A2SF: ACCUMULATIVE ATTENTION SCORE WITH FORGETTING FACTOR

Anonymous authors
Paper under double-blind review

ABSTRACT

Transformer-based large language models (LLMs) face memory bottlenecks during inference due to key-value (KV) caches. The direction of recent researches has been to identify and discard redundant tokens. Existing approaches often rely on attention scores to remove low-contributing tokens. However, because they adopt a fixed observation window size, they fail to guarantee input-level optimality and frequently suffer performance degradation across different environments. To overcome these limitations, we propose A2SF (accumulative Attention Score with Forgetting Factor), a token selection method that applies a forgetting factor to accumulative attention scores. A2SF defines a accumulative formula that gradually forgets past attention contributions over time, thereby generalizing fixed-window approaches. Furthermore, it employs reinforcement learning to dynamically explore appropriate forgetting factors for each sentence. Experimental results demonstrate that by dynamically adapting to the characteristics of each input, A2SF more effectively selects critical tokens, thereby achieving 4–9% performance improvements over competitive baselines under a highly constrained token budget.

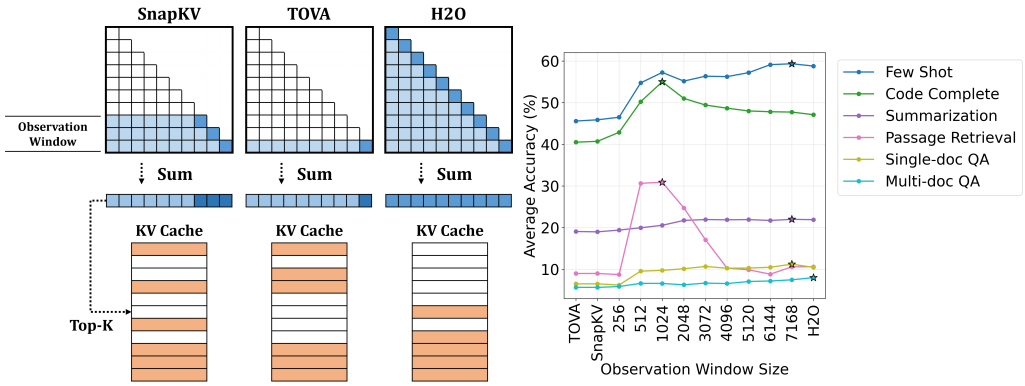
1 INTRODUCTION

Large language models (LLMs) have recently driven remarkable advances in natural language processing and generation, playing a central role in applications such as summarization, translation, document generation, and conversational systems. In particular, auto-regressive Transformer decoder architectures, exemplified by the LLaMA family, have gained wide adoption in both industry and research due to their ability to generate sequential tokens that ensure contextual coherence and expressive power.

However, this decoder structure stores the Key and Value of every generated token in the KV cache and repeatedly references them during subsequent computations. As the sequence length increases, KV cache memory usage grows linearly, creating severe bottlenecks in GPU memory capacity and bandwidth. These constraints can lead to slower processing speeds, restricted batch sizes, and even computation failures due to memory exhaustion, posing major challenges for large-scale deployment and real-world service integration of LLMs.

To alleviate these issues, recent studies have focused on token pruning techniques, which reduce KV cache memory usage by discarding less important tokens. Notable examples include H2O, SnapKV, and TOVA. These methods commonly leverage attention scores to identify high-contribution tokens and remove those deemed redundant. To preserve recent information, they often combine this with local window attention, thereby mitigating performance degradation.

Nonetheless, existing approaches differ in how they incorporate historical attention scores into the token selection process. TOVA considers only the most recent attention scores. H2O aggregates all past attention scores uniformly, thereby reflecting long-term history. SnapKV takes a middle ground, summing scores within a fixed observation window of limited size, illustrated in Figure 1a. Our analysis shows that these choices directly affect the distribution of selected tokens, and the optimal window size varies depending on the task. Smaller windows favor tokens generated more recently, whereas larger windows capture long-range contextual contributions. Consequently, fixed-window methods struggle to consistently achieve optimal performance across diverse tasks.



(a) Examples of existing accumulative attention score methods. The observation window specifies the range of past queries whose attention scores are included in the accumulation. TOVA considers only the most recent attention scores. SnapKV adopts an intermediate window size, summing scores within a fixed observation window. H2O uniformly aggregates all past attention scores.

(b) Task-specific accuracy across different observation window sizes. Stars (★) denote the highest accuracy achieved for each task. The optimal window size varies across tasks, with Few Shot and Code Complete performing best at mid-sized windows (1024), while Summarization, Retrieval, and QA tasks benefit from larger windows (7168).

Figure 1: Existing method examples and task-specific accuracy of accumulative attention score methods.

To address this limitation, we propose A2SF (accumulative Attention Score with Forgetting Factor), a generalized extension of fixed-window methods that introduces a forgetting factor into the accumulation process. A2SF defines a forgetting function that gradually attenuates past attention contributions as a function of their temporal distance from the current generation step, thereby formalizing the observation window size mathematically. This enables the model to flexibly adapt token selection to input-specific characteristics, achieving both stable performance and memory savings beyond what fixed-window approaches can provide. Furthermore, we propose a strategy that leverages reinforcement learning to automatically search for and adaptively adjust the forgetting factor in response to the characteristics of each input sequence.

The contributions of this work to KV cache compression can be summarized as follows:

- We define a formulation that generalizes existing methods based on fixed observation windows.
- We propose a method to determine the formulation's hyperparameter in accordance with input characteristics.
- We validate the effectiveness of our approach through comprehensive experiments, showing consistent accuracy gains across diverse tasks and context lengths.

The remainder of this paper is organized as follows. Section 2 reviews related work and highlights the limitations of existing methods. Section 3 analyzes the effect of observation window size in accumulative attention score approaches. Section 4 presents the core idea and mathematical formulation of A2SF, together with a reinforcement learning strategy for selecting an appropriate forgetting factor. Section 5 reports experimental results across diverse tasks and model settings, demonstrating the effectiveness of the proposed method.

2 Related Works

2.1 KV Cache Compression

Research on improving memory efficiency in decoder-based large language models (LLMs) can largely be divided into two directions: system-level approaches and cache compression techniques. Representative system-level methods include vLLM (Kwon et al., 2023), LMCache (Yao et al.,

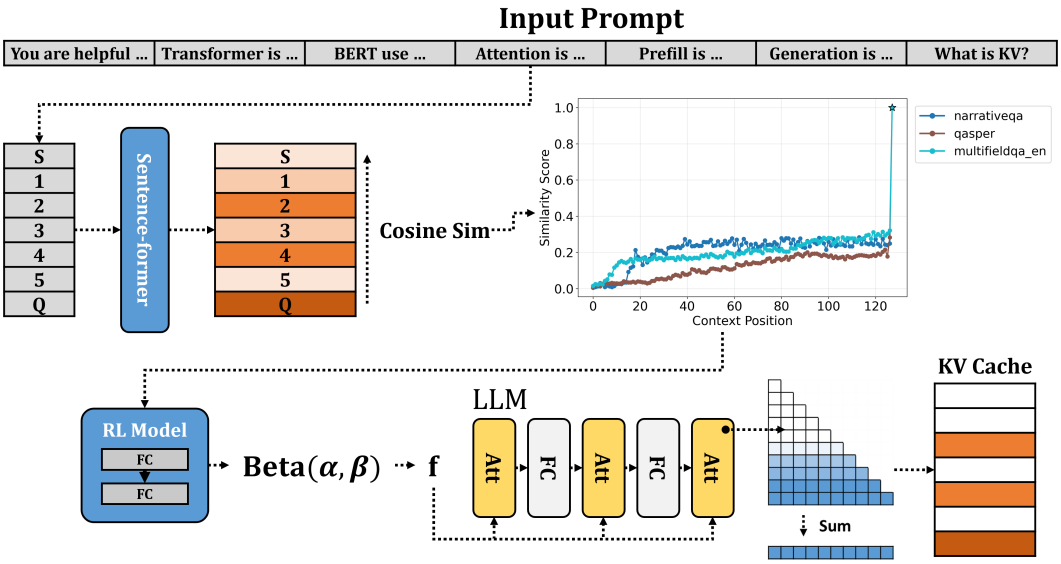


Figure 2: A2SF Overview. The input prompt is first divided into sentence-level units, which are embedded using a Sentence Transformer. These embeddings are compared with the final query sentence to construct a semantic state representation, which is then fed into a reinforcement learning (RL) model. The RL agent samples a forgetting factor f. Based on this factor, token indices are selected to update the KV cache.

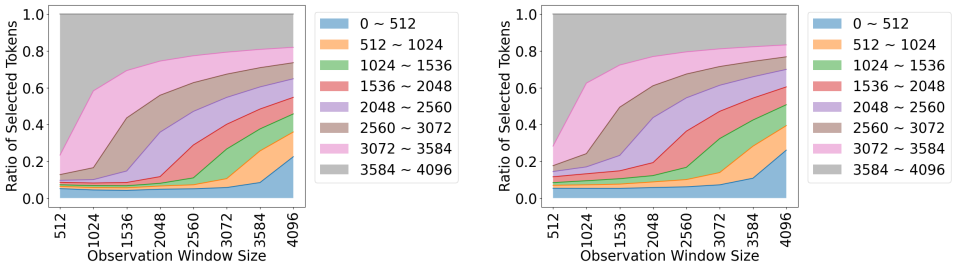
2025), and FlexGen (Sheng et al., 2023). These approaches manage KV caches at the page level to reduce fragmentation and offload parts of the cache to CPUs or SSDs to reduce GPU memory usage. However, as the cache grows, offloading inevitably increases data transfer overhead and resource consumption. This highlights the need for methods that directly reduce the size of the KV cache itself.

Meanwhile, a significant body of work focuses on token removal techniques, which generally evaluate the importance of each token and remove those deemed unnecessary. One notable example is H2O (Heavy-Hitter Oracle) (Zhang et al., 2024), which accumulates attention scores across all past timesteps to compute token importance, discarding tokens with low cumulative scores. While this approach captures long-range context, it risks retaining excessive historical information. In contrast, TOVA (Token Omission via Attention) (Oren et al., 2024) considers only the most recent attention scores, emphasizing recency and enabling removal of outdated tokens—though at the cost of weakening long-term dependencies. SnapKV takes a middle ground, accumulating attention scores within a fixed observation window to strike a balance between recency and long-range context.

Other methods aim to mitigate information loss without entirely discarding tokens. Get More with LESS (Dong et al., 2024) compresses discarded tokens into low-dimensional representations for storage, while No Token Left Behind (Paiss et al., 2022) applies variable quantization levels based on token importance, retaining all tokens while still lowering memory usage. These strategies mitigate the risk of context loss introduced by pruning.

Dynamic budget allocation has also been explored. AdaKV (Feng et al., 2024) assigns variable memory resources to tokens depending on input context, while PyramidKV (Cai et al., 2024) applies different token retention policies across layers and heads, allowing the model to adapt flexibly to diverse scenarios. These methods enable more efficient and context-aware resource allocation.

Architectural innovations provide another line of improvement. DuoAttention (Xiao et al., 2024) partitions attention heads into two groups: one processes all tokens, while the other operates only on pruned tokens, striking a balance between efficiency and accuracy. PoD (Pruning on Demand) (Ma et al., 2024) reduces redundancy by sharing attention outputs across adjacent layers. However, such methods often require structural modifications, which can limit their practicality.



(a) Passage Retrieval (b) Summarization

Figure 3: Distribution of 128 selected tokens under varying observation window sizes for Passage Retrieval and Summarization (input length fixed at 4096). Tokens are grouped into segments of 512 for analysis. Smaller windows concentrate selections on recent tokens, whereas larger windows result in a more distributed selection across the entire context.

In summary, while existing attention score–based studies vary in how they utilize scores, they all rely on fixed policies. In contrast, we propose A2SF, which introduces a forgetting factor that gradually attenuates the contribution of past attention scores over time. This removes the need for fixed observation windows and enables input-adaptive token selection. Moreover, techniques such as information compression and dynamic budget allocation can be integrated with A2SF to further improve both memory efficiency and accuracy.

2.2 Reinforcement Learning

Reinforcement Learning (RL) is a earning paradigm for decision-making in which an agent interacts with an environment by observing a state, selecting an action, and receiving a reward. Through interaction, the agent learns a policy that maximizes expected cumulative reward. The design of state and action spaces is critical: states must capture relevant context while remaining tractable, and actions may be discrete or continuous depending on the task.

Among policy optimization methods, Proximal Policy Optimization (PPO) (Schulman et al., 2017) is widely adopted for its stability and efficiency. PPO constrains policy updates within a trust region using a clipped objective, preventing destabilizing changes while maintaining learning progress. In continuous control settings, the policy is often modeled as a probability distribution. While Gaussian distributions are common, they are unbounded. When actions must lie strictly within [0,1], the Beta distribution is more appropriate. Combining PPO with a Beta distribution (Petrazzini & Antonelo, 2021) thus enables smooth, continuous action sampling restricted to the unit interval, making it well-suited for tasks requiring bounded control.

3 Observation

To examine the impact of attention score observation window size on model performance, we conducted experiments on the LongBench (Bai et al., 2023) benchmark using the LLaMA3 (Grattafiori et al., 2024) 8B model. The budget of local tokens was fixed at 16 and the number of selectable tokens at each step was constrained to a budget of 112, resulting in the total budget of 128.

3.1 Observation Window and Accuracy

Figure 1b presents task-wise accuracy under different observation window sizes, where stars (★) mark the highest accuracy achieved for each task. The results clearly indicate that the optimal window size varies across tasks. For example, the Few Shot and Code Completion tasks achieved their best performance with a window size of 1024, while tasks such as Summarization, Retrieval, and QA exhibited stable accuracy at much larger windows. These findings demonstrate that applying a single fixed window size across all tasks is suboptimal.

3.2 Token Selection Patterns

To investigate the cause of these accuracy differences, we analyzed the distribution of selected tokens. Figure 3 shows the distribution of 128 selected tokens under different observation window sizes for Passage Retrieval and Summarization, with the input length fixed at 4096. The input sequence was partitioned into segments of 512 tokens for analysis.

Two distinct patterns emerged. First, token selection was strongly dependent on the observation window size. When the window size was 512, approximately 80% of the selected tokens fell within the most recent 512 tokens, while extending the window to 2048 shifted the same proportion of selections to the most recent 2048 tokens. This phenomenon arises from the property of self-attention whereby each token assigns the highest score to itself, and the effective selection range is therefore determined by the extent to which these self-alignment terms are included. As a result, small windows concentrate selection on recent tokens, whereas large windows distribute selections more evenly across the entire context. Second, differences across tasks were negligible. Comparing Figure 3a with Figure 3b revealed highly similar distributions, with no substantial deviations. This indicates that fixing the observation window size structurally predetermines the selection probability of each token position, regardless of the task.

These results suggest that the positions of important tokens may vary not only across tasks but also across individual sentences. Thus, dynamically adjusting the observation window to ensure that these critical tokens fall within the observed range becomes essential.

3.3 Key Observations

From the above analysis, we derive the following insights:

- Observation window size directly influences performance, and the optimal size varies by task.
- Smaller windows lead to a focus on recent tokens, while larger windows promote a more uniform coverage of the overall context.
- These patterns are governed primarily by the observation window itself, rather than task characteristics.
- Consequently, a fixed window size cannot be optimal for all tasks, and adaptive window adjustment is required.

These limitations highlight the constraints of existing fixed-window approaches. To address them, this paper proposes A2SF, which incorporates a forgetting factor. By progressively forgetting the contribution of past attention scores over time, A2SF enables a continuous adjustment of the effective window size and facilitates more flexible token selection tailored to task-specific characteristics.

4 A2SF

4.1 Forgetting Factor

The existing token pruning based on the accumulative Attention Score (A2S) can be defined as follows:

$$S = \mathrm{Softmax}\left(\frac{QK^{\mathsf{T}}}{\sqrt{d}} + M\right) \tag{1}$$

$$A_q = \sum_{k=n_{seq}-w_o}^{n_{seq}} S_{q,k} \tag{2}$$

$$\mathrm{Idx} = \mathrm{Top}_k(A) \tag{3}$$

$$\mathrm{KV\_Cache} = \big(\mathrm{Gather}(K, \mathrm{Idx}), \mathrm{Gather}(V, \mathrm{Idx})\big) \tag{4}$$

Algorithm 1 A2SF RL Training Pipeline

Input:
$P \in \mathbb{S}$ ▷ input prompt
$\mathcal{I}_{\text{target}}$ ▷ target token indices from attention maps
θ_π ▷ policy network parameters
Output: $\mathcal{R}$ ▷ reward

// Context-to-State Encoding
$\mathbf{s} \leftarrow \text{EncodeState}(P)$ ▷ Algorithm 2

// Policy Action Selection
$\alpha, \beta, V \leftarrow \pi_{\theta_\pi}(\mathbf{s})$
$f \sim \text{Beta}(\alpha, \beta)$ ▷ sample forgetting factor
$f \leftarrow \text{Clamp}(f, \epsilon, 1-\epsilon)$

// A2SF Inference with Forgetting Factor
$\mathcal{I}_{\text{selected}} \leftarrow \text{ForwardWithA2SF}(P, f, Model_{target})$

// Policy Update (PPO)
$\mathcal{R} \leftarrow \text{Jaccard}(\mathcal{I}_{\text{selected}}, \mathcal{I}_{\text{target}})$
$\text{Advantage} \leftarrow \mathcal{R} - V$
$\text{Ratio} \leftarrow \frac{\pi_{\theta_\pi}(f|\mathbf{s})}{\pi_{\theta_{\text{old}}}(f|\mathbf{s})}$
$\mathcal{L}_{\text{policy}} \leftarrow -\min(\text{Ratio} \cdot \text{Advantage}, \text{Clamp}(\text{Ratio}, 1-\epsilon, 1+\epsilon) \cdot \text{Advantage})$
$\mathcal{L}_{\text{value}} \leftarrow \text{MSE}(V, \mathcal{R})$
$\mathcal{L}_{\text{entropy}} \leftarrow \mathcal{H}[\pi_{\theta_\pi}(f|\mathbf{s})]$ ▷ entropy loss for exploration
$\theta_\pi \leftarrow \theta_\pi - \eta \nabla_{\theta_\pi}(\mathcal{L}_{\text{policy}} + \lambda_{\text{value}}\mathcal{L}_{\text{value}} + \lambda_{\text{entropy}}\mathcal{L}_{\text{entropy}})$
return $\mathcal{R}$

For clarity, we omit layer and head indices. In actual implementation, the computation is executed at the head level, where each head maintains its own KV cache. M denotes the causal mask used in Transformer decoders, k is the capacity of the KV cache (i.e., the number of tokens preserved), and n_{seq} represents the input sequence length. The parameter w_o indicates the observation window size: $w_o = 1$ corresponds to TOVA, $w_o = 16$ to SnapKV, and $w_o = n_{\text{seq}}$ to H2O.

In this work, we extend Equation (2) as follows:

$$A_q = \sum_{q=0}^{n_{seq}} f^{n_{seq}-q} S_{q,k} \tag{5}$$

Here, $0 \leq f \leq 1$ is the newly introduced forgetting factor. This factor applies an exponential decay to attention scores as a function of their distance from the current step. In other words, tokens closer to $q = 0$ (earlier in the sequence) receive lager weights $f^{n_{\text{seq}}}$, reducing their contribution to near zero. With sufficiently small $f^{n_{\text{seq}}}$, older tokens are treated as if they fall outside the observation window. When $f = 0$, the model ignores the history entirely, which is equivalent to TOVA, whereas when $f = 1$, it considers the entire history, which corresponds to H2O.

By applying the forgetting factor in this way, the effective observation window emerges naturally depending on the value of f. Adjusting f allows the model to flexibly set the optimal observation range according to input, without relying on a fixed window size.

4.2 Search Method

To identify an appropriate forgetting factor for a given input prompt, we formulate the problem within a reinforcement learning (RL). First, the prompt is divided into fixed-length sentence units, which are then embedded using a Sentence Transformer. Next, we compute cosine similarities between the final sentence and all preceding sentences typically containing the question of the model's

response, thereby extracting a semantic distribution across the input. This distribution is used as the state representation for the RL agent. Based on this state, the agent outputs a probability distribution over the forgetting factor.

Since the forgetting factor is a continuous value between 0 and 1, the policy network is designed to support continuous action sampling by combining Proximal Policy Optimization (PPO) with a Beta distribution.

During training, the agent receives rewards based on the Jaccard similarity between the token indices selected under the sampled forgetting factor and those indices that actually achieved high attention scores during generation. Through this feedback, the agent learns to favor forgetting factors that preserve important tokens, ultimately acquiring task-adaptive token selection strategies. The overall RL pipeline and training procedure are summarized in Algorithm 1 and Algorithm 2.

4.3 Overhead of Additional Components

The components required for A2SF(Sentence Transformer and the RL model) introduce additional computation and memory overhead. The all-MiniLM-L6-v2 Wang et al. (2020) model used in this work is a lightweight BERT-based architecture with six layers and a hidden size of 384, occupying approximately 45.5 MB in FP16. As a result, it imposes minimal computational cost and can run efficiently even on a CPU. Similarly, the RL model consists of two fully connected layers with an internal dimension of 512 and two head layer for action, which also operates smoothly on a CPU without issue. Consequently, these auxiliary models do not place additional burden on GPU memory, and their impact on the overall system performance remains negligible.

4.4 Optimization of Prefill Computation

Methods such as H2O, which accumulate all attention scores, require completing attention computations over the entire sequence during the prefill stage. In contrast, approaches like SnapKV and TOVA can reduce prefill cost by pre-eviction: attention is precomputed only for queries within observation range, after which the corresponding keys and values are compressed and used in the main attention process. A similar optimization can be applied to A2SF. Because A2SF repeatedly multiplies by a fractional forgetting factor between 0 and 1, the values decay exponentially and eventually become negligible beyond a certain depth. By introducing a predefined threshold, we can determine an effective window size beyond which token contributions can be safely ignored. Applying this principle enables A2SF to reduce prefill computation efficiently, in a manner analogous to SnapKV.

$$f^n < T, \quad n > \frac{T}{\ln(f)} \qquad (T \approx e^{-20}) \tag{6}$$

$$A_q = \sum_{q=n}^{n_{\text{seq}}} f^{\,n_{\text{seq}}-q}\, S_{q,k} \tag{7}$$

5 Evaluation

5.1 Evaluation Environment

The performance of A2SF was evaluated using the LLaMA3 8B model on the LongBench benchmark and the NIAH (Needle In A Haystack) benchmark. The maximum input sequence length was set to 7,500 tokens, and the token budget at each step was fixed at 128. For transforming prompts into state vectors, the sentence-level token length was set to 64. The training dataset was constructed by randomly sampling 10 sentences per task from the full 3,750 sentences in LongBench, resulting in a total of 160 sentences. During training, the weight for value loss was set to 0.5 and for entropy loss to 0.05. The state representations were generated using the all-MiniLM-L6-v2 Sentence Transformer on huggingface. All inference experiments were conducted on NVIDIA RTX 4090 (24GB) GPUs, and training was performed on a single NVIDIA A100 (80GB) GPU.

Algorithm 2 Context-to-State Feature Encoding

Input: $P \in \mathbb{S}$ (prompt), $w = 64$, $d = 32$
Output: $\mathbf{s} \in \mathbb{R}^d$ (state vector)

$\mathbf{T} \leftarrow \text{Tokenize}(P)$
$\mathbf{C} \leftarrow \text{SplitIntoChunks}(\mathbf{T}, w)$
$\mathbf{E} \leftarrow \text{SentenceTransformer}(\mathbf{C})$
$\mathbf{e}_{\text{last}} \leftarrow \mathbf{E}[-1]$
$\mathbf{s} \leftarrow [\,\text{CosineSimilarity}(\mathbf{E}[i], \mathbf{e}_{\text{last}})\,]_{i=1}^{|\mathbf{E}|}$
if $|\mathbf{s}| > d$ **then**
  $\mathbf{s} \leftarrow \mathbf{s}[-d:]$
else if $|\mathbf{s}| < d$ **then**
  $\mathbf{s} \leftarrow \text{Zeros}(d - |\mathbf{s}|) \,\|\, \mathbf{s}$
end if
return $\mathbf{s}$

Table 1: Accuracy results on the LongBench benchmark with LLaMA3 8B. The maximum input length is set to 7,500 tokens, and the token budget is fixed at 128. "SnapKV(1024)" denotes SnapKV with an observation window size of 1024.

Method	Code	Few Shot	Single QA	Multi QA	Summary	Passage	Overall
TOVA	40.52	45.60	6.51	5.68	19.08	9.00	20.60
SnapKV(16)	40.73	45.89	6.52	5.66	19.00	9.00	20.70
SnapKV(1024)	**55.02**	57.26	9.75	6.61	20.57	30.90	28.40
H2O	47.12	58.80	10.46	**8.01**	**21.92**	10.62	25.82
A2SF	51.03	**61.39**	**11.24**	7.63	20.41	**31.62**	**29.21**

Table 2: Accuracy results on the LongBench benchmark. The token budget is fixed at 512.

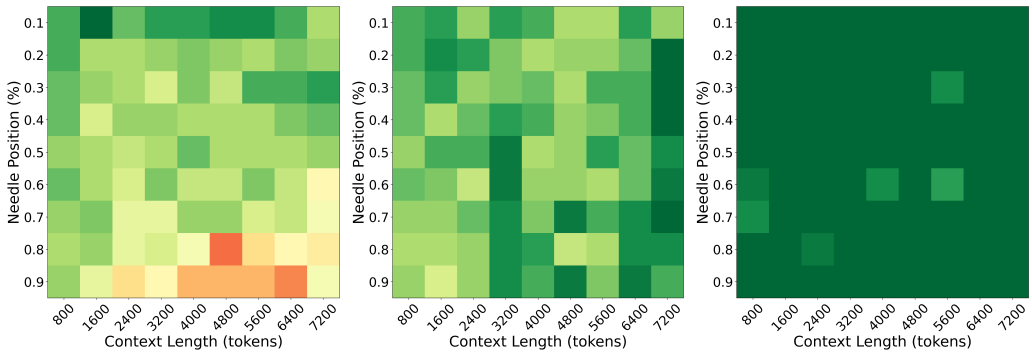
Method	Code	Few Shot	Single QA	Multi QA	Summary	Passage	Overall
TOVA	51.14	56.84	8.14	5.53	22.36	23.0	26.68
SnapKV(16)	51.24	56.95	8.25	5.52	22.33	23.0	26.73
SnapKV(1024)	**57.90**	63.58	11.82	7.32	22.92	35.12	31.43
H2O	56.20	**66.80**	11.89	**7.81**	**23.97**	23.97	30.62
A2SF	57.12	66.72	**12.08**	7.65	23.15	**35.41**	**31.76**

5.2 LongBench Results

Table 1 and 2 report a comparative evaluation of A2SF against existing methods on LongBench. Overall, A2SF consistently surpasses baseline approaches, demonstrating robust performance across all task categories. These results indicate that A2SF effectively retains essential contextual information while eliminating redundant tokens, thereby sustaining reliable performance even under constrained KV cache budgets.

At the task level, A2SF exhibits pronounced advantages in retrieval-oriented settings, where striking an appropriate balance between recency and long-range context is crucial. A2SF employs the forgetting factor to adaptively adjust the contextual range, effectively capturing important sentences irrespective of their position and ensuring stable accuracy. In summarization tasks, although another method attained the highest score, A2SF remained competitive, suggesting that its adaptive mechanism does not compromise performance in generation-oriented scenarios.

Collectively, these findings demonstrate that A2SF, in contrast to fixed-window or heuristic-based approaches, provides a more balanced and reliable solution across a wide spectrum of tasks.



(a) H2O (b) SnapKV(1024) (c) A2SF

Figure 4: Results on the NIAH benchmark. The x-axis denotes the length of the input sentence, while the y-axis represents the relative position of the needle within the sentence. This demonstrate the model's capability to capture sentence-level understanding.

5.3 Needle In A Haystack

Figure 4 compares the performance of H2O, SnapKV(1024), and A2SF on the NIAH (Needle-in-a-Haystack) benchmark. NIAH evaluates a model's ability to accurately retrieve a specific sentence within a long context, directly reflecting how well the model can discard irrelevant tokens while preserving essential information.

As shown in Figure 4a, H2O exhibits generally low accuracy with a clear degradation in performance, highlighting the limitations of a fixed heuristic-based approach that cannot adapt to varying contextual conditions. SnapKV (Figure 4b) delivers more stable results than H2O, but its performance remains inconsistent.

In contrast, A2SF (Figure 4c) achieves the highest overall accuracy and maintains relatively uniform performance regardless of context length or the position of the target sentence. By dynamically adjusting the effective context range through the forgetting factor, A2SF effectively avoids two common pitfalls: missing important information with overly short windows and incorporating excessive noise with overly long windows.

Overall, A2SF demonstrates the most stable and robust performance on the NIAH benchmark, clearly validating the strength of its adaptive token selection mechanism in selectively preserving critical information across diverse contextual settings.

6 Conclusion

We identified the limitations of conventional accumulative attention–based token selection methods that rely on fixed observation windows. To address this limitation, we introduced A2SF which incorporates a forgetting factor. A2SF apply a accumulative formulation that progressively forgets the contribution of past attention scores over time. Furthermore, by employing reinforcement learning with a PPO–Beta policy, it dynamically searches for forgetting factors suited to the characteristics of the input context.

Experimental results on the LongBench and NIAH benchmarks demonstrate that A2SF maintains stable accuracy across a wide range of tasks and context lengths, achieving up to a 9% performance improvement over existing methods. Notably, even under strict KV cache budgets, A2SF effectively preserves critical tokens, reducing memory usage while minimizing accuracy degradation. Moreover, the proposed approach can be combined with techniques such as dynamic budget allocation across layers or heads, as well as token offloading and compression strategies, offering further potential for accuracy gains.

7 Reproducibility Statement

Sections 4.2 and 4.3 describe the RL training procedure and model architecture in detail. In addition, Section 5.1 presents the datasets, models, and experimental settings used for performance evaluation. We also specify all datasets employed for training as well as the hardware configurations used.

References

Yushi Bai, Xin Lv, Jiajie Zhang, Hongchang Lyu, Jiankai Tang, Zhidian Huang, Zhengxiao Du, Xiao Liu, Aohan Zeng, Lei Hou, et al. Longbench: A bilingual, multitask benchmark for long context understanding. *arXiv preprint arXiv:2308.14508*, 2023.

Zefan Cai, Yichi Zhang, Bofei Gao, Yuliang Liu, Tianyu Liu, Keming Lu, Wayne Xiong, Yue Dong, Baobao Chang, Junjie Hu, et al. Pyramidkv: Dynamic kv cache compression based on pyramidal information funneling. *arXiv preprint arXiv:2406.02069*, 2024.

Harry Dong, Xinyu Yang, Zhenyu Zhang, Zhangyang Wang, Yuejie Chi, and Beidi Chen. Get more with less: Synthesizing recurrence with kv cache compression for efficient llm inference. *arXiv preprint arXiv:2402.09398*, 2024.

Yuan Feng, Junlin Lv, Yukun Cao, Xike Xie, and S Kevin Zhou. Ada-kv: Optimizing kv cache eviction by adaptive budget allocation for efficient llm inference. *arXiv preprint arXiv:2407.11550*, 2024.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, et al. The llama 3 herd of models. *arXiv e-prints*, pp. arXiv–2407, 2024.

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

Da Ma, Lu Chen, Situo Zhang, Yuxun Miao, Su Zhu, Zhi Chen, Hongshen Xu, Hanqi Li, Shuai Fan, Lei Pan, et al. Compressing kv cache for long-context llm inference with inter-layer attention similarity. *arXiv preprint arXiv:2412.02252*, 2024.

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

Roni Paiss, Hila Chefer, and Lior Wolf. No token left behind: Explainability-aided image classification and generation. In *European Conference on Computer Vision*, pp. 334–350. Springer, 2022.

Irving GB Petrazzini and Eric A Antonelo. Proximal policy optimization with continuous bounded action space via the beta distribution. In *2021 IEEE symposium series on computational intelligence (SSCI)*, pp. 1–8. IEEE, 2021.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

Ying Sheng, Lianmin Zheng, Binhang Yuan, Zhuohan Li, Max Ryabinin, Beidi Chen, Percy Liang, Christopher Ré, Ion Stoica, and Ce Zhang. Flexgen: High-throughput generative inference of large language models with a single gpu. In *International Conference on Machine Learning*, pp. 31094–31116. PMLR, 2023.

Wenhui Wang, Furu Wei, Li Dong, Hangbo Bao, Nan Yang, and Ming Zhou. Minilm: Deep self-attention distillation for task-agnostic compression of pre-trained transformers. *Advances in neural information processing systems*, 33:5776–5788, 2020.

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.

Jiayi Yao, Hanchen Li, Yuhan Liu, Siddhant Ray, Yihua Cheng, Qizheng Zhang, Kuntai Du, Shan Lu, and Junchen Jiang. Cacheblend: Fast large language model serving for rag with cached knowledge fusion. In *Proceedings of the Twentieth European Conference on Computer Systems*, pp. 94–109, 2025.

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

A THE USE OF LARGE LANGUAGE MODELS (LLMS)

No but not all. We only used LLMs to polish som sentences.