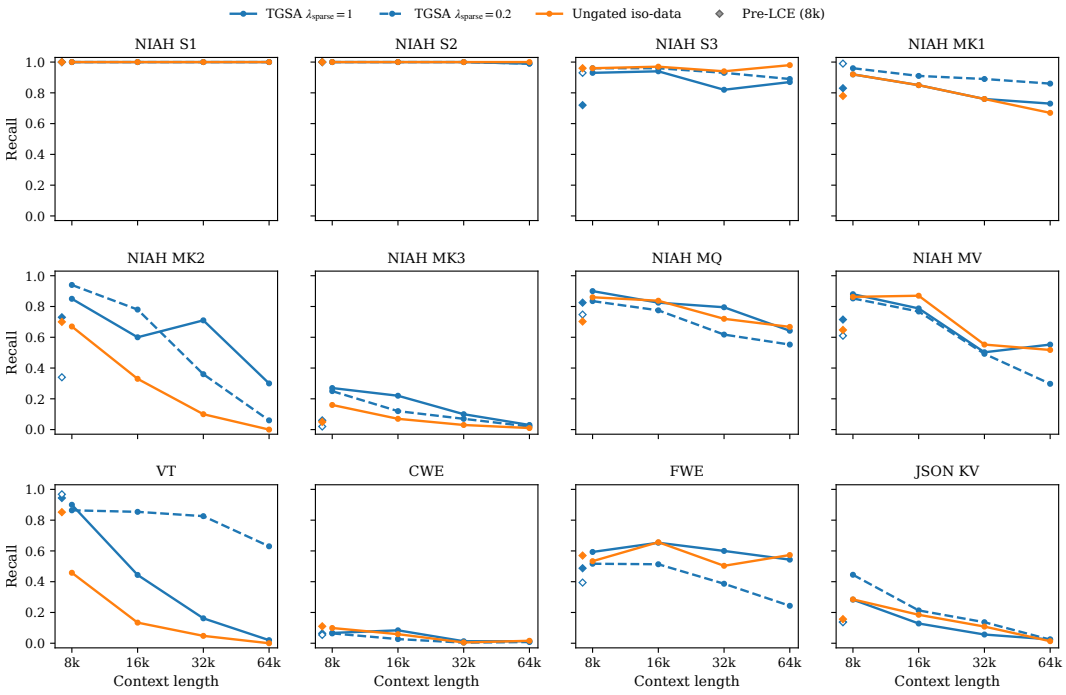


Figure 9: HELMET recall at 1.7B for TGSA ($\lambda_{\text{sparse}} = 0.2$ and $\lambda_{\text{sparse}} = 1$) and the ungated iso-data baseline. Lines show post-LCE checkpoints; diamonds show pre-LCE recall at 8K.

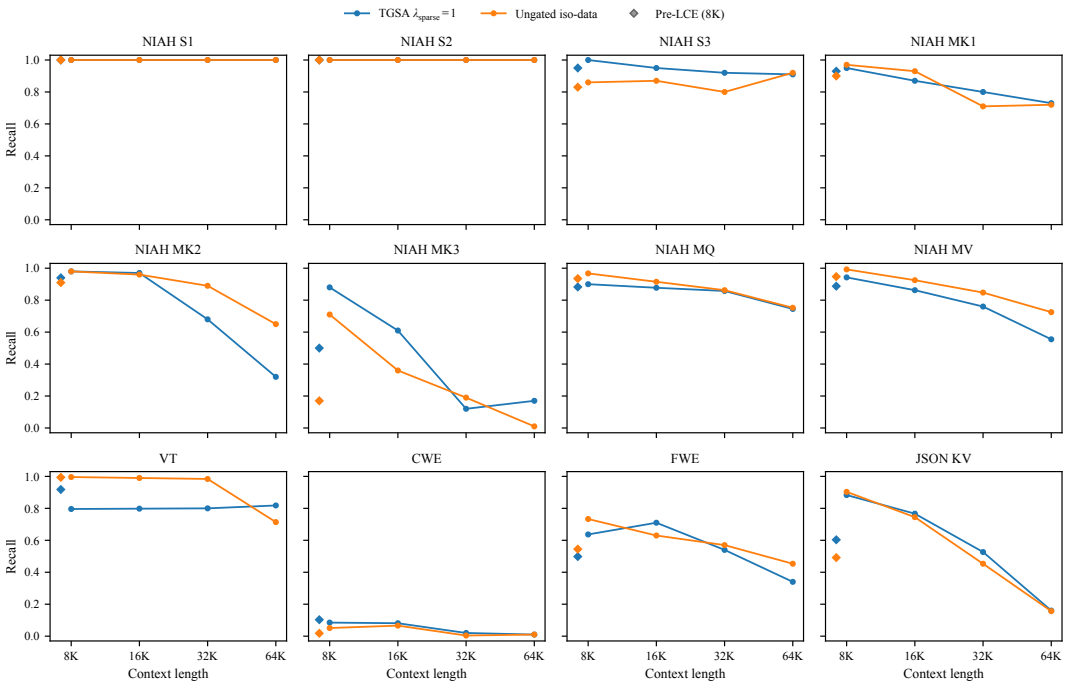


Figure 10: HELMET recall at 4B for TGSA $\lambda_{\text{sparse}} = 1$ and the ungated iso-data baseline. Lines show post-LCE checkpoints; diamonds show pre-LCE recall at 8K.

J GATE MASK VISUALIZATIONS

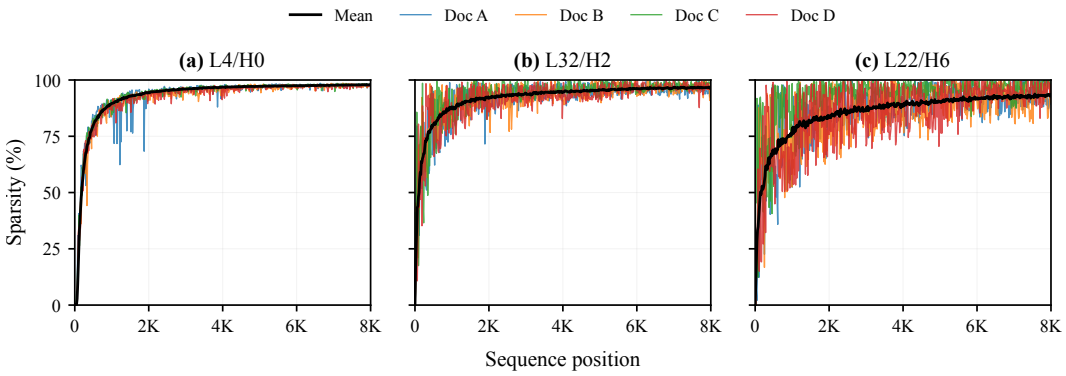


Figure 11: Measured sparsity for three gate heads as a function of sequence position, with **(a)** low, **(b)** median, and **(c)** high inter-document variation. The black curves show the mean sparsity across 128 documents, with colored lines representing four sample documents.

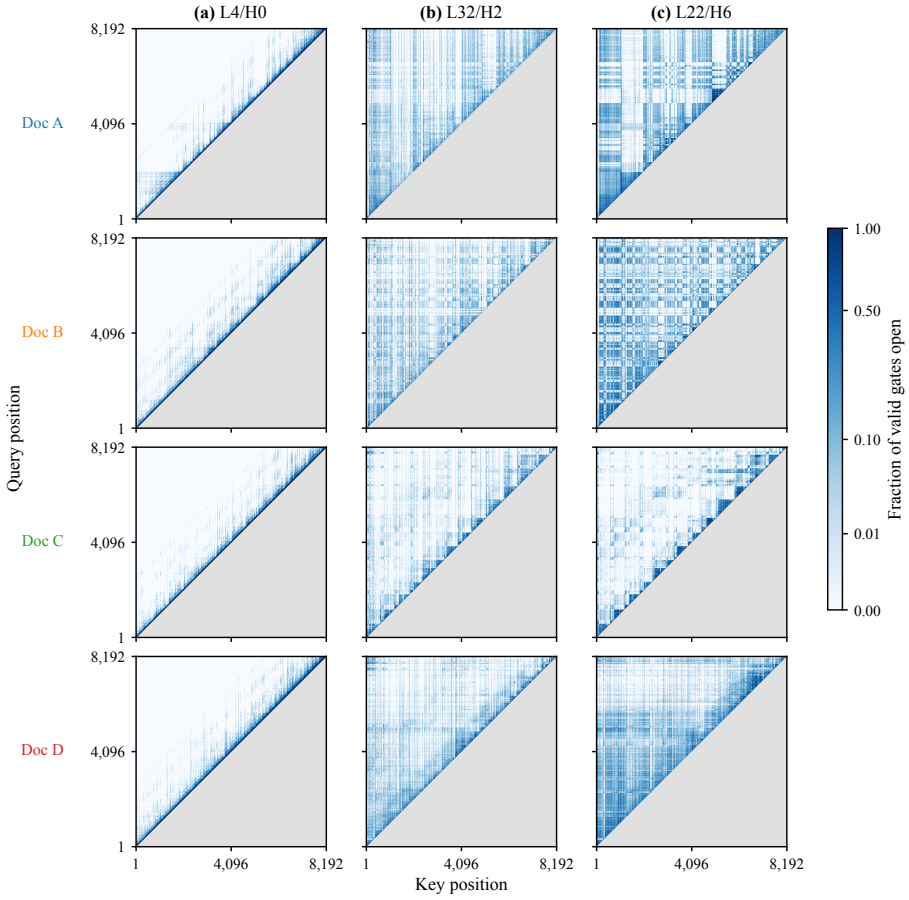


Figure 12: Gating masks for the same heads and documents as shown in Fig. 11. We take 8×8 patches of gates and set the color of each patch proportional to the fraction of gates in the patch that were open; note that the colorscale is not uniform, to distinguish a fully sparse patch from a partially sparse one.