BLOCK-ATTENTION FOR LOW-LATENCY RAG

East Sun[*]
tianjibai@gmail.com

Yan Wang[*†]
yanwang.branden@gmail.com

Tian Lan
lantiangmftby@gmail.com

ABSTRACT

We introduce Block-Attention, an attention mechanism designed to address the increased inference latency in Retrieval-Augmented Generation (RAG) scenarios. Its main idea lies in dividing the input sequence into blocks, where each block calculates its key-value (KV) states independently except for the final block. In RAG scenarios, by defining each passage as a block, Block-Attention enables us to pre-compute the KV states for all passages and cache them in memory.

The implementation involves block segmentation, positional encoding calculation, and fine-tuning the LLM to adapt to the Block-Attention mechanism. Experiments on four RAG benchmarks demonstrate that after block fine-tuning, the Block Attention model can achieve performance comparable to (68.4% vs 67.9% on Llama3) or even better (62.8% vs 59.6% on Mistral) than self-attention models. Notably, Block-Attention reduces the TTFT to a very low level. It only takes 45 ms to output the first token for an input sequence with a total length of 32K. Compared with the self-attention model, the time consumption is reduced by 98.7%.[1]

1 INTRODUCTION

Retrieval-Augmented Generation (RAG) (Li et al., 2022) is a crucial technology for mitigating knowledge hallucination in Large Language Models (LLMs). By utilizing retrieval technology, LLMs can seamlessly access passages stored in external databases, grounding their responses in the content of these passages. To the best of our understanding, RAG has emerged as the most effective method for infusing specific domain knowledge into LLMs in real-world scenarios.

However, everything has two sides, and RAG is no exception. Generally, for each user query, in order to ensure that a passage with the "correct knowledge" is successfully recalled, we retrieve multiple passages (between 5 to 30 in most scenarios) and incorporate them into the input prompt of the LLM. Within this new and significantly longer input prompt, the inference latency, particularly the time to first token (TTFT), of a RAG-LLM will be much higher than that of a non-RAG-LLM.

Given that the content of passages in the external databases is already determined, one might naturally consider the feasibility of caching them for fast inference. Nonetheless, for autoregressive LLMs, the KV states are inherently context-dependent, which means the KV states for the same passage will vary in different contexts. As a result, when encountering an unseen query, the model must undertake a complete re-encoding of the KV states to ensure the accuracy of its predictions.

In this paper, we propose the Block-Attention method, which reduces the TTFT of RAG-LLMs to a level nearly equivalent to that of non-RAG LLMs, while fully maintaining the same accuracy level. As shown in Figure 1, the main idea of Block-Attention is to divide the entire input sequence into several blocks. Each block independently calculates its KV states through self-attention, without any attention to other blocks. Only the final block is able to attend other blocks. When utilizing Block-Attention in RAG scenarios, we may achieve substantial benefits by defining each passage as a single block and caching its KV states in the memory. The Block-Attention method has effectively

[*] Equal Contribution
[†] Corresponding Author
[1]Codes are available at: https://github.com/TemporaryLoRA/Block-Attention

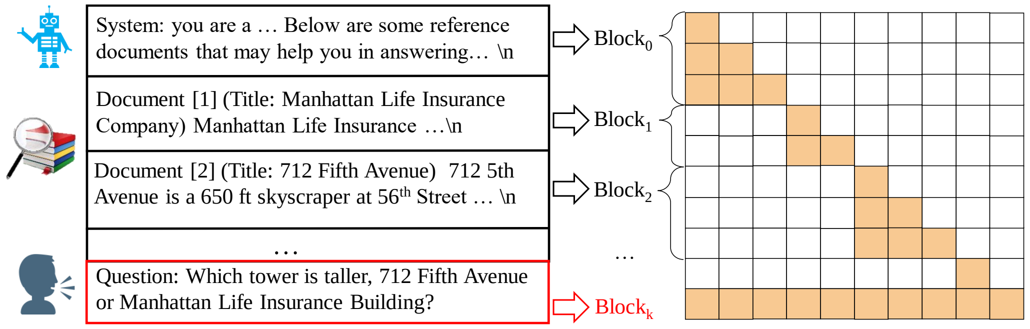

System: you are a … Below are some reference documents that may help you in answering… \n
Document [1] (Title: Manhattan Life Insurance Company) Manhattan Life Insurance …\n
Document [2] (Title: 712 Fifth Avenue) 712 5th Avenue is a 650 ft skyscraper at 56th Street … \n
…
Question: Which tower is taller, 712 Fifth Avenue or Manhattan Life Insurance Building?
Block0
Block1
Block2
…
Blockk

Figure 1: The Block-Attention Masks

reduced the average TTFT in experiments. When the length of the input sequence is 32K and the length of the user input is 50, the Block-Attention method can achieve a 98.7% reduction in TTFT (from 3638 ms to 45 ms), along with a 99.8% reduction in Flops of first token. [2]

The implementation of Block-Attention can be easily achieved through the following steps: 1) Encode all blocks except the last one separately; 2) calculating the positional encoding for each based on its position in the input prompt; 3) Integrating these blocks to compute the KV states for the final block. However, the primary challenge in utilizing Block-Attention is that the LLMs have not been exposed to such an attention mechanism during their training, leading to difficulties in accurately interpreting the input prompt. In our preliminary experiments, we attempted to directly implement Block-Attention in the LLMs without updating any parameters. Unfortunately, this approach resulted in a substantial decrease in performance, with the average accuracy of Llama3 on four RAG benchmarks falling from 67.9% to 48.0%.

To address this challenge, we implemented a fine-tuning process for the LLMs to adapt to the Block-Attention mechanism. Our experiments demonstrated that, after approximately 100 - 1000 fine-tuning steps, the Block-Attention model achieved a full recovery of its original accuracy across both in-domain and out-domain scenarios, impressively increasing from 48.0% to 68.4%. This outcome underscores the Block-Attention LLMs' capability to uphold inference accuracy while significantly enhancing inference efficiency in RAG scenarios.

We evaluate Block-Attention on four RAG benchmarks. Experimental results demonstrate that after fine-tuning, the average accuracy of the Block-Attention model on the benchmarks can remain comparable to (68.4% vs 67.9% on Llama3) or even slightly exceed (62.8% vs 59.6% on Mistral) the performance of self-attention models. In terms of efficiency, we counted the TTFT of the block-attention model when the length of user input is 50 and the total length of the input sequence gradually increases. We found that the longer the total length, the more obvious the improvement of block-attention on efficiency. When the length of the input sequence reaches 32K, the TTFT of the block-attention model is only 45 ms, which is 1.3% of that of the self-attention model.

2 Block-Attention

2.1 Main Idea

Let $\mathcal{S} = \{s_0, s_1, ..., s_n\}$ represents the input sequence, where each s represents a token. We denote the KV states associated with $\mathcal{S}$ as $\mathcal{K} = \{k_0, k_1, ..., k_n\}$ and $\mathcal{V} = \{v_0, v_1, ..., v_n\}$, respectively. For an auto-regressive model Θ_{LLM}, since the computation of the KV states is dependent on their preceding tokens, when a text block $\{s_i, ..., s_j\}$ changes to a new text block $\{s'_i, ..., s'_m\}$, for the new sequence $\mathcal{S}' = \{s_0, ..., s'_i, ..., s'_m, s_{j+1}, ..., s_n\}$, its KV states become $\mathcal{K}' = \{k_0, ..., k'_i, ..., k'_m, k'_{j+1}, ..., k'_n\}$ and $\mathcal{V}' = \{v_0, ..., v'_i, ..., v'_m, v'_{j+1}, ..., v'_n\}$. It is evident that although only one block $\{s_i, ..., s_j\}$ has changed, due to the auto-regressive nature of LLMs, the KV states of all subsequent blocks must be re-encoded.

[2]Given that the KV cache is already a mature and low-cost technology (Qin et al., 2024), in this paper we do not take the cost of KV cache into account

In the scenario of RAG, a given passage frequently recurs in the retrieval results for numerous queries, making it a natural consideration to explore the possibility of reusing its KV states. Unfortunately, the auto-regressive nature of the encoding approach, as previously outlined, imposes constraints that render the KV states nearly non-reusable across distinct queries. Our research is dedicated to examining a noval attention mechanism, Block-Attention, that designed to necessitate the re-computation of only the altered text blocks between two input sequences, thereby achieving an outcome equivalent to that of full sequence re-encoding.

As illustrated in Figure 1, the essence of Block-Attention is to divide the input sequence $\mathcal{S}$ into several independent blocks. Each block autonomously calculates its KV states through self-attention, without considering other blocks. The final block, however, has the unique capability to integrate information from preceding blocks. A primary advantage of this method is the modular independence it provides: when a block b_i is updated to b'_i, re-encoding only the KV states of the affected block $k_{b'_i}$, $v_{b'_i}$, and those of the final block k_{b_k}, v_{b_k}, is sufficient to obtain the updated sequence's KV states.

To develop a Block-Attention LLM capable of precise inference, we must tackle three key challenges:

- How do we segment blocks?
- How should the positional encoding be calculated for each block?
- How can the LLM be adapted to the Block-Attention mechanism?

These issues will be addressed in detail in Sections 2.2, 2.3, and 2.4, respectively.

2.2 Block Segmentation

The primary principle of block division is to segment semantically independent parts of the prompt into separate blocks. In RAG scenarios, since the retrieved passages are originally mutually independent, it is natural to divide them into different blocks. Therefore, let's go back to the left part of Figure 1, where we allocate each passage to a single block and designate the user's query as the final block. This principle extends to other scenarios as well. For example, in the context of code generation tasks, a function may be treated as one block; in multi-turn dialogues, each turn could be segmented into an individual block. In this paper, our primary focus is on the application of Block-Attention in RAG, with the exploration of other scenarios reserved for future research.

2.3 Positional Encoding

The second problem is to re-encoding the positional information. Although the same passage may appear in multiple input prompts, its position generally varies. Therefore, when we attempt to reuse the KV states of a block, we need to re-encode its positional information. The process of re-encoding is simple and straightforward: taking the rotary positional encoding (RoPE) as an example, assume we wish to change the positional encoding of a block $b = \{s_i, ..., s_j\}$ to $b' = \{s_{i_\triangle}, ..., s_{j_\triangle}\}$, then we only need three steps:

1) For the token s_i, its positional encoding vector $f(s_i, i)$ is calculated using the following formula:

$$f(x_i, i) = \begin{pmatrix} \cos i\theta_0 & -\sin i\theta & \cdots & 0 & 0 \\ \sin i\theta_0 & \cos i\theta & \cdots & 0 & 0 \\ 0 & 0 & \cdots & \cos i\theta_{\frac{d}{2}-1} & -\sin i\theta_{\frac{d}{2}-1} \\ 0 & 0 & \cdots & \sin i\theta_{\frac{d}{2}-1} & \cos i\theta_{\frac{d}{2}-1} \end{pmatrix} \begin{pmatrix} p_0 \\ p_1 \\ \cdots \\ p_{d-1} \end{pmatrix} \tag{1}$$

2) we set x_i's positional encoding to zero, which is equivalent to rotating it counterclockwise by $i\theta$ degrees

$$f(x_i, 0) = \begin{pmatrix} \cos i\theta_0 & \sin i\theta & \cdots & 0 & 0 \\ -\sin i\theta_0 & \cos i\theta & \cdots & 0 & 0 \\ 0 & 0 & \cdots & \cos i\theta_{\frac{d}{2}-1} & \sin i\theta_{\frac{d}{2}-1} \\ 0 & 0 & \cdots & -\sin i\theta_{\frac{d}{2}-1} & \cos i\theta_{\frac{d}{2}-1} \end{pmatrix} f(x_i, i) \tag{2}$$

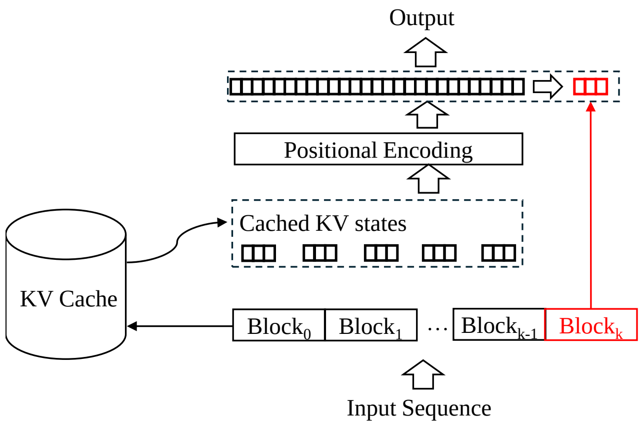


Figure 2: The Inference Pipeline of Block-Attention Model

3) Then, by performing a clockwise rotation of $(i_\triangle)\theta$ degrees, we obtain the final positional encoding vector.

$$f(x_{i_\triangle}, i_\triangle) = f(x_i, i_\triangle) \\ \begin{pmatrix} \cos{(i_\triangle)\theta_0} & -\sin{(i_\triangle)\theta} & \cdots & 0 & 0 \\ \sin{(i_\triangle)\theta_0} & \cos{(i_\triangle)\theta} & \cdots & 0 & 0 \\ 0 & 0 & \cdots & \cos{(i_\triangle)\theta_{\frac{d}{2}-1}} & -\sin{(i_\triangle)\theta_{\frac{d}{2}-1}} \\ 0 & 0 & \cdots & \sin{(i_\triangle)\theta_{\frac{d}{2}-1}} & \cos{(i_\triangle)\theta_{\frac{d}{2}-1}} \end{pmatrix} f(x_i, 0) \tag{3}$$

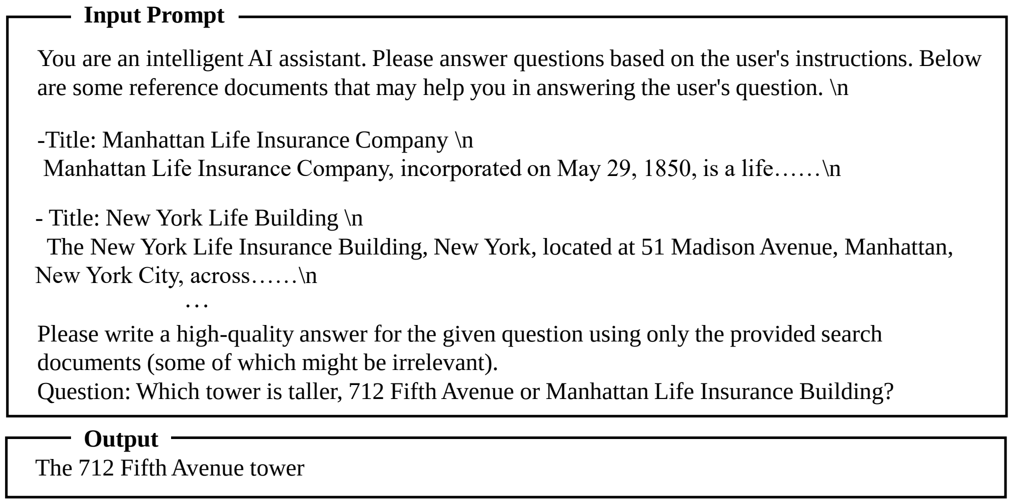
For the remaining tokens within block b, namely $s_{i+1}, \ldots, s_j$, we can re-encode their positional information in a similar manner. Although the formulas presented above may seem complex, the principle is quite straightforward: **first set the positional encoding to zero, and then rotate it to the updated position**. One might wonder why we do not simply rotate by $(i_\triangle - i)\theta$ degrees directly? The reason is to mitigate the potential for errors in updating positional encodings within practical applications: in the KV cache, the positional encoding of the initial token of each block is standardized to zero, and with only the updated positional index $i_\triangle$, we can readily determine their new positional encoding vectors as per Equation 3.

2.4 Block Fine-tune

Due to the LLM's reliance on full self-attention during the training phase, a direct switch to Block-Attention during inference might result in a significant discrepancy between the training and inference states. Our preliminary findings indicate that introducing Block-Attention without subsequent fine-tuning could precipitate a substantial decrease in performance, with the average accuracy dropping significantly from 67.9% to 48.0%. Adapting the LLM to Block-Attention through fine-tuning, which we refer to as "block fine-tune," is quite straightforward. The only difference from the standard SFT process is the modification of the traditional lower-triangular attention mask matrix to the attention mask matrix depicted in the right part of Figure 1. With this masking matrix, tokens in all blocks except the last are restricted to attending only to information within their own block, thereby ensuring consistency between training and inference.

2.5 Inference

In inference, the Block-Attention model retrieves the KV states of blocks from the KV cache and concatenates them into the complete input KV states. The detailed process of the inference stage is depicted in Figure 2. Initially, we query and extract the KV states of the first $k-1$ blocks from the cache. Then, based on the position of each block within the input sequence, we calculate their positional encoding using Equation 3. Finally, using the KV states of the first $k-1$ blocks, we compute the KV states of the last block as well as the model output.



Input Prompt

You are an intelligent AI assistant. Please answer questions based on the user's instructions. Below are some reference documents that may help you in answering the user's question. \n

-Title: Manhattan Life Insurance Company \n
Manhattan Life Insurance Company, incorporated on May 29, 1850, is a life……\n

- Title: New York Life Building \n
The New York Life Insurance Building, New York, located at 51 Madison Avenue, Manhattan, New York City, across……\n

…

Please write a high-quality answer for the given question using only the provided search documents (some of which might be irrelevant).
Question: Which tower is taller, 712 Fifth Avenue or Manhattan Life Insurance Building?

Output

The 712 Fifth Avenue tower

Figure 3: The Inference Pipeline of Block-Attention Model

3 Experiments

After the above analysis, most readers may have the following two concerns about Block-Attention: 1) Can the Block-Attention model achieve the same accuracy as self-attention in RAG scenarios? 2) How much can the Block-Attention mechanism improve the RAG efficiency? The following experimental results will reveal the answers to these two questions for everyone. In Sections 3.1, we analyze the accuracy of Block-Attention models. Meanwhile, in Section 3.2, we demonstrate the efficiency of Block-Attention.

Datasets We evaluate the performance of the Block-Attention method on four RAG benchmarks: Natural Questions (NQ) (Kwiatkowski et al., 2019), TriviaQA (TQA) (Joshi et al., 2017), HotpotQA (HQA) (Yang et al., 2018), and 2WikiMultiHopQA (2Wiki) (Ho et al., 2020). Specifically, we employ the training datasets of TQA and 2Wiki for fine-tuning models, followed by evaluations on all four respective test sets.

In training, we have randomly sampled 20,000 instances from both the TQA and 2wiki datasets to form our training set. The model's input comprises (1) the question and (2) 10 retrieved passages. To obtain these 10 passages, we employed the Contriver[3] to identify the 10 most relevant passages from the provided dataset. Additionally, due to the fact that the original answers in the TQA and 2wiki datasets may not be included in the retrieved passages, this could lead the model to overlook the content of the retrieved passages and generate outputs directly. Instead, we utilize the outputs from GPT-4 as the model's output to ensure consistency between the retrieved passages and the final output.

Following Kandpal et al. (2023) and Liu et al. (2024), we use accuracy as our primary evaluation metric, judging whether any of the correct answers appear in the predicted output. To mitigate biases arising from output length, we set a maximum token limit of 200 for the output sequences.

Prompt Format The format of input prompt follows Liu et al. (2024). For retrieved passages, we concatenate them in ascending order of retrieval score. An example is shown in Figure 3.

Base Model & Baselines We implement the Block-Attention mechanism on two base language models: Llama3-8B-base[4] and Mistral-7B-v3.0[5]. The models, after block fine-tuning, are denoted as *Llama3-block* and *Mistral-block*. In addition to these, we construct two strong baselines and two weak baselines.

[3]https://github.com/facebookresearch/contriever
[4]https://huggingface.co/meta-llama/Meta-Llama-3-8B
[5]https://huggingface.co/mistralai/Mistral-7B-v0.3

	TQA	2wiki	NQ	HQA	Average
Llama3-vanilla	72.8	73.4	55.7	69.5	67.9
Llama3-block-wo-ft	62.7	42.1	43.2	39.6	48.0
Llama3-block	73.0	73.3	56.2	68.5	68.4
Mistral-vanilla	66.6	69.5	46.0	56.4	59.6
Mistral-block-wo-ft	51.9	44.6	32.1	27.8	38.7
Mistral-block	70.7	69.0	50.5	58.6	62.8

Table 1: Accuracy of different models on different datasets. Among them, TQA and 2wiki serve as in-domain test sets, while NQ and HQA are out-of-domain test sets. We have not utilized these latter two datasets for fine-tuning. Furthermore, please note that the "average" column does not depict a simple arithmetic mean of the preceding four columns, as the number of samples contained in each test set varies.

- **Strong baselines:** *Llama3-vanilla* and *Mistral-vanilla*. These models are derived from fine-tuning the base language models on the training dataset. They can be considered as the upper bounds for the Block-Attention model. We aim for the Block-Attention model's performance to approach these as closely as possible.
- **Weak baselines:** *Llama3-block-wo-ft* and *Mistral-block-wo-ft*. These models transition the attention mechanism of Llama3-vanilla and Mistral-vanilla to a Block-Attention mechanism **without any block fine-tuning**. the outcomes of these models will represent the lower bounds for the Block-Attention model's effectiveness, given that they have not undergone any adaptation to Block-Attention during the training phase.

Please note that we have not utilized instruct-based models, such as Llama3-instruct, as our base language model. The reason is our inability to entirely re-implement the fine-tuning processes of these models, which precludes a direct comparison between self-attention and Block-Attention methodologies under identical experimental conditions. Instead, we perform vanilla fine-tuning and block fine-tuning on the same models using the same dataset, ensuring a fair comparison.

Setup All experiments are done with 8 NVIDIA A800 GPUs, we set the learning $\alpha = 2 \times 10^{-5}$, batch size $b = 64$ and Num. of epochs $n = 1$ for all models. We use a linear learning rate warmup for the first 20 steps. Its training is in bfloat16 format, using Deepspeed ZeRO Stage 2, Flash-Attention V2 (for sft training) and SDPA (for block training).

3.1 Main Results

From the results in Table 1, we can draw two important conclusions: 1) It is not advisable to directly switch from self-attention to Block-Attention, as it will lead to a sharp drop in the accuracy of the model. 2) However, if we use the Block-Attention mechanism in the fine-tuning stage, then the resultant model has almost the same performance as the self-attention model or is even slightly better on some datasets.

The first conclusion is easy to understand: The model has never seen an input sequence in the Block-Attention manner during the training process. Therefore, the sharp drop makes sense. Next, we will focus on analyzing the second conclusion. From Table 1, we can easily find that the models trained with two attention methods, namely *Llama3-vanilla* and *Llama3-block*, not only have comparable effects on in-domain test sets but also have little difference on out-domain test sets. This indicates that in the RAG scenario, it is completely feasible to replace self-attention with Block-Attention, and there will be almost no performance loss.

Moreover, the experimental results on Mistral show some interesting results: the accuracy of Block-Attention models actually slightly outperformed the results of self-attention models on two datasets. It raised a conjecture: is Block-Attention a better attention mechanism than the self-attention mechanism? Intuitively, the semantics of each retrieved passage are mutually independent, and the self-attention method may cause the representation of the passage to be interfered by the context and

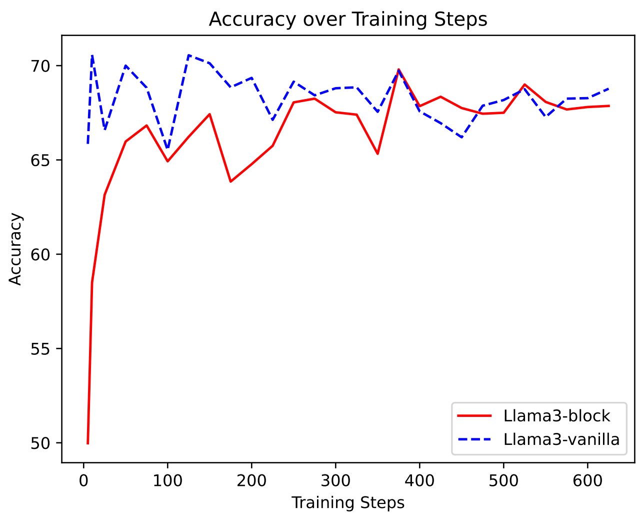


Figure 4: The accuracy of model checkpoint at different steps

not accurate enough. However, we still need sufficient experiments to verify the correctness of this assertion, so this is only a conjecture and not our conclusion.

Finally, one may still be interested in knowing exactly how many training steps are needed for the model to adapt to the Block-Attention mechanism. Therefore, we counted the accuracy of the Llama3 model on the four test sets at different fine-tuning steps and plotted it in Figure 4. We find that at the beginning stage of fine-tuning, there is a huge performance difference between the two models. It makes sense because the model needs more training steps to adapt to the Block-Attention manner. After about 400 training steps, the model completely adapts to the new attention mechanism, and the two models show comparable accuracy.

3.2 Efficiency

In the previous section, we already addressed our first concern: After fine-tuning, the Block-Attention model can achieve similar or even better performance than the self-attention model. In this section, we focus on our second concern: How much can the Block-Attention mechanism reduce the TTFT and the Flops of first token?

To quantify the effects of the Block-Attention mechanism on the TTFT, we show in Table 2 the respective TTFTs and Flops of the *Llama3-block* and *Llama3-vanilla* when the KV states of all retrieved passages have been pre-computed and cached in memory. Obviously, the acceleration effect is gratifying: Once the length of the input sequence is 512 and the length of user input is 50, using Block-Attention can reduce TTFT by 48% and reduce Flops by 90.1%. As the total length increases, when it is equal to 32K, the acceleration effect reaches an astonishing 98.7%, and the consumption of Flops is even reduced by 99.8%. We may simply conclude the results as: with greater text comes greater necessity for Block-Attention.

3.3 Discussion

From the experimental results, we can easily figure out the effects of Block-Attention on existing RAG-based applications: Under the existing technical paradigm, developers need to deliberately balance the trade-off between accuracy and inference speed. Therefore, they have to limit the number of retrieved passages to a certain number, such as 3 to 10. With Block-Attention, the impact of the number of retrieved passages on inference speed will be greatly reduced, which will empower the developers to freely choose the number of retrieved passages that gives the optimal effect without any hesitation.

Prompt Length	0	512	1K	2K	4K	8K	16K	32K
TTFT-vanilla	26	50	87	167	330	691	1515	3638
TTFT-block	26	26(48%)	26(71%)	26(84%)	27(91%)	29(95%)	34(97%)	45(98.7%)
Flops-vanilla	7.5e+11	7.6e+12	1.5e+13	3.0e+13	6.1e+13	1.2e+14	2.45e+14	4.9e+14
Flops-block	7.5e+11	7.5e+11	7.5e+11	7.5e+11	7.5e+11	7.5e+11	7.5e+11	7.5e+11
Reduction	-	90.1%	95.0%	97.5%	98.7%	99.3%	99.6%	99.8%

Table 2: The Time and Flops consumed by the first token of a user input with a length of 50 tokens under different total length of the retrieved passages

Furthermore, some dynamic RAG paradigms such as ReAct (Yao et al., 2023) have been proven to enhance the general task-solving ability of the model in most scenarios. However, compared to vanilla RAG, such paradigms need to iteratively compute the KV states of retrieved passages for the same user input, which results in a disastrous inference latency. With the help of Block-Attention, the additional latency consumption in this aspect can be reduced to a level that users can tolerate. Developers will be able to safely use such paradigms in online scenarios to optimize the reasoning ability of LLM.

4 Conclusion

We introduce Block-Attention to optimize the inference efficiency of LLM in RAG scenarios. Its essence lies in separately calculating the KV states of independent blocks in the input prompt. Block-Attention enables us to pre-compute the KV states for all passages and cache them in memory, thus avoiding repeated computation of the KV states of the same passage during inference.

Our experimental results across various RAG benchmarks demonstrated the profound impact of Block-Attention. We showed that Block-Attention can maintain the original reasoning accuracy of LLM while significantly reducing its TTFT and Flops in RAG scenarios. The effectiveness of Block-Attention becomes increasingly apparent as the number of passages increases or the frequency of retrievals increases. When considering the implementation of Block-Attention in your applications, bear in mind this guiding principle: With greater text comes greater necessity for Block-Attention.

We also want to note that Block-Attention plays an important role in many scenarios and is not limited to RAG only. We only introduced its effects on RAG because of that, due to some confidentiality reasons, we cannot disclose how we use it in our industrial applications. We look forward to researchers from the community being able to further explore the potential of Block-Attention and apply it to more appropriate scenarios.

References

Xanh Ho, Anh-Khoa Duong Nguyen, Saku Sugawara, and Akiko Aizawa. Constructing a multi-hop QA dataset for comprehensive evaluation of reasoning steps. In Donia Scott, Nuria Bel, and Chengqing Zong (eds.), *Proceedings of the 28th International Conference on Computational Linguistics*, pp. 6609–6625, Barcelona, Spain (Online), December 2020. International Committee on Computational Linguistics. doi: 10.18653/v1/2020.coling-main.580. URL `https://aclanthology.org/2020.coling-main.580`.

Mandar Joshi, Eunsol Choi, Daniel Weld, and Luke Zettlemoyer. TriviaQA: A large scale distantly supervised challenge dataset for reading comprehension. In Regina Barzilay and Min-Yen Kan (eds.), *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 1601–1611, Vancouver, Canada, July 2017. Association for Computational Linguistics. doi: 10.18653/v1/P17-1147. URL `https://aclanthology.org/P17-1147`.

Nikhil Kandpal, Haikang Deng, Adam Roberts, Eric Wallace, and Colin Raffel. Large language models struggle to learn long-tail knowledge, 2023. URL `https://arxiv.org/abs/2211.08411`.

Tom Kwiatkowski, Jennimaria Palomaki, Olivia Redfield, Michael Collins, Ankur Parikh, Chris Alberti, Danielle Epstein, Illia Polosukhin, Jacob Devlin, Kenton Lee, Kristina Toutanova, Llion Jones, Matthew Kelcey, Ming-Wei Chang, Andrew M. Dai, Jakob Uszkoreit, Quoc Le, and Slav Petrov. Natural questions: A benchmark for question answering research. *Transactions of the Association for Computational Linguistics*, 7:452–466, 2019. doi: 10.1162/tacl_a_00276. URL https://aclanthology.org/Q19-1026.

Huayang Li, Yixuan Su, Deng Cai, Yan Wang, and Lemao Liu. A survey on retrieval-augmented text generation, 2022. URL https://arxiv.org/abs/2202.01110.

Nelson F. Liu, Kevin Lin, John Hewitt, Ashwin Paranjape, Michele Bevilacqua, Fabio Petroni, and Percy Liang. Lost in the middle: How language models use long contexts. *Transactions of the Association for Computational Linguistics*, 12:157–173, 2024. doi: 10.1162/tacl_a_00638. URL https://aclanthology.org/2024.tacl-1.9.

Ruoyu Qin, Zheming Li, Weiran He, Mingxing Zhang, Yongwei Wu, Weimin Zheng, and Xinran Xu. Mooncake: A kvcache-centric disaggregated architecture for llm serving, 2024. URL https://arxiv.org/abs/2407.00079.

Zhilin Yang, Peng Qi, Saizheng Zhang, Yoshua Bengio, William W. Cohen, Ruslan Salakhutdinov, and Christopher D. Manning. Hotpotqa: A dataset for diverse, explainable multi-hop question answering, 2018. URL https://arxiv.org/abs/1809.09600.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models, 2023. URL https://arxiv.org/abs/2210.03629.