Octopus: Gated Selective Attention for Memory-Bounded Long-Context Inference in Large Language Models

Anonymous ACL submission

Abstract

Transformer inference becomes increasingly memory-bound as the Key–Value (KV) cache grows linearly with sequence length. While subquadratic architectures offer constant-memory inference, they rely on aggressive state compression that degrades performance on complex reasoning tasks. We propose **OCTOPUS**, a framework that confers fixed-memory inference onto pretrained Transformers without the information loss of linearization. OCTOPUS retrofits attention layers with **Gated Selective Attention**, a learnable module that enforces an **adaptive sparsity policy** over the context history. By dynamically scoring and retaining only high-utility KV states, this mechanism transforms the unbounded cache into a compact, evolving memory budget that filters out uninformative noise. Empirically, on the GSM8K benchmark, it outperforms state-of-the-art linearized baselines by over **36 points** under identical memory constraints. Remarkably, OCTOPUS also surpasses its own full-cache teacher, demonstrating that learned sparse retention serves as an effective regularizer for long-horizon reasoning. Code: Link.

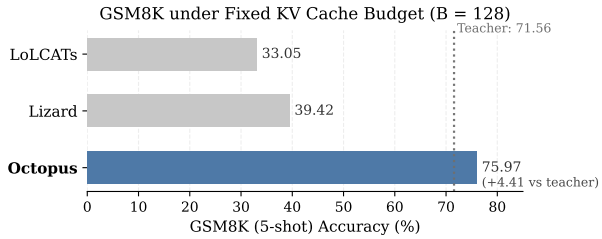

Figure 1: **GSM8K reasoning under a fixed KV cache budget (B=128).** OCTOPUS not only substantially outperforms existing budgeted baselines, but also exceeds the full-cache teacher with a fixed KV cache.

1 Introduction

Transformer architectures are the dominant backbone of modern Large Language Models (LLMs) (Touvron et al., 2023; Yang et al., 2025; Grattafiori et al., 2024; Achiam et al., 2023), but their inference efficiency degrades sharply with context length. Standard self-attention requires storing Key-Value (KV) pairs for every token in the sequence, causing the cache to grow linearly with context length. In long-context scenarios, the KV cache quickly dominates GPU memory and makes inference increasingly memory-bound, limiting practical deployment (Li et al., 2024; Zhang et al., 2023; Liu et al., 2025).

To mitigate the KV bottleneck, recent work has shifted toward subquadratic architectures, including State Space Models (SSMs) (Gu and Dao, 2023; Dao and Gu, 2024) and linear attention variants (Yang et al., 2024), which enable constant-memory inference ($\mathcal{O}(1)$ complexity) by compressing the entire history into a fixed-size recurrent state. However, this aggressive compression introduces a key trade-off: forcing all past information into a single state often discards the high-fidelity details needed for associative recall and complex multi-step reasoning, leaving these models behind Transformers on challenging in-context retrieval and reasoning benchmarks (Waleffe et al.; Zhang et al., 2025). Recent evaluations under matched pretraining budgets consistently show substantial performance gaps, especially on tasks requiring precise long-range dependency tracking (Waleffe et al.).

A complementary direction attempts to inherit Transformer capabilities without expensive pretraining by linearizing pretrained Transformers. Methods such as LoLCATs (Zhang et al., 2025) and Lizard (Van Nguyen et al., 2025) approximate softmax attention using a combination of global memory vectors and local sliding windows. While effective for efficiency, these methods force a pretrained softmax-based model into a fundamentally different architectural paradigm. This architectural mismatch makes exact recovery of the teacher's capabilities difficult, often resulting in persistent qual-

ity gaps on tasks requiring precise long-range dependency tracking (Van Nguyen et al., 2025; Wang et al., 2024).

In this work, we propose **OCTOPUS**, a novel framework that confers **fixed-cache inference** onto pretrained Transformers. Our key observation is simple: *in autoregressive model, each generated token representation already aggregates information from the entire prefix through self-attention, and thus already acts as an aggregated summary of the preceding context*. This suggests that the full KV history is often redundant. If the model can learn which KV states are truly useful for future predictions, then long-horizon inference can be made memory-bounded while retaining Transformer expressiveness. Octopus retrofits each attention layer with **Gated Selective Attention**, by augmenting standard self-attention with a lightweight gate that dynamically assigns a utility score to KV states. These scores are then used to enforce a sparsity constraint that encourages the model to rely on a small subset of high-utility tokens. At inference time, Octopus maintains a fixed KV cache by retaining only the highest-utility states, transforming the KV cache from an unbounded history into a compact, evolving state. Importantly, the learned scoring mechanism is **budget-controllable**: the same trained model can adjust cache size at runtime to satisfy different memory constraints without retraining. Beyond memory savings, Gated Selective Attention addresses a second limitation of indiscriminate caching. Standard attention computes a weighted sum over the entire history, which can allocate probability mass to weakly relevant or noisy tokens degrades long-context reasoning (Ye et al., 2024). By restricting computation to a curated set of high-utility KV states, Octopus can act as an implicit regularizer, filtering uninformative context and improving robustness in low-memory regimes.

Empirically, our results demonstrate a step-change in efficiency-performance trade-offs. On the GSM8K reasoning benchmark, OCTOPUS achieves a massive **+36 point improvement** over state-of-the-art linearized baselines (e.g., Lizard) under identical memory budgets.

Overall, our key contributions are as follows:

- We propose OCTOPUS, a framework that retrofits pretrained Transformers with budget-controllable, fixed-memory inference without pretraining from scratch.
- We demonstrate that Gated Selective Attention (GSA) is superior to both heuristic pruning and architectural linearization. In fixed-budget settings, OCTOPUS matches or exceeds the full-cache teacher on complex reasoning tasks, suggesting that utility-based gating effectively converts the memory-fidelity trade-off into a noise-reduction advantage.

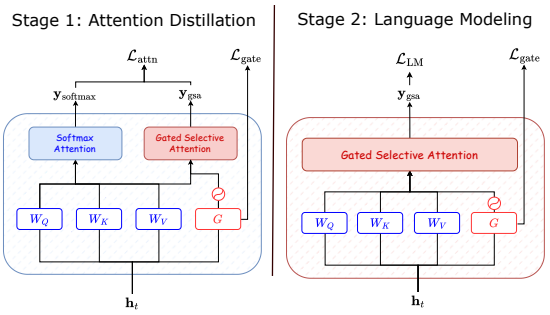

Figure 2: Two-stage training of Gated Selective Attention (GSA). **Stage 1: Attention Distillation (left).** We freeze the pretrained Transformer backbone and train only the gating module G. GSA is optimized to match the teacher's softmax attention outputs via $\mathcal{L}_{\text{attn}}$, while enforcing sparsity and decisiveness through the gate regularizer $\mathcal{L}_{\text{gate}}$. **Stage 2: Language Modeling (right).** Softmax attention is fully replaced by GSA in all layers, and the gates are fine-tuned for next-token prediction using $\mathcal{L}_{\text{LM}}$ under the same sparsity constraint.

2 Preliminaries

2.1 Causal Self-Attention and KV Caching

We consider a standard Transformer decoder layer (Vaswani et al., 2017) operating on a sequence $x_{1:T}$. Let $h_t \in \mathbb{R}^d$ denote the layer input at position t. The model computes queries, keys, and values:

$$q_t = W_Q h_t, \quad k_t = W_K h_t, \quad v_t = W_V h_t,$$

where $W_Q, W_K, W_V \in \mathbb{R}^{d\times d}$ are learnable projection matrices. In causal self-attention, the query q_t attends to all past keys and values (including itself) to compute the output o_t:

$$o_t = \sum_{i=1}^{t} \frac{\exp(q_t^\top k_i/\sqrt{d})}{\sum_{j=1}^{t}\exp(q_t^\top k_j/\sqrt{d})} v_i$$

A token-as-summary view. Because o_t aggregates information over the entire prefix $1{:}t$, the representation at position t encodes the history $x_{1:t}$ through repeated attention and MLP mixing across layers. Thus, later-token representations can be viewed as progressively updated *summaries* of the prefix, suggesting that retaining *every* past KV pair may be redundant for future prediction.

2.2 Memory-Bounded Inference

We formalize *memory-bounded inference* as maintaining a bounded memory state S_t under a fixed budget B, independent of the sequence length:

$$S_t = f(S_{t-1}, k_t, v_t), \qquad \text{Size}(S_t) \le B.$$

Existing approaches typically satisfy this constraint via truncation or compression.

Sliding window attention. A simple policy keeps only the most recent B tokens (First-In First-Out eviction):

$$S_t = S_{t-1} \cup \{(k_t, v_t)\} \setminus \{(k_{t-B}, v_{t-B})\}.$$

This preserves exact KV states (high fidelity) but imposes a hard locality bias: dependencies beyond the window are irrevocably forgotten regardless of semantic importance.

State compression (linear-time models). Subquadratic architectures (e.g., linear attentions and SSMs (Dao and Gu, 2024; Yang et al., 2024)) avoid storing discrete tokens by compressing history into a fixed-size recurrent state:

$$S_t = g_t \odot S_{t-1} + \phi(k_t)\, v_t^\top$$

where S_t has constant size and $\phi(\cdot)$ is a feature map. Here g_t is a *data-dependent* gate (either a scalar $g_t \in [0,1]$ or a vector $g_t \in [0,1]^d$) computed from the current input, which controls how much of the previous state is retained (i.e., the importance of past information) versus forgotten. While this formulation retains unbounded context in principle, the representation is inherently lossy: forcing all historical detail into a single aggregate state can degrade the precise, token-level retrieval often required for complex reasoning.

Our perspective. These strategies expose a fidelity-memory trade-off: sliding windows retain exact KV states but discard long-range dependencies, while linear-time models retain unbounded history only through lossy state compression. **OCTOPUS** takes a different route. Inspired by the data-dependent gating used in linear-time models, we keep standard softmax self-attention but augment it with a lightweight gate that assigns a scalar utility score to each KV state. We use these scores to enforce a learned sparsity pattern, retaining only high-utility KV pairs under a fixed cache budget for future prediction, achieving memory-bounded inference without collapsing history into a single recurrent state.

3 Method: OCTOPUS

3.1 Overview

OCTOPUS retrofits a pretrained Transformer by augmenting each standard causal softmax self-attention layer with a lightweight gating module. The resulting **Gated Selective Attention (GSA)** preserves the base attention computation, but predicts a per-token *utility score* for KV states and uses these scores to induce a learned sparsity pattern over the KV cache. Throughout, we keep all pretrained model parameters frozen and optimize only the gating modules.

3.2 Gated Selective Attention

Consider a Transformer layer with H query heads and H_{kv} key/value heads (grouped-query attention). Let d_h be the head dimension and let $h_t \in \mathbb{R}^d$ denote the layer input at position t. Standard projections produce:

$$q_t^{(a)} = W_Q^{(a)} h_t, \quad k_t^{(b)} = W_K^{(b)} h_t, \quad v_t^{(b)} = W_V^{(b)} h_t,$$

where $a \in \{1, \dots, H\}$ indexes query heads and $b \in \{1, \dots, H_{kv}\}$ indexes KV heads.

Utility gating. We augment each attention layer ℓ with a lightweight gating module $G^{(\ell)}$, implemented as a small MLP that maps the current hidden state to a scalar utility per KV head:

$$s_t = G(h_t) \in \mathbb{R}^{H_{kv}}, \quad g_t = \sigma(s_t) \in (0,1)^{H_{kv}},$$

where $\sigma(\cdot)$ is the sigmoid function.

Gated Attention Logits. For query head a, let $\pi(a) \in \{1, \dots, H_{\text{kv}}\}$ denote its corresponding KV head under grouped-query attention.[1] OCTOPUS *reweights* each KV state by a gating score $g_i \in [0,1]$, which serves as a per-token *utility score*. Concretely, the attention weight from position t to a cached token $i \le t$ is reweighted by the token utility score $g_i^{(\pi(a))}$, so that low-utility tokens ($g_i \approx 0$) contribute negligibly regardless of their content similarity, while high-utility tokens ($g_i \approx 1$) are retained and emphasized for future prediction:

$$\alpha_{t,i}^{(a)} \propto g_i^{(\pi(a))} \exp\left(\langle q_t^{(a)}, k_i^{(\pi(a))} \rangle / \sqrt{d_h} \right),$$

Crucially, GSA is parallelizable and compatible with modern hardware accelerators. While conceptually the gate acts as a multiplicative factor on

[1] Assuming query heads are evenly grouped, $\pi(a) = \left\lfloor \frac{a-1}{H/H_{\text{kv}}} \right\rfloor + 1$.

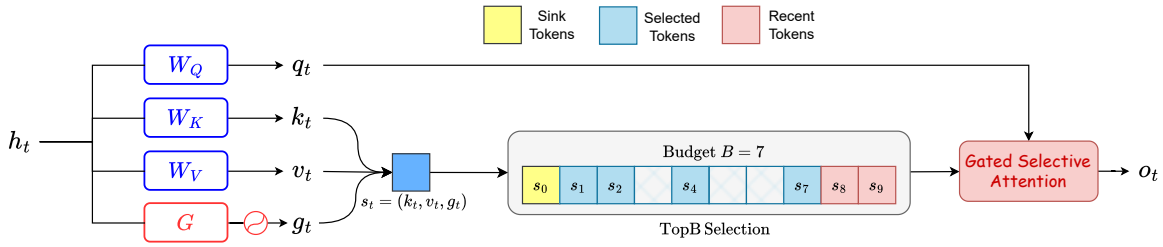

Figure 3: **Inference with a budgeted KV cache.** At decoding step t, we compute (q_t, k_t, v_t) and a gate score g_t from the current hidden state h_t. Each token contributes a KV state $s_t = (k_t, v_t, g_t)$ to the cache. Under a fixed budget B, we reserve slots for *sink tokens* and a *recent window*, and fill the remaining slots by selecting the Top-B states ranked by utility g. Gated Selective Attention then attends over this fixed-size cache to produce the output o_t.

the attention probabilities, implementing this as a separate element-wise multiplication would require materializing the full $N \times N$ attention matrix. To address this, we express the gate reweighting via the *log-space trick*, turning the multiplicative factor β into an additive logit bias $\log \beta$:

$$\alpha_{t,i}^{(a)} \propto \exp\Big(\langle q_t^{(a)},\, k_i^{(\pi(a))}\rangle / \sqrt{d_h} + \log\big(g_i^{(\pi(a))}\big)\Big),$$

Now, OCTOPUS becomes natively compatible with optimized kernels such as **FlashAttention-2** (Dao, 2023). We provide the complete PyTorch implementation utilizing FlexAttention in Appendix A.

Attention output. Given the gated attention weights $\alpha_{t,i}^{(a)}$, the attention output is computed in the standard way as a softmax-weighted sum over values:

$$o_t^{(a)} = \sum_{i \le t} \alpha_{t,i}^{(a)}\, v_i^{(\pi(a))}.$$

The gating mechanism biases the logits so that low-utility tokens receive negligible probability mass, yielding an effectively sparse attention distribution while preserving standard softmax attention.

3.3 Training Objective

We freeze the pretrained Transformer backbone and optimize only the lightweight gating modules in Gated Selective Attention (GSA). Our goal is to learn utility scores that reweight attention toward informative KV states and induce a global ranking of token importance, enabling flexible cache truncation at inference.

Sparsity regularization. To enforce memory-bounded behavior, we encourage the gates to be both *sparse* and *decisive* (utilities are close to 0 or 1 rather than ambiguous). We achieve this with a sparsity-entropy penalty:

$$\mathcal{L}_{\text{gate}} = \mathbb{E}_\ell \left[g^{(\ell)} + \lambda_{\text{ent}}\, \mathcal{H}\Big(g^{(\ell)}\Big) \right],$$

where $\mathcal{H}(\cdot)$ is the (Bernoulli) entropy:

$$\mathcal{H}(g) = -g \log g - (1-g)\log(1-g).$$

The first term penalizes large average gate values, pushing most utilities toward 0 and thus promoting global sparsity. The entropy term discourages uncertain gates, which encourages near-binary decisions. Crucially, this sparsity objective serves to **decouple the training objective from the inference budget**. By forcing the gates to be decisive during training, the model learns a robust, intrinsic ranking of token utility rather than overfitting to a specific window size. This ensures that the cache budget B becomes a flexible inference-time hyperparameter: the same trained model can scale down to strict memory constraints or scale up to larger budgets simply by adjusting the selection threshold, without requiring retraining.

Two-stage training. We adopt a two-stage optimization procedure to learn stable, utility-based gate scores while keeping the pretrained Transformer backbone fixed. We train the gating modules using a two-stage procedure: attention distillation followed by language modeling, as summarized in Fig. 2 and formalized below.

Stage 1: Attention Distillation. We first train the gates to match the pretrained teacher's full softmax attention behavior, while regularizing the gates with $\mathcal{L}_{\text{gate}}$. We compute both teacher attention and gated attention (Eq. (3.2)) and optimize:

$$\mathcal{L}_{\text{stage1}} = \lambda_{\text{attn}}\, \mathcal{L}_{\text{attn}} + \lambda_{\text{gate}}\, \mathcal{L}_{\text{gate}},$$

where $\mathcal{L}_{\text{attn}} = \frac{1}{L} \sum_{\ell=1}^{L} \Big\| A_{\text{gated}}^{(\ell)} - A_{\text{softmax}}^{(\ell)} \Big\|_2^2$.

Stage 2: Language Modeling. We then replace softmax attention with gated selective attention in all layers and fine-tune the gates for next-token

prediction, while retaining the same sparsity regularizer:

$$\mathcal{L}_{\text{stage2}} = \mathcal{L}_{\text{LM}} + \lambda_{\text{gate}}\, \mathcal{L}_{\text{gate}}.$$

3.4 Inference with Budgeted KV Cache

Figure 3 illustrates the inference-time mechanism. At inference time, OCTOPUS performs autoregressive decoding under a fixed KV cache budget B. Importantly, the cache budget B is an inference-time parameter. The same trained model can operate under different memory constraints by adjusting B without retraining, enabling flexible trade-offs between memory usage and performance. For each layer ℓ, we store keys, values, and their corresponding gate scores produced at token creation time.

Gate-based Cache Pruning. After generating token t, let $\mathcal{C}_t^{(\ell)} = \{s_i^{(\ell)} := (k_i^{(\ell)}, v_i^{(\ell)}, g_i^{(\ell)})\}_{i=1}^{t}$ denote the KV cache at layer ℓ, where each state is associated with a learned utility score $g_i^{(\ell)}$. If $|\mathcal{C}_t^{(\ell)}| > B$, we prune the cache by retaining the top-B states ranked by utility:

$$\mathcal{C}_t^{(\ell)} \leftarrow \text{TopB}\Big(\mathcal{C}_t^{(\ell)};\, \{g_i^{(\ell)}\}\Big), \qquad |\mathcal{C}_t^{(\ell)}| \le B.$$

In practice, we reserve a small subset of the budget for (i) initial *sink tokens* and (ii) a sliding window of the most *recent tokens*, and apply the utility-based selection to the remaining slots. This ensures stable attention behavior while enforcing a strict memory bound. By coupling gated attention with gate-based cache pruning, OCTOPUS transforms the unbounded KV cache of standard Transformers into a compact, evolving state. As a result, inference memory remains constant with respect to sequence length while preserving access to high-utility historical context.

4 Experiments

We design our evaluation to answer two primary questions regarding efficiency and scalability:

1. **Efficiency vs. Linearization (Q1):** Can OCTOPUS deliver superior reasoning performance compared to state-of-the-art subquadratic architectures (e.g., Lizard, Mamba-variants) when constrained to an identical memory footprint?

2. **Robustness vs. Eviction (Q2):** Does our learned utility policy scale to long-context mathematical reasoning, where heuristic eviction policies (e.g., H2O, SnapKV) typically fracture semantic dependencies?

4.1 Experimental Setup

Baselines. To address Q1 and Q2, we categorize our baselines into two distinct groups:

- **For Q1 (Architecture Comparison):** We compare against **LoLCATs** (Zhang et al., 2025), **Liger** (Lan et al., 2025), and **Lizard** (Van Nguyen et al., 2025). These methods represent the state-of-the-art in linearizing pretrained Transformers into recurrent forms with constant inference memory.

- **For Q2 (Policy Comparison):** We compare against **H2O** (Zhang et al., 2023), **SnapKV** (Li et al., 2024), and **R-KV** (Cai et al., 2025). These are representative heuristic eviction strategies (based on accumulated attention or clustering) applied to the standard Transformer.

- **Oracle:** The Full-Cache Teacher serves as the performance upper bound. Note that this baseline is *unbounded*: for a sequence of length L, it stores all L tokens. In our long-context experiments, this results in cache sizes up to 32K tokens, whereas OCTOPUS is capped at $\{1\text{K}, 2\text{K}, 4\text{K}\}$.

We adopt a two-stage evaluation regime:

- **Standard Regime (Fixed Budget):** To compare with subquadratic models, we retrofit **LLaMA-3-8B** and train on Alpaca-Cleaned (Taori et al., 2023) [2] for 40M tokens. We evaluate on standard commonsense and reasoning benchmarks.

- **Long-Context Regime (Scalability):** To evaluate long-range robustness, we retrofit **Qwen3-1.7B** and **Qwen3-4B**. We finetune on the `open-r1/Mixture-of-Thoughts` (Bercovich et al., 2025) [3] dataset with sequences up to 32k tokens.

For inferene, We reserve $S{=}4$ slots for initial sink tokens and a sliding window of $W{=}32$ recent tokens to preserve local syntax. The remaining budget $B - (S + W)$ is allocated to the top-ranked tokens by learned utility.

[2] https://huggingface.co/datasets/yahma/alpaca-cleaned
[3] https://huggingface.co/datasets/open-r1/Mixture-of-Thoughts

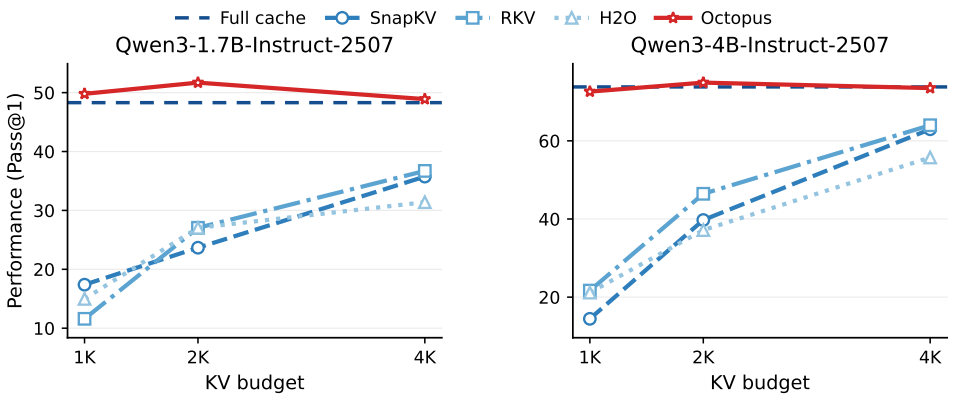

Figure 4: **AIME 2024 long-context math reasoning under budgeted KV caches.** We evaluate Qwen3-1.7B and Qwen3-4B with KV budgets $\{1K, 2K, 4K\}$ (Pass@1). OCTOPUS consistently matches the full-cache teacher and substantially outperforms eviction-based baselines under tight budgets, demonstrating robust budget scalability on long-horizon reasoning.

4.2 Q1: Efficiency on Standard Benchmarks

We first evaluate performance under a strict, fixed memory budget (B=128) on commonsense tasks (HellaSwag (Zellers et al., 2019)) and reasoning benchmarks (GSM8K (Cobbe et al., 2021), MMLU (Hendrycks et al., 2020)). This low-budget regime rigorously tests the model's ability to compress context.

Comparison with Subquadratic Baselines. As shown in Table 1, OCTOPUS achieves a decisive advantage where prior linearization methods struggle. On GSM8K, OCTOPUS outperforms the strongest subquadratic baseline (Lizard) by over 36 percentage points (75.97% vs. 39.42%). This highlights a critical architectural distinction: while linear models successfully approximate global statistics for commonsense tasks, their compressed state fails to retain the precise, token-level definitions required for multi-step math. OCTOPUS's selective cache addresses this by preserving the exact high-utility tokens needed for deduction.

Surpassing the Teacher via Noise Filtering. Counter-intuitively, OCTOPUS outperforms its own full-cache teacher on GSM8K (**75.97%** vs. **71.56%**). Standard attention attends to all previous tokens, allowing irrelevant distractor tokens to dilute the probability mass. By enforcing a learned sparsity policy, OCTOPUS effectively filters out this noise. This confirms our hypothesis that for reasoning tasks, a curated, high-utility memory is often superior to a complete but noisy history.

Model	Training Tokens	GSM8K	MMLU	Hella.
Teacher (Unbounded Cache)				
LLaMA-3-8B	–	71.56	66.60	79.10
Constant Memory / Fixed Budget (B=128)				
LoLCATs	40M	33.05	52.80	79.70
Liger	20M	–	43.40	76.30
Lizard	40M	39.42	61.20	79.30
Octopus (Ours)	**40M**	**75.97**	**65.74**	**81.70**

Table 1: **Strict Budget Comparison (B=128).** OCTOPUS yields a step-change improvement over linearized baselines (+36.5 points over Lizard on GSM8K) using a comparable computational budget. Notably, while baselines like Lizard recover teacher performance on HellaSwag, OCTOPUS surpasses the teacher across all metrics.

4.3 Q2: Long-Context Math Reasoning

We next evaluate whether the learned utility policy scales to *long-context* regimes where the signal-to-noise ratio is significantly lower. We report Pass@1 accuracy on **AIME 2024** (Art of Problem Solving), comparing OCTOPUS against heuristic eviction policies (SnapKV, H2O, R-KV) across varying KV budgets ($\{1K, 2K, 4K\}$).

Figure 4 illustrates that eviction-based methods degrade sharply under aggressive budgets. This failure stems from their reliance on rigid heuristics rather than true utility. **H2O** (Zhang et al., 2023) prioritizes tokens with high *accumulated* attention scores, effectively assuming that past popularity predicts future relevance. This often discards "sleeping" dependencies—tokens that are historically quiet but become critical for a specific future deduction. Similarly, **SnapKV** (Li et al., 2024) relies on attention patterns observed within a limited

Configuration	Acc (%)	Δ	Key Observation
Full Method (Ours)	**74.8**	–	*Stable convergence, best performance*
w/o Stage 1 (Joint Train)	64.0	-10.8	High instability; optimization difficulty
w/o Sink Tokens	73.1	-1.7	Minor degradation in deep layers
w/o Recent Window	68.0	-6.8	Loss of local syntactic coherence

Table 2: **Component-wise Ablation Study.** We evaluate the impact of the training strategy and fixed retention anchors. "No Stage 1" refers to training the gates jointly with the LM from scratch.

window to cluster and compress the cache. This approach struggles in reasoning tasks where dependency structures are highly dynamic and cannot be inferred solely from a static observation window. In contrast, OCTOPUS remains robust across all budgets. Consistent with our findings in Section 4.2, we observe that OCTOPUS can exceed the teacher's performance at moderate budgets (e.g., 2K on Qwen3-4B). This reinforces the "regularization via sparsity" effect: moderate budgets force the model to prioritize high-utility states, effectively suppressing the noise inherent in long contexts that would otherwise distract the full-attention mechanism.

5 Ablation and Analysis

To validate the effectiveness of our proposed framework, we conduct a comprehensive ablation study evaluating the training strategy, the importance of retention anchors, and the granularity of the gating mechanism. We also analyze the controllability of the model under varying KV cache budgets. Unless otherwise stated, all experiments are conducted on the validation set using a standard budget of $K = 128$.

We first assess the contribution of our two-stage training pipeline and the fixed retention anchors (Sink Tokens and Recent Window). The results are summarized in Table 2.

Impact of Two-Stage Training. A key finding is the necessity of our decoupled training strategy. Removing Stage 1 and attempting to train the gating mechanism jointly with the LM (or directly on a frozen LM without warm-up) results in a sharp performance drop to 64.0%. We observed significant instability during these runs. Without the initial warm-up, the gate network struggles to identify salient features, often collapsing into a trivial solution where the cache is populated with noise. This confirms that Stage 1 is crucial for initializing a robust policy before fine-tuning.

Importance of Anchors. Removing the sink tokens results in a moderate drop to 73.1%, suggesting that while our dynamic gate can learn to retain high-norm tokens to some extent, explicit sinks ensure stability in attention computations. However, removing the recent window tokens is far more detrimental (68.0%): without a guaranteed local window, the model can no longer reliably preserve short-range context, leading to disrupted local coherence and degraded syntactic consistency.

Sensitivity to Anchor Hyperparameters Our method incorporates two fixed attention anchors: a set of initial "sink tokens" (S) to stabilize the softmax distribution, and a local sliding window (W) to capture immediate syntactic dependencies. To verify that our performance stems from the learned gating policy rather than these heuristics, we evaluate the model's sensitivity to variations in S and W under a fixed total cache budget of $K = 128$.

Sinks (S)	Window (W)	Avg. Acc (%)
0	32	73.1
1	32	74.5
4	**32**	**74.8**
8	32	74.8
4	0	67.1
4	16	73.5
4	64	74.1

Table 3: **Sensitivity to Anchor Sizes.** We vary the number of sink tokens (S) and the recent window size (W) with a fixed total budget of $K = 128$.

Minimal Sink Tokens Required. As shown in Table 3, removing sink tokens ($S = 0$) leads to a performance drop to 73.1%, consistent with observations in StreamingLLM regarding attention sink phenomena. However, increasing S from 1 to 4 yields marginal gains, and further increasing to 8 yields no improvement. This confirms that sink to-

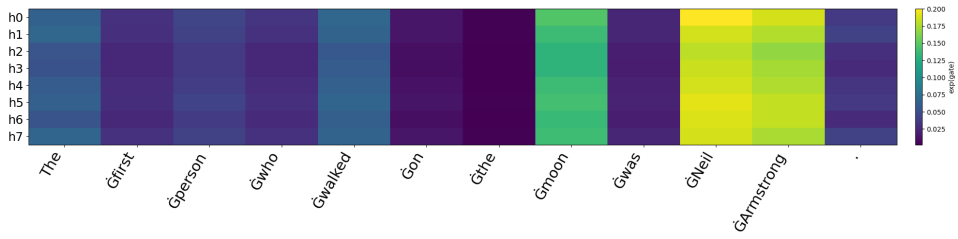

Figure 5: **Visualization of Learned Gating Behavior.**

kens serve purely as a numerical stabilizer, and the model does not rely on them for semantic content.

The "Crowding Out" Effect of Windows. The local window (W) plays a more dynamic role. Removing it ($W = 0$) causes a sharp decline to 68.0%, proving that the learned gate cannot easily replicate the dense, high-frequency attention required for immediate bigram statistics. Crucially, however, simply making the window larger is not better. Increasing W to 64 *reduces* accuracy to 74.1%. Because the total budget is fixed at $K = 128$, a larger forced window leaves fewer slots for the learned gating mechanism to retain important long-range information. This trade-off validates our core contribution: the learned gate is more value-dense than a naive sliding window, and performance is maximized when the gate is given the majority of the budget.

Visualization of Gating Behavior. To understand how our model achieves high efficiency, we visualize the learned gating scores in Figure 5. Using the sample sentence *"The first person who walked on the moon was Neil Armstrong,"* we observe that the gating mechanism successfully learns a semantic retention policy without explicit supervision. The gate assigns near-maximum scores (yellow) to information-rich entities like *"Neil"*, *"Armstrong"*, and *"Moon"*, ensuring they are preserved in the limited cache. Conversely, functional stop words such as *"The"*, *"who"*, and *"was"* are aggressively suppressed (dark blue). This confirms that the model does not rely on simple heuristics (like position) but actively evaluates the semantic utility of each token before admission.

6 Related Work

Subquadratic Architectures and Linearization. A primary avenue for mitigating the computational bottlenecks of Transformers is the development of subquadratic architectures. State Space Models (SSMs) like Mamba (Gu and Dao, 2023; Dao and Gu, 2024) and linear attention variants reduce the quadratic complexity of self-attention to linear time by compressing context into a fixed-size recurrent state. Recent linearization frameworks, such as Lizard (Van Nguyen et al., 2025), attempt to bridge the gap by distilling pretrained Transformers into these subquadratic forms, introducing learnable recurrent gates to manage memory dynamics. However, while these architectures offer constant-memory inference, they often require expensive retraining or suffer from limited associative recall compared to full-attention mechanisms, particularly on tasks requiring precise retrieval from long contexts.

Heuristic KV Cache Eviction. To address the memory growth in standard Transformers without fundamental architectural changes, various KV cache compression strategies have been proposed. Heuristic methods such as H2O (Zhang et al., 2023) and SnapKV (Li et al., 2024) impose a fixed memory budget by evicting tokens based on attention scores, operating on the assumption that past attention predicts future utility.

7 Conclusion

We introduce a memory-bounded inference framework that replaces indiscriminate KV caching with a learnable, head-specific admission policy. Unlike static heuristics, our lightweight gating mechanism dynamically evaluates token utility, allowing models to retain semantic "heavy hitters" while filtering noise. Experiments on GSM8K and MMLU demonstrate that our approach matches or surpasses full-cache performance with a budget of just 128 tokens. Our analysis confirms that this efficiency arises from the learned gate's ability to prioritize long-term information while minimal fixed anchors preserve local stability. This work provides a principled, scalable solution for deploying long-context LLMs on memory-constrained hardware.

8 Limitations

While OCTOPUS effectively reduces memory fragmentation and enhances reasoning through noise filtering, several limitations remain. First, unlike training-free heuristic methods (e.g., H2O, SnapKV), our approach requires a distinct fine-tuning phase to adapt the gating mechanism, incurring additional computational overhead before deployment. Second, as a distillation-based framework, our performance is inherently upper-bounded by the quality of the teacher model; OCTOPUS cannot synthesize reasoning capabilities absent in the teacher, but rather optimizes the retention of those existing capabilities under constraints. Finally, the eviction policy is destructive; once a token is dropped, it cannot be recovered. While our two-stage training minimizes gating errors, extreme compression rates may still risk discarding information that becomes relevant only much later in the sequence.

References

Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, and 1 others. 2023. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*.

Art of Problem Solving. AIME Problems and Solutions.

Yushi Bai, Xin Lv, Jiajie Zhang, Hongchang Lyu, Jiankai Tang, Zhidian Huang, Zhengxiao Du, Xiao Liu, Aohan Zeng, Lei Hou, and 1 others. 2024. Longbench: A bilingual, multitask benchmark for long context understanding. In *Proceedings of the 62nd annual meeting of the association for computational linguistics (volume 1: Long papers)*, pages 3119–3137.

Akhiad Bercovich, Itay Levy, Izik Golan, Mohammad Dabbah, Ran El-Yaniv, Omri Puny, Ido Galil, Zach Moshe, Tomer Ronen, Najeeb Nabwani, Ido Shahaf, Oren Tropp, Ehud Karpas, Ran Zilberstein, Jiaqi Zeng, Soumye Singhal, Alexander Bukharin, Yian Zhang, Tugrul Konuk, and 114 others. 2025. Llama-nemotron: Efficient reasoning models. *Preprint*, arXiv:2505.00949.

Zefan Cai, Wen Xiao, Hanshi Sun, Cheng Luo, Yikai Zhang, Ke Wan, Yucheng Li, Yeyang Zhou, Li-Wen Chang, Jiuxiang Gu, and 1 others. 2025. R-kv: Redundancy-aware kv cache compression for reasoning models. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, and 1 others. 2021. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*.

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

Tri Dao and Albert Gu. 2024. Transformers are ssms: Generalized models and efficient algorithms through structured state space duality. *arXiv preprint arXiv:2405.21060*.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, and 1 others. 2024. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*.

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

Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. 2020. Measuring massive multitask language understanding. *arXiv preprint arXiv:2009.03300*.

Disen Lan, Weigao Sun, Jiaxi Hu, Jusen Du, and Yu Cheng. 2025. Liger: Linearizing large language models to gated recurrent structures. In *Forty-second International Conference on Machine Learning*.

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

Jiaheng Liu, Dawei Zhu, Zhiqi Bai, Yancheng He, Huanxuan Liao, Haoran Que, Zekun Wang, Chenchen Zhang, Ge Zhang, Jiebin Zhang, and 1 others. 2025. A comprehensive survey on long context language modeling. *arXiv preprint arXiv:2503.17407*.

Rohan Taori, Ishaan Gulrajani, Tianyi Zhang, Yann Dubois, Xuechen Li, Carlos Guestrin, Percy Liang, and Tatsunori B. Hashimoto. 2023. Stanford alpaca: An instruction-following llama model. `https://github.com/tatsu-lab/stanford_alpaca`.

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, and 1 others. 2023. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*.

Chien Van Nguyen, Ruiyi Zhang, Hanieh Deilamsalehy, Puneet Mathur, Viet Dac Lai, Haoliang Wang, Jayakumar Subramanian, Ryan A Rossi, Trung Bui, Nikos Vlassis, and 1 others. 2025. Lizard: An efficient linearization framework for large language models. *arXiv preprint arXiv:2507.09025*.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. *Advances in neural information processing systems*, 30.

Roger Waleffe, Wonmin Byeon, Duncan Riach, Brandon Norick, Vijay Korthikanti, Tri Dao, Albert Gu, Ali Hatamizadeh, Sudhakar Singh, Deepak Narayanan, and 1 others. An empirical study of mamba-based language models, 2024. *URL https://arxiv. org/abs/2406.07887*.

Junxiong Wang, Daniele Paliotta, Avner May, Alexander Rush, and Tri Dao. 2024. The mamba in the llama: Distilling and accelerating hybrid models. *Advances in Neural Information Processing Systems*, 37:62432–62457.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, and 1 others. 2025. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*.

Songlin Yang, Bailin Wang, Yikang Shen, Rameswar Panda, and Yoon Kim. 2024. Gated linear attention transformers with hardware-efficient training. In *Forty-first International Conference on Machine Learning*.

Tianzhu Ye, Li Dong, Yuqing Xia, Yutao Sun, Yi Zhu, Gao Huang, and Furu Wei. 2024. Differential transformer. *arXiv preprint arXiv:2410.05258*.

Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. 2019. HellaSwag: Can a machine really finish your sentence? In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 4791–4800. Association for Computational Linguistics.

Michael Zhang, Simran Arora, Rahul Chalamala, Benjamin Frederick Spector, Alan Wu, Krithik Ramesh, Aaryan Singhal, and Christopher Re. 2025. LoLCATs: On low-rank linearizing of large language models. In *The Thirteenth International Conference on Learning Representations*.

Zhenyu Zhang, Ying Sheng, Tianyi Zhou, Tianlong Chen, Lianmin Zheng, Ruisi Cai, Zhao Song, Yuandong Tian, Christopher Ré, Clark Barrett, and 1 others. 2023. H2o: Heavy-hitter oracle for efficient generative inference of large language models. *Advances in Neural Information Processing Systems*, 36:34661–34710.

A Implementation Details

We provide the PyTorch implementation of the OCTOPUS attention mechanism below. By utilizing the `flex_attention` API, we can inject the learned gating scores as a fused modification to the attention scores before the softmax operation. This avoids the need for custom CUDA kernels while retaining the memory and speed benefits of FlashAttention.

A.1 Training Protocol

Our training process consists of two distinct epochs (phases):

Phase 1: Attention Distillation (Warm-up). In the first epoch, we treat the frozen full-attention model as a teacher. The goal is to initialize the gates such that the sparse selection mimics the dense attention distribution. We minimize the Mean Squared Error (MSE) between the OCTOPUS gated attention scores and the standard Softmax attention scores. During this phase, the causal language modeling head is ignored.

Phase 2: Gated Generative Fine-tuning. In the second epoch, we switch to the standard Next Token Prediction objective. The model uses its learned gates to perform inference, and gradients are backpropagated from the cross-entropy loss through the gates. This adapts the gating policy to prioritize tokens that are semantically necessary for generation, rather than just mimicking the teacher's attention pattern:

$$\mathcal{L}_{\text{phase2}} = \mathcal{L}_{\text{CE}} + \lambda_{\text{sparsity}} \cdot \mathcal{L}_{\text{gate}} \tag{1}$$

A.2 Experimental Configurations

We evaluate our method on two distinct scales. The specific hyperparameters for each setup are detailed below:

Standard Context (Alpaca). For standard instruction following benchmarks, we train on the `unsloth/alpaca-cleaned` dataset.

- **Base Model:** Meta-Llama-3-8B-Instruct
- **Sequence Length:** 2048 tokens
- **Hardware:** 8 $\times$ NVIDIA H100 (80GB) GPUs
- **Micro Batch Size:** 2 (Global Batch Size 16)
- **Optimization:** AdamW ($\beta_1 = 0.9, \beta_2 = 0.95$), LR=$1e^{-4}$

Long Context (R1 Mixture of Thoughts). To evaluate long-range dependency retention, we train on the `open-r1/Mixture-of-Thoughts` dataset.

- **Base Model:** Qwen3-1.7B/Qwen3-4B

- **Sequence Length:** 32,768 tokens (32K)
- **Hardware:** 8 × NVIDIA H100 (80GB) GPUs with activation checkpointing enabled
- **Gradient Clipping:** 1.0

B Additional Evaluation on LongBench

Dataset	Full KV	OCTOPUS (25%)
Single-Doc QA		
NarrativeQA	32.37	30.50
Qasper	24.12	22.80
MultiFieldQA-en	27.55	28.15
MultiFieldQA-zh	21.0	22.40
Multi-Doc QA		
HotpotQA	16.46	19.90
2WikiMQA	15.88	16.85
Musique	11.64	13.15
DuReader (zh)	30.44	31.60
Summarization		
GovReport	35.90	33.40
QMSum	23.86	24.15
MultiNews	24.89	26.75
VCSUM (zh)	15.40	16.10
Few-Shot Learning		
TREC	70.60	73.50
TriviaQA	92.86	90.25
SAMSum	42.54	43.85
LSHT (zh)	44.45	46.50
Synthetic / Code		
PassageRetrieval-en	95.82	99.10
PassageRetrieval-zh	78.44	76.20
Passage Count	6.26	2.50
LCC	51.22	53.80
RepoBench-P	44.46	54.10

Table 4: **LongBench Evaluation (Octopus vs. Full Cache).** We compare OCTOPUS (operating at 25% KV cache budget) against the Full KV baseline. Despite using significantly less memory, OCTOPUS matches or exceeds the teacher's performance on the majority of tasks.

To demonstrate the robustness of OCTOPUS across diverse long-context tasks, we conduct an evaluation on the LongBench benchmark (Bai et al., 2024). In Table 4, we compare our method using a fixed 25% KV cache budget against the standard Full KV baseline (100% memory usage). OCTOPUS achieves a comparable performance with the Full KV baseline. This result reinforces our hypothesis that the learned gating mechanism acts as an effective noise filter, discarding irrelevant context that can distract the attention mechanism in full-cache settings, particularly in multi-document QA and retrieval tasks.

Table 5: **Effect of Gating Granularity.** Comparison of sharing gating logic across the model vs. specific heads. The drastic drop in Scalar and Per-Layer performance highlights that attention patterns are highly head-specific.

Granularity	Parameters	Accuracy
Scalar (Per-Token)	$1\times$	11.0
Per-Layer	$L\times$	23.0
Per-Head (Ours)	$L \times H\times$	**74.8**

C Further Analysis

While Figure 5 demonstrates the general selection capability, we further analyze the distinct specialization of different attention heads. We hypothesize that different attention heads require distinct caching strategies. To test this, we vary the granularity of the gating scalar: (1) **Scalar (Per-Token):** A single score shared across all layers and heads. (2) **Per-Layer:** A score shared across all heads within a layer. (3) **Per-Head (Ours):** Unique scores for every head. As shown in Table 5, the results are stark. The scalar approach fails completely, performing near random chance. The Per-Layer approach performs poorly at 23%. This confirms that token importance is not a global property, a token critical for one head may be irrelevant to another. Coarse-grained gating (Scalar/Per-Layer) forces these diverse mechanisms to share a single policy, leading to the observed collapse in performance.

Listing 1: PyTorch implementation of Octopus Gated Attention using FlexAttention.

```python
from typing import Optional
import torch
import torch.nn as nn
# We disable compile debug for cleaner multi-GPU inference in this example
torch.nn.attention.flex_attention._FLEX_ATTENTION_DISABLE_COMPILE_DEBUG = True
from torch.nn.attention.flex_attention import flex_attention, create_block_mask

def flex_octopus_attention(
    query_states: torch.Tensor,
    key_states: torch.Tensor,
    value_states: torch.Tensor,
    log_gated_states: torch.Tensor,
    scaling: float,
    q_start_pos: int = 0,
) -> torch.Tensor:
    """
    Hardware-efficient Octopus Attention via FlexAttention.
    Args:
        query_states: [batch, num_heads, q_len, head_dim]
        key_states:   [batch, num_kv_heads, kv_len, head_dim]
        value_states: [batch, num_kv_heads, kv_len, head_dim]
        log_gated_states: Log-sigmoid gate scores [batch, num_kv_heads, kv_len]
        scaling: Attention scaling factor
    """
    q_len = query_states.shape[2]
    kv_len = key_states.shape[2]

    # This injects log(beta) into the attention logits efficiently.
    # Mathematically: exp(score + log_beta) = exp(score) * beta
    def octopus_score_mod(score, b, h, q_idx, kv_idx):
        num_heads = query_states.shape[1]
        num_kv_heads = key_states.shape[1]
        group_size = num_heads // num_kv_heads

        # Map query head 'h' to the correct KV head (handling GQA)
        kv_head = h // group_size

        # Retrieve the gate value for the target token (kv_idx)
        log_gate = log_gated_states[b, kv_head, kv_idx]

        # Additive bias in log-space equivalent to multiplicative gating
        return score + log_gate

    # -- Causal Masking --
    # Ensures queries only attend to current and past keys.
    # q_start_pos handles shifting during autoregressive decoding steps.
    block_mask = create_block_mask(
        lambda b, h, q, k: (q_start_pos + q) >= k,
        B=None, H=None, Q_LEN=q_len, KV_LEN=kv_len,
        device=query_states.device,
    )

    # -- Execution --
    # Compute attention with fused gating and causal masking in one kernel.
    attention_output = flex_attention(
        query_states, key_states, value_states,
        score_mod=octopus_score_mod,
        block_mask=block_mask,
        scale=scaling,
        enable_gqa=True,
    )

    return attention_output.transpose(1, 2).contiguous()

# Optional: Compilation for further speedup
compiled_flex_octopus_attention = torch.compile(flex_octopus_attention, dynamic=
    False)
```