Attention-Gate: Adaptive In-Context KV-Cache Eviction in LLMs

Anonymous ACL submission

Abstract

The KV-Cache technique has become the standard for the inference of large language models (LLMs). This paper enables a novel dynamic KV-Cache eviction policy by injecting lightweight *Attention-Gates* (AGs) into the model to maximize the utilization efficiency of KV-Cache. AG accepts the *global* context as input and yields eviction flags for each token. The self-attention modules in the model proceed according to the flags and cache only a subset of the KV states for next token prediction. The Attention-Gates can yield various flags for different heads and layers and be easily tuned on top of a pre-trained LLM via continual pre-training or supervised fine-tuning. The computational and memory overhead introduced by Attention-Gates can be minimal. We conduct empirical evaluations across multiple scenarios, showing that our method significantly reduces redundant KV-Cache memory usage while maintaining competitive performance.

1 Introduction

Large language models (LLMs) (Dubey et al., 2024; Team et al., 2024; Chiang et al., 2023) have achieved remarkable success across a wide range of tasks. A key technique in LLM inference is KV-Cache, which stores transient attention keys and values to avoid recomputation. However, as the size of LLMs continues to increase and the demand for handling long-context queries grows, the KV-Cache has emerged as a significant bottleneck. Storing attention states for numerous tokens can lead to considerable memory overhead and increased data movement across the memory hierarchy.

Studies have shown that sparsity is a natural phenomenon in attention mechanisms, with many tokens being redundant for inference (Zhang et al., 2024). This suggests that retaining all tokens in the KV-Cache is unnecessary. Existing works have explored this insight to compress KV-Cache using static strategies or hinging on accumulative attention scores. For example, streamingLLM (Xiao et al., 2024) retains a fixed window of beginning and recent tokens in the KV-Cache, but it struggles to flexibly adapt to specific contexts (e.g., in sentiment analysis, retaining the token "cute" in "a cute cat" is crucial, while in object recognition, the token "cat" would be more important). H2O (Zhang et al., 2024), on the other hand, employs a token-adaptive approach, using local accumulative attention scores to determine which tokens to evict. However, the local perspective can introduce attention bias (Oren et al., 2024), with a tendency to over-prioritize either the initial or recent tokens.

To overcome these challenges, we introduce a learnable neural network module, Attention-Gate (AG), for adaptive *in-context* eviction. For each self-attention within the model, we insert an AG preceding it, which maps the sequence of token features into token-wise eviction flags. The flags indicate whether the token should be excluded from the subsequent self-attention, and evicted tokens do not require their KV states to be cached. AGs can seamlessly embrace pre-trained LLMs, tuned by minimizing the language modeling loss. Ideally, AGs automatically learn to discern the most relevant tokens for the current context without manual intervention. In practice, we implement the AG as a self-attention layer with much fewer heads than the original self-attention in the model (e.g., 4 v.s. 32) to minimize the extra overhead.

As illustrated in Figure 1, AG can generate different eviction strategies across different layers and attention-heads for different tokens, demonstrating its adaptability to the diverse requirements of each component in the model. Importantly, AGs enjoy high training efficiency, e.g., only four NVIDIA 4090 GPUs and a dataset of 5,000 samples are required when applying AGs to LLaMA2-7B (Touvron et al., 2023) following a continual pre-training (CPT) recipe. This alleviates concerns about the

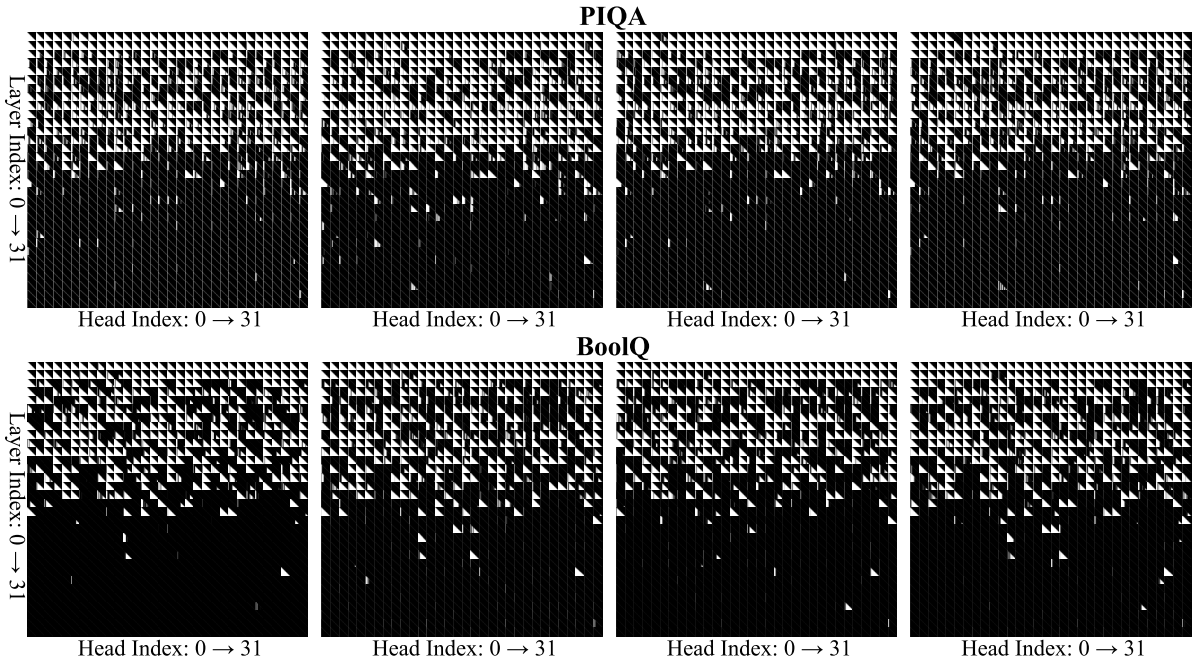

Figure 1: KV-Cache eviction patterns across different layers and attention-heads, visualized for 4 samples from the PIQA dataset (**top row**) and 4 samples from the BoolQ dataset (**bottom row**), using AG fine-tuned Llama2-7B models. Black areas represent tokens that are neither computed nor stored in the KV-Cache. *The variability of eviction patterns across tasks, prompts, layers, and attention-heads demonstrates the dynamic nature of our method.* A common trend observed is that deeper layers tend to mask more KV-Cache states, with some in deeper layers being entirely masked.

computational overhead of trainable eviction strategies (Zhang et al., 2024; Chen et al., 2024).

Empirically, extensive experiments show that our method can outperform traditional training-free eviction strategies, such as streamingLLM and H2O, in accuracy and token eviction rates under both CPT and supervised fine-tuning (SFT) training recipes. In particular, when trained by SFT, our method not only evicts a significant number of redundant tokens but also maintains or surpasses the performance of LoRA-finetuned LLMs. For example, on the RTE dataset (Bar-Haim et al., 2006), our approach improves accuracy by 13.9% while evicting 62.8% of tokens. This demonstrates that selective token eviction can enhance performance.

2 Related Work

As large language models (LLMs) scale in size and input sequence length, optimizing their efficiency has become increasingly important, particularly in addressing space and time complexity. A significant bottleneck lies in the attention mechanism, which demands considerable computational and memory resources, especially for long sequences.

Traditional KV-Cache Eviction Strategies To address both memory and computational challenges, KV-Cache eviction has emerged as an effective strategy. Existing approaches predominantly rely on parameter-free heuristics.

Static strategies, such as those used in Sparse Transformers (Child et al., 2019), employ fixed pruning patterns, such as Strided and Fixed Attention. While effective in some cases, these approaches are not adaptive to specific contexts, often sacrificing accuracy. streamingLLM (Xiao et al., 2024) tackles the *Attention Sink* phenomenon, where attention scores concentrate on initial tokens, by retaining these tokens along with a fixed window of recent tokens. While this improves performance, static approaches generally lack the flexibility needed to adapt to different tokens, attention-heads, or layers.

Strategies using accumulative attention scores offer more flexibility by dynamically identifying important tokens. For instance, SpAtten (Wang et al., 2021) employs Accumulative Attention Scores (A2S), which sum the softmax outputs for each token to measure its importance. This approach allows selective token pruning in subsequent layers, effectively reducing computational complex-

ity without the need for retraining. H2O (Zhang et al., 2024) extends this concept to decoder-based models, using local A2S statistics for adaptive eviction in autoregressive generation. However, H2O suffers from the attention bias issue (Oren et al., 2024), particularly in long-context inputs. Several follow-up works have aimed to address this limitation. NACL (Chen et al., 2024) introduces random eviction to mitigate attention bias, while A2SF (Jo and Shin, 2024) incorporates a Forgetting Factor. However, none of these approaches fully resolves the underlying problem. Despite these limitations, some studies (Adams et al., 2024) suggest that H2O remains an optimal solution in many scenarios.

More Adaptive Strategies Although strategies based on accumulative attention scores provide more flexibility than static methods, they still have notable limitations. For instance, H2O (Zhang et al., 2024) applies the same token eviction ratio across all attention heads, restricting the adaptability of the method. FastGen (Ge et al., 2023), on the other hand, introduces a different approach by hybridizing KV-Cache compression policies and applying adaptive strategies to each attention head. However, it focuses on the decoding stage and neglects the importance of the prefilling stage. Learnable eviction strategies, on the other hand, offer greater flexibility by enabling different layers and attention heads to adopt heterogeneous eviction policies. However, such strategies have been relatively underexplored, likely due to concerns about the computational overhead they may introduce (Zhang et al., 2024; Chen et al., 2024). Nonetheless, task-specific training is essential for optimizing performance across different contexts. For example, a recent approach (Anagnostidis et al., 2024) introduces a learnable mechanism for dropping uninformative tokens, but it faces difficulties in batched generation and does not account for continual pre-training or decoding-only LLMs. Despite these challenges, learnable strategies have a strong potential to improve performance across a variety of tasks by allowing models to adapt eviction strategies to meet task-specific requirements.

3 Method

This section first briefly reveals multi-head self-attention and KV-Cache, then describes the *Attention-Gate* (AG) mechanism for in-context KV-Cache eviction. An illustrative overview of AG is presented in Figure 2.

3.1 Preliminary

Multi-Head Attention (MHA) (Vaswani et al., 2017) is a core component of Transformer, as used by most LLMs. MHA enables the model to capture dependencies across different tokens in a sequence. Specifically, for an input sequence $X \in \mathbb{R}^{n\times d}$, where n represents the sequence length and d denotes the dimensionality of the hidden states, the output of MHA is computed as:

$$\mathrm{MHA}(X) = [\mathrm{H}_1(X), \mathrm{H}_2(X), \ldots, \mathrm{H}_h(X)]\, W^O \,,$$

where $\{\mathrm{H}_i\}_{i=1}^h$ refers to the h attention heads and

$$\begin{aligned}
\mathrm{H}_i(X) &= \mathrm{Attn}\left(XW_i^Q, XW_i^K, XW_i^V\right) \\
&= \mathrm{Attn}\left(Q_i, K_i, V_i\right) \\
&= \mathrm{Softmax}\left(\frac{Q_i K_i^\top}{\sqrt{d_k}} - \mathrm{INF}(\mathbb{1} - M_i)\right) V_i \\
&= A_i V_i \,. \qquad (1)
\end{aligned}$$

Here, $W_i^Q, W_i^K \in \mathbb{R}^{d\times d_k}$, $W_i^V \in \mathbb{R}^{d\times d_v}$, and $W^O \in \mathbb{R}^{hd_v\times d}$ are learned projection matrices. INF is a large constant, $\mathbb{1}$ is a matrix of ones, M_i is the mask applied to head H_i, and A_i represents the attention scores for head H_i.

KV-Cache is employed during the inference of auto-regressive transformers, which stores the key and value information from previous time steps, allowing efficient reuse and reducing recomputation. The inference process can be divided into two stages: prefilling and decoding.

In the prefilling stage, the input sequence $X^{(\le n)} = \left[x^{(1)}, x^{(2)}, \ldots, x^{(n)}\right] \in \mathbb{R}^{n\times d}$ passes through MHA, and the corresponding key-value pairs $K_i^{(\le n)}$ and $V_i^{(\le n)}$ for head H_i are stored in KV-Cache. These are expressed as:

$$\begin{aligned}
K_i^{(\le n)} &= \left[k_i^{(1)}, \cdots, k_i^{(n)}\right] \,, \\
V_i^{(\le n)} &= \left[v_i^{(1)}, \cdots, v_i^{(n)}\right] \,,
\end{aligned}$$

where $k_i^{(t)} = x^{(t)}W_i^K$ and $v_i^{(t)} = x^{(t)}W_i^V$. After prefilling, the next token $x^{(n+1)}$ is generated.

In the decoding stage, $x^{(n+1)}$ is input to generate $x^{(n+2)}$ for the first step. During this process, only $k_i^{(n+1)}, v_i^{(n+1)}$ need to be computed for head H_i. These are then concatenated with the cached $K_i^{(\le n)}$ and $V_i^{(\le n)}$ to form $K_i^{(\le n+1)}$ and $V_i^{(\le n+1)}$, which are used to complete the current MHA computation and update the KV-Cache. The process repeats token by token until the generation is complete.

KV-Cache plays a critical role in improving the efficiency of LLM inference. However, the size of the KV-Cache grows with the input sequence length, leading to substantial memory overhead. Efficiently managing KV-Cache while maintaining model performance has become a key challenge in scaling LLMs to longer contexts.

3.2 Issues of Traditional Eviction Strategies

Various KV-Cache eviction strategies have been developed to mitigate the above issue, but they still suffer from limitations.

Lack of Adaptability Static approaches, such as streamingLLM (Xiao et al., 2024), lack adaptability across tokens, attention-heads, layers, tasks, and models. H2O (Zhang et al., 2024) addresses some of these by introducing token-level and head-level adaptability. However, it still applies a uniform eviction ratio across all attention heads, overlooking the significant variation in attention scores across different heads, as demonstrated by FastGen (Ge et al., 2023). Without finer-grained flexibility, it is then possible to retain unnecessary information, leading to reduced efficiency.

Absence of Global Context KV-Cache eviction strategies should ideally be context-aware, as the importance of the same token can vary significantly depending on the surrounding context. However, existing strategies relying on accumulative attention scores, such as H2O, NACL, and A2SF (Chen et al., 2024; Jo and Shin, 2024; Zhang et al., 2024), are primarily based on local statistics of the context. While methods like NACL and A2SF attempt to mitigate this by introducing various adjustments to reduce the misjudgment of token importance and the resulting biased retention (Oren et al., 2024), the root issue remains unsolved.

Inherent Inefficiency Methods like H2O and FastGen (Ge et al., 2023) are inefficient due to their sequential, token-by-token eviction at each decoding step. And, H2O computes attention scores before deciding which tokens to evict, wasting computation on soon-to-be discarded tokens.

3.3 Attention-Gate

To address these issues, we develop *Attention-Gate* (AG), a lightweight, trainable module positioned before the MHA layer to generate adaptive, context-aware *eviction flags* for the tokens. The flags determine which tokens in the KV-Cache of each head of the MHA should join the computation of attention scores and be retained in the KV-Cache.

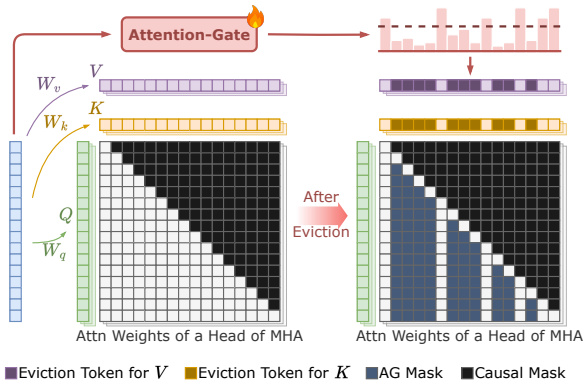

Figure 2: An overview of Attention-Gate (AG) for KV-Cache eviction. AG is a lightweight learnable module placed before each MHA layer. Given the input hidden states, it determines for each head whether to retain or discard the key and value tokens in the KV-Cache. In the attention weights, this corresponds to masking out columns for the evicted keys, while keeping the diagonal intact to ensure the query interacts with its own key.

AG includes (i) a lightweight multi-head attention with h-dim outputs, denoted as MHA$'$, to enable the awareness of *global context* of the sequence[1], and (ii) a gating mechanism G to enable adaptive eviction flags for the attention heads in the subsequent MHA layer. Notably, MHA$'$ has much fewer heads compared to MHA, i.e., $h' \ll h$.

Specifically, given the hidden states $X \in \mathbb{R}^{n \times d}$ yielded by the preceding module, AG outputs a binary mask for the heads in the subsequent MHA:

$$\text{AG}(X) = \text{G}\left(\text{MHA}'(X), \tau\right) \in \{0,1\}^{n \times h}, \quad (2)$$

where τ is the gating threshold used as:

$$\text{G}(s, \tau) = \begin{cases} 1, & \text{if Sigmoid}(s) > \tau \\ 0, & \text{otherwise} \end{cases}. \quad (3)$$

The KV for t-th token of the t-th head within the subsequent MHA layer are then retained if $\text{AG}(X)_i^{(t)} = 1$. Meanwhile, in the attention matrix, the columns corresponding to evicted tokens are masked out.[2] This way, AG selectively determines which tokens are retained or discarded for each attention head.

[1] To distinguish it from the vanilla MHA, all symbols in MHA$'$ are marked with a prime ($'$).

[2] The diagonal elements of the attention matrix, where a token attends to itself, are always preserved.

3.4 Training of AG

Eviction Loss We introduce the *Eviction Loss* to encourage the model to maintain the eviction ratio close to a target value β, which is defined as:

$$\ell_{\text{evict}} = \alpha \cdot \left| \overline{\text{AG}} - \beta \right|, \tag{4}$$

where $\overline{\text{AG}}$ denotes the average of all $\text{AG}(X)$ within the model. In particular, α adjusts the intensity of KV-Cache eviction, while β ensures that eviction does not become overly aggressive. Besides, various layers and heads have enough freedom to adjust their own eviction frequency. This loss function works alongside the auto-regressive loss to balance token eviction and model performance.

Initialization We initialize the AG parameters using Xavier initialization (Glorot and Bengio, 2010). Additionally, a small constant $\gamma \geq 0$ can optionally be added inside Sigmoid in Equation (3), ensuring that the initial retention probabilities are close to 1. This encourages the model to retain most tokens early in training.

Handling Non-Differentiability Directly applying the threshold-based gating mechanism from Section 3.3 would lead to non-differentiable gradients during training. To resolve this, we employ the *Straight-Through Estimator* (STE) (Yin et al., 2019), which allows gradients to flow through discrete decisions by approximating them during the backward pass. Specifically, during backpropagation, instead of using the hard 0 or 1 values obtained from comparing against the threshold, we utilize the smooth output of the Sigmoid function.

For more details, please refer to Section 4 and Appendix A.

3.5 Complexity Analysis

AG Module For input $X \in \mathbb{R}^{n \times d}$, the FLOPs for AG are:

$$\underset{\text{AG}}{\text{FLOPs}} = \mathcal{O}(n^2 d'_k h') \,.$$

MHA Module Without AG, original MHA's FLOPs are:

$$\underset{\text{original MHA}}{\text{FLOPs}} = \mathcal{O}(n^2 d_k h) \,. \tag{5}$$

After AG processing, $t\%$ of the KV-Cache tokens are discarded, leaving $(1 - t\%)$ for attention computation:

$$\underset{\text{MHA after AG}}{\text{FLOPs}} = \mathcal{O}\big((1 - t\%) n^2 d_k h\big) \,.$$

Combined AG & MHA the total FLOPs are:

$$\underset{\text{AG \& MHA}}{\text{FLOPs}} = \mathcal{O}\big(n^2 \big(d'_k h' + (1 - t\%) d_k h\big)\big) \,. \tag{6}$$

Efficiency The reduction in FLOPs depends on three factors: (i) Reduction in token count ($t\%$): Higher values of $t\%$ result in a larger reduction in the quadratic term of the original MHA. (ii) Head configuration ($h' < h$): The AG module must have significantly fewer heads (h') compared to the original MHA (h) to ensure its overhead is small. (iii) Head dimension ratio ($d'_k < d_k$): A smaller head dimension (d'_k) for AG further reduces its contribution to total FLOPs. The analysis above is further supported by our empirical results in Figure 3.

3.6 Discussion

Global v.s. Local AG utilizes MHA$'$ to inherently incorporate global information of the context. This is essential, as determining redundancy or relevance often requires a global understanding of the sequence. Alternatively, a simpler approach is to employ a linear transformation to generate eviction flags. This method relies solely on the hidden state of each token itself without incorporating information from other tokens in the sequence. While the local approach is computationally cheaper, as shown in Table 3, its performance is not guaranteed. This limitation highlights the challenges faced by methods that rely on local statistics of the context, as discussed in Section 3.2.

Prefilling or Decoding Our current studies mainly apply AG to the prefilling stage following NACL (Chen et al., 2024), where the full sequence is available. By making eviction decisions before the MHA layers, AG effectively manages the KV-Cache during this phase. AG can be easily extended to the decoding phase by maintaining an extra KV-Cache for MHA$'$, which is still very economical because MHA$'$ has far fewer heads than MHA.

4 Experiments

This section consists of three parts. First, we evaluate the performance of AG in two scenarios: continual pre-training (CPT) and supervised fine-tuning (SFT) (Section 4.1 & 4.2). Second, we provide a visualization of selected examples to demonstrate the core characteristics of AG (Section 4.3). Finally, we conduct ablation studies to provide further insights into the effectiveness of AG (Section 4.4). Additional results are provided in Appendix A.1.

	Metric	PIQA	ARC-C	ARC-E	RTE	COPA	BoolQ	HellaSwag	MMLU	Avg.	Metric	LongBench
Llama2-7B-cpt	Acc.	72.69	32.88	50.62	50.54	57.00	64.77	42.19	26.64	49.67	Score	23.42
streamingLLM	Acc.	72.42	31.53	**49.74**	50.90	54.00	61.31	37.75	26.66	48.04	Score	4.61
H2O	Acc.	72.20	30.85	49.38	**51.99**	55.00	**62.42**	41.45	26.45	48.72	Score	4.85
Ours	Acc.	**76.33**	**32.20**	48.32	50.18	**59.00**	60.46	**64.23**	**28.54**	**52.41**	Score	**13.71**
	%Evict.	43.12	46.54	45.15	48.60	**55.37**	**50.16**	**61.10**	**70.36**	**52.55**	%Evict.	**68.55**

Table 1: Performance comparison of Llama2-7B and various KV-Cache eviction strategies *after continual pre-training*. For baselines, (W_q, W_k, W_v, W_o) are made trainable, while in our method, the AG module is also trainable. Higher values indicate better performance for all metrics. Acc. refers to accuracy. %Evict. refers to the mean KV-Cache eviction ratio, representing the percentage of tokens evicted from KV-Cache. The eviction ratio is fixed at 50% for the baseline methods. In contrast, our method achieves better performance (average accuracy and score) while maintaining a higher average %Evict..

4.1 Continual Pre-training

Models & Datasets We use Llama2-7B (Touvron et al., 2023) as our primary base model. Additionally, we validate the feasibility of our approach on Mistral-7B (Jiang et al., 2023), with results provided in Table 5. We select a subset of the RedPajama dataset (Computer, 2023), comprising approximately 5,000 samples [3], to serve as the training set. To assess the effectiveness of our method, we evaluate it on widely recognized benchmarks: PIQA (Bisk et al., 2020), ARC-C (Clark et al., 2018), ARC-E (Clark et al., 2018), RTE (Bar-Haim et al., 2006), COPA (Roemmele et al., 2011), BoolQ (Clark et al., 2019), HellaSwag (Zellers et al., 2019), MMLU (Hendrycks et al., 2021), and LongBench (Bai et al., 2023). All evaluations are conducted in a zero-shot setting, with performance assessed using OpenCompass (Contributors, 2023).

Training Details & Baselines During CPT, (W_q, W_k, W_v, W_o) are made trainable using LoRA (Hu et al., 2021). In our method, AG is also trainable.[4] For the Llama2-7B model, $\tau = 0.5$, $\gamma = 2$, $\alpha = 5$, and $\beta = 0.4$. The model was trained for a single epoch. We compare to streamingLLM (Xiao et al., 2024), a representative of static strategies, and H2O (Zhang et al., 2024), which represents methods based on accumulative attention scores.[5] For baseline methods, the eviction ratio is fixed at 50%.

[3] Specifically, we sampled 4,997 samples proportionally from each subset of the RedPajama dataset.

[4] We tested a version where only AG is trainable, with other parameters frozen. Results are shown in Table 4.

[5] Adams et al. (2024) suggest that H2O is an optimal solution in many scenarios. Due to its strong performance and the difficulty of reproducing other training-free methods, no additional training-free baselines were included.

	Metric	PIQA	ARC-C	RTE	COPA	BoolQ	OBQA	Avg.
Fine-tuned Llama2-7B	Acc.	82.92	60.34	64.98	92.00	88.10	78.80	77.86
Ours ($\alpha = 1$)	Acc.	82.15	59.66	64.26	93.00	86.82	78.80	77.45
	%Evict.	66.16	48.31	65.47	45.40	67.46	67.17	60.00
Ours ($\alpha = 0.5$)	Acc.	81.50	57.63	74.01	95.00	87.00	79.20	79.06
	%Evict.	64.96	36.45	62.80	34.77	67.31	66.65	55.49

Table 2: Performance of Llama2-7B with LoRA fine-tuning and our method on six downstream tasks. Our method makes the AG modules learnable. Two settings for α (0.5 and 1) are tested. Our method maintains comparable or better accuracy while achieving a higher eviction ratio, demonstrating its *task-specific adaptability* in managing token eviction.

Results As shown in Table 1, our method can better balance performance and KV-Cache eviction. It consistently outperforms the baseline strategies in performance across most tasks while achieving a higher mean eviction ratio. Moreover, our added computational overhead is minimal. The CPT was conducted on only 5,000 samples and trained for just one epoch. This efficiency can be attributed to the fact that our method does not need to learn new knowledge from scratch but rather focuses on learning effective token retention strategies, leveraging the existing capabilities of the pre-trained model.

4.2 Supervised Fine-tuning

Model & Tasks We also use Llama2-7B (Touvron et al., 2023), and evaluate on six widely recognized downstream tasks: PIQA, ARC-C, RTE, COPA, BoolQ, and OpenBookQA (Mihaylov et al., 2018). For each task, we fine-tune model using the respective training set and evaluate its performance on the corresponding test set.

Implementation Details The selection of trainable parameters follows Section 4.1. The hyper-

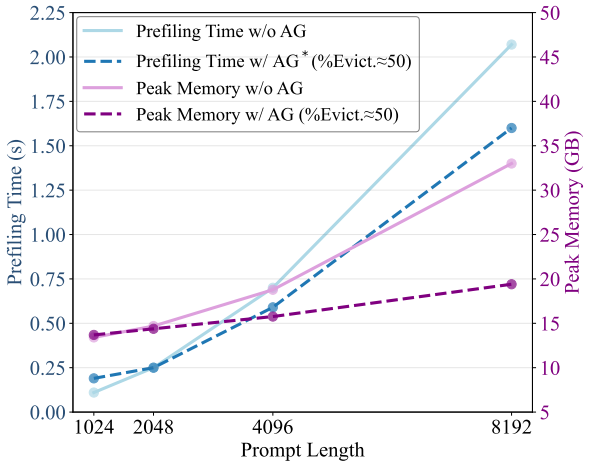

Figure 3: Comparison of peak memory usage and prefilling time between the LLaMA2-7B model (without AG) and the proposed implementation (with AG and ∼50% eviction) across varying prompt lengths.

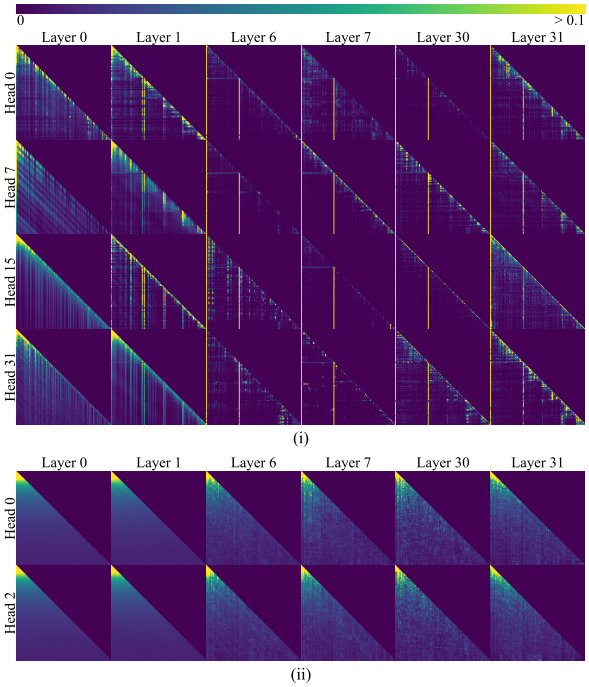

Figure 4: Attention patterns in Llama2-7B after fine-tuning on BoolQ. (i) MHA heads across layers before eviction; (ii) AG attention scores across layers.

parameters are $\tau = 0.3$, $\gamma = 0$, $\alpha = 1$ or 0.5, and $\beta = 0.28$. Training is performed using the AdamW optimizer (Loshchilov and Hutter, 2017) with a learning rate of 5e-5 for 2 epochs per dataset.

Results As shown in Table 2, our method achieves a strong balance between accuracy and KV-Cache eviction. With $\alpha = 1$, it maintains competitive accuracy compared to the fine-tuned Llama2-7B baseline while achieving a high mean eviction ratio of 60.00%. With $\alpha = 0.5$, the eviction ratio decreases to 55.49%, but the average accuracy improves. In tasks like RTE and COPA, it even surpasses the baseline. This suggests that effective token eviction helps the model focus on relevant information. Additionally, performance varies across tasks under the same settings. For instance, ARC-C is more challenging to evict compared to OpenBookQA, leading to a larger accuracy drop post-eviction. This highlights the importance of *task-specific* KV-Cache eviction policies.

Inference Efficiency We evaluated the inference efficiency of the LLaMA2-7B model with and without the AG module, as shown in Figure 3. Our method achieves significant memory savings, especially as prompt lengths increase. Although we have not implemented specialized sparse MHA kernels and the current prefilling implementation (marked with * in the legend) uses a suboptimal for-loop over attention heads, AG still maintains stable prefilling time and even shows a decreasing trend as prompt length grows. These empirical results align well with the analysis presented in Section 3.5. We believe there is room for further acceleration through kernel fusion techniques, which we leave for future work.

4.3 Attention Pattern

We visualize the attention patterns of both MHA and AG mechanisms in Llama2-7B after fine-tuning on the BoolQ dataset using a selected sample (Figure 4). The complete visualization is available in Appendix A.3. From the figure, we derive insights in two key areas:

(i) MHA Attention Patterns Before Eviction Multiple MHA heads across layers exhibit diverse attention behaviors—such as vertical, horizontal, and diagonal patterns—particularly in the first two layers where heterogeneity is most pronounced. As layers deepen, attention patterns become progressively sparser, shifting from dense activations in early layers to more focused patterns in later ones. Notably, bright yellow vertical lines in deeper layers consistently highlight critical tokens essential for inference. These correspond to the *Heavy Hitters* identified in H2O (Zhang et al., 2024), emphasizing tokens that significantly contribute to attention scores. Our method preserves these critical tokens in deeper layers, maintaining their importance throughout the network.

	Metric	PIQA	ARC-C	RTE	COPA	BoolQ	OBQA	Avg.
(1)	Acc.	82.15	59.66	64.26	93.00	86.82	78.80	77.45
	%Evict.	66.16	48.31	65.47	45.40	67.46	67.17	60.00
(2-1)	Acc.	81.88	57.63	65.70	91.00	87.52	77.40	76.86
	%Evict.	63.92	36.38	62.73	24.38	65.22	63.57	52.70
(2-2)	Acc.	82.15	53.90	62.45	89.00	87.31	77.40	75.37
	%Evict.	58.97	31.47	59.77	20.32	63.02	59.17	48.79
(3-1)	Acc.	81.45	53.36	58.84	88.00	86.73	78.40	74.46
	%Evict.	61.75	33.55	61.34	19.24	64.59	59.59	50.01
(3-2)	Acc.	83.03	53.90	59.93	89.00	87.16	76.40	74.90
	%Evict.	58.68	24.23	32.23	12.28	59.40	55.54	40.39
(4-1)	Acc.	81.66	55.25	66.06	88.00	86.85	78.00	75.97
	%Evict.	49.52	36.92	46.85	28.74	56.02	60.32	46.40
(4-2)	Acc.	82.75	55.93	79.06	82.00	86.33	78.40	77.41
	%Evict.	53.31	44.38	51.20	47.95	61.98	61.73	53.43
(5)	Acc.	82.54	54.58	57.40	81.00	87.71	74.80	73.01
	%Evict.	1.06	0.46	0.81	0.26	1.38	1.16	0.86
(6-1)	Acc.	83.08	50.85	65.34	82.00	87.31	73.20	73.63
	%Evict.	65.15	40.96	64.29	21.37	67.49	63.69	53.83
(6-2)	Acc.	81.61	53.56	60.29	82.00	87.37	74.20	73.17
	%Evict.	65.66	44.48	65.14	24.28	68.18	63.44	55.20

Table 3: Ablation study on AG configurations, reporting accuracy (Acc.) and KV-Cache eviction ratio (%Evict.) under various settings.

(ii) MHA′ Attention Patterns The attention scores produced by MHA′ exhibit a clear transition from high-resolution focus in early layers to lower-resolution, distilled representations of in-context information in deeper layers. This suggests that deeper layers of MHA′ rely less on detailed global context, as earlier layers have already sufficiently refined the relevant information. This behavior indicates potential for improving efficiency by reducing the number of attention heads or feature dimensions in the deeper layers of MHA′.

4.4 Ablation

We investigate the effects of different configurations of the AG mechanism, including the number of heads, head dimensions, and eviction strategies. Detailed results are summarized in Table 3.

Number of Heads Reducing the number of AG heads from 4 (setting (1)) to 2 or 1 (settings (2-1) and (2-2)) leads to decreases in both accuracy and eviction ratio. This indicates that the capacity of AG is closely tied to the number of heads.

Head Dimensions Similarly, decreasing the dimensionality of AG heads (settings (3-1) and (3-2)) results in lower eviction capabilities and accuracy, highlighting the importance of maintaining sufficient head dimensionality.

Layer-wise Guided Eviction Strategies Settings (4-1) and (4-2) investigate using the previous layer's hidden states and AG module to guide eviction decisions in the current layer, introducing inter-layer dependency to enable parallelism. In (4-1), eviction starts from the second layer onward: the first layer does not evict, while each subsequent layer uses the eviction information from the immediately preceding layer (e.g., the first layer predicts eviction for the second layer, the second for the third, and so forth). In (4-2), eviction begins from the third layer, with the second layer guiding the third, the third guiding the fourth, etc. This design stems from the observation that the first two layers retain most KV pairs after standard AG training, as shown in Figure 1, where these early layers exhibit minimal masking. By skipping eviction in these initial layers, computational resources are saved without significantly impacting performance. Although these guided strategies introduce parallelism and reduce computation in early layers, they result in a slight decrease in accuracy and eviction ratio compared to setting (1), indicating a trade-off between efficiency and effectiveness.

Replacement of MHA′ with Linear Layer Setting (5) replaces the MHA′ in AG with a simple linear layer to determine eviction. The sharp drop in eviction effectiveness and accuracy compared to (1) underscores the necessity of using attention-like mechanisms to capture global in-context information for successful eviction.

Recent Token Retention Settings (6-1) and (6-2) vary the number of recent tokens retained at 5 and 10, respectively. The results suggest that increasing the number of recent tokens does not necessarily improve performance within the AG framework. Future work could explore more sophisticated approaches for managing recent tokens, such as learnable weighted strategies.

5 Conclusion

In conclusion, the proposed Attention-Gate mechanism offers a flexible and adaptive solution to KV-Cache eviction in large language models. By dynamically identifying and discarding less important tokens in a data-driven manner, Attention-Gate addresses the limitations of static and attention-score-based strategies, providing efficient context-aware eviction. This mechanism integrates seamlessly with pre-trained models and can be easily tuned, making it a practical and effective method for enhancing both performance and memory efficiency in various tasks.

Limitations

This work primarily focuses on the prefilling stage. Due to limited computational resources, experiments were not conducted on larger-scale models.

References

Griffin Adams, Faisal Ladhak, Hailey Schoelkopf, and Raja Biswas. 2024. Cold compress: A toolkit for benchmarking kv cache compression approaches.

Sotiris Anagnostidis, Dario Pavllo, Luca Biggio, Lorenzo Noci, Aurelien Lucchi, and Thomas Hofmann. 2024. Dynamic context pruning for efficient and interpretable autoregressive transformers. *Advances in Neural Information Processing Systems*, 36.

Yushi Bai, Xin Lv, Jiajie Zhang, Hongchang Lyu, Jiankai Tang, Zhidian Huang, Zhengxiao Du, Xiao Liu, Aohan Zeng, Lei Hou, Yuxiao Dong, Jie Tang, and Juanzi Li. 2023. Longbench: A bilingual, multitask benchmark for long context understanding. *Preprint*, arXiv:2308.14508.

Roy Bar-Haim, Ido Dagan, Bill Dolan, Lisa Ferro, Danilo Giampiccolo, Bernardo Magnini, and Idan Szpektor. 2006. The second pascal recognising textual entailment challenge. In *Proceedings of the second PASCAL challenges workshop on recognising textual entailment*, volume 1. Citeseer.

Yonatan Bisk, Rowan Zellers, Ronan Le Bras, Jianfeng Gao, and Yejin Choi. 2020. Piqa: Reasoning about physical commonsense in natural language. In *Thirty-Fourth AAAI Conference on Artificial Intelligence*.

Yilong Chen, Guoxia Wang, Junyuan Shang, Shiyao Cui, Zhenyu Zhang, Tingwen Liu, Shuohuan Wang, Yu Sun, Dianhai Yu, and Hua Wu. 2024. Nacl: A general and effective kv cache eviction framework for llms at inference time. *arXiv preprint arXiv:2408.03675*.

Wei-Lin Chiang, Zhuohan Li, Zi Lin, Ying Sheng, Zhanghao Wu, Hao Zhang, Lianmin Zheng, Siyuan Zhuang, Yonghao Zhuang, Joseph E. Gonzalez, Ion Stoica, and Eric P. Xing. 2023. Vicuna: An open-source chatbot impressing gpt-4 with 90%* chatgpt quality.

Rewon Child, Scott Gray, Alec Radford, and Ilya Sutskever. 2019. Generating long sequences with sparse transformers. *arXiv preprint arXiv:1904.10509*.

Christopher Clark, Kenton Lee, Ming-Wei Chang, Tom Kwiatkowski, Michael Collins, and Kristina Toutanova. 2019. BoolQ: Exploring the surprising difficulty of natural yes/no questions. In *Proceedings of NAACL-HLT 2019*.

Peter Clark, Isaac Cowhey, Oren Etzioni, Tushar Khot, Ashish Sabharwal, Carissa Schoenick, and Oyvind Tafjord. 2018. Think you have solved question answering? try arc, the ai2 reasoning challenge. *arXiv:1803.05457v1*.

Together Computer. 2023. Redpajama: An open source recipe to reproduce llama training dataset.

OpenCompass Contributors. 2023. Opencompass: A universal evaluation platform for foundation models. https://github.com/open-compass/opencompass.

Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, et al. 2024. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*.

Suyu Ge, Yunan Zhang, Liyuan Liu, Minjia Zhang, Jiawei Han, and Jianfeng Gao. 2023. Model tells you what to discard: Adaptive kv cache compression for llms. *arXiv preprint arXiv:2310.01801*.

Xavier Glorot and Yoshua Bengio. 2010. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pages 249–256. JMLR Workshop and Conference Proceedings.

Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. 2021. Measuring massive multitask language understanding. *Proceedings of the International Conference on Learning Representations (ICLR)*.

Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. 2021. Lora: Low-rank adaptation of large language models. *arXiv preprint arXiv:2106.09685*.

Albert Q Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, et al. 2023. Mistral 7b. *arXiv preprint arXiv:2310.06825*.

Hyun Rae Jo and Dong Kun Shin. 2024. A2sf: Accumulative attention scoring with forgetting factor for token pruning in transformer decoder. *arXiv preprint arXiv:2407.20485*.

Ilya Loshchilov and Frank Hutter. 2017. Decoupled weight decay regularization. *Preprint*, arXiv:1711.05101.

Todor Mihaylov, Peter Clark, Tushar Khot, and Ashish Sabharwal. 2018. Can a suit of armor conduct electricity? a new dataset for open book question answering. In *EMNLP*.

Matanel Oren, Michael Hassid, Yossi Adi, and Roy Schwartz. 2024. Transformers are multi-state rnns. *arXiv preprint arXiv:2401.06104*.

Melissa Roemmele, Cosmin Adrian Bejan, and Andrew S. Gordon. 2011. Choice of plausible alternatives: An evaluation of commonsense causal reasoning. In *2011 AAAI Spring Symposium Series*.

Gemma Team, Morgane Riviere, Shreya Pathak, Pier Giuseppe Sessa, Cassidy Hardin, Surya Bhupatiraju, Léonard Hussenot, Thomas Mesnard, Bobak Shahriari, Alexandre Ramé, et al. 2024. Gemma 2: Improving open language models at a practical size. *arXiv preprint arXiv:2408.00118*.

Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, et al. 2023. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*.

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.

Hanrui Wang, Zhekai Zhang, and Song Han. 2021. Spatten: Efficient sparse attention architecture with cascade token and head pruning. In *2021 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, pages 97–110. IEEE.

Guangxuan Xiao, Yuandong Tian, Beidi Chen, Song Han, and Mike Lewis. 2024. Efficient streaming language models with attention sinks. *Preprint*, arXiv:2309.17453.

Penghang Yin, Jiancheng Lyu, Shuai Zhang, Stanley Osher, Yingyong Qi, and Jack Xin. 2019. Understanding straight-through estimator in training activation quantized neural nets. *arXiv preprint arXiv:1903.05662*.

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*.

Zhenyu Zhang, Ying Sheng, Tianyi Zhou, Tianlong Chen, Lianmin Zheng, Ruisi Cai, Zhao Song, Yuandong Tian, Christopher Ré, Clark Barrett, et al. 2024. H2o: Heavy-hitter oracle for efficient generative inference of large language models. *Advances in Neural Information Processing Systems*, 36.

A Additional Experiments

A.1 Additional Results for Continual Pre-training

In this section, we perform continual pre-training on Llama2-7B using the same training data and hyperparameter settings described in Section 4.1. The baselines are training-free, while in our method, only the AG module is trainable. The results are shown in Table 4.

A.2 Results of Continual Pre-training on Mistral

We conducted continual pre-training on Mistral-7B (Jiang et al., 2023) using 5,000 samples from RedPajama (Computer, 2023), and the results are shown in Table 5. Compared to the performance of Llama2-7B presented in Table 4, Mistral's performance slightly declined. We hypothesize that this may be due to the distribution of RedPajama's data being less suited to Mistral. Additionally, this raises the question of whether KV-Cache eviction is model-dependent, and whether its effectiveness is related to the model's expressive power. Although the parameter counts of Mistral-7B and Llama2-7B are similar, Mistral-7B significantly outperforms Llama2-7B. This could suggest that Mistral is utilizing more tokens or scoring them with finer granularity, which results in fewer redundant tokens and thus makes eviction less effective. Furthermore, it is possible that Mistral's use of grouped-query attention (GQA), which inherently involves compression, may make it more challenging to increase the eviction ratio effectively in this context.

A.3 More Visualization

Figure 5 provides a comprehensive view of the layers and attention heads from Figure 4.

	Metric	PIQA	ARC-C	ARC-E	RTE	COPA	BoolQ	HellaSwag	Avg.
Llama2-7B	Acc.	76.33	37.29	51.32	51.99	62.00	69.94	68.16	59.58
Local	Acc.	69.97	31.86	48.68	51.99	60.00	57.86	37.08	51.06
streamingLLM	Acc.	72.69	33.22	51.15	50.18	**63.00**	62.05	40.85	53.31
H2O	Acc.	75.9	33.22	**52.03**	**52.71**	47.00	67.37	66.32	56.36
Ours	Acc.	**76.17**	**33.90**	49.03	52.35	**63.00**	**67.52**	**66.33**	**58.33**
	%Evict.	**54.29**	**51.03**	**51.05**	46.70	40.02	**57.75**	**52.16**	**51.87**

Table 4: Performance comparison of Llama2-7B and various KV-Cache eviction strategies across seven tasks. *Our approach trains only the AG module during continual pre-training, keeping other components frozen.* The table reports accuracy (Acc.) for Llama2-7B and all eviction methods, with Llama2-7B serving as the upper bound for accuracy. Metric %Evict. refers to the mean KV-Cache eviction ratio, representing the percentage of tokens evicted from the KV-Cache. The eviction ratio is fixed at 50% for the baseline methods, including a local strategy (retaining only recent tokens), streamingLLM, and H2O. In contrast, our method achieves higher average accuracy while maintaining a higher average %Evict..

	Metric	PIQA	ARC-C	ARC-E	RTE	COPA	BoolQ	HellaSwag	Avg.
Mistral-7B	Acc.	80.09	42.37	63.14	48.01	76	64.22	73.02	63.84
Ours	Acc.	75.90	34.24	55.2	48.01	65	62.2	67.91	58.35
	Eviction	37.14	39.48	37.80	40.93	45.27	44.68	50.92	42.32

Table 5: Performance comparison between Mistral-7B and Ours across various tasks.

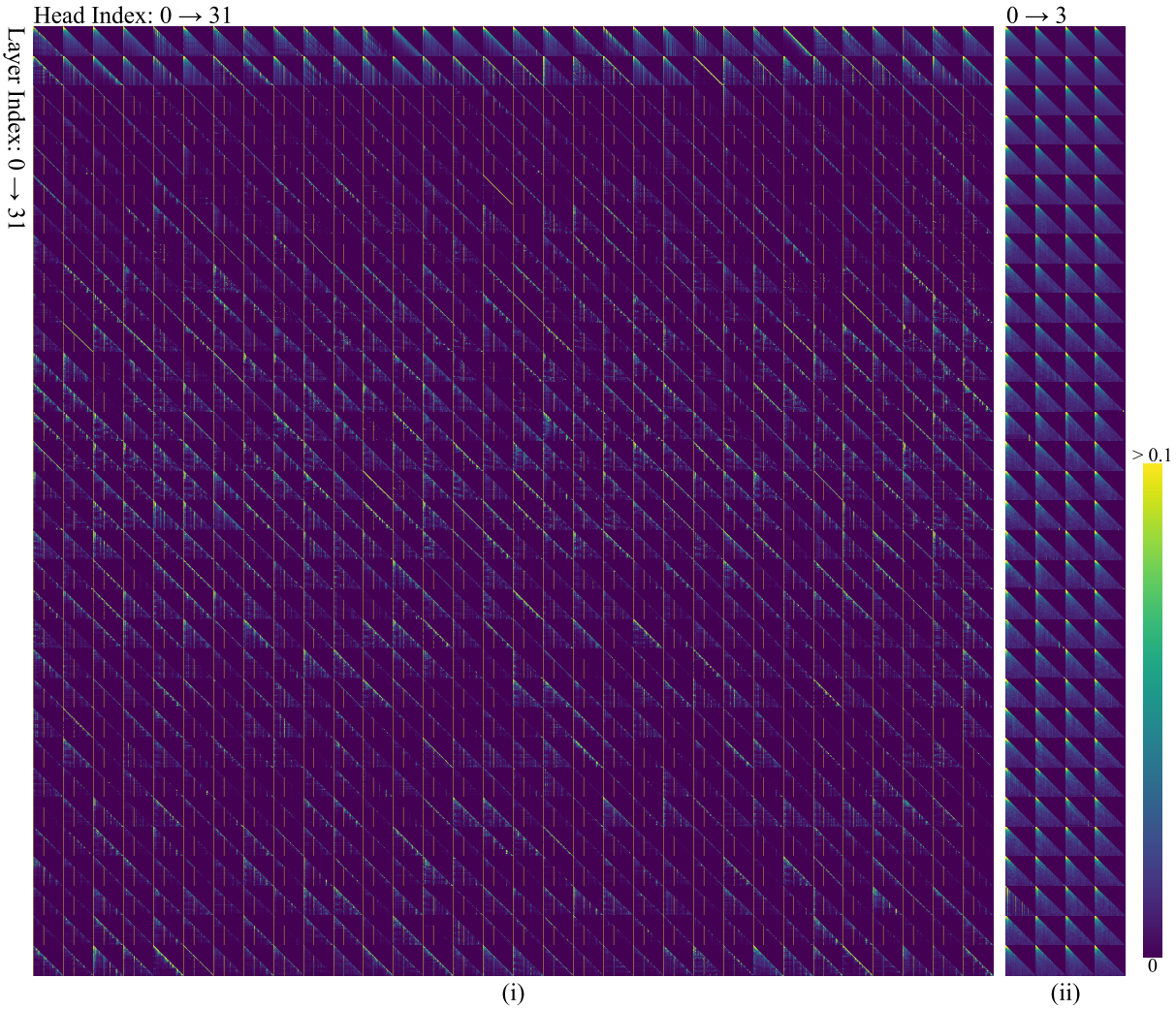

Figure 5: The complete version of Figure 4. Two key observations emerge in (i): 1. the first two layers are denser compared to the subsequent layers, and 2. bright-yellow vertical lines, representing critical tokens for inference, consistently appear across heads in deeper layers.