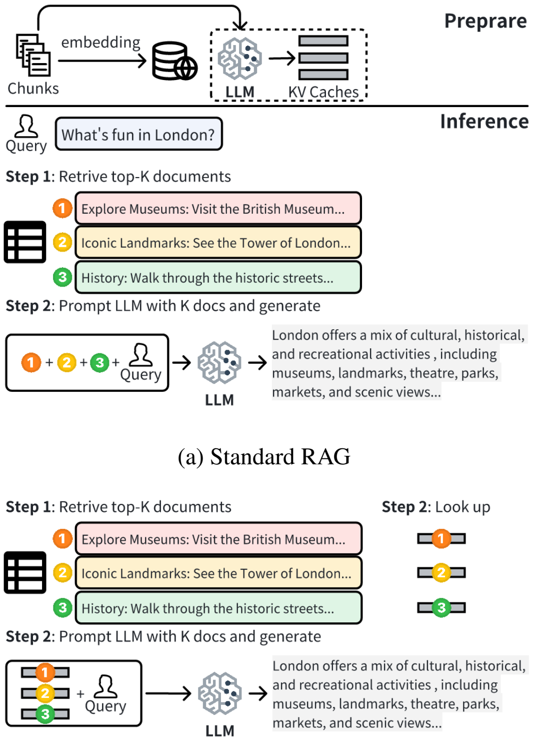
TurboRAG: Accelerating Retrieval-Augmented Generation with Precomputed KV Caches for Chunked Text

Songshuo Lu **Hua Wang** **Yutian Rong** **Zhi Chen** **Yaohua Tang**
Moore Threads AI
lusongshuo97@gmail.com wangtianyu.di@gmail.com
rytapril@foxmail.com cswccz@gmail.com tangyaohua28@gmail.com

Abstract

Current Retrieval-Augmented Generation (*RAG*) systems concatenate and process numerous retrieved document chunks for prefill which requires a large volume of online computation, therefore leading to significant latency in time-to-first-token (*TTFT*). To reduce the computation overhead as well as TTFT, we introduce *TurboRAG*, a *hybrid offline-online* paradigm that **(i)** pre-computes chunk-level key-value (*KV*) caches, **(ii)** stitches them together at inference time using independent-attention and reordered-RoPE techniques, and **(iii)** preserves answer quality without changing the model architecture. Our approach is applicable to most existing large language models and their applications without any requirement in modification of models and inference systems. Experimental results across a suite of RAG benchmarks demonstrate that TurboRAG reduces TTFT by up to 9.4x compared to the conventional RAG systems (on an average of 8.6x), but reserving comparable performance to the standard RAG systems.


Preprare
embedding
Chunks
LLM
KV Caches
Inference
Query
What's fun in London?
Step 1: Retrive top-K documents
1 Explore Museums: Visit the British Museum...
2 Iconic Landmarks: See the Tower of London...
3 History: Walk through the historic streets...
Step 2: Prompt LLM with K docs and generate
1 + 2 + 3 + Query
LLM
London offers a mix of cultural, historical, and recreational activities , including museums, landmarks, theatre, parks, markets, and scenic views...

(a) Standard RAG

Step 1: Retrive top-K documents
Step 2: Look up
1 Explore Museums: Visit the British Museum...
2 Iconic Landmarks: See the Tower of London...
3 History: Walk through the historic streets...
Step 2: Prompt LLM with K docs and generate
1 2 3 + Query
LLM
London offers a mix of cultural, historical, and recreational activities , including museums, landmarks, theatre, parks, markets, and scenic views...

(b) TurboRAG

Figure 1: Pipeline of Standard RAG and TurboRAG. TurboRAG precompute the KV cache for each chunk of text and reuse during RAG inference.

1 Introduction

Retrieval–Augmented Generation (RAG) couples a large language model (LLM) with a dense retriever so that generation can be grounded in external knowledge (Lewis et al., 2020; Chen et al., 2024). While effective, the conventional *concatenate-then-prefill* paradigm imposes three salient bottlenecks:

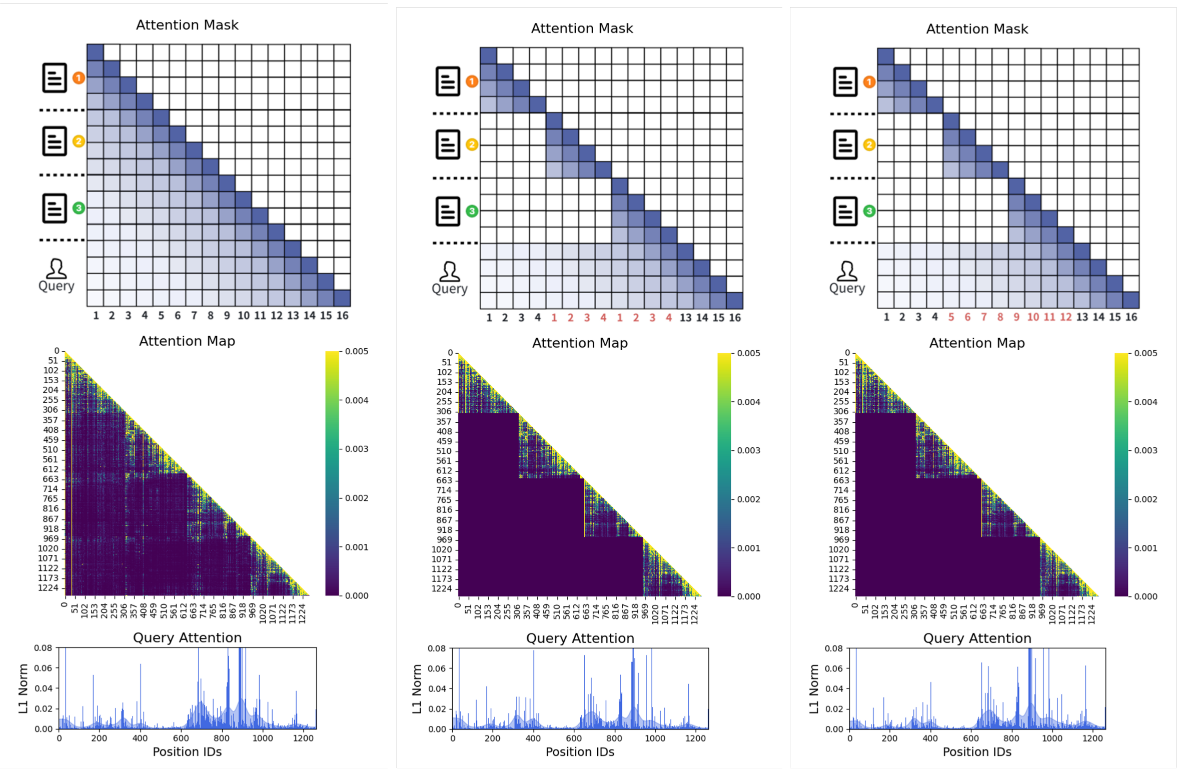
1. **Redundant recomputation.** Frequently retrieved chunks must be re-encoded on every query, duplicating key–value (KV) cache computation.

2. **Quadratic prefill cost.** Concatenating k chunks enlarges the input length by $\mathcal{O}(k)$; self-attention therefore scales quadratically, inflating TTFT and overall latency (Borgeaud et al., 2022).

3. **Restricted batch size.** Long concatenated contexts consume disproportionate GPU memory, limiting per-device batch size and thereby throughput.

These issues stem from the current prefill paradigm that computes a single KV cache for the entire concatenated document set online. A natural question arises: *can we transform prefill into a hybrid offline–online process by precomputing chunk-level KV caches once and reusing them*

Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing, pages 6588–6601
November 4-9, 2025 ©2025 Association for Computational Linguistics



(a) Casual Attention

(b) **Composite Positions**

(c) **Reordered Positions**

Figure 2: The first row presents three distinct setting of attention mask matrices and position IDs. (a) Lower triangular casual attention, where the entire context is attended to. (b) **Independent Attention** and **Composite Positions**, which use the original position IDs for each chunk. (c) **Independent Attention** and **Reordered Positions**, where each document can only attend to itself and rearrange the position IDs for tokens in chunk to standard monotone increasing numbers. In the second and third rows, we present an instance of RAG to visualize and analyze the distribution of the attention matrices under different settings, as well as the distribution of attention scores from the query to the context chunks. This instance consists of four text chunks and a user query, as detailed in Appendix A. In the standard setting shown in the first column of second row, it can be observed that the attention scores between different chunks are quite sparse; each document primarily focuses on its internal information. Furthermore, in the third row, the distribution of attention scores from the query to the context chunks indicates that even when the attention between documents is fully masked, the distribution of attention scores from the query to the documents does not exhibit significant variation, remaining concentrated in the documents that contain relevant information.

across queries? The main technical obstacle is that naively stitching caches produces inconsistent attention masks and position indices, degrading accuracy. To address them, we start from two empirical observations in real-world RAG workloads:

- **Sparse inter-chunk attention.** Figure 2a shows that cross-chunk attention weights are negligible in typical RAG settings. The text contents between most documents are actually independent.
- **RoPE depends only on relative offsets.** For rotary position embeddings (RoPE) (Su et al., 2024), the absolute token index is irrelevant; only pairwise distance matters.

Building on these insights, we propose **TurboRAG**, a hybrid offline–online RAG framework that:

1. precomputes and stores KV caches for each passage offline;
2. retrieves the relevant caches at inference time and stitches them using the independent mask and reordered positions;
3. performs a lightweight supervised fine-tuning so the base LLM can seamlessly consume the new cache layout.

Compared to the conventional RAG system, experimental results across the LongBench multi-document QA benchmarks demonstrate that TurboRAG reduces TTFT by up to **9.4×** (8.6× on average), with comparable accuracy to the baseline. Simultaneously, during online inference, TurboRAG reduces computational resource utilization by **98.5 %** compared to standard RAG, which significantly increases the maximum supported batch size and enhances throughput. Additionally, regression experiments indicate that TurboRAG does not exhibit any significant degradation in other general capabilities compared to standard RAG.

These gains make the method especially appealing in two real-world scenarios: (1) **Large-scale real-time user support**—for example web-based customer service assistants with relatively fixed documents. Because every passage already has a KV cache, user queries achieve a 100 % hit-rate, eliminating redundant prefill computation. TurboRAG can significantly enhance the user experience by optimizing latency and efficiency. (2) **Resource-constrained on-device assistants**—for instance, a personal laptop or edge workstation equipped with a modest, heavily time-shared GPU. Because that accelerator must also serve other local workloads (e.g. IDEs, browsers, video rendering), compute head-room is scarce. By storing chunk-level caches on disk and eliminating prefill FLOPs, TurboRAG keeps TTFT well below one second even when only a small fraction of the GPU is available, delivering smooth interaction without relying on cloud resources.

Contributions

- We design a novel pipeline that decomposes the prefill stage of conventional RAG systems into offline and online phases to notably reduce the overhead of KV cache computation.
- We propose simple yet effective techniques to handle attention mask and position IDs so that model accuracy is maintained.
- We achieve a substantial improvement of 9.4x in TTFT over the state-of-the-art multi-document QA benchmarks without compromising accuracy.

2 Related Work

Retrieval-Augmented Generation (RAG) couples a dense retriever with a large language model (LLM) to ground generation in external knowledge, and has become the de-facto solution for knowledge-intensive NLP tasks (Lewis et al., 2020). Subsequent studies confirm its gains across question answering, code synthesis and creative writing (Borgeaud et al., 2022; Jiang et al., 2024; Trivedi et al., 2022; Ram et al., 2023).

To curb the latency induced by long concatenated contexts, one strand of work reduces the amount of text delivered to the decoder. Sparse RAG prunes low-utility passages with an auxiliary LLM, achieving faster inference without loss of quality (Zhu et al., 2024). Fusion-in-Decoder (FiD) encodes passages independently before decoder-level fusion, thus avoiding quadratic cross-passage attention (Izacard and Grave, 2020). Parallel Context Windows batch sliding windows but rely on heuristic position shifts to maintain coherence (Ratner et al., 2022).

A second line accelerates decoding itself. Linear or sparse-attention Transformers such as Linformer (Wang et al., 2020), Reformer (Kitaev et al., 2020) and Performer (Choromanski et al., 2020) turn the quadratic cost sub-quadratic, though often at some accuracy penalty on multi-document inputs. Complementary techniques compress the key-value (KV) cache on-the-fly: CacheGen encodes caches into compact bitstreams (Liu et al., 2024), H2O evicts low-utility "Heavy-Hitter" tokens (Zhang et al., 2024), and ChunkKV keeps only semantically salient sub-chunks (Liu et al., 2025). Most recently, speculative-decoding variants draft responses with a small model and verify them with a larger model—Speculative RAG (Wang et al., 2024b) and RASD (Quan et al., 2025)—reducing the computational cost of decoding. These speculative methods reduce decoding FLOPs, whereas TurboRAG lowers prefill FLOPs via cache reuse; both directions are orthogonal and could be combined.

A third branch focuses on cache reuse. RAGCache saves query-level KV states but requires an exact context match, leading to low hit rates when passage order changes (Jin et al., 2024). CacheBlend precomputes per-passage caches but must pipeline disk I/O to hide latency (Yao et al., 2025). TurboRAG advances this idea by introducing RoPE-consistent position reordering and an independent-attention mask, enabling order-agnostic cache stitching with negligible accuracy loss.

Concurrent with our work, several accelerators have emerged which are complementary to, or share conceptual underpinnings with, TurboRAG. For instance, Marconi (Pan et al., 2024) focuses on cross-request prefix caching; this is complementary, as a TurboRAG-stitched context for a popular document set can itself be cached by Marconi for a second-level speedup. Similarly, FlashForge (Wang et al., 2025) optimizes decoding-phase attention kernels, which is orthogonal to TurboRAG's prefill savings; the two can be stacked to reduce both TTFT and time-per-output-token. Notably, Block-Attention (Ma et al., 2024), developed independently in the same period, also utilizes chunk-level attention and RoPE re-ordering. This represents a conceptually analogous approach to our independent-attention and reordered-RoPE techniques, validating the efficacy of this strategy for blocked-context processing.

Finally, position-handling techniques such as RoPE extrapolation (Su et al., 2024) and position interpolation (Chen et al., 2023) extend context length during training. TurboRAG achieves low-latency, high-quality answers simply by masking cross-chunk attention and reordering position IDs at inference time. Loading the precomputed passage caches from storage to GPU is much faster than computing long-context prefill. Furthermore, a targeted fine-tune can be added to eliminate the small accuracy bias introduced by the changed attention mechanism.

In summary, prior art either reduces how much text is retrieved, how expensively it is decoded, or reuses caches under strict ordering. TurboRAG contributes a complementary axis—KV caches reuse with correct positional semantics—while remaining compatible with retrieval sparsification, cache compression, and speculative decoding. To the best of our knowledge, this is the first work that redesigns the RAG inference paradigm by transforming the online computation of document KV caches into an offline process. Unlike techniques that accelerate the online prefill computation itself, our hybrid approach aims to eliminate it for retrieved content entirely. This approach significantly reduces the computational complexity of the RAG systems and could become a powerful enabler for LLM applications that have restricted latency constraints.

3 Methodology

This section presents TurboRAG, a novel approach to improve the performance of conventional RAG systems without sacrificing accuracy. We formalize the problem in Section 3.1 and discuss the differences in the attention mask matrix and position IDs between TurboRAG and existing RAG systems in Section 3.2. Section 3.3 explains how we trained the model to adapt to the new attention mask matrix and position IDs. We introduce the TurboRAG inference pipeline in Section 3.4.

3.1 Problem Formalization

Conventionally, given a user query q, we retrieve top k document chunks, $[c_1, \dots, c_k]$, and send them to a LLM that sequentially generates the textual outputs. We denote the number of tokens in x as $\text{len}(x)$ and we assume $len(c_i) = l$. In existing RAG, we first compute the prefill using q and the concatenated c, denoted as a concatenated context sequence $[c_1, \dots, c_k, q]$, to obtain the corresponding hidden states $\boldsymbol{X}^c$. At each decoding step t, the model computes attention scores based on $\boldsymbol{X}^c$. Let $\boldsymbol{X} = [\boldsymbol{X}_1, \boldsymbol{X}_2, \dots, \boldsymbol{X}_t]$ be the hidden states of the tokens generated so far, where $\boldsymbol{X}_t$ is the hidden state for the current token being generated. The model computes the query $\boldsymbol{Q}_t$, key $\boldsymbol{K}_i$, and value $\boldsymbol{V}_i$ matrices for context at position i:

$$\boldsymbol{Q}_t = \boldsymbol{X}_t \boldsymbol{W}_Q, \boldsymbol{K}_i = \boldsymbol{X}_i^c \boldsymbol{W}_K, \boldsymbol{V}_i = \boldsymbol{X}_i^c \boldsymbol{W}_V \quad (1)$$

Here, $\boldsymbol{W}_Q$, $\boldsymbol{W}_K$, and $\boldsymbol{W}_V$ are the learned weight matrices. The attention score is computed using the dot product of the query and the key, scaled by the square root of the dimension of the key vectors d:

$$\text{Attention\_scores} = \frac{\boldsymbol{Q}_t \boldsymbol{K}_i^T}{\sqrt{d}} \quad (2)$$

For RoPE, it is necessary to multiply $\boldsymbol{Q}_t$ and $\boldsymbol{K}_i$ by their corresponding position embedding separately as shown in Equation 3:

$$\boldsymbol{Q}_t' = \begin{pmatrix} q_0 \\ q_1 \\ q_2 \\ q_3 \\ \vdots \\ q_{d-2} \\ q_{d-1} \end{pmatrix} \oplus \begin{pmatrix} \cos t\theta_0 \\ \cos t\theta_0 \\ \cos t\theta_1 \\ \cos t\theta_1 \\ \vdots \\ \cos t\theta_{d/2-1} \\ \cos t\theta_{d/2-1} \end{pmatrix} + \begin{pmatrix} -q_1 \\ q_0 \\ -q_3 \\ q_2 \\ \vdots \\ -q_{d-1} \\ q_{d-2} \end{pmatrix} \oplus \begin{pmatrix} \sin t\theta_0 \\ \sin t\theta_0 \\ \sin t\theta_1 \\ \sin t\theta_1 \\ \vdots \\ \sin t\theta_{d/2-1} \\ \sin t\theta_{d/2-1} \end{pmatrix} \quad (3)$$

where $\theta_m = 10000^{-2m/d}$. A benefit of this equation is that the position embedding for $\boldsymbol{Q}$ and $\boldsymbol{K}$ can be computed independently. Furthermore, the final result of the multiplication of the two position embeddings is solely dependent on the positional difference between them. Since this is an autoregressive model, we need to apply a causal mask to ensure that the model does not attend to future tokens. This is typically achieved by multiplying with a lower triangular masking matrix:

$$\text{Attention\_scores} = \text{Attention\_scores} * \boldsymbol{M} \quad (4)$$

where $\boldsymbol{M}$ is the masking matrix. $\boldsymbol{K}^{'}$ and $\boldsymbol{V}$ are generally referred to as *KV cache*, which is stored for the subsequent computation of attention scores in the later regressive decoding. The attention scores are then normalized using the *softmax* function to obtain attention weights. Finally, the output for the current token is computed as a weighted sum of the value vectors.

3.2 Position ID Rearrangement

This section presents the technique we developed to ensure that the concatenated KV cache computed offline for each document is as effective as the KV cache computed using the whole originally retrieved documents. Figure 2 illustrates the differences in the attention mask matrix and position IDs between the two methods.

The online concatenation of the KV cache requires that there is no cross-attention between multiple document chunks during inference, which is a significant distinction from the lower triangular mask matrix employed by the current RAG system. We denote this new attention modality in Figure 2c as **Independent Attention**, which effectively simulates the scenario of retrieving the KV caches and concatenating them. As illustrated in Figure 2c, cross-attention between documents are all set to zero, and when decoding the answer, attention scores are computed among query, answer and all documents.

Another issue arising from TurboRAG is the computation of position embeddings. The key cache computed for each c_i are denoted as $\boldsymbol{K}^{c_i}$. If the KV caches are simply concatenated, all $\boldsymbol{K}^{c_i}$ will consist of position IDs ranging from 0 to l. Consequently, the finally combined IDs will be represented as $[0, \ldots, l, 0, \ldots, l, 0, \ldots, l]$, which we refer to as **composite positions**. This presents a problem: when decoding at step t, the positional difference between an element in $\boldsymbol{K}^{c_i}$ and t does not correspond to the actual token index difference. For instance, the third element in $\boldsymbol{X}^{c_2}$ at this point has a positional difference of $t-3$, while the actual token index difference should be $t-(l+3)$.

To resolve this issue, we rearrange the positions of all key cache to obtain $[0, \ldots, l, l+1, \ldots, 2l, 2l{+}1, \ldots, k{\cdot}l]$. We refer to this new positions arrangement as **reordered positions**. Session 2 indicates that RoPE encodes "only the relative pair-wise offset" $t-i$, not the absolute position index, so any permutation that preserves those offsets leaves the rotary phase term unchanged. Consequently, once the per-chunk $\boldsymbol{K}, \boldsymbol{V}$ from Equation 1 are saved, we need only re-apply Equation 3 with the new indices to obtain the concatenated $\boldsymbol{K}'$; the corresponding $\boldsymbol{Q}'$ is produced exactly as in standard RAG. Implementation details are given in Appendix B.

However, the new attention mask matrix and position embedding could lead to a accuracy drop in question-answering tasks. To mitigate this issue, we need to specifically train the model to make the LLM be able to handle this new setting. To compare the effects of different positional indices, we will conduct experiments on both **reordered positions** and **composite positions** in Section 4. Next, we will introduce the training details.

3.3 Adapting LLMs for Precomputed Cache Concatenation

Standard supervised fine-tuning (SFT) typically employs the attention mask matrix and position embeddings shown in Figure 2a to fine-tune the LM using the data with the above format. However, to make sure that the pretrained LM can accommodate to new patterns exhibited in the mask matrix and position embedding during inference, TurboRAG used the mask matrix and position embedding in Figure 2b and Figure 2c to fine-tune the LM. After the fine-tuning, the LM would be able to see the same context KV cache produced from training while conducting inference. Therefore, it would not experience the accuracy regression in question-answering tasks.

3.4 The TurboRAG Pipeline

With the fine-tuned LLM, the inference pipeline of TurboRAG is enumerated as follows (Figure 1b):

1. **Document Encoding (offline)**: The documents are encoded into embedding vectors using a transformer-based model like Bert(Devlin et al., 2019). These document embeddings are stored in a vector index to facilitate efficient similarity search.

2. **Document Prefill (offline)**: Use an LLM to perform prefill offline. It computes the KV caches for each document and saves them in the database.

3. **Query Encoding**: The input query is encoded into a vector using the same Bert model.
4. **Retrieval**: The encoded query is used to perform a similarity search in the vector database to retrieve the most relevant documents.
5. **Contextual KV cache Formation (online)**: Retrieve the stored KV cache corresponding to the documents and concatenate them in the way demonstrated in Figure 2. The combined KV cache forms a comprehensive context for the query.
6. **KV Cache Prefill (online)**: The LLM processes prefill using the combined KV caches for the input query.
7. **Response Generation (online)**: After the prefill phase is accomplished, the LLM starts to generate the response and return to the user.

It is evident that the usage process of TurboRAG is fundamentally consistent with that of standard RAG, making it highly convenient to use. We will be releasing the modified implementation code as open source.

4 Experiments

This section evaluates performance and accuracy of a number of TurboRAG model variants against the conventional RAG models. Specifically, we seek to answer the questions below in this section:

- How does TurboRAG perform on document question-answering (QA)?
- What is the overall TTFT performance of TurboRAG compared against the Näive RAG system on popular benchmarks?
- How large is the regression in the general capabilities of TurboRAG models?
- How efficient is TurboRAG in scaling inference batch sizes?

4.1 Experiment Setup

We selected gpt-4o-2024-08-06 as the baseline due to its excellence in many benchmark suites. For brevity, we refer the conventional RAG system as "Naïve RAG". We also fine-tuned two models for TurboRAG, namely TurboRAG-composite and TurboRAG-reordered corresponding to **composite positions** and **reordered positions**, respectively. All three models are fine-tuned on a dataset composed of 50% document QA data and 50% general tasks (e.g., code, dialogue, reasoning). All data are publicly accessible. For a detailed composition of the dataset, please refer to Appendix D.

Training Setup We base our training on Qwen2-7B(Yang et al., 2024), performing SFT on the aforementioned dataset. The fine-tuning was conducted on 32 NVIDIA A100 80GB GPUs with a batch size of 256 samples, using a learning rate of 1e-5 and the AdamW optimizer(Loshchilov, 2017). To quantify the overhead, this entire SFT process completes in approximately 888 GPU-hours. At current on-demand cloud rates, this represents a one-time cost under $3,000 USD, a manageable figure comparable to standard adaptation costs for 7B-parameter models. Both Naïve RAG and TurboRAG models were trained using the same data proportions to ensure comparability.

4.2 Document QA Accuracy

Let's first evaluate the accuracy of document QA via intensive study on RGB Benchmark(Chen et al., 2024), a bilingual benchmark designed to test a model's ability to answer questions on retrieved documents. We followed the testing methodology provided by the official guidelines and let each query extract five documents during the evaluation. In addition, we also measured the accuracy with varying noise levels from 0.2 to 0.8 (e.g., *Noise Ratio* = 0.6 means 3 out of 5 retrieved documents are irrelevant or noisy). In order reveal the effectiveness of fine-tuning, we gauged accuracy of each TurboRAG configuration with and without fine-tuning.

As Table 1 shows, TurboRAG-reordered is robust even without any fine-tuning: its average accuracy falls by only 4.2% (92.6 vs 96.8) and never more than 6% even at the highest noise ratio 0.8, so it can be used out-of-the-box in many applications. By contrast, TurboRAG-composite incurs a larger drop (5.8%) and nearly 20% as the task difficulty increases. Because the relative offsets that RoPE relies on are no longer preserved with duplicated position IDs. After fine-tuning, TurboRAG-reordered and TurboRAG-composite can effectively maintain the benchmark accuracy gap within 1% compared to the Naïve RAG. They also demonstrated comparable performance to GPT-4o across both Chinese and English datasets even under high-noise conditions. This highlights the effectiveness of the proposed modifications in preserving high accu-

Model	Chinese					English				
	0.2	0.4	0.6	0.8	Avg.	0.2	0.4	0.6	0.8	Avg.
GPT-4o-2024-08-06	98.3	98.0	96.6	87.7	95.2	99.0	99.3	98.3	96.3	98.2
Naïve RAG	99.0	98.0	96.7	87.3	95.3	99.7	99.3	99.3	94.3	98.2
TurboRAG-composite w/o fine-tuning	98.3	96.3	93.7	79.0	91.8	98.0	96.3	91.3	75.0	90.2
TurboRAG-reordered w/o fine-tuning	98.0	96.7	93.3	81.3	92.3	98.0	97.3	90.7	85.7	92.9
TurboRAG-composite	99.0	97.3	96.0	86.7	94.8	99.3	98.0	96.7	92.7	96.7
TurboRAG-reordered	98.7	97.3	96.0	90.7	**95.7**	99.0	98.3	96.0	93.7	**96.8**

Table 1: Performance comparison of different models under various noise ratios in English and Chinese in RGB.

racy when leveraging KV cache in document QA tasks. Additional experimental data on RGB can be found in Table 2, which also includes details on the multi-document integration tasks in the RGB dataset. The results show that even for queries requiring information synthesis across multiple documents, TurboRAG-reordered achieves accuracy comparable to that of Näive RAG. These results confirm that (i) the reordered-position scheme is immediately usable, and (ii) a lightweight SFT step suffices to eliminate any residual gap for either masking strategy.

Chinese				
Model	0.2	0.4	0.6	Avg.
Naïve RAG	50	46	29	42
TurboRAG-composite w/o fine-tuning	35	27	18	27
TurboRAG-reordered w/o fine-tuning	30	21	20	24
TurboRAG-composite	53	41	32	42
TurboRAG-reordered	56	44	32	44

English				
Model	0.2	0.4	0.6	Avg.
Naïve RAG	57	48	36	47
TurboRAG-composite w/o fine-tuning	40	27	27	31
TurboRAG-reordered w/o fine-tuning	31	23	19	24
TurboRAG-composite	58	48	34	47
TurboRAG-reordered	57	51	34	47

Table 2: Performance comparison of different models under various noise ratios in RGB Information Integration Task.

To validate that our method proposed techniques are also directly applicable to long text input cases, we inspected TurboRAG's accuracy on an additional long-text RAG benchmark dataset, LongBench(Bai et al., 2023). As shown in Table 3, TurboRAG also exhibits comparable answer accuracy to that of Naïve RAG in such use scenarios.

In all experiments, the performance of TurboRAG-composite was consistently inferior to that of TurboRAG-reordered, particularly in more challenging contexts such as LongBench. This observation further validates the necessity of maintaining the accuracy of relative positional differences in positional encoding.

To further validate the generality of our approach, we conducted identical experiments on LLaMA-3.1-8B(Dubey et al., 2024). As shown in Appendix F, the results are consistent, confirming the effectiveness of our method across RoPE-based models.

4.3 General Capability Regression

To ensure that the non-standard attention masks and position IDs usded in fine-tuning does not negatively affect the models' general capabilities, we accomplished regression tests using the OpenCompass[1] benchmark on various mainstream tasks. As summarized in Table 4, the modifications had minimal impact on the base capabilities of the models. TurboRAG-reordered showed strong generalization across tasks, with no significant performance degradation compared to Naïve RAG.

4.4 TTFT Performance

Now we assess the impact of TurboRAG on inference speed. All models are evaluated on the

[1] https://github.com/open-compass/opencompass

Subcategory (Metric)	**Context Token**	**Query Token**	**Score**			**TTFT (ms)**		
			Naïve	Turbo Composite	Turbo Reordered	Naïve	Turbo Reordered	Speedup
MuSiQue (F1)	16349	18.8	22.12	23.64	27.37	1610	171	**9.4x**
2WikimQA (F1)	7553	17.0	35.02	34.28	39.51	709	101	**7.0x**
DuReader (Rouge-L)	10642	6.0	34.57	33.37	33.03	1007	116	**8.7x**
HotpotQA (F1)	13453	20.1	40.21	35.78	45.28	1333	147	**9.1x**
Avg.	11999	15.5	32.99	31.76	**36.29**	1165	**134**	**8.6x**

Table 3: Performance of Naive RAG and TurboRAG on LongBench multi-document QA (subcategories).

Metric	Naïve RAG	TurboRAG -reordered	Sub
MMLU	69.57	70.73	+1.16
TriviaQA	56.90	56.47	-0.43
GSM-8K	79.12	79.45	+0.33
MATH	39.54	40.58	+1.04
HumanEval	58.26	57.32	-0.94
AlpacaEval2	7.83	8.32	+0.49

Table 4: Regression experiments of Naïve RAG and TurboRAG. Evaluated by OpenCompass.

LongBench dataset, with specific focus on its multi-document QA tasks. The experiments were conducted on the Huggingface *transformers*[2] using FlashAttention2(Dao, 2023) and an NVIDIA A100 80GB GPU. As shown in Table 3, TurboRAG-reordered improves the performance of TTFT by 8.6x on average, with a peak speedup of 9.4x, compared to Naïve RAG for long-documents processing. This reduction substantiates that TurboRAG can significantly reduce TTFT, thereby enhancing user experience, and consequently enables the expansion of RAG applications to cases with stringent latency requirement. The main reason of reduction in the TTFT is that the online computation overhead of KV caches for long text is largely alleviated as TurboRAG shifts the KV cache computation for each document to offline processing. Table 10 in Appendix E shows that TurboRAG can still achieve 2.42x speedup on short documents.

4.5 Batch Scaling

Compared to Naïve RAG, TurboRAG requires to transfer KV cache from CPU to GPU, which may introduce extra communication overhead that degrades performance measured by TTFT. To evaluate the magnitude of the communication cost, we carried out experiments under a fixed total recall text length of 8192 and a query length of 128. We gathered a series of TTFT numbers with batch size ranging from 1 to 8 in two settings. One transferred the KV cache from CPU to GPU using PCIE Gen4, while the other assumed that the KV cache was prefetched to the GPU memory thereby excluding the impact of communication. Additionally, we measured the computational load for both Naïve RAG and TurboRAG under different settings. The method for measuring is detailed in Appendix G.

Batch Size	**Metric**	**Naïve RAG**	**Turbo RAG**	**Speed-up**	**Turbo w/o h2d**	**Speedup w/o h2d**
1	TTFT (ms) TFLOPs	711 136.36	175 2.09	**4.1x**	44 2.09	**16.1x**
2	TTFT (ms) TFLOPs	1408 272.72	325 4.19	**4.3x**	56 4.19	**25.1x**
4	TTFT (ms) TFLOPs	2842 545.46	666 8.39	**4.3x**	97 8.39	**29.3x**
6	TTFT (ms) TFLOPs	4373 818.20	928 12.58	**4.7x**	134 12.58	**32.6x**
8	TTFT (ms) TFLOPs	5812 1090.93	1429 16.78	**4.1x**	177 16.78	**32.8x**

Table 5: Generation throughput and latency.

From Table 5, it is evident that as the batch size increases, the speedup ratio (decrease in TTFT) also increases without any degradation in performance. When the batch size is small, the pressure on computational resources is insufficient, resulting in a TTFT speedup value of only 16.1x between Naïve RAG and TurboRAG. As the batch size increases, GPU becomes over-utilized for naive RAG, thus leading to substantially higher latency in TTFT compared to TurboRAG. Table 5 also illustrates that, even in scenarios requiring the transfer of the KV cache from host to device (h2d),

[2] https://huggingface.co/

Seq Length	Query Length	Batch Size	Naïve	Turbo
256	128	1	44.00	41.62
256	128	2	68.19	195.96
256	128	4	127.19	165.73
256	128	8	242.31	120.62
512	128	1	59.16	37.16
512	128	2	101.84	47.58
512	128	4	205.61	133.14
512	128	8	398.18	179.94
1024	128	1	97.89	48.79
1024	128	2	186.02	89.08
1024	128	4	359.95	139.70
1024	128	8	711.19	189.81

Table 6: TTFT (ms) for different context lengths and batch sizes on an A100 GPU.

Model Size	Batch Size	Naïve TTFT(ms)	Turbo TTFT(ms)	Δ (ms)	Speedup
1.5B	1	197	85	112	2.31
	2	377	88	289	4.28
	4	742	150	592	4.94
	8	1479	292	1187	5.06
3B	1	363	96	267	3.78
	2	715	117	598	6.11
	4	1417	199	1218	7.12
	8	2855	491	2364	5.81
14B	1	1413	272	1141	5.19
	2	2921	549	2372	5.32
	4	5852	1022	4830	5.72
	8	11888	2452	9436	4.84
32B (2 x GPUs)	1	2923	383	2540	7.63
	2	5766	711	5055	8.10
	4	11558	1319	10239	8.76
	8	23205	2884	20321	8.04
72B (4 x GPUs)	1	6157	653	5504	9.42
	2	12349	1099	11250	11.23
	4	24840	2333	22507	10.64
	8	50595	5600	44995	9.03

Table 7: TTFT for different model sizes of Qwen-2.

TurboRAG still achieves a fourfold speed improvement compared to Naïve RAG. In addition to latency, we analyzed the computational overhead in terms of TFLOPs. Our measurements show that TurboRAG achieves an approximate 98.46% reduction in TFLOPs compared to naïve RAG, highlighting its profound efficiency.

We provide detailed empirical results in Table 6, which includes TTFT comparisons for shorter context lengths. Furthermore, Table 7 demonstrates that TurboRAG's speedup is consistent across various scales of the Qwen-2 model family, from 1.5B to 72B parameters. Additionally, if each text chunk contains 200 tokens, recalling 5 segments results in a total of 1000 tokens. According to the experimental results, even with a batch size of 1, a commendable speedup of up to two times can be achieved.

5 Conclusion and Discussion

This paper presented a novel approach to training and utilizing RAG that significantly reduces the time required for prefill computations. Our method, which involves pre-computing and concatenating the KV caches of retrieved text fragments without inter-document cross-attention, effectively streamlines a critical bottleneck in the generation pipeline. It is worth noting that other optimization techniques, such as KV cache compression, are orthogonal to our method and can be used in conjunction to further reduce latency and ease storage pressure. Our work raises a fundamental question about the necessity of computationally intensive cross-attention between different fragments, suggesting that direct query-document interactions are often sufficient for effective context aggregation. Furthermore, our findings offer insights into the adaptability of LLMs to varied positional embeddings, drawing a parallel to context-extension techniques and underscoring the models' flexibility.

Beyond these architectural insights, a practical deployment must address systems-level challenges. For instance, managing dynamic knowledge bases could involve strategies such as versioned cache entries, an incremental offline pipeline for asynchronous updates, and a graceful fallback to online-prefill to ensure high availability. Storage pressure could be further managed by using 4/8-bit quantization, yielding a 2-4x reduction in footprint. The design of a high-throughput KV-cache store, another critical component, would likely leverage technologies such as fast local NVMe SSDs for storage, lightweight LSM-tree indexing for sub-millisecond lookups, and bundling chunks to optimize DMA transfers into GPU memory. In multi-node setups, frameworks like vLLM and technologies like GPUDirect-RDMA could be used to further manage memory and hide network latency. For future work, a systematic study could explore the performance boundaries of this approach across different task domains.

Limitations

This section discusses some limitations this paper has that we intentionally leave as the future work to further improve.

Limitation 1: **Storage overhead.** TurboRAG deliberately trades space for latency. Taking Qwen2-7B as an example, a 512-token chunk produces a FP16 KV cache of $2 \times 2 \times 28 \times 8 \times 128 \times 512 = 28$ MB, so caching one million chunks needs $\sim$28 TB of disk. Current nearline HDDs cost roughly \$10-\$20 per TB [3], meaning the entire 28 TB repository is well under \$600—an order of magnitude cheaper than the GPUs ordinarily provisioned to meet the same sub-second latency target. Moreover, most practical deployments cache far fewer passages: an internal customer-service knowledge base or a personal laptop's local archive typically contains $10^4 - 10^5$ passages, translating to only 0.3-3 TB (or 40-400 GB after 8-bit KV quantisation). Only at web-scale corpora with tens of billions of passages would storage grow beyond a few petabytes and call for additional strategies such as tiered object storage or aggressive cache compression; for the vast majority of latency-sensitive workloads, the modest disk budget is amply justified by the multi-fold reduction in TTFT. Besides, we have noticed an increasing number of works to handle KV cache compression (Wang et al., 2024a; Liu et al., 2024; Zhang et al., 2024), which can effectively reduce the storage requirements and are orthogonal to our work. Integrating these KV cache compression techniques into TurboRAG will be our next direction of work. Beyond disk storage, the process of loading the KV cache from disk to memory in TurboRAG also puts pressure on memory usage.

Limitation 2: **Model fine-tuning.** Another Issue is that the current pipeline still requires fine-tuning of the model, which limits its applicability and prevents it from being directly used on newly emerging state-of-the-art LLMs. We are currently exploring ways to reduce or even eliminate this dependency on fine-tuning.

References

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

Sebastian Borgeaud, Arthur Mensch, Jordan Hoffmann, Trevor Cai, Eliza Rutherford, Katie Millican, George Bm Van Den Driessche, Jean-Baptiste Lespiau, Bogdan Damoc, Aidan Clark, et al. 2022. Improving language models by retrieving from trillions of tokens. In *International conference on machine learning*, pages 2206–2240. PMLR.

Jiawei Chen, Hongyu Lin, Xianpei Han, and Le Sun. 2024. Benchmarking large language models in retrieval-augmented generation. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pages 17754–17762.

Shouyuan Chen, Sherman Wong, Liangjian Chen, and Yuandong Tian. 2023. Extending context window of large language models via positional interpolation. *arXiv preprint arXiv:2306.15595*.

Krzysztof Choromanski, Valerii Likhosherstov, David Dohan, Xingyou Song, Andreea Gane, Tamas Sarlos, Peter Hawkins, Jared Davis, Afroz Mohiuddin, Lukasz Kaiser, et al. 2020. Rethinking attention with performers. *arXiv preprint arXiv:2009.14794*.

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

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. Bert: Pre-training of deep bidirectional transformers for language understanding. *Preprint*, arXiv:1810.04805.

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

Gautier Izacard and Edouard Grave. 2020. Leveraging passage retrieval with generative models for open domain question answering. *arXiv preprint arXiv:2007.01282*.

Wenqi Jiang, Shuai Zhang, Boran Han, Jie Wang, Bernie Wang, and Tim Kraska. 2024. Piperag: Fast retrieval-augmented generation via algorithm-system co-design. *arXiv preprint arXiv:2403.05676*.

Chao Jin, Zili Zhang, Xuanlin Jiang, Fangyue Liu, Xin Liu, Xuanzhe Liu, and Xin Jin. 2024. Ragcache: Efficient knowledge caching for retrieval-augmented generation. *arXiv preprint arXiv:2404.12457*.

Nikita Kitaev, Łukasz Kaiser, and Anselm Levskaya. 2020. Reformer: The efficient transformer. *arXiv preprint arXiv:2001.04451*.

Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, et al. 2020. Retrieval-augmented generation

[3] A 16 TB Exos X18 enterprise drive lists for \$210 on Amazon (\$13/TB), while an 18 TB WD Red Pro sells for about \$350 (\$19/TB).

for knowledge-intensive nlp tasks. *Advances in Neural Information Processing Systems*, 33:9459–9474.

Xiang Liu, Zhenheng Tang, Peijie Dong, Zeyu Li, Bo Li, Xuming Hu, and Xiaowen Chu. 2025. Chunkkv: Semantic-preserving kv cache compression for efficient long-context llm inference. *arXiv preprint arXiv:2502.00299*.

Yuhan Liu, Hanchen Li, Yihua Cheng, Siddhant Ray, Yuyang Huang, Qizheng Zhang, Kuntai Du, Jiayi Yao, Shan Lu, Ganesh Ananthanarayanan, et al. 2024. Cachegen: Kv cache compression and streaming for fast large language model serving. In *Proceedings of the ACM SIGCOMM 2024 Conference*, pages 38–56.

I Loshchilov. 2017. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*.

Dongyang Ma, Yan Wang, and Lan Tian. 2024. Block-attention for efficient prefilling. *arXiv preprint arXiv:2409.15355*.

Rui Pan, Zhuang Wang, Zhen Jia, Can Karakus, Luca Zancato, Tri Dao, Yida Wang, and Ravi Netravali. 2024. Marconi: Prefix caching for the era of hybrid llms. *arXiv preprint arXiv:2411.19379*.

Guofeng Quan, Wenfeng Feng, Chuzhan Hao, Guochao Jiang, Yuewei Zhang, and Hao Wang. 2025. Rasd: Retrieval-augmented speculative decoding. *arXiv preprint arXiv:2503.03434*.

Ori Ram, Yoav Levine, Itay Dalmedigos, Dor Muhlgay, Amnon Shashua, Kevin Leyton-Brown, and Yoav Shoham. 2023. In-context retrieval-augmented language models. *Transactions of the Association for Computational Linguistics*, 11:1316–1331.

Nir Ratner, Yoav Levine, Yonatan Belinkov, Ori Ram, Inbal Magar, Omri Abend, Ehud Karpas, Amnon Shashua, Kevin Leyton-Brown, and Yoav Shoham. 2022. Parallel context windows for large language models. *arXiv preprint arXiv:2212.10947*.

Jianlin Su, Murtadha Ahmed, Yu Lu, Shengfeng Pan, Wen Bo, and Yunfeng Liu. 2024. Roformer: Enhanced transformer with rotary position embedding. *Neurocomputing*, 568:127063.

Harsh Trivedi, Niranjan Balasubramanian, Tushar Khot, and Ashish Sabharwal. 2022. Interleaving retrieval with chain-of-thought reasoning for knowledge-intensive multi-step questions. *arXiv preprint arXiv:2212.10509*.

Sinong Wang, Belinda Z Li, Madian Khabsa, Han Fang, and Hao Ma. 2020. Linformer: Self-attention with linear complexity. *arXiv preprint arXiv:2006.04768*.

Zheng Wang, Boxiao Jin, Zhongzhi Yu, and Minjia Zhang. 2024a. Model tells you where to merge: Adaptive kv cache merging for llms on long-context tasks. *arXiv preprint arXiv:2407.08454*.

Zhibin Wang, Rui Ning, Chao Fang, Zhonghui Zhang, Xi Lin, Shaobo Ma, Mo Zhou, Xue Li, Zhongfeng Wang, Chengying Huan, et al. 2025. Flashforge: Ultra-efficient prefix-aware attention for llm decoding. *arXiv preprint arXiv:2505.17694*.

Zilong Wang, Zifeng Wang, Long Le, Huaixiu Steven Zheng, Swaroop Mishra, Vincent Perot, Yuwei Zhang, Anush Mattapalli, Ankur Taly, Jingbo Shang, et al. 2024b. Speculative rag: Enhancing retrieval augmented generation through drafting. *arXiv preprint arXiv:2407.08223*.

An Yang, Baosong Yang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Zhou, Chengpeng Li, Chengyuan Li, Dayiheng Liu, Fei Huang, et al. 2024. Qwen2 technical report. *arXiv preprint arXiv:2407.10671*.

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

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

Yun Zhu, Jia-Chen Gu, Caitlin Sikora, Ho Ko, Yinxiao Liu, Chu-Cheng Lin, Lei Shu, Liangchen Luo, Lei Meng, Bang Liu, et al. 2024. Accelerating inference of retrieval-augmented generation via sparse context selection. *arXiv preprint arXiv:2405.16178*.

A Document Q&A Example

Query	When is the premiere of ’Carole King & James Taylor: Just Call Out My Name’?
Document 1	Duke capped off a remarkable season by beating UCF 30-13 on Wednesday in the Military Bowl - the program’s first bowl win since 2018. With the win, Duke got to nine wins for the first time since 2014. Mike Elko has done one of the best coaching jobs in the country in his first season with the Blue Devils. The program was barely competitive in David Cutcliffe’s final seasons on the job, going a combined 5-18 (1-17 ACC) in his final two years. With Wednesday’s win, Duke finished the season 9-4 overall with a 5-3 mark in ACC play. It was just the third season in school history that the Blue Devils had finished with a winning conference record and won a bowl game. Washington: After going 4-8 in 2021, Washington capped off a tremendous turnaround by beating Texas 27-20 in the Alamo Bowl. With the win, Washington finished the season with 11 wins - the most it has had in a season since 2016. That’s the year the Huskies reached the College Football Playoff...
Document 2	Personal Preference is a 1987 board game created by Donal Carlston that involves guessing the order in which a player prefers foods, activities, people, and other items compared to one another. The game was published by Broderbund in the United States, Playtoy Industries in Canada, and Parker Brothers International in Britain. An updated version by the original creator was launched on Kickstarter on May 1, 2023. The new version contains updated cultural references and new categories. The game contains cards in four categories: Food & Drink, Activities, People, and Potpourri (miscellaneous). Each card has a photo or drawing on each side and text indicating what that side represents (e.g., chocolate éclairs, climbing a mountain, Harrison Ford, spy novels). Each round, one player draws four cards from one category, or one from each category, depending on the player’s position on the board...
Document 3	However, the concert tour took place in honor of the 40th anniversary. The two might have aged since they first performed together but neither Carole King nor James Taylor have lost a beat in all these years!The concert film includes the following songs:(You Make Me Feel Like) A Natural WomanSomething in the Way She MovesSo Far Away-Carolina in My MindCountry RoadSmackwater JackWhere You Lead (lyrics changed up as the city they’re playing in replaces New York)Your Smiling FaceBeautifulShower The PeopleWay Over YonderSweet Baby James (this kicks off the second half of the film)Up on the RoofIt’s Too LateFire and RainI Feel the Earth MoveYou’ve Got a FriendHow Sweet It Is (To Be Loved by You)You Can Close Your EyesMexico (end credits)DIRECTOR: Frank MarshallFEATURING: Carole King, James Taylor, Danny Kortchmar, Peter Asher, Russ Kunkel, Leland SklarADDITIONAL MUSICIANS: Andrea Zonn, Arnold McCuller, Kate Markowitz, Robbie KondorCarole King & James Taylor: Just Call Out My Name premiered January 2, 2022, at 9:00pm ET/PT on CNN...
Document 4	I was also raised to see the correlation between life and the game of football and how the process of preparation leads to success in both.” Jason earned a bachelors in history, government and philosophy at Adams State in 2005, and a masters in criminal justice administration from the University of Phoenix in 2007. He added a second master’s in educational methods from the University of Tulsa in 2012. He was a defensive coordinator at the University of Montana, a co-defensive coordinator at Adams State, a defensive coordinator at Valdosta State and the Colorado School of Mines, a defensive advisor at Temple University, served as a defensive assistant at Oklahoma State for two years - after a two-season stay with fellow FBS program Tulsa as outside linebackers coach...

B Positional-Encoding Options in TurboRAG

In our paper, we discuss two approaches for handling positional encodings in TurboRAG: Composite Positions and Reordered Positions:

Composite Positions. This approach applies RoPE during the precomputation of each chunk's key-value cache, making it straightforward to implement.

Reordered Positions. Here, RoPE is applied at inference time instead of during precomputation. The additional computational cost is negligible, as shown in the pseudocode from Qwen-2. The only additional step is the single call to `apply_single_rotary_pos_emb` on the stitched `key_states`.

```
# query_states, key_states =
     apply_rotary_pos_emb(query_states,
     key_states, cos, sin, position_ids)
query_states = apply_single_rotary_pos_emb(
     query_states, cos, sin, position_ids)
if past_key_value is not None:
    # ...
    cache_kwargs = {"sin": sin, "cos": cos} #
         Specific to RoPE models
    key_states, value_states = past_key_value.
         update(key_states, value_states, self.
         layer_idx, cache_kwargs)

    full_position_ids = torch.arange(
            0, past_key_value.seen_tokens, dtype=
                 torch.long, device=query_states.
                 device
        )
    full_position_ids = full_position_ids.
         unsqueeze(0)
else:
    full_position_ids = position_ids

key_states = apply_single_rotary_pos_emb(
     key_states, cos, sin, full_position_ids)
```

C Data Format

You are an accurate and reliable AI assistant capable of answering questions by referencing external documents. Please note that the external documents may not always be related to the question. The documents are as follows:
`<|doc_start|>{chunk_1}<|doc_end|>`
`<|doc_start|>{chunk_2}<|doc_end|>`
`<|doc_start|>{chunk_3}<|doc_end|>`
...
If the information in the documents contain the correct answer, you will provide an accurate response. If the documents do not contain the answer, you will refuse to answer.

Question: `{que}`

D Data Proportions

Data Type	Sampling Ratio
Document Q&A	50%
General Dialogue	25%
Reasoning	10%
Code	10%
Others	5%

Table 8: Sampling Ratios of Different Data Types during Model Fine-tuning

Data Name	Language	Quantity
glave-rag-v1	English	51,153
MS Marco	English	10,000
HotpotQA	English	17,796
BaiduSTI	Chinese	4,032
DuReader	Chinese	30,000
BaiduBaike	Chinese	13,615
Wiki	Chinese	9,265

Table 9: Specific Data and Quantities of Document Q&A

E Supplementary Information for RGB

Model	Context Tokens	TTFT (ms)	Speedup
Naïve RAG	743	87	**2.42x**
TurboRAG		36	

Table 10: Comparison of TTFT in RGB for Naïve RAG and TurboRAG.

F LLaMA-3.1-8B Experimental Results

In this section, we present the experimental results on the LLaMA-3.1-8B model, further validating the effectiveness of our method across RoPE-based models.

Chinese					
Model	0.2	0.4	0.6	0.8	Avg.
Naïve RAG	99.7	99.0	98.0	91.7	97.1
TurboRAG -reordered	99.0	97.7	95.3	88.0	95.0
English					
Model	0.2	0.4	0.6	0.8	Avg.
Naïve RAG	99.0	99.0	99.0	96.7	98.4
TurboRAG -reordered	99.0	97.7	96.7	96.0	97.4

Table 11: Performance comparison of different models based on LLaMA-3.1-8B under various noise ratios in English and Chinese in RGB.

Chinese				
Model	0.2	0.4	0.6	Avg.
Naïve RAG	61	50	44	51.7
TurboRAG-reordered	56	49	40	48.3
English				
Model	0.2	0.4	0.6	Avg.
Naïve RAG	71	64	52	62.3
TurboRAG-reordered	68	66	52	62.0

Table 12: Performance comparison of different models based on LLaMA-3.1-8B under various noise ratios in RGB Information Integration Task.

G Computational Load Calculation

Here, we present the method for calculating the FLOPS, while omitting the computation of the `lm_head` (due to its relatively small proportion). Let n_{input} denote the number of input tokens and n_{context} the context length. For a large language model (LLM) employing the Swiglu activation function, the key parameters are:

$$L, \quad H, \quad K, \quad d_h, \quad d, \quad d_{\text{mlp}},$$

where

- L is the number of layers,
- H is the number of attention heads,
- K is the number of key-value heads,
- d_h is the head dimension,
- d is the hidden size, and
- d_{mlp} is the intermediate (MLP) size.

For each token, the per-layer computational costs are defined as follows:

1. **QKV Transformation:**

   The cost C_{qkv} is given by

   $$C_{\text{qkv}} = 2\,d\,\Big(H + 2K\Big)\,d_h.$$

2. **Attention Mechanism:**

   The cost C_{attn} is expressed as

   $$C_{\text{attn}} = 2\,H\,d_h\,n_{\text{context}}.$$

3. **Output Projection:**

   The cost C_o is given by

   $$C_o = 2\,d^2.$$

4. **Multilayer Perceptron (MLP):**

   The cost C_{mlp} can be represented as

   $$C_{\text{mlp}} = 6\,d\,d_{\text{mlp}}.$$

Therefore, the total computational cost (FLOPS) is expressed as:

$$\text{FLOPS} = n_{\text{input}}\,L\,\Big(C_{\text{qkv}} + C_{\text{attn}} + C_o + C_{\text{mlp}}\Big).$$