RECYCLED ATTENTION: EFFICIENT INFERENCE FOR LONG-CONTEXT LANGUAGE MODELS

Anonymous authors
Paper under double-blind review

ABSTRACT

Processing long-context input imposes a heavy computational burden when deploying large language models. Recently proposed inference-time methods accelerate generation by attending only to local context. Despite its efficiency gains, this approach fails to capture all relevant information in the input, showing substantial performance drop in long-context benchmarks. We propose *recycled attention*, an efficient and effective method which alternates between full context attention and attention over a subset of input tokens. When performing partial attention, we leverage the attention pattern of a nearby token that has performed full attention and attend only to the top K most attended tokens. We evaluate our methods on RULER, a suite of tasks designed to comprehensively evaluate long-context abilities, and long-context language modeling tasks. Applying our inference method to off-the-shelf LLMs achieves comparable speedup to baselines which only consider local context while improving the performance by 2x. We further experiment with continued pre-training the model with recycled attention to improve the performance-efficiency trade-off.

1 INTRODUCTION

Recent large language models (LLMs) are trained to ingest extremely long inputs and generate long outputs (Meta, 2024; Gemini, 2024) to support a wide range of applications. However, deploying such long-context LLMs can be very costly. As the context length increases, LLMs see a linear increase in memory to store the Key-Value (KV) cache and a quadratic increase in time for attention computation. These two factors lead to high latency during inference; Adnan et al. (2024) showed that as context length increased 16x for the MPT-7B model (MosaicML, 2023), the inference latency increased by 50x, where 40% of the increase was due to the data movement of the KV cache.

To improve efficiency, prior work put a limitation on the size of KV cache, i.e. the number of past tokens that are available at each generation step. This leads to a meaningful gain along two axes: memory requirement and time for attention computation. To form a smaller KV cache, they make a locality assumption, only keeping most recent input tokens (Beltagy et al., 2020; Child et al., 2019) along with a fixed number of globally available initial tokens (i.e., StreamingLLM (Xiao et al., 2023)). Another line of work (e.g., H_2O (Zhang et al., 2024), Keyformer (Adnan et al., 2024)) maintains a dynamically constructed fixed sized KV cache by identifying *key* past tokens from observed attention patterns and dynamically evicting the rest during generation.

These approaches reported little degradation in perplexity-based evaluation for the next token prediction task. However, they show a significant drop in performance (Sun et al., 2024) on long-context benchmarks that require synthesizing information from non-local contexts (Hsieh et al., 2024). For example, on the simple needle-in-a-haystack (NIAH) task, both StreamingLLM (Xiao et al., 2023) and H_2O (Zhang et al., 2024) report less than 8% accuracy compared to 100% for vanilla attention. Keeping a smaller KV cache is problematic when LLMs is tasked with synthesizing information from long context, going beyond next token prediction where local contexts suffice. Once a key input token is eliminated from the KV cache (either through locality assumption or by eviction during the generation process), there is no way to recover access to the eliminated token. When LLMs are tasked to generate long text, it gets harder to predict which input tokens are useful in advance.

In this work, we propose a novel approach, Recycled Attention, that focuses on reducing inference time while comprehensively capturing long-context inputs. We keep the full KV cache throughout

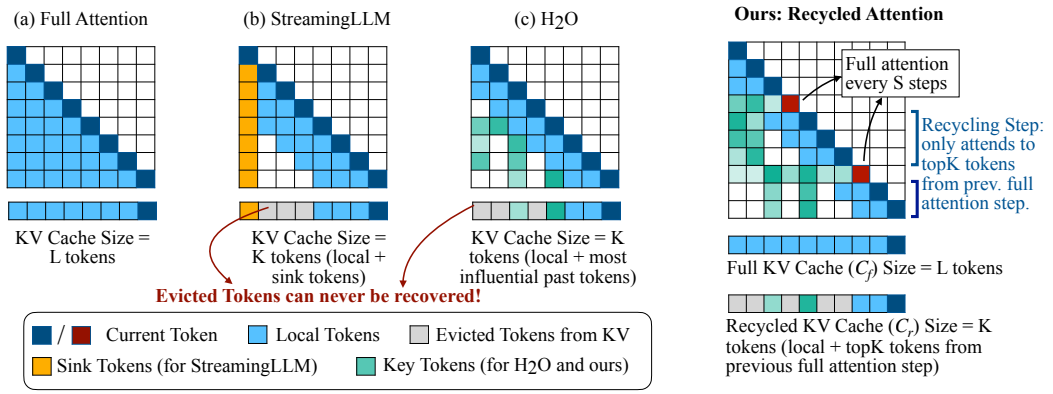

Figure 1: Illustration of our Recycled Attention method (right) compared to baselines (left). Our approach alternates between full attention steps (i.e. over all past tokens) and *recycled* attention steps (i.e. over a reduced KV cache of *key* tokens) during generation. By restricting the full attention computation to once in every S steps, Recycled Attention is able to achieve comparable speedups to baseline models with smaller KV without degrading performance on long-context benchmarks.

the inference (thus no gain in memory footprint), but perform attention over a dynamically constructed smaller KV cache, retaining gain in inference speedups. Our method flexibly alternates between two modes of generation: generation that involves an attention over the full KV cache and generation that computes an attention over a subset of tokens (see Figure 1). We choose this subset of tokens by taking top K attended tokens from the most recent generation step involving attention over the full KV cache (thus the term *recycling* attention). In this work, we have a fixed strategy for alternation: full attention every S steps and recycled attention for the next $S-1$ steps. Our design choices is supported by the analysis that neighboring tokens during generation place high attention mass over a similar subset of past tokens. Our work (no KV eviction, dynamically constructed smaller KV) establishes a middle ground between full attention (no KV eviction, high latency, high performance) and sparse attention (KV eviction, reduced latency, low performance).

We evaluate our approach in language modeling task and RULER (Hsieh et al., 2024) benchmark, a suite of tasks designed to evaluate long-context models, as well as datasets from LongBench(Bai et al., 2023). Applying our inference method to two off-the-shelf LLMs (Meta, 2024; Yang et al., 2024a) achieves comparable speedup to prior work with limited KV cache while improving the performance on long-context benchmark by 2x. We further experiment to continued pre-training the model with recycled attention, bringing further gains. To summarize, our contributions are

- We propose *recycled attention*, an inference-time method to accelerate generation with long input.
- We comprehensively evaluate our methods to two long-context models and a suite of long-context tasks, including downstream tasks and language modelling tasks. We find that our method achieves up to 2x wall clock time speedup while preserving performance, especially on downstream tasks which require access to information throughout the input.
- We investigate further improvements: continued pre-training LLM with recycled attention and deciding when to perform full attention based on query similarity.

2 Recycled Attention for Long-Context LLMs

2.1 Problem Setting and Notation

Let M be a language model trained to estimate the conditional probability of all output sequences given an input x. At inference time, M generates an output $\hat{y} \sim M(x)$ in two steps: (1) **Pre-filling stage**: M ingests the input $x = x_1, \cdots x_L$ and stores the KV cache for all L tokens across all layers of the transformer model, and (2) **Generation stage**: generate one token y_i at a time from the conditional distribution $P_M(y_i|x, y_1 \cdots y_{i-1})$. At each step, the model attends to the KV cache of all previous tokens, and also updates the KV cache to include the current token's key-value pairs.

Our goal is to reduce the inference latency during this second stage of the generation process. There are two main factors that contribute to this increased latency; first, the attention computation increases quadratically with input length L. Second, a large L necessitates maintaining a large KV cache of past tokens, and 40% of the inference latency can be attributed to the data movement of this large KV cache from the GPU HBM (Adnan et al., 2024).

Prior approaches (Xiao et al., 2023; Zhang et al., 2024; Adnan et al., 2024) achieve inference time speed-ups by limiting the size of the KV cache to a fixed size K. StreamingLLM (Xiao et al., 2023) constructs this fixed size KV cache by retaining only the initial and recent tokens (illustrated in Figure 1(b)) while H_2O (Zhang et al., 2024) retain a mix of local tokens and *key* past tokens dynamically identified during generation (see Figure 1(c)). For both approaches, once tokens are evicted from the KV cache, they cannot be recovered in subsequent generation steps. This can be particularly catastrophic in long-context scenarios where key tokens are challenging to identify in advance, e.g. cases where instructions inquiring about the past tokens are located towards the end of the input. Consequently, methods that evict tokens from the KV cache often report poor performance on benchmarks like RULER (Hsieh et al., 2024) that are designed to test reasoning and information synthesis capabilities over long-contexts.

Instead of permanently evicting tokens for all future steps, we ask: **can we distinguish between important and unimportant tokens for the attention computation for the next S time steps?** Our key hypothesis is that consecutive tokens in a sequence likely place the majority of the attention weights over a similar subset of tokens in the context, and this can be leveraged to increased inference efficiency. We test this hypothesis for the LLaMA-3.1-8B model in the subsection below.

2.2 Attention Mass Overlap between Neighboring Tokens

Setting: We randomly sample five examples from the Arxiv split of the RedPajama dataset (Together, 2023) and compute the attention weights over past tokens for all layers and all time steps. Next, for time step $t = 8K$, we identify the topK($= 1024$) past tokens based on attention weights independently for each layer. Then, for subsequent attention computations for tokens at times steps $t = 8K + i$, varying i from 1 to 10, we compute the fraction of the attention mass placed on $t = 8K$'s topK tokens. Figure 2 shows this attention recovery rate for different step i from the full attention step, averaged across all layers of the transformer model (shown in blue). The graph clearly demonstrates that the topK tokens at $t = 8K$ include the past tokens that contribute, on average, more than 90% of the attention weights at subsequent times $t = 8K + i$. Based on this observation, our key idea is to alternate between full attention over the entire KV cache of past tokens every S steps, and a more time-efficient attention over only K tokens for the next $S - 1$ steps, where these K tokens are selected to be the highest weighted tokens during the previous full attention step. We call this strategy Recycled Attention, as we recycle the topK tokens from a previous time step in lieu of full attention.

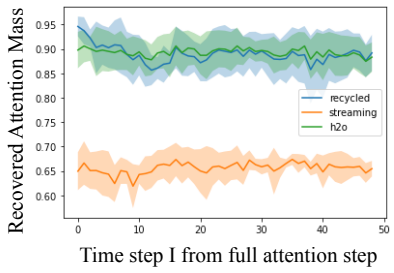

Figure 2: Fraction of the total attention mass recovered at time $t = T + i$ by the topK past tokens in the KV cache, where these topK tokens are selected based on attention scores at $t = T$. Compared to StreamingLLM, topK tokens recover a larger fraction of the total attention mass.

Note that the graph in Figure 2 also reports the fraction of the full attention weight placed on tokens corresponding to StreamingLLM's cache of similar size K (shown in orange), comprising of the initial "sink" tokens and the local tokens (see Figure 1b for KV construction strategy). Compared to our proposed strategy, StreamingLLM reports a much lower attention mass recovery rate (~ 0.65 compared to 0.9 for our approach) and is consequently worse at approximating full attention.

2.3 Methodology and Implementation

Given a language model M and a sequence of input tokens $x_1, ..., x_L$, we present the pseudocode for generating the output sequence of tokens using Recycled Attention in Figure 3.

Algorithm 1: Generation using Recycled Attention

Input: Language model M, input $x_1, \ldots, x_L$
Hyperparameters: recycle cache size K, stride S
Output: Sequence of generated tokens O

Initialization

1. Prefill M with $x_1, \ldots, x_L$
   1. Initialize full KV cache C_f
   2. Let $a_L \leftarrow$ attention scores for $x_L$over past tokens for all layers.
   3. Initialize recycle KV cache $C_r \leftarrow$ reverse(arg top-k$_{x \in C_f}$ $\mathbf{a_L}$)
2. $O \leftarrow []$ // Initialize empty output sequence

Generation

3. **for** $i \in 1,2,\cdots, T$ **do**
4. **if** $(i \mod S \neq 0)$ **then** // **Recycle Steps**
5. $o \sim M(C_r)$, O.append(o) // Generate using recycle KV
6. $C_r = C_r[1:] + \text{KV}(L+i)$ // Update recycle KV C_r
7. **else if** $(i \mod S = 0)$ **then** // **Full Attention Step**
8. $C_f \leftarrow [C_f; C_r[-S:]]$ // Update C_f with *new* tokens in C_r
9. $o, \mathbf{a} \leftarrow M(C_f)$, O.append(o) // Generate using full attention
10. $C_r \leftarrow$ reverse(arg top-k$_{x \in C_f}$ $\mathbf{a}$)// Re-initialize C_r with topK tokens
11. **return** O

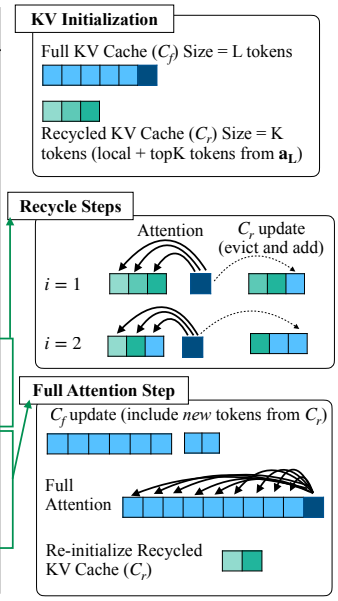

Figure 3: Pseudocode for Recycled Attention. We use $M(C)$ to denote performing a forward pass with the language model while computing attention over the key-value pairs in cache C.

Our approach maintains two separate KV caches C_f and C_r (size $\propto K$), corresponding to the full and recycle attention steps respectively. Given input $x_1, ..., x_L$, we first prefill M using the vanilla full attention computation and initialize our full KV cache C_f with the first L tokens. We also obtain the attention scores $\mathbf{a_L}$ for the last token x_L at each layer of model M. We initialize our recycle KV cache C_r with the key-value pairs of the topK tokens at each layer based on attention scores.

At each recycle steps, i.e. $S-1$ contiguous steps after every full attention step, we generate the next token $y_t \sim M(C_r)$ using the smaller recycled KV cache C_r to compute attention. This leads to a reduction in both the attention computation FLOPs as well as the latency due to movement of KV cache (we only move the smaller KV C_r instead of the larger full KV C_f, where $|C_r| << |C_f|$). As C_r is updated with the KV cache of the new input tokens (i.e. the generated token from previous step) in the forward pass, we remove the recycled token which received the lowest attention score from C_r to maintain a fixed size.

At each full attention step that occurs every S steps, we first update the C_f KV cache with the key-value pairs of the $S-1$ tokens from the recycle step. Next, we generate the next token $y_t \sim M(C_f)$ using the full KV cache C_f. Finally, we follow the same procedure as above to reset the recycle cache C_r with the topK tokens from each layer of the current time step.

Compatibility with Flash Attention FlashAttention (Dao, 2024) improves standard attention computation on GPU by reducing data movement, significantly improving the memory and speed efficiency. It achieves this by directly producing the output for the attention blocks, without storing the $O(L^2)$ attention matrix. However, we rely on these attention scores to select the topK tokens during the full attention steps and construct our recycled KV cache C_r (lines 9-10 of Algorithm 3). To make our method compatible with FlashAttention, we implement an extra step to re-compute the attention score when we perform full attention. As we only perform full attention at stride of S, this does not introduce significant overhead. Additionally, note that other methods that use attention patterns (e.g. H_2O) will also show reduced speed-gain when using in conjunction with FlashAttention.

Memory and time requirements Table 1 shows a comparison of the memory and attention compute requirements for our Recycle Attention method and baseline approaches. Recycle Attention uses a similarly sized KV cache memory compared to vanilla attention ($L+K$ vs L, where $K << L$) but larger than efficient KV strategies like StreamingLLM and H_2O . Our method substantially re-

Table 1: Comparison of the time and memory requirements of Recycle Attention and baseline approaches. Suppose an LLM ingests a sequence of L input tokens and generates T output tokens. We report the memory requirement for storing the input KV cache and time required to generate the T output tokens. Let K denote the size of the reduced KV cache of baseline non-vanilla approaches. We use the same size K for our recycle KV cache in Recycle Attention and use S to denote the stride. Additionally, we report NIAH performance measured for the Llama-3.1-8B model, with $K = 4096$ and $L = 32,768$ for all approaches.

	Vanilla	H_2O	StreamingLLM	SnapKV	Recycled Attention (Ours)
Memory	L	K	K	K	$L + K$
Time	$T \times L$	$T \times K$	$T \times K$	$T \times (K + T)$	$T \times \frac{L}{S} + T \times K$
NIAH Accuracy	100	8	7	77	98

duces the time for the attention computation compared to vanilla attention by setting the recycle KV size $K << L$. Our strategy also allows us to remain performant on tasks such as NIAH compared to vanilla attention, in contrast with other KV eviction strategies. We provide a detailed comparison of wallclock times and performances on various tasks against baslines in Section 3.

3 Experimental Settings

We evaluate our method on two long-context language models **Llama-3.1-8b** (Meta, 2024) and **Qwen-2-7b** (Yang et al., 2024a). **Llama-3.1** is pre-trained with 8K tokens, followed by a continued pre-training stage to increase the context window to 128K. **Qwen-2** is continued pre-trained with up to 32k tokens, and adopted YARN (Peng et al., 2024) and Dual Chunk Attention (An et al., 2024) to enable processing of up to 128k tokens. As both models employ Grouped Query Attention (Ainslie et al., 2023), we use a single aggregated attention score for all query heads (max over all query heads) in the same group to identify the top K tokens.[1]

3.1 Tasks

We evaluate our approach on language modeling and a suite of downstream proxy tasks for long-context evaluation (Hsieh et al., 2024). For both tasks, report the task performance and inference speed measured by wall clock time.

Downstream tasks We test our method on RULER (Hsieh et al., 2024), a suite of tasks designed to evaluate long-context models. It includes tasks that require retrieval capabilities (e.g. Needle-in-a-Haystack) as well as those that require aggregating information over the long context. We follow Hsieh et al. (2024) and evaluate our methods on 13 tasks from four categories of RULER. We evaluate on context length of 32K and 64K, with 100 examples for each {task, context length}. We additionally report on two tasks from LongBench in Section A.2 in the Appendix.

Language Modeling We evaluate language modeling perplexity on the Arxiv and Book split of RedPajama (Together, 2023), and PG19 (Rae et al., 2019). We evaluate on 16k and 100k context size for the two respectively. We report results on 100 sequences for each domain. Following prior work (Yen et al., 2024b), we report the perplexity on the last 256 tokens of each sequence.

3.2 Baselines

We compare Recycle Attention against the following baselines: (1) **Vanilla** attention baseline which uses the entire KV cache to generate tokens. (2) **StreamingLLM** (Xiao et al., 2023) inferences by attending to a KV cache consisting of "sink tokens" and recent tokens, discarding all other tokens. Following previous work, we maintain a cache with 4 sink tokens and K - 4 recent tokens. (3)

[1]Our ablations show that taking the max outperforms other aggregation method such as mean, or relying solely on one of the query head in the group. We detail this more in Table 7 in the Appendix.

Table 2: Performance on the RULER benchmark for LLama-3.1-8B and Qwen-2-7B. The results show that Recycled Attention achieves a comparable speedup to prior approaches, while substantially outperforming them based on accuracy across all settings.

Method	stride	K	LLama-3.1 32K Acc ↑	LLama-3.1 32K time(s) ↓	LLama-3.1 64K Acc ↑	LLama-3.1 64K time(s) ↓	QWEN-2 32K Acc ↑	QWEN-2 32K time(s) ↓	QWEN-2 64K Acc ↑	QWEN-2 64K time(s) ↓
Vanilla	-	-	90	1.71	82	2.40	79	2.55	57	4.93
H_2O	-	4096	21	2.15	11	2.29	16	1.94	11	1.94
StreamingLLM	-	4096	22	**1.23**	17	**1.21**	21	**1.17**	11	**1.19**
StreamingLLM++	50	4096	22	1.25	17	1.33	21	1.21	11	1.29
SnapKV (kernel=7, w=32)	-	4096	72	1.64	62	1.73	57	1.43	**31**	1.60
Recycled	50	4096	63	1.27	50	1.29	32	1.21	20	1.20
Recycled (kernel=7)	50	4096	**79**	1.26	**65**	1.29	**58**	1.20	**31**	1.20

StreamingLLM++: we also implement a modified version of StreamingLLM that is equivalent to our Recycled Attention method in terms of both computation and memory requirements. Similar to our approach, StreamingLLM++ performs full attention at a stride S, i.e. every S steps, to match the attention operations of Recycle Attention. (4) **H_2O** (Zhang et al., 2023) maintains a KV cache which contains recent tokens and "heavy hitters", defined by high cumulative attention scores. We set the heavy hitter size and recent cache size to be $K/2$. (5) **SnapKV** (Li et al., 2024) considers the average attention scores of the last few tokens ("observation window") in the prompt to decide the KV cache to keep. It further applies max pooling over consecutive tokens' attention score, instead of relying on the token's attention score, to decide token to keep. We set the observation window size to 32 and kernel size to 7 following Li et al. (2024).[2]

3.3 Inference settings

We prefill the model with the input and measure wall clock time for the generation phase for each of the method. We generate 50 tokens for the RULER tasks and 256 tokens for the language modeling task. We perform our experiments on 1 A100 80GB GPU with Flash Attention (Dao, 2024). We report a fixed set of K and S for the tasks, and perform ablation study on varying these two hyperparameters in Section A.1 in the Appendix. We include a variant of Recycled Attention with max pooling with kernel size=7 to cluster KV cache, same as SnapKV.

4 Results

RULER For this set of experiments, we set $K = 4096$ for all baseline models, where applicable. For Recycled attention and Streaming++, we set stride $S = 50$.

Aggregate accuracy results and the generation time per example for the RULER benchmark are reported in Table 2. All methods aiming to achieve inference speedup by evicting tokens from the KV cache permanently (e.g. StreamingLLM, H_2O) shows substantial degradation. **Recycled attention significantly outperforms other non-vanilla approaches by over 2x in terms of aggregate accuracy**. However, we note that the performance degrades substantially compared to vanilla attention. In terms of speed-up, our method achieves similar speedup to StreamingLLM/StreamingLLM+, followed by H_2O model. As input context length increases, inference time for vanilla method scales linearly, while the other methods' inference time remain at the same ballpark with a fixed K.

Note that H_2O relies on calculating attention scores at each time step to identify the "heavy hitter" tokens which FlashAttention does not store. Thus, the inference speed-up is not as significant in certain setting (with 1/8 KV cache size for 32K input) when used with Flash Attention. For our Recycled Attention approach, we only explicitly re-compute the attention score every S steps, which do not introduce as heavy an overhead.

[2] Our proposed method of constructing the top K cache is equivalent to SnapKV with a window size of 1 and kernel size of 1. While it is not feasible to apply a window size greater than 1 for our method as it will require access to attention scores at *each* decoding step, it is possible to apply the kernel method to our approach.

Table 3: Per-task performance of Llama-3.1-8B on RULER subtasks. For non-vanilla methods, we set the $K = 4096$.

Method	niah_single	multi_key	multi_query	multi_value	fwe	vt	cwe	qa
Context size: 32K								
Vanilla	100	98	99	99	93	99	65	61
H_2O	7	7	6	6	78	38	39	34
Streaming	8	13	13	13	93	12	4	42
StreamingLLM++	8	14	13	13	93	18	8	50
SnapKV (kernel=7, w=32)	77	68	99	98	83	99	56	61
Recycled	98	35	59	37	90	99	20	59
Recycled (kernel=7)	99	60	98	99	83	99	44	63
Context size: 64K								
Vanilla	100	90	96	99	91	98	3	54
H_2O	3	2	2	4	52	3	8	20
Streaming	8	7	7	8	90	5	0	33
StreamingLLM++	8	7	7	8	90	5	0	35
SnapKV (kernel=7, w=32)	74	42	90	88	71	92	5	48
Recycled	80	30	26	17	79	95	3	51
Recycled (kernel=7)	96	41	85	84	72	93	4	49

As RULER consists of a diverse range of task, we report per-task fine-grained performance for Llama-3.1 in Table 3. Recycled attention performs the best at tasks that require retrieving a piece of information in the context (including Needle-in-a-haystack (NIAH), Question Answering (QA) and Variable Tracking (VT)). H_2O and StreamingLLM suffers at these tasks as the information falls out of the KV cache. However, recycled attention does not perform well for task that requires aggregating the information in the context, such as frequent word extraction (fwe), lagging behind H_2O and StreamingLLM. Performances for tasks which requires retrieving for multiple pieces of information (multiple keys or multiple queries) are worse compared to the task with (single key, single value) when using the same K. We later show in ablation study in Section A.1 increasing the size of K leads to improvements in such tasks.

Language Modeling For context size 16K, we fix $K = 2048$ and $S = 10$ both LLaMA-3.1 and QWEN-2. For context size $100K$, we report results using $K = 2048$ and $32,768$, and $S = 256$.

Table 4 outlines the perplexity-based performance of the baselines and recycled attention approach. For LLaMA-3.1, recycled attention achieves better perplexity and comparable inference speeds compared to StreamingLLM when the KV cache size is 1/8 of a 16K context. This shows that the model benefits from attending to tokens outside of local context window. We observe that H_2O outperforms our approach by a small margin, but at the cost of a substantially higher inference time per example (10.77 for H_2O vs 6.05 for our method). Overall, **our recycled attention approach achieves a better trade-off between inference speeds and task accuracy compared to non-vanilla approaches for both LLaMA-3.1 and QWEN-2 models** for the 16K context size setting.

When we scale up the context length to 100K, we find differing trends between the LLaMA-3.1 and QWEN-2 models. For LLaMA-3.1, we observe that recycled attention reports better perplexity but worse inference speeds compared to non-vanilla baseline methods. However, baseline approaches outperform recycle attention for the QWEN-2 model. We analyze the attention pattern to investigate this in Section A.1.

5 Continued pre-training with recycled attention

So far, we use the off-the-shelf LLMs as is, only modifying the inference method. This creates a discrepancy between model training and inference assuming LLM is trained with vanilla full attention setting. In this section, we experiment with continued pre-training the model with recycled

Table 4: Perplexity results on language modeling task for LLama-3.1-8B and QWEN-2-7B. We report performances for Arxiv (the first number) and Book (the second number) and PG19.

Method	K	Stride	LLama-3.1-8B time(s) ↓	LLama-3.1-8B PPL ↓	QWEN-2-7B time(s) ↓	QWEN-2-7B PPL ↓
Context size: 16 K (Arxiv and Book)						
Vanilla	-	-	7.63	2.22 / 7.07	8.85	2.33 / 8.26
H_2O	2048	-	10.77	2.48 / 7.60	11.57	2.68 / 9.02
StreamingLLM	2048	-	**6.92**	2.62 / 7.94	**5.71**	2.75 / 9.10
StreamingLLM++	2048	10	7.21	2.59 / 7.88	6.08	2.71 / 9.05
SnapKV (kernel=7, w=32)	2048	-	7.77	2.48 / 7.68	6.90	2.65 / 8.97
Recycled	2048	10	7.14	2.36 / 7.49	6.33	2.57 / 9.01
Recycled (kernel=7)	2048	10	7.14	**2.32 / 7.40**	6.33	**2.47 / 8.73**
Context size: 100 K (PG19)						
Vanilla	-	-	18.11	8.24	40.42	13.28
H_2O	2048	-	10.56	17.04	9.96	**13.39**
StreamingLLM	2048	-	**5.94**	9.53	**5.72**	13.58
StreamingLLM++	2048	256	6.04	9.53	5.92	13.58
Recycled	2048	256	6.10	**9.31**	5.90	14.90
H_2O	32,768	-	26.89	8.63	23.55	13.36
StreamingLLM	32,768	-	**13.38**	8.55	**15.81**	**12.31**
StreamingLLM++	32,768	256	13.43	8.55	15.87	12.32
Recycled	32,768	256	13.52	8.46	15.89	13.50

attention, with the goal of teaching the models to adapt to attending over discontinuous tokens in the recycled cache.

Data We sample 200k data from the Arxiv split of RedPajama dataset[3] and filter out sequences with less than 8192 tokens. We split the data into 80%, 10% and 10% train/dev/test splits, resulting in 120k training data.

Training We train the model to adapt to recycled attention when predicting a sequence with 8142 with a prefilling length of 8092, $K = 2048$ and a stride of 50. Concretely, for the last 50 tokens t_i in the sequence, attention is calculated over the 2048 tokens that received the highest attention score according to t_{8092}, as well as $\{t_{8093}, ... \ t_i\}$. For $\{t_0, ... \ t_{8092}\}$, attention is calculated with regular causal mask. We train the model with next token prediction loss for all the tokens in the sequence. We report implementation details in Section A.3 in the Appendix.

Comparison sytstems We compare fine-tuning with other inference methods (Vanilla, StreamingLLM, StreamingLLM++). For each method, we report the base performance from the pre-trained checkpoint (Base) and the performance after continued fine-tuning (+CPT).

Results We report the results of continued pre-training in Table 5 for both the language modelling and the 14 RULER tasks. We see that continued pre-training does not improve performance with vanilla inference method, likely as the model is highly optimized in this setting and trained with this data. We also observe very little performance gain through continued pre-training in other inference methods (StreamingLLM, StreamingLLM++). Yet, with Recycled Attention, we see a meaningful gain from continued pre-training in two stride setting (25, 50). Continued pre-training achieves a lower perplexity and higher accuracy with higher stride (50) compared to base model with a smaller stride (25), leading to a better performance-efficiency trade-off.

6 New section: Dynamic Stride

Our experiments in Section 3 employs a fixed schedule for all layers. In this section, we explore a dynamic scheduler to alternate between full and recycling attention steps. Intuitively, if the query vector of a particular layer and head for the current step is similar to the query vector of the most

[3] https://huggingface.co/datasets/togethercomputer/RedPajama-Data-1T

Table 5: Results on continued pre-training LLaMA-3.1. The context length is 8K and we decode non-vanilla methods with $K = 2048$. We report perplexity on the last 50 tokens.

Model	Method	Stride	Dev PPL	Test PPL	RULER Acc
Base	Vanilla	-	2.83	2.68	93
+ CPT	Vanilla	-	2.83	2.68	93
Base	Streaming	-	3.20	3.14	37
+ CPT	Streaming	-	3.19	3.14	37
Base	Streaming++	50	3.19	3.13	37
+ CPT	Streaming++	50	3.18	3.13	36
Base	Recycled	50	3.09	2.96	83
+ CPT	Recycled	50	3.01	2.87	84
Base	Recycled	25	3.03	2.90	83
+ CPT	Recycled	25	2.97	2.81	85

recent full attention step, the attention pattern should be similar. Based on this, for dynamic scheduling, we only perform the full attention step when this similarity falls below a threshold.

Approach At every S^{th} decode step, we first determine whether we *need* to perform full attention instead of always performing full attention by default. We calculate the cosine similarity between query vectors of the input token t averaged across all query heads in layer l, with the averaged query vector of the most recent full attention step for that layer. If the similarity is higher than a threshold s, we decode with recycle cache, and otherwise use full attention for layer l. Our approach offers the flexibility of using different schedules for different layers, but uses the same schedule for all heads in the same layer. Since we perform this similarity check every S steps, setting threshold $s = 1$ is equivalent to decoding with the fixed stride S. We perform the comparison only every S steps to reduce computational overhead; we call this query comparison (QC) stride.

Setup We run experiments with Llama-3.1-8B on the Arxiv and Books corpus. As before, we report perplexity and decoding time measured on one A100 with batch size of 1 for the last 256 tokens of each test sequence. We run dynamic scheduler with two different query comparison strides $\{5, 10\}$ and a similarity threshold of 0.8. We compare against Recycled Attention with fixed strides 10 and 15. For RULER, we report performance on two tasks which require generating longer outputs (*multi-query* and *multi-value*). We run a dynamic scheduler with two different query comparison strides 10 and a similarity threshold of $\{0.8, 0.9\}$. We compare against Recycled Attention with fixed strides $\{10, 15\}$. For dynamic schedules, we report the effective stride across layers, i.e. the average stride at which full attention is performed.

Dynamic stride strategy improves perplexity compared to fixed strategy when using similar decoding times. Table 6 reports our results. We observe that using dynamic strides improves the performance-efficiency trade-off across all settings. Compared to fixed stride of 10 (row 2), dynamic stride with query comparison stride of 5 (row 3) achieves lower perplexity with a slightly faster decoding time on both domains. Similarly, employing a dynamic stride with query comparison stride of 10 (last row) achieves better or on-par performance with less decoding time compared to having a fixed stride of 15 (row 4). We observe a similar trend for RULER tasks. This better trade-off can be attributed to the larger effective stride size, i.e. less frequent full attention steps, that result from using dynamic schedules. Overall, our experiment demonstrates that dynamically deciding when to refresh the recycle cache can improve performance when using similar decode times.

7 Related Work

Efficient inference methods There are multiple paths to improve decoding efficiency of long-context LMs. Prior work (Dao, 2024) achieves significant gain in inference latency by optimizing attention computations on GPUs. A line of work (Xiao et al., 2022) achieves efficiency through quantization of KV caches. We note that these are orthogonal to and can be combined with our approach. Other lines of work introduce changes to model architecture, which involves further training the model: Cai et al. (2024a) adds extra decoding heads to predict multiple subsequent tokens in parallel to further speed-up speculative decoding (Leviathan et al., 2022). Yen et al. (2024a) encodes

Table 6: Results comparing fixed stride and dynamic stride based on query similarity.

Method	Schedule	Language Modeling					
		Time	PPL *Arxiv*	Stride	Time	PPL *Book*	Stride
Vanilla	-	7.63	2.22	-	7.63	7.07	-
Recycled	Fixed	7.17	2.36	10	7.17	7.49	10
Recycled	Dynamic (QC = 5, s=0.8)	7.07	**2.32**	25	7.07	**7.42**	24
Recycled	Fixed	6.88	2.41	15	6.94	7.53	15
Recycled	Dynamic (QC = 10, s=0.8)	6.86	2.36	32	6.83	7.54	31
		RULER					
		Time	Acc *multi-query*	Stride	Time	Acc *multi-value*	Stride
Vanilla	-	1.71	99	-	1.71	99	-
Recycled	Fixed	1.48	69	10	1.48	44	10
Recycled	Dynamic (QC=10, s=0.9)	1.32	69	15	1.32	44	17
Recycled	Fixed	1.43	62	15	1.43	37	15
Recycled	Dynamic (QC=10, s=0.8)	1.26	62	36	1.24	40	38

chunks of long context in parallel with an encoder model, which are used as input to the decoder model. We note that our method can be used as a training-free method, and show that it is possible to fine-tune the model to further improve the performance-efficiency trade-off.

Dynamic KV cache Recent work (Sun et al., 2024) introduces a hierarchical speculative decoding method, which uses the model with a small KV cache constructed with attention pattern as draft model for the model with the full cache. While we share the motivation of using the attention pattern to construct a smaller KV cache, we directly leverages the dynamic cache to accelerate inference and study the performance-efficiency trade-off. Another line of recent work (Xiao et al., 2024a) proposes building a dynamic KV cache by mapping distant tokens into chunks and retrieving chunks that are similar to the current token, with the focus of extending the context size of the language model. Quest (Tang et al., 2024) uses the minimal and maximal key values to estimate import tokens for the query embedding of the current input token and load the KV cache of these tokens to decode.

KV cache eviction As performing attention over the full KV cache imposes a high memory and computation burden, KV cache eviction methods have been extensively studied. Strategies include keeping only "sink" and recent tokens in the KV cache (Xiao et al., 2023); or tokens with high accumulative attention scores (Zhang et al., 2024). Building on the idea of attention-based eviction strategy, PyramidInfer(Yang et al., 2024b) retains different number of tokens per layer. Another line of work proposed query-aware eviction strategies, using the attention scores of the last few tokens in the prompt to select tokens to keep (Li et al., 2024; Cai et al., 2024b; Chen et al., 2024). Other works design eviction strategies based on attention patterns of different heads (Ge et al., 2024; Xiao et al., 2024b) or different layers (Yang et al., 2024b).

8 Conclusion

We propose recycled attention, an inference-time method which maintains a small, dynamic KV cache based on attention patterns of neighboring tokens. Our work follows a line of work leveraging the locality assumption during the attention computation. Instead of using locality to directly decide which tokens to attend to (only selecting nearby tokens), we recycle the attention pattern of nearby tokens, allowing more flexible and dynamic sparse attention patterns. We apply our method to two off-the-shelf long-context model and show that our method reduces inference wall-clock time while better preserving performance compared to prior methods which keep a KV cache of recent tokens. Finally, we show that continued pre-training the model with recycled attention and employing a dynamic stride can further improve the performance-efficiency trade-off.

REFERENCES

Muhammad Adnan, Akhil Arunkumar, Gaurav Jain, Prashant Nair, Ilya Soloveychik, and Purushotham Kamath. Keyformer: Kv cache reduction through key tokens selection for efficient generative inference. *Proceedings of Machine Learning and Systems*, 6:114–127, 2024.

Joshua Ainslie, James Lee-Thorp, Michiel de Jong, Yury Zemlyanskiy, Federico Lebr'on, and Sumit K. Sanghai. Gqa: Training generalized multi-query transformer models from multi-head checkpoints. *ArXiv*, abs/2305.13245, 2023. URL https://api.semanticscholar.org/CorpusID:258833177.

Chen An, Fei Huang, Jun Zhang, Shansan Gong, Xipeng Qiu, Chang Zhou, and Lingpeng Kong. Training-free long-context scaling of large language models. *ArXiv*, abs/2402.17463, 2024. URL https://api.semanticscholar.org/CorpusID:268032518.

Chenxin An, Shansan Gong, Ming Zhong, Xingjian Zhao, Mukai Li, Jun Zhang, Lingpeng Kong, and Xipeng Qiu. L-eval: Instituting standardized evaluation for long context language models, 2023. URL https://arxiv.org/abs/2307.11088.

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. Longbench: A bilingual, multitask benchmark for long context understanding. *arXiv preprint arXiv:2308.14508*, 2023.

Iz Beltagy, Matthew E. Peters, and Arman Cohan. Longformer: The long-document transformer. *arXiv:2004.05150*, 2020.

Tianle Cai, Yuhong Li, Zhengyang Geng, Hongwu Peng, Jason D. Lee, De huai Chen, and Tri Dao. Medusa: Simple llm inference acceleration framework with multiple decoding heads. *ArXiv*, abs/2401.10774, 2024a. URL https://api.semanticscholar.org/CorpusID:267061277.

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. *CoRR*, 2024b.

Yilong Chen, Guoxia Wang, Junyuan Shang, Shiyao Cui, Zhenyu Zhang, Tingwen Liu, Shuohuan Wang, Yu Sun, Dianhai Yu, and Hua Wu. NACL: A general and effective KV cache eviction framework for LLM at inference time. In Lun-Wei Ku, Andre Martins, and Vivek Srikumar (eds.), *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 7913–7926, Bangkok, Thailand, August 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.acl-long.428. URL https://aclanthology.org/2024.acl-long.428.

Rewon Child, Scott Gray, Alec Radford, and Ilya Sutskever. Generating long sequences with sparse transformers. *ArXiv*, abs/1904.10509, 2019. URL https://api.semanticscholar.org/CorpusID:129945531.

Tri Dao. FlashAttention-2: Faster attention with better parallelism and work partitioning. In *International Conference on Learning Representations (ICLR)*, 2024.

Tim Dettmers, Mike Lewis, Sam Shleifer, and Luke Zettlemoyer. 8-bit optimizers via block-wise quantization. *CoRR*, abs/2110.02861, 2021. URL https://arxiv.org/abs/2110.02861.

Suyu Ge, Yunan Zhang, Liyuan Liu, Minjia Zhang, Jiawei Han, and Jianfeng Gao. Model tells you what to discard: Adaptive KV cache compression for LLMs. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=uNrFpDPMyo.

Gemini. Google. gemini 1.5: Unlocking multimodal understanding across millions of tokens of context. *arXiv preprint arXiv:2403.05530*, 2024.

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

Marzena Karpinska, Katherine Thai, Kyle Lo, Tanya Goyal, and Mohit Iyyer. One thousand and one pairs: A "novel" challenge for long-context language models, 2024. URL https://arxiv.org/abs/2406.16264.

Tomáš Kočiský, Jonathan Schwarz, Phil Blunsom, Chris Dyer, Karl Moritz Hermann, Gábor Melis, and Edward Grefenstette. The NarrativeQA reading comprehension challenge. *Transactions of the Association for Computational Linguistics*, 6:317–328, 2018. doi: 10.1162/tacl_a_00023. URL https://aclanthology.org/Q18-1023.

Yaniv Leviathan, Matan Kalman, and Yossi Matias. Fast inference from transformers via speculative decoding. In *International Conference on Machine Learning*, 2022. URL https://api.semanticscholar.org/CorpusID:254096365.

Yuhong Li, Yingbing Huang, Bowen Yang, Bharat Venkitesh, Acyr Locatelli, Hanchen Ye, Tianle Cai, Patrick Lewis, and Deming Chen. SnapKV: LLM knows what you are looking for before generation. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024. URL https://openreview.net/forum?id=poE54GOq2l.

Meta. The llama 3 herd of models. *ArXiv*, abs/2407.21783, 2024. URL https://api.semanticscholar.org/CorpusID:271571434.

NLP Team MosaicML. Introducing mpt-7b: A new standard for open-source, commercially usable llms, 2023. URL www.mosaicml.com/blog/mpt-7b. Accessed: 2023-05-05.

Bowen Peng, Jeffrey Quesnelle, Honglu Fan, and Enrico Shippole. YaRN: Efficient context window extension of large language models. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=wHBfxhZu1u.

Jack W Rae, Anna Potapenko, Siddhant M Jayakumar, Chloe Hillier, and Timothy P Lillicrap. Compressive transformers for long-range sequence modelling. *arXiv preprint*, 2019. URL https://arxiv.org/abs/1911.05507.

Hanshi Sun, Zhuoming Chen, Xinyu Yang, Yuandong Tian, and Beidi Chen. Triforce: Lossless acceleration of long sequence generation with hierarchical speculative decoding. *arXiv preprint arXiv:2404.11912*, 2024.

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

Together. Redpajama: an open dataset for training large language models, 2023. URL https://github.com/togethercomputer/RedPajama-Data.

Harsh Trivedi, Niranjan Balasubramanian, Tushar Khot, and Ashish Sabharwal. MuSiQue: Multihop questions via single-hop question composition. *Transactions of the Association for Computational Linguistics*, 10:539–554, 2022. doi: 10.1162/tacl_a_00475. URL https://aclanthology.org/2022.tacl-1.31.

Chaojun Xiao, Pengle Zhang, Xu Han, Guangxuan Xiao, Yankai Lin, Zhengyan Zhang, Zhiyuan Liu, Song Han, and Maosong Sun. Infllm: Unveiling the intrinsic capacity of llms for understanding extremely long sequences with training-free memory. *arXiv*, 2024a.

Guangxuan Xiao, Ji Lin, Mickael Seznec, Julien Demouth, and Song Han. Smoothquant: Accurate and efficient post-training quantization for large language models. *ArXiv*, abs/2211.10438, 2022. URL https://api.semanticscholar.org/CorpusID:253708271.

Guangxuan Xiao, Yuandong Tian, Beidi Chen, Song Han, and Mike Lewis. Efficient streaming language models with attention sinks. *ArXiv*, abs/2309.17453, 2023. URL https://api.semanticscholar.org/CorpusID:263310483.

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*, 2024b.

An Yang, Baosong Yang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Zhou, Chengpeng Li, Chengyuan Li, Dayiheng Liu, Fei Huang, Guanting Dong, Haoran Wei, Huan Lin, Jialong Tang, Jialin Wang, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Ma, Jin Xu, Jingren Zhou, Jinze Bai, Jinzheng He, Junyang Lin, Kai Dang, Keming Lu, Ke-Yang Chen, Kexin Yang, Mei Li, Min Xue, Na Ni, Pei Zhang, Peng Wang, Ru Peng, Rui Men, Ruize Gao, Runji Lin, Shijie Wang, Shuai Bai, Sinan Tan, Tianhang Zhu, Tianhao Li, Tianyu Liu, Wenbin Ge, Xiaodong Deng, Xiaohuan Zhou, Xingzhang Ren, Xinyu Zhang, Xipin Wei, Xuancheng Ren, Yang Fan, Yang Yao, Yichang Zhang, Yunyang Wan, Yunfei Chu, Zeyu Cui, Zhenru Zhang, and Zhi-Wei Fan. Qwen2 technical report. *ArXiv*, abs/2407.10671, 2024a. URL https://api.semanticscholar.org/CorpusID:271212307.

Dongjie Yang, Xiaodong Han, Yan Gao, Yao Hu, Shilin Zhang, and Hai Zhao. PyramidInfer: Pyramid KV cache compression for high-throughput LLM inference. In Lun-Wei Ku, Andre Martins, and Vivek Srikumar (eds.), *Findings of the Association for Computational Linguistics ACL 2024*, pp. 3258–3270, Bangkok, Thailand and virtual meeting, August 2024b. Association for Computational Linguistics. doi: 10.18653/v1/2024.findings-acl.195. URL https://aclanthology.org/2024.findings-acl.195.

Howard Yen, Tianyu Gao, and Danqi Chen. Long-context language modeling with parallel context encoding. *ArXiv*, abs/2402.16617, 2024a. URL https://api.semanticscholar.org/CorpusID:268032128.

Howard Yen, Tianyu Gao, and Danqi Chen. Long-context language modeling with parallel context encoding. In *Association for Computational Linguistics (ACL)*, 2024b.

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

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.

Yanli Zhao, Andrew Gu, Rohan Varma, Liangchen Luo, Chien chin Huang, Min Xu, Less Wright, Hamid Shojanazeri, Myle Ott, Sam Shleifer, Alban Desmaison, Can Balioglu, Bernard Nguyen, Geeta Chauhan, Yuchen Hao, and Shen Li. Pytorch fsdp: Experiences on scaling fully sharded data parallel. *Proc. VLDB Endow.*, 16:3848–3860, 2023. URL https://api.semanticscholar.org/CorpusID:258297871.

A Appendix

A.1 Ablating K and S

Our method depends on two hyperparameters, the size of the recycle cache K and the stride S which governs how often we perform full attention and update the recycle KV cache. Here, we analyze the impact of varying these two values for Llama-3.1. We experiment with 100 ArXiv sequences with $L = 16,354$ and 14 RULER tasks with $L = 32,768$. Results are reported in Table 10 and Figure 4. We see that Recycled Attention outperforms baselines with similar inference time budget for both tasks. For example, Recycled Attention with $K = 2048, S = 16$ achieves better perplexity than StreamingLLM with $K = 4096$. In fact, Recycled Attention with $K = 4096$ achieves better accuracy than StreamingLLM with a larger $K = 8192$ for RULER. Overall, we find that increasing K is more effective than decreasing the stride S. While decreasing stride S generally benefits Recycled Attention, it has negligible effect on StreamingLLM++. This shows that the improvement does not merely come from performing full attention, but also from refreshing the recycle cache.

Table 7: Results comparing different methods to aggregate attention scores for GQA models. We evaluate perplexity on sequences of length 16K for Llama-3.1-8B, where 4 query heads share the same KV head. We experiment with taking the attention score of the first query head, the average attention scores of the four query heads and the max of the four query heads to select top K KV cache.

Method	K	Stride	Agg	Arxiv PPL	Book PPL
Vanilla	-	-	-	2.22	7.07
StreamingLLM	2048	-	-	2.62	7.94
StreamingLLM++	2048	10	-	2.57	7.85
Retrieval	2048	10	First	2.43	7.62
Retrieval	2048	10	Mean	2.39	7.51
Retrieval	2048	10	Max	**2.36**	**7.49**

Table 8: Updated: Recycled attention and baseline performances when varying K and S on RULER tasks with 32K context length (left) and Arxiv with 16K context length (right). We report results for LLama-3.1-8B with decoding time measured on a single A100 machine.

Method	K	S	PPL	Time	K	S	Accuracy	Time
	ArXiv Perplexity (16K)				*RULER performance (32K)*			
Vanilla	-	-	2.22	7.39	-	-	90	1.71
Streaming	2048	-	2.62	6.64	4096	-	22	1.23
Streaming++	2048	32	2.61	6.61	4096	50	22	1.25
Recycled	2048	32	2.48	6.72	4096	50	63	1.27
Streaming++	2048	16	2.59	6.77	4096	10	22	1.4
Recycled	2048	16	2.40	6.90	4096	10	65	1.48
Streaming	4096	-	2.44	6.94	8192	-	26	1.46
Streaming++	4096	32	2.43	7.05	8192	50	26	1.47
Recycled	4096	32	2.33	7.12	8192	50	70	1.48

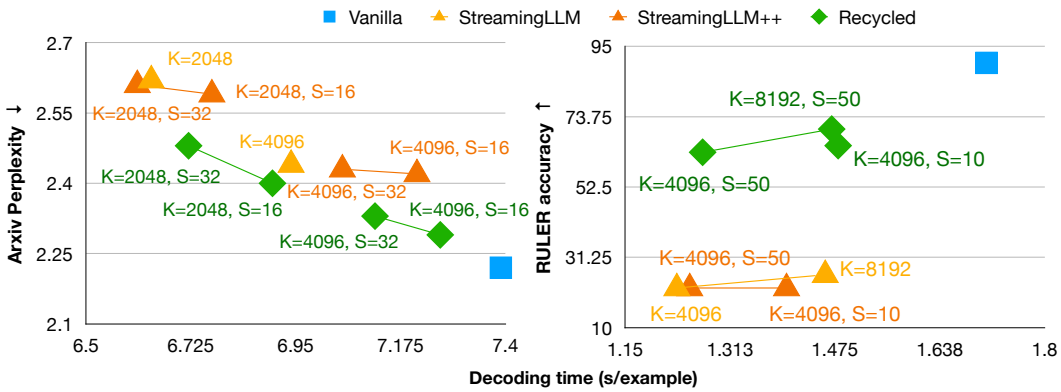

Figure 4: New: Recycled Attention and baseline performances when varying K and S on Arxiv with 16K context length (left) and RULER tasks with 32K context length (right) for Llama-3.1-8B. Recycled Attention achieves better performance than baselines (StreamingLLM, StreamingLLM++) with the same or less decoding time for both task.

Figure 5: Recovery rate of StreamingLLM and recycled attention on 5 samples from the Arxiv split of RedPajama (Left: LLaMA-3.1, Right: Qwen-2). We calculate recovery rate with a prefill length of 8K, K of 1024 and $S = 256$.

A.2 LongBench experiments

We evaluate our method on two long-context tasks with average input length is longer than 10K from LongBench(Bai et al., 2023): NarrativeQA(Kočiský et al., 2018), Musique(Trivedi et al., 2022). We generate up to 50 tokens for each task.

Results Experiment results are reported in Table 9. We find that Recycled Attention with kernel size of 7 performs comparably or better compared to SnapKV (the best performing baseline), with faster decoding speed.

Table 9: NEW:Performance on two datasets from LongBench. benchmark for LLama-3.1-8B and Qwen-2-7B. We set $K = 2048$ for all tasks.

Method	stride	LLama-3.1 NarrativeQA F1 ↑	LLama-3.1 NarrativeQA time(s) ↓	LLama-3.1 Musique F1 ↑	LLama-3.1 Musique time(s) ↓	QWEN2 NarrativeQA F1 ↑	QWEN2 NarrativeQA time(s) ↓	QWEN2 Musique F1 ↑	QWEN2 Musique time(s) ↓
Vanilla	-	40	2.93	31	1.38	15	9.33	41	1.71
H_2O	-	29	4.37	29	1.62	11	4.03	28	1.98
StreamingLLM	-	36	2.47	25	1.2	9	2.30	33	1.12
StreamingLLM++	50	36	2.54	25	1.22	9	2.30	33	1.15
Recycled	50	39	2.32	31	1.22	**13**	2.28	40	1.15
Recycled (kernel=7)	50	**41**	2.32	**32**	1.23	**13**	2.28	40	1.15
SnapKV (kernel=7, w=32)	-	40	3.21	**32**	1.47	**13**	2.86	**41**	1.33

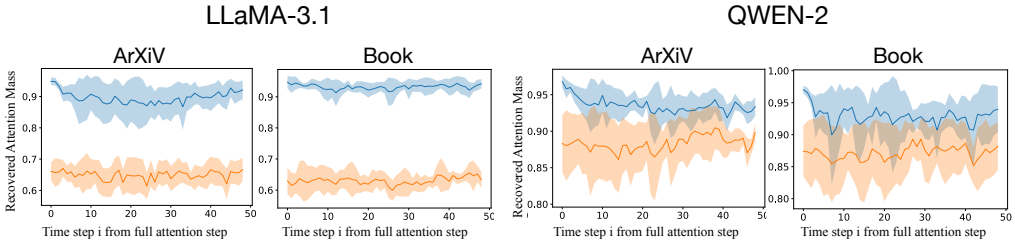

Figure 6: Recovery rate of StreamingLLM and recycled attention on 5 samples each from the Arxiv and Book split of RedPajama (Left: LLaMA-3.1, Right: Qwen-2). We calculate recovery rate with a prefill length of 8K, K of 1024 and $S = 50$.

A.3 Continued Pre-training implementation details

Implementation details We train Llama-3.1 for one epoch with a global batch size of 64 and a learning rate of 5e-6. We use 20 warm-up steps and a linear schedule with 0 weight decay. We use the AdamW Optimizer. We use Fully Sharded Data Parallel (Zhao et al., 2023) and 8-bit optimizer (Dettmers et al., 2021) to improve training efficiency. Training is done on 4 H100 80 GB GPUs.

A.4 Attention pattern analysis

We analyze the **recovery rate** of recycled attention and StreamingLLM for LLaMA-3.1 and QWEN-2 (similar to the setting in Section 2). Figure 6 shows the aggregated attention recovery rate. We observe a consistent trend across the two domains. While for both models recycled attention recovers more attention mass than StreamingLLM, the gap between the two methods is much smaller for QWEN-2.

A.5 Dynamic Stride analysis

We reported an aggregated effective stride in Table 6. We further investigate the effective stride patterns across different layers, shown in Figure 7. We can see that our method enables setting a different stride at different layers, with the earlier layer having a larger stride.

A.6 Ruler configuration

We follow the suite of evaluation tasks introduced in Hsieh et al. (2024), which consists of the 13 tasks[4]. We group them based on the types:

[4] https://github.com/hsiehjackson/RULER

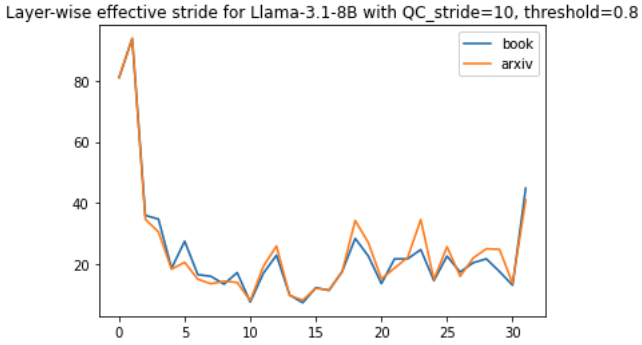

Figure 7: Layer-wise effective stride for LLaMA-3.1-8B with query similarity dynamic strides.

Table 10: Recycled attention and baseline performances when varying K and S on RULER tasks with 32K context length (left) and Arxiv with 16K context length (right). We report results for LLama-3.1-8B with decoding time measured on a single A100 machine.

Method	**K**	**S**	**PPL**	**Time**	**K**	**S**	**Accuracy**	**Time**
	ArXiv Perplexity (16K)				*RULER performance (32K)*			
Vanilla	-	-	2.22	7.39	-	-	90	1.71
Streaming	2048	-	2.62	6.64	4096	-	22	1.23
Streaming++	2048	32	2.61	6.61	4096	50	22	1.25
Recycled	2048	32	2.48	6.72	4096	50	63	1.27
Streaming++	2048	16	2.59	6.77	4096	10	22	1.4
Recycled	2048	16	2.40	6.90	4096	10	65	1.48
Streaming	4096	-	2.44	6.94	8192	-	26	1.46
Streaming++	4096	32	2.43	7.05	8192	50	26	1.47
Recycled	4096	32	2.33	7.12	8192	50	70	1.48

Single NIAH An NIAH-styled task with one key and one value to retrieve. We include three variations of the task with different types of key, value and haystack.

Multi-key NIAH An NIAH-styled task with distracting keys. We include three variations of the task with different types of key, value and haystack.

Multi-values NIAH An NIAH-styled task with multiple values corresponding to the key.

Multi-queries NIAH An NIAH-styled task with multiple queries, each corresponding to a distinct key.

Variable Tracking A NIAH-styled task that requires tracing through multiple hops.

Common word extraction and **Frequent word extraction** require extracting the words based on the pattern in a list of words.

Question Answering A task that requires answering a question given a set of documents. We include two variations of the tasks, corresponding to two question answering datasets.

We refer the readers to Hsieh et al. (2024) for detailed description and examples of each task.

A.7 Limitations and Future Work

Proposed method While we focus on accelerating inference speed, our method does not reduce memory requirement for using long-context LLMs, which can be a bottleneck for certain use cases. Our method is focused on a setting where we generate long output given a long input. When the

output length is very small, the efficiency gain will be minimal. In this study, we focus on employing a fixed stride across all layers and explore dynamic scheduling based on query-similarity. Setting a custom stride per layer, or exploring other methods for deciding when to recycle the cache could be future avenue to improve performance.

Experimental Settings We have conducted experiment with two open-sourced long-context models and two evaluation tasks setting. We did not test out more language models and other long-context benchmarks (An et al., 2023; Karpinska et al., 2024) given our limited compute resources. Finally, our method is not limited to the language domain. Future work can explore applying recycled attention to other modalities, for instance, vision transformers.