Fast LLM Inference with Parallel Prompting

Anonymous ACL submission

Abstract

This paper presents a new method for efficiently decoding multiple queries over the same content in Transformer language models. This is particularly useful for tasks that have many prompts with the shared prefix, as document question answering with a large number of questions for each document. Traditional methods prompt the language model with each query independently in a batch or combine multiple questions together into one larger prompt. However, both approaches are based on the autoregressive fashion with one token per homogeneous forward pass, which uses inefficient matrix-vector products for every sequence in the batch. These methods also encounter issues such as a duplicate key-value cache, quality degradation, or redundant memory when large key-value (KV) caches are accessed from memory, which leads to wasted GPU memory and decreased performance. Our proposed method addresses these challenges by decoding queries in parallel, replacing matrix-vector products with more efficient matrix-matrix products, improving efficiency without compromising result quality. Experimental results demonstrate that our method increases throughput effectively in multiple downstream tasks, providing a reliable solution for prompt inference in language models.

1 Introduction

As transformer-based large language models (LLMs) (Vaswani et al., 2023) are deployed at increasingly large scales, optimizing the inference has been a key focus for many recent works such as FlashAttention (Dao et al., 2022), speculative decoding (Chen et al., 2023) and multi-token prediction (Gloeckle et al., 2024). As research continues to expand its capabilities and applications, the importance of efficiency in LLM inference becomes increasingly critical.

The remarkable ability of LLMs has led to their widespread adoption across various domains (Zhou et al., 2024; Yuan et al., 2024; Miao et al., 2023a). As a result, while LLMs are increasingly deployed in environments demanding high reliability such as in healthcare (Qureshi et al., 2023), legal interpretations (Sun, 2023), finance (Wu et al., 2023), education (Kasneci et al., 2023), and code assistant (Chen et al., 2021) settings, the ability to streamline processing while maintaining accuracy becomes paramount (Hadi et al., 2023; Zhou et al., 2024; Yuan et al., 2024; Miao et al., 2023a).

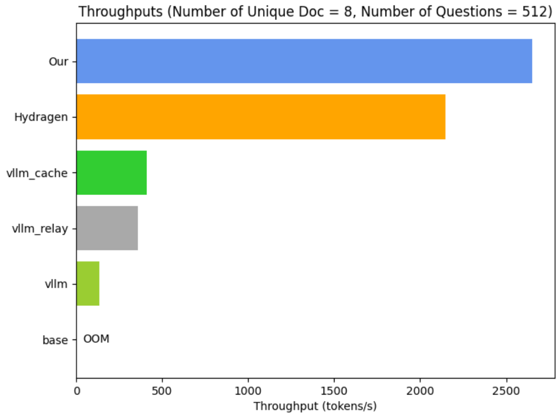

Figure 1: CodeLlama-7b-Instruct attention inference Throughput w.r.t. number of unique documents (A100-SXM4-80GB GPU). We set the length of content to 256, the number of total queries is 512, for each unique content is 64, the length of each query to 12, the length of generated token to 5.

In many applications, tasks often involve multiple queries over the same content. This scenario is prevalent in fields such as education, medical care (Qureshi et al., 2023), and legal consulting (Sun, 2023), where LLMs must be queried multiple times over the same content. The motivation for developing and refining LLMs to handle this scenario is rooted in the practical demands and efficiency required across several critical fields. In education, for instance, students might query an

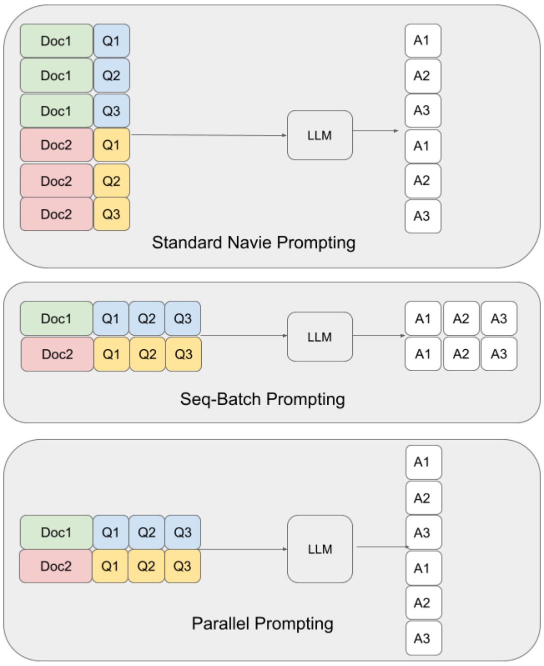

Figure 2: Prompting methods of LLM.

LLM multiple times to gain deeper insights into a particular topic or to understand complex concepts through varied perspectives. This facilitates an enriched learning experience, allowing users to engage more thoroughly with the content without starting from scratch for each new question. In the medical field, consistency and continuity in information are vital. Doctors, nurses, and medical researchers may need to run successive queries on patient data or medical literature to make informed decisions, diagnose conditions, or explore treatment options. Ensuring the LLM can process these queries efficiently and contextually aware can significantly streamline workflows, reduce errors, and ultimately enhance patient care. Legal consulting requires navigating complex, often massive, bodies of text. Legal professionals frequently need to parse through large documents and discuss various aspects of a legal case in a precise and consistent manner. Leveraging LLMs to handle multiple queries over shared contexts can save significant time and reduce the cognitive load on legal practitioners, allowing them to focus on nuanced legal strategies and client interactions. An LLM capable of processing these tasks across a static shared context can refine its responses, offering more precise and relevant answers, which is particularly beneficial in dynamic fields where real-time information processing is critical.

Improving the efficiency of prompting with shared content for LLMs can have a significant impact. With growing demand comes the necessity for LLMs to efficiently handle long prompts containing more shared content, many recent works focus on optimizing LLM inference in this scenario, such as RelayAttention (Zhu et al., 2024), PromptCache (Gim et al., 2024), Hydragen (Juravsky et al., 2024). Many LLM serving systems such as vLLM (Kwon et al., 2023) and SGLang (Zheng et al., 2024) also optimize the inference in this scenario by caching the previous queries.

Overall, the ability of LLMs to seamlessly handle multiple queries over the same content enhances their utility, efficiency, and reliability, making them indispensable tools across various professional fields. This capability not only optimizes the user experience by maintaining context and continuity but also expands the potential applications of LLMs in solving complex, real-world problems.

2 Patterns of prompting

The traditional inference process of LLMs in the scenario poses limitations due to its autoregressive nature. The naive approach is to either prompt the LLM with each prompt independently in a batch or to combine them all into one bigger prompt(Cheng et al., 2023; Lin et al., 2023). Both approaches do not exploit the parallel capabilities of a GPU in the generation stage due to the fact that generating every new token for each sequence requires one forward pass. Additionally, each method has its own additional drawbacks. Naive batched inference stores the KV cache multiple times for every sequence, even they share the exact same content prefixes, leading to redundant storage of the prefix key and value vectors, a problem which we will call **KV cache duplication**. Some related works, including vLLM with PagedAttention (Kwon et al., 2023) and the Prompt Caching technique (Gim et al., 2024), which consolidates identical input KV caches into one physical block across different queries. Another related work, SGLang (Zheng et al., 2024) with the RadixAttention algorithm, examines incoming requests to identify the longest previously processed subsequence, thereby preventing redundant computations of overlapping keys and values. Despite the fact that the system prompt is common to all requests, the hidden states, represented as key-value pairs, are repeatedly read from DRAM by current attention algorithms like

PagedAttention, RadixAttention, and FlashAttention (Dao et al., 2022), separately for each request in the batch. Consequently, this approach only minimizes the time needed for query processing (the prefilling phase) but does not decrease the time required for generating new tokens (the decoding phase).

Recent works like Hydragen (Juravsky et al., 2024) and RelayAttention (Zhu et al., 2024) optimize the attention computation for LLM generation with shared content by utilizing the benefit of efficiency of matrix multiplications in modern GPUs. For document question tasks, Hydragen's multiple levels of sharing system does not work well since it requires each document to ask the same number of questions, and the length of questions is restricted in the current version of their current released code, which is the initial release. Incorporating a similar idea into vLLM service systems, relayAttention (Zhu et al., 2024) assumes that all requests share the same system prompt, which implies the serving process provides only one application. For prompting questions with different documents in a batch, a hybrid batch with multiple sharing groups is still not supported based on the current implementation.

Generating reliable outputs for multiple queries with one prompt makes it even more challenging. SeqBatch Prompting in Figure 2 with many queries sequentially all at once within a bigger prompter often causes the degraded performance (Cheng et al., 2023; Lin et al., 2023), which we will refer to as **prompt interference**. This inevitably leads to a severe performance decrease in the language model (Liu et al., 2024), and improving efficiency in this setting will have a significant impact.

To address these bottlenecks, we introduce a novel and simple method for efficient parallel decoding of multiple prompts to a transformer language model. These prompts can be done all at once in parallel. Our approach benefits from increased parallelization (textbfparallel decoding), and removes both problems of prompt interference and KV cache duplication. Specifically, our work not only increases the throughput of generation and reduces memory consumption during processing but also maintains the generation quality of language models.

To summarize, we make the following contributions:

- We propose a simple and effective method leveraging parallel prompting in LLM that allows a single LLM prompt to infer multiple answers for various questions simultaneously.
- We provide a mechanism to further optimize generation latency and throughput with batch parallel generation.
- We conduct experiments with multiple downstream datasets, generate synthetic data, and show our method achieves improvements in throughput and computational resource management, offering a robust solution for different tasks in LLMs.

3 Method

We formulate the problem as follows. Suppose we have a context C and N sentence queries $q_1, \ldots, q_n$ for the context.

Let the generation function of original model be $LLM.GEN()$, and suppose the current batch of data with batch size N is $Q = \{q_1, q_2, ..., q_n\}$, the answers to each data are $A = \{a_1, a_2, ..., a_n\}$. In the situation of standard batch prompting multiple questions Q based on the same context C from the auto-regressive language model, the final answer for q_n can be formulated as:

$$a_i = \text{LLM.GEN}(C, q_i) \quad (1)$$

In order to improve the inference efficiency, Seq-Batch Prompting in Figure 2 combines all question into one bigger prompt. The final answer for q_n with Seq-Batch Prompting can be formulated as:

$$a_i = \text{LLM.GEN}(C, Q, a_{1:i-1}) \quad (2)$$

However, the answer A_n to the data Q_n is not only conditioned on the task specification but also on $\{a_1, a_2, ...a_{n-1}\}$, which can be viewed as the context of a_n. Therefore, all of the generated answers have a unique effect for the following ones in the batch prompting method, which we refer to as the prompt interference problem.

To tackle this problem, the simplest way is to construct a mask matrix MASK for each answer that makes sure that that answer only pays attention to its corresponding question and the shared context. With the specialized attention mask, we are able to compute attention over the shared context and corresponding question as a standalone operation for every answer. While this specialized attention mask does not improve efficiency on its own (in fact, it introduces additional work to initialize a mask for each answer), it can allow us to compute cross-attention much more efficiently over a batch of sequences in the following generation stage.

$$a_i = \text{LLM.GEN}(C, Q, M, a_{1:i-1}) \tag{3}$$

However, though the prompt interference problem is solved, we still face the efficiency problem. Since next-token prediction remains an inefficient way of generating answers for all independent questions and restricts the LLM's world knowledge and reasoning capabilities. More precisely, next-token prediction assumes left-to-right dependencies in language, i.e., a later-appearing token depends on all earlier-appearing tokens but overlooks the existence of independent dependencies.

We explore a parallel prediction method in which we merge each independent query vector together into one attention operation over a single prompting sequence, then feed it into the language model to predict future tokens in parallel. The following sections will succinctly introduce our method, encompassing both the prefilling and generation stages.

3.1 Prefilling prompt with Independent questions

In the prefill stage of our method , the model encodes the prompt in parallel within a single forward pass. During this phase, the LLM takes a prepacked prompt sequence with a modified masking in Figure 3 and position encoding to extract the corresponding KV-cache values. Each question's position index follows the end token index of context, which ensures the correct position embedding passing into the model. If the attention status of context is already precached, the prefill process can also be done by providing the context attention status as past kv-cache. We provide the pseudo-codes for our generation process in algorithm 1 and a detailed parallel process in algorithm 2.

3.2 Parallel Generation

Recall that given the sequence of queries $Q \in \mathbb{R}^{N_q \times d}$, keys $K \in \mathbb{R}^{N_{kv} \times d}$, values $V \in \mathbb{R}^{N_{kv} \times d}$, the transformer model computes the attention output $O \in \mathbb{R}^{N_q \times d}$ as follows:

$$O = \text{Attention}(Q, K, V) = \text{softmax}(\frac{QK^T}{\sqrt{d}})V \tag{4}$$

During generation, the Q matrix is $1 \times d$, a vector. With causal masking, this usually becomes:

$$\begin{aligned} O &= \text{Attention}(Q, K, V) && (5) \\ &= \text{softmax}\left(\frac{QK^T}{\sqrt{d}} + M\right)V && (6) \end{aligned}$$

Each entry in M is $-\infty$ or 0 for masked or non-masked entries in the attention matrix, respectively.

Since all questions are independent and share a common context, we are able to generate the probability distribution of answers simultaneously. To achieve this, we need to allow the model to generate N tokens at once in each forward pass of the generation stage, which means increasing the number of query vectors in the attention computation by making Q a matrix of dimension $N \times d$.

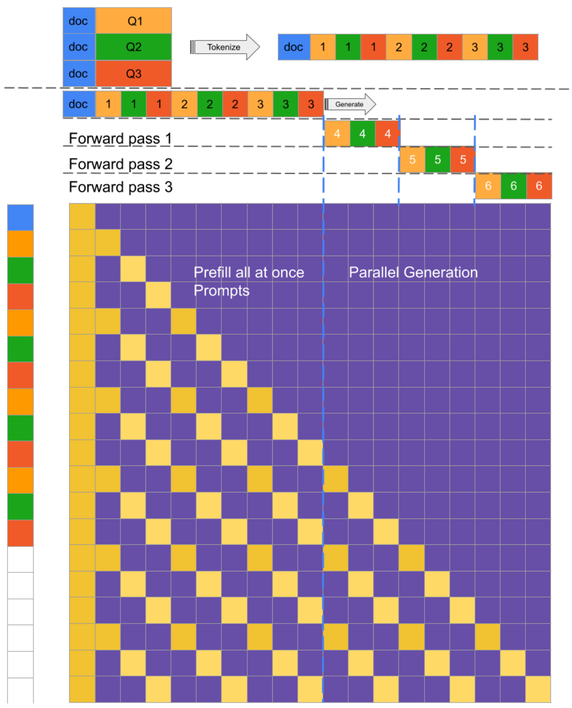

Figure 3: Overview of independent masking prefill and parallel generation.

During the decoding phase, our method generates tokens for different questions simultaneously. In the process of parallel generation, each forward pass would generate N new tokens which is also the number of questions. In Figure 3, the number of questions is three and different colors represent different questions, the number in block represents the position of tokens in the normal prompt sequences. Since the position of each generated token should be followed by the provided prefix tokens, we have to record the position of all last input tokens and add 1 for them. Also, in order to seamlessly generate the full answers to the provided questions, we update the generated tokens to their corresponding positions in the inputs prompt. Since all the mask attention structures are already defined from the prefill stage, the model only needs to update them with the same pattern in the generation stage.

Algorithm 1 Parallel_Batch_Prompting

```
Function Parallel_Batch_Prompting(shared prefix Doc, unique suffixes Q_all, batch size N, parallel size P, LLM)
    Initialize i ← 0
    Initialize N_p ← N/P
    Initialize 3d_mask ← N_p * torch.tril()
    while i ≤ len(Q_all) do
        Q_n ← Q_all[i : i + N]
        Q_np ← parallize_interleave(Q_n, P)
        prompts ← prepare_input(Doc, Q_np, N_p)
        masks ← 3d_mask * padding.mask
        answers, output_mask ← LLM.parallel_generate()
        for n ← 1 to N_p do
            for p ← 1 to P do
                final_answer.append(LLM.decode())
            end
        end
        i = i + N
    end
```

Algorithm 2 Parallel_Generate

```
Function Parallel_Generate(self, inputs prompts, 3d masks masks, parallel size P, LLM)
    Initialize finished ← False
    Initialize self.inputs_ids ← self.prompts
    Initialize self.position_ids
    Initialize self.masks
    while True do
        outputs ← self.LLM()
        parallel_logits ← outputs[−P :]
        parallel_tokens ← argmax(parallel_logits)
        input_ids ← concat(input_ids, parallel_tokens)
        if stopping_criteria(input_ids, P) then
            finished ← True
            break
        self.prepare_parellel_mask(P)
        self.prepare_parellel_position(P)
    end
    return self.input_ids, self.masks
```

3.3 Batching

The use of batching is a crucial technique to enhance throughput in LLM inference. Through batched decoding, each forward pass of the model processes the latest token from multiple sequences concurrently rather than just one. This approach amplifies the arithmetic intensity of transformer components, such as the multilayer perceptron (MLP) blocks, and facilitates the use of hardware-friendly matrix multiplications.

However, the computation intensity of attention does not inherently benefit from batching, as each sequence possesses its distinct key and value matrix. Consequently, while other model components can leverage tensor cores during batched decoding, attention is required to be computed using numerous independent matrix-vector products. Our parallel generation technique aims to address this by enhancing the computation intensity of attention.

RealyAttention (Zhu et al., 2024) does not support batching with different prefixes, as it necessitates a more complex implementation of fused operators in CUDA for hybrid batching with multiple sharing groups. Hydragen (Juravsky et al., 2024) requires a batched document with the same number of questions, and it also has a question length constraint with its implementation.

Our method integrates seamlessly with the batching technique. By batching texts with multiple unique documents and corresponding questions, efficiency can be improved further. Parallel generation with batching provides two distinct advantages: firstly, inference throughput is further amplified by batching with multiple unique prefix documents; secondly, it enables the balancing of batch size and sequence length for model input, optimizing overall performance.

4 Experiments

The experiments are organized into three subsections: main experiments, analytical study, and ablation study.

The main experiments focus on the throughput of our generation method compared to various baseline techniques in reading comprehension tasks with Llama 3-8B model (Grattafiori et al., 2024). It serves to validate the motivating principles behind our approach. Initially, we compare the accuracy of token predictions made using our method against baseline methods like standard batch prompting and seq-batch prompting. Our findings show that our method maintains high prediction accuracy across different datasets. Additionally, we analyze the throughput in the generation phase relative to the more advanced methods to further substantiate our motivation.

Both the analytical experiments and the ablation study are conducted on smaller model sizes such as CodeLlama-7b-Instruct (Rozière et al., 2024)

Dataset	Avg. #tokens(Doc)	Avg. #tokens(Q)	Avg. #Q per Doc
SQuAD	556	24	5
QuAC	2,628	18	7
DROP	761	26	16

Table 1: Average number of shared tokens of each document with one shot demonstration, average number of tokens for questions, and average number of questions each shared document .

and Sheared-LLaMA-1.3B (Xia et al., 2024) and LLaMa-160m (Miao et al., 2023b). These subsections aim to demonstrate the reliability and effectiveness of our approach. It optimizes the processing efficiency of LLMs to manage larger, more context-rich inputs without a loss in performance.

More detailed information is available in the Appendix A. Across all models, we employ a consistent parallel generation method to predict the next set of multiple-answer tokens.

All experiments are conducted on a single NVIDIA A100-80GB GPU. Our implementations rely on PyTorch, using the HuggingFace architecture (Wolf et al., 2020).

4.1 Datasets

We evaluate our method on three popular datasets: SQUAD(Rajpurkar et al., 2016), QuAC(Choi et al., 2018), and DROP(Dua et al., 2019) with Llama 3-8b (Grattafiori et al., 2024). Many recent works like RelayAttention with vLLM (Zhu et al., 2024)and Hydragen (Juravsky et al., 2024) have a huge performance improvement when the number of questions is huge (bigger than 100) and the shared content is very long (tokens bigger than 1000). However, we noticed that the popular downstream tasks with parallel questions have a much shorter shared document length and a much smaller size of questions. The statistic is summarized in Table 1. The benefits of their methods can not be fully utilized under this circumstance. Instead, our parallel generation method can work better in this scenario.

To further show the effectiveness of our parallel prompting method, we also evaluate our method on one constructed synthetic data following the Hydragen paper (Juravsky et al., 2024) with different lengths and numbers of unique documents and various numbers of questions. To demonstrate the throughput benefits of using our method to answer questions about a long document, we generate data that contains arbitrary facts from which question/answer pairs can be easily generated. The content of the document is a subset of War and Peace (Tolstoy, 1869), modified to include procedurally generated sentences of the form "The {animal} named {name} has {body part} that is {color}." The questions are of the form "What color is the {body part} of the {animal} named {name}?", where the answer is {color}. We construct various questions inference tasks and various lengths of shared content from War and Peace (plus five for the few-shot examples) and concatenate these few shot examples at the end of the document.

Dataset	**Method**	**Time(s)**	**F1(%)**
SQuAD	Standard	590	87.2
	SeqBatch	393	84.2
	Hydragen	1077	87.1
	vLLM	351	87.4
	vLLM-RA	365	87.3
	Parallel	168	87.2
QuAC	Standard	1799	34.0
	SeqBatch	462	29.1
	Hydragen	832	34.0
	vLLM	843	32.8
	vLLM-RA	468	32.8
	Parallel	317	33.9
DROP	Standard	654	58.1
	SeqBatch	834	42.5
	Hydragen	316	58.2
	vLLM	393	58.5
	vLLM-RA	203	58.5
	Parallel	111	58.1

Table 2: Comparison of generation time and performance with different methods on average of five times with Llama 3 8B model on A100-80G.

4.2 Evaluation on Downstream Tasks

The Table 2 compares the generation time of standard, batch prompting, Hydragen, vLLM, vLLM with relay attention and our parallel prompting methods. The result shows that parallel prompting performs consistently better than standard and batch prompting on the latency of generation while remains the same quality of outputs as the standard prompting over all datasets.

We use the current latest version, 0.6.4, of the vLLM package, which uses the PagedAttention algorithm. vLLM avoids redundant storage of the prefix, allowing much larger batch sizes to be tested. Additionally, because of this non-redundant storage, PagedAttention can achieve a higher GPU cache hit rate when reading the prefix, reducing the cost of redundant reads. We consider comparing the vLLm with the Prefix Cache method in our constructed synthetic data since it will not be a fair comparison with other methods without the caching technique, especially when we use the same one-shot example for the downstream tasks in each dataset.

4.3 Analytical Study on Synthetic Data

We constructed synthetic data with various document lengths and a number of unique documents with questions to evaluate our method. The Figure 4 compares the throughput of standard, batch prompting, Hydragen, vLLM, vLLM with RelayAttention, vLLM with prefix cache and our parallel prompting on the generated synthetic data. As the number of unique shared docs increases, our parallel generation method outperforms other methods without the decrease in generation quality.

The performance of LLM's generation can be affected by various factors. We also run experiments with various configurations with CodeLlama-7b-Inst (Rozière et al., 2024) and Sheared-LLaMA-1.3B (Xia et al., 2024). For example, the length of shared documents, questions, and answers. Different model sizes and GPUs could also affect generation performance. More detailed results can be found in Appendix A.

Number of Questions We run our benchmarks on CodeLlama-7b-Instruct (Rozière et al., 2024) with one A100-80GB GPU with various numbers of questions and documents. In Table 3, we fix the document length to 512 tokens and sweep over the question size from a range while generating five tokens per question. When the batch size is small, non-attention operations contribute significantly to decoding time, with all methods reaching at least half of the throughput of no-attention upper bound. At these small batch sizes, most methods have similar throughputs, and some methods spend more time staging document KV cache. However, as the batch size grows at a certain level, attention over the prefix becomes increasingly expensive, and our parallel generation save more time for attention computation. As a result, our method begins to outperform the other baselines. A certain number of parallelized questions work better than others in our experiment; more detailed analysis is in Appendix A.

Batch Size We ran a few experiments on querying a range of fixed prompts with different batch sizes. Interestingly, maximizing the parallel size(minimizing the batch size) is only sometimes ideal. This situation happens for all of our models with various sizes (7B (Grattafiori et al., 2024),1B (Rozière et al., 2024),160m (Miao et al., 2023b)). In Table 3 , we see the best throughput performance is reached by 256 parallel sizes when queries 128 prompts. In Table 5and Table 6, the parallel size also not always be the maxized one in different GPUs (NVIDIA-A100-SXM4-80GB i, NVIDIA-GeForce-RTX-3090). We assume that the best number of parallel sizes balances the cost of computation in the arithmetic intensity of the transformer components such as the multilayer perceptron (MLP) blocks and intensity of attention.

Document Length Now, we run a similar experiment, except now we hold the number questions in the list $[2, 4, 8, 16, 32, 64]$ of each document as one constant number 128 and sweep over the shared prefix length among the list $[128, 256, 512]$ in Figure 5 Figure 6 Figure 7. Even though the throughput decreases as the prefix grows, with our parallel generation method, throughput is less unaffected when the prefix content grows under a certain level below 1000 tokens. We perform more in-depth sweeps over different models, prefix lengths, batch sizes, and numbers of generated tokens in Appendix A - for smaller models and more parallel questions, the speedup can exceed Table 4.

5 Related work

Recent advancements in language modeling have delved into the prediction of multiple tokens simultaneously to enhance both efficiency and performance. Notable works such as (Miao et al., 2024; Leviathan et al., 2023; Wu et al., 2024) focus on speculative decoding methods, where potential future sequences are built and verified to expedite inference. Similarly, (Gloeckle et al., 2024) and (Cai et al., 2024) propose predicting multiple future tokens using different output heads, thereby speeding up the inference process.

Efforts to increase throughput in LLM inference

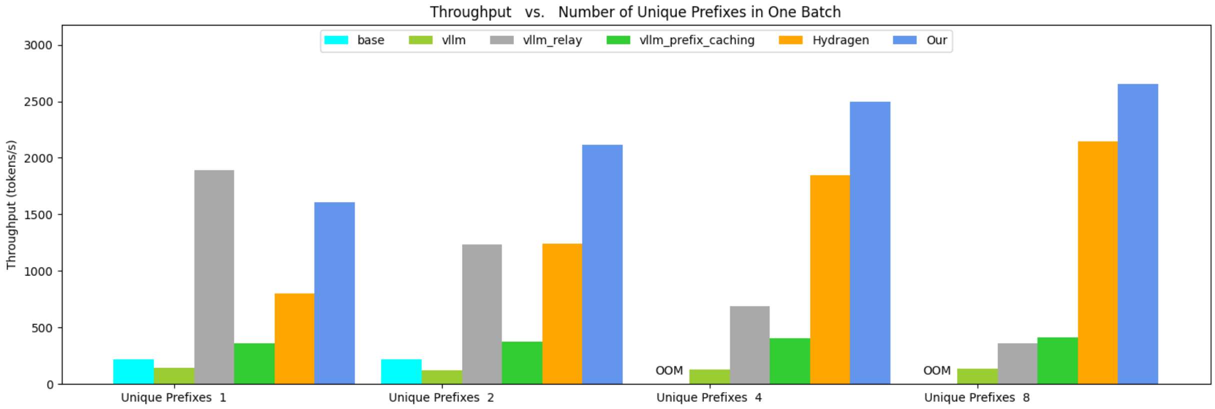

Figure 4: Throughputs of different methods when the number of unique documents changes in the LLM inference. CodeLlama-7b-Instruct attention inference Throughput w.r.t. number of unique documents (A100-SXM4-80GB GPU). We set the length of context to 256, the number of total queries is 512, for each context is 64, the length of each query to 12, the length of generated token to 5.

have led to various innovative techniques aimed at optimizing GPU utilization and improving throughput. (Dao et al., 2022) and (Sheng et al., 2023) aim to improve memory usage efficiency, enabling higher throughput in generative inference tasks. (Jin et al., 2023) schedules prompts based on estimated output sequence lengths to optimize GPU usage. (Gim et al., 2024) proposes reusing precomputed caches in a predefined schema to reduce latency. (Sun et al., 2024) applies dynamic sparse KV caching in decoding to accelerate long sequence generation.

Efficient prompting techniques could also increase the throughput of LLM.(Cheng et al., 2023) groups multiple questions in a single prompt, though it will lead to performance degradation when the number of questions increases. (Zhao et al., 2024) enhances throughput during the prefilling stage by prepacking data. (Ning et al., 2024) uses the skeleton of the answer to batch-generate the final answer.

To avoid the KV cache duplication, existing work (Kwon et al., 2023) vLLM uses its PagedAttention and paged memory management to point multiple identical input prompts to only one physical block across multiple queries. Also, (Juravsky et al., 2024) proposes a decomposition of attention computation of shared prefixes and unique suffixes. (Lu et al., 2024) increases efficiency by sharing cache in the encoder-decoder model for decomposable tasks.

Compared with the above methods, our work introduces a novel inference technique that allows LLMs to handle multiple questions within a single prompt efficiently, leveraging GPU parallel capacity to improve inference throughput and memory utilization without degrading reasoning performance.

6 Conclusion

We introduce an efficient parallel prompting method for decoding prompt queries in parallel. We conduct experiments with multiple down stream datasets, generate synthetic data, and show our method achieves improvements in throughput and computational resource management, offering a robust solution for different tasks in LLMs.

Limitations

Our parallel generation method is not highly optimized for querying with extremely long shared content prefixes. However, it can be improved with other techniques like prefix cache. Our approach requires a modified causal mask as one extra input for the model, which may not be available or may require additional steps to implement it. Due to budget and hardware constraints, we could not experiment with our approach on larger open-sourced LLMs.

References

Tianle Cai, Yuhong Li, Zhengyang Geng, Hongwu Peng, Jason D Lee, Deming Chen, and Tri Dao. 2024. Medusa: Simple llm inference acceleration framework with multiple decoding heads. *arXiv preprint arXiv:2401.10774*.

Charlie Chen, Sebastian Borgeaud, Geoffrey Irving, Jean-Baptiste Lespiau, Laurent Sifre, and John Jumper. 2023. Accelerating large language model decoding with speculative sampling. *arXiv preprint arXiv:2302.01318*.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde De Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, et al. 2021. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*.

Zhoujun Cheng, Jungo Kasai, and Tao Yu. 2023. Batch prompting: Efficient inference with large language model apis. *arXiv preprint arXiv:2301.08721*.

Eunsol Choi, He He, Mohit Iyyer, Mark Yatskar, Wentau Yih, Yejin Choi, Percy Liang, and Luke Zettlemoyer. 2018. Quac: Question answering in context. *arXiv preprint arXiv:1808.07036*.

Tri Dao, Dan Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. 2022. Flashattention: Fast and memory-efficient exact attention with io-awareness. *Advances in Neural Information Processing Systems*, 35:16344–16359.

Dheeru Dua, Yizhong Wang, Pradeep Dasigi, Gabriel Stanovsky, Sameer Singh, and Matt Gardner. 2019. Drop: A reading comprehension benchmark requiring discrete reasoning over paragraphs. *arXiv preprint arXiv:1903.00161*.

In Gim, Guojun Chen, Seung-seob Lee, Nikhil Sarda, Anurag Khandelwal, and Lin Zhong. 2024. Prompt cache: Modular attention reuse for low-latency inference. *Proceedings of Machine Learning and Systems*, 6:325–338.

Fabian Gloeckle, Badr Youbi Idrissi, Baptiste Rozière, David Lopez-Paz, and Gabriel Synnaeve. 2024. Better & faster large language models via multi-token prediction. *arXiv preprint arXiv:2404.19737*.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, Amy Yang, Angela Fan, Anirudh Goyal, Anthony Hartshorn, Aobo Yang, Archi Mitra, Archie Sravankumar, Artem Korenev, Arthur Hinsvark, Arun Rao, Aston Zhang, Aurelien Rodriguez, Austen Gregerson, Ava Spataru, Baptiste Roziere, Bethany Biron, Binh Tang, Bobbie Chern, Charlotte Caucheteux, Chaya Nayak, Chloe Bi, Chris Marra, Chris McConnell, Christian Keller, Christophe Touret, Chunyang Wu, Corinne Wong, Cristian Canton Ferrer, Cyrus Nikolaidis, Damien Allonsius, Daniel Song, Danielle Pintz, Danny Livshits, Danny Wyatt, David Esiobu, Dhruv Choudhary, Dhruv Mahajan, Diego Garcia-Olano, Diego Perino, Dieuwke Hupkes, Egor Lakomkin, Ehab AlBadawy, Elina Lobanova, Emily Dinan, Eric Michael Smith, Filip Radenovic, Francisco Guzmán, Frank Zhang, Gabriel Synnaeve, Gabrielle Lee, Georgia Lewis Anderson, Govind Thattai, Graeme Nail, Gregoire Mialon, Guan Pang, Guillem Cucurell, Hailey Nguyen, Hannah Korevaar, Hu Xu, Hugo Touvron, Iliyan Zarov, Imanol Arrieta Ibarra, Isabel Kloumann, Ishan Misra, Ivan Evtimov, Jack Zhang, Jade Copet, Jaewon Lee, Jan Geffert, Jana Vranes, Jason Park, Jay Mahadeokar, Jeet Shah, Jelmer van der Linde, Jennifer Billock, Jenny Hong, Jenya Lee, Jeremy Fu, Jianfeng Chi, Jianyu Huang, Jiawen Liu, Jie Wang, Jiecao Yu, Joanna Bitton, Joe Spisak, Jongsoo Park, Joseph Rocca, Joshua Johnstun, Joshua Saxe, Junteng Jia, Kalyan Vasuden Alwala, Karthik Prasad, Kartikeya Upasani, Kate Plawiak, Ke Li, Kenneth Heafield, Kevin Stone, Khalid El-Arini, Krithika Iyer, Kshitiz Malik, Kuenley Chiu, Kunal Bhalla, Kushal Lakhotia, Lauren Rantala-Yeary, Laurens van der Maaten, Lawrence Chen, Liang Tan, Liz Jenkins, Louis Martin, Lovish Madaan, Lubo Malo, Lukas Blecher, Lukas Landzaat, Luke de Oliveira, Madeline Muzzi, Mahesh Pasupuleti, Mannat Singh, Manohar Paluri, Marcin Kardas, Maria Tsimpoukelli, Mathew Oldham, Mathieu Rita, Maya Pavlova, Melanie Kambadur, Mike Lewis, Min Si, Mitesh Kumar Singh, Mona Hassan, Naman Goyal, Narjes Torabi, Nikolay Bashlykov, Nikolay Bogoychev, Niladri Chatterji, Ning Zhang, Olivier Duchenne, Onur Çelebi, Patrick Alrassy, Pengchuan Zhang, Pengwei Li, Petar Vasic, Peter Weng, Prajjwal Bhargava, Pratik Dubal, Praveen Krishnan, Punit Singh Koura, Puxin Xu, Qing He, Qingxiao Dong, Ragavan Srinivasan, Raj Ganapathy, Ramon Calderer, Ricardo Silveira Cabral, Robert Stojnic, Roberta Raileanu, Rohan Maheswari, Rohit Girdhar, Rohit Patel, Romain Sauvestre, Ronnie Polidoro, Roshan Sumbaly, Ross Taylor, Ruan Silva, Rui Hou, Rui Wang, Saghar Hosseini, Sahana Chennabasappa, Sanjay Singh, Sean Bell, Seohyun Sonia Kim, Sergey Edunov, Shaoliang Nie, Sharan Narang, Sharath Raparthy, Sheng Shen, Shengye Wan, Shruti Bhosale, Shun Zhang, Simon Vandenhende, Soumya Batra, Spencer Whitman, Sten Sootla, Stephane Collot, Suchin Gururangan, Sydney Borodinsky, Tamar Herman, Tara Fowler, Tarek Sheasha, Thomas Georgiou, Thomas Scialom, Tobias Speckbacher, Todor Mihaylov, Tong Xiao, Ujjwal Karn, Vedanuj Goswami, Vibhor Gupta, Vignesh Ramanathan, Viktor Kerkez, Vincent Gonguet, Virginie Do, Vish Vogeti, Vítor Albiero, Vladan Petrovic, Weiwei Chu, Wenhan Xiong, Wenyin Fu, Whitney Meers, Xavier Martinet, Xiaodong Wang, Xiaofang Wang, Xiaoqing Ellen Tan, Xide Xia, Xinfeng Xie, Xuchao Jia, Xuewei Wang, Yaelle Goldschlag, Yashesh Gaur, Yasmine Babaei, Yi Wen, Yiwen Song, Yuchen Zhang, Yue Li, Yuning Mao, Zacharie Delpierre Coudert, Zheng Yan, Zhengxing Chen, Zoe Papakipos, Aaditya Singh, Aayushi Srivastava, Abha Jain, Adam Kelsey, Adam Shajnfeld, Adithya Gangidi, Adolfo Victoria, Ahuva Goldstand, Ajay Menon, Ajay Sharma, Alex Boesenberg, Alexei Baevski, Allie Feinstein, Amanda Kallet, Amit Sangani, Amos Teo, Anam Yunus, Andrei Lupu, Andres Alvarado, Andrew Caples, Andrew Gu, Andrew Ho, Andrew Poulton, Andrew Ryan, Ankit Ramchandani, Annie Dong, Annie Franco, Anuj Goyal, Aparajita Saraf, Arkabandhu Chowdhury, Ashley Gabriel, Ashwin Bharambe, Assaf Eisenman, Azadeh Yazdan, Beau James, Ben Maurer, Benjamin Leonhardi, Bernie Huang, Beth Loyd, Beto De Paola, Bhargavi

Paranjape, Bing Liu, Bo Wu, Boyu Ni, Braden Hancock, Bram Wasti, Brandon Spence, Brani Stojkovic, Brian Gamido, Britt Montalvo, Carl Parker, Carly Burton, Catalina Mejia, Ce Liu, Changhan Wang, Changkyu Kim, Chao Zhou, Chester Hu, Ching-Hsiang Chu, Chris Cai, Chris Tindal, Christoph Feichtenhofer, Cynthia Gao, Damon Civin, Dana Beaty, Daniel Kreymer, Daniel Li, David Adkins, David Xu, Davide Testuggine, Delia David, Devi Parikh, Diana Liskovich, Didem Foss, Dingkang Wang, Duc Le, Dustin Holland, Edward Dowling, Eissa Jamil, Elaine Montgomery, Eleonora Presani, Emily Hahn, Emily Wood, Eric-Tuan Le, Erik Brinkman, Esteban Arcaute, Evan Dunbar, Evan Smothers, Fei Sun, Felix Kreuk, Feng Tian, Filippos Kokkinos, Firat Ozgenel, Francesco Caggioni, Frank Kanayet, Frank Seide, Gabriela Medina Florez, Gabriella Schwarz, Gada Badeer, Georgia Swee, Gil Halpern, Grant Herman, Grigory Sizov, Guangyi, Zhang, Guna Lakshminarayanan, Hakan Inan, Hamid Shojanazeri, Han Zou, Hannah Wang, Hanwen Zha, Haroun Habeeb, Harrison Rudolph, Helen Suk, Henry Aspegren, Hunter Goldman, Hongyuan Zhan, Ibrahim Damlaj, Igor Molybog, Igor Tufanov, Ilias Leontiadis, Irina-Elena Veliche, Itai Gat, Jake Weissman, James Geboski, James Kohli, Janice Lam, Japhet Asher, Jean-Baptiste Gaya, Jeff Marcus, Jeff Tang, Jennifer Chan, Jenny Zhen, Jeremy Reizenstein, Jeremy Teboul, Jessica Zhong, Jian Jin, Jingyi Yang, Joe Cummings, Jon Carvill, Jon Shepard, Jonathan McPhie, Jonathan Torres, Josh Ginsburg, Junjie Wang, Kai Wu, Kam Hou U, Karan Saxena, Kartikay Khandelwal, Katayoun Zand, Kathy Matosich, Kaushik Veeraraghavan, Kelly Michelena, Keqian Li, Kiran Jagadeesh, Kun Huang, Kunal Chawla, Kyle Huang, Lailin Chen, Lakshya Garg, Lavender A, Leandro Silva, Lee Bell, Lei Zhang, Liangpeng Guo, Licheng Yu, Liron Moshkovich, Luca Wehrstedt, Madian Khabsa, Manav Avalani, Manish Bhatt, Martynas Mankus, Matan Hasson, Matthew Lennie, Matthias Reso, Maxim Groshev, Maxim Naumov, Maya Lathi, Meghan Keneally, Miao Liu, Michael L. Seltzer, Michal Valko, Michelle Restrepo, Mihir Patel, Mik Vyatskov, Mikayel Samvelyan, Mike Clark, Mike Macey, Mike Wang, Miquel Jubert Hermoso, Mo Metanat, Mohammad Rastegari, Munish Bansal, Nandhini Santhanam, Natascha Parks, Natasha White, Navyata Bawa, Nayan Singhal, Nick Egebo, Nicolas Usunier, Nikhil Mehta, Nikolay Pavlovich Laptev, Ning Dong, Norman Cheng, Oleg Chernoguz, Olivia Hart, Omkar Salpekar, Ozlem Kalinli, Parkin Kent, Parth Parekh, Paul Saab, Pavan Balaji, Pedro Rittner, Philip Bontrager, Pierre Roux, Piotr Dollar, Polina Zvyagina, Prashant Ratanchandani, Pritish Yuvraj, Qian Liang, Rachad Alao, Rachel Rodriguez, Rafi Ayub, Raghotham Murthy, Raghu Nayani, Rahul Mitra, Rangaprabhu Parthasarathy, Raymond Li, Rebekkah Hogan, Robin Battey, Rocky Wang, Russ Howes, Ruty Rinott, Sachin Mehta, Sachin Siby, Sai Jayesh Bondu, Samyak Datta, Sara Chugh, Sara Hunt, Sargun Dhillon, Sasha Sidorov, Satadru Pan, Saurabh Mahajan, Saurabh Verma, Seiji Yamamoto, Sharadh Ramaswamy, Shaun Lindsay, Shaun Lindsay, Sheng Feng, Shenghao Lin, Shengxin Cindy Zha, Shishir Patil, Shiva Shankar, Shuqiang Zhang, Shuqiang Zhang, Sinong Wang, Sneha Agarwal, Soji Sajuyigbe, Soumith Chintala, Stephanie Max, Stephen Chen, Steve Kehoe, Steve Satterfield, Sudarshan Govindaprasad, Sumit Gupta, Summer Deng, Sungmin Cho, Sunny Virk, Suraj Subramanian, Sy Choudhury, Sydney Goldman, Tal Remez, Tamar Glaser, Tamara Best, Thilo Koehler, Thomas Robinson, Tianhe Li, Tianjun Zhang, Tim Matthews, Timothy Chou, Tzook Shaked, Varun Vontimitta, Victoria Ajayi, Victoria Montanez, Vijai Mohan, Vinay Satish Kumar, Vishal Mangla, Vlad Ionescu, Vlad Poenaru, Vlad Tiberiu Mihailescu, Vladimir Ivanov, Wei Li, Wenchen Wang, Wenwen Jiang, Wes Bouaziz, Will Constable, Xiaocheng Tang, Xiaojian Wu, Xiaolan Wang, Xilun Wu, Xinbo Gao, Yaniv Kleinman, Yanjun Chen, Ye Hu, Ye Jia, Ye Qi, Yenda Li, Yilin Zhang, Ying Zhang, Yossi Adi, Youngjin Nam, Yu, Wang, Yu Zhao, Yuchen Hao, Yundi Qian, Yunlu Li, Yuzi He, Zach Rait, Zachary DeVito, Zef Rosnbrick, Zhaoduo Wen, Zhenyu Yang, Zhiwei Zhao, and Zhiyu Ma. 2024. The llama 3 herd of models.

Muhammad Usman Hadi, Rizwan Qureshi, Abbas Shah, Muhammad Irfan, Anas Zafar, Muhammad Bilal Shaikh, Naveed Akhtar, Jia Wu, Seyedali Mirjalili, et al. 2023. A survey on large language models: Applications, challenges, limitations, and practical usage. *Authorea Preprints*.

Yunho Jin, Chun-Feng Wu, David Brooks, and Gu-Yeon Wei. 2023. s^3: Increasing gpu utilization during generative inference for higher throughput. *Advances in Neural Information Processing Systems*, 36:18015–18027.

Jordan Juravsky, Bradley Brown, Ryan Ehrlich, Daniel Y Fu, Christopher Ré, and Azalia Mirhoseini. 2024. Hydragen: High-throughput llm inference with shared prefixes. *arXiv preprint arXiv:2402.05099*.

Enkelejda Kasneci, Kathrin Seßler, Stefan Küchemann, Maria Bannert, Daryna Dementieva, Frank Fischer, Urs Gasser, Georg Groh, Stephan Günnemann, Eyke Hüllermeier, et al. 2023. Chatgpt for good? on opportunities and challenges of large language models for education. *Learning and individual differences*, 103:102274.

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

Yaniv Leviathan, Matan Kalman, and Yossi Matias. 2023. Fast inference from transformers via speculative decoding. In *International Conference on Machine Learning*, pages 19274–19286. PMLR.

Jianzhe Lin, Maurice Diesendruck, Liang Du, and Robin Abraham. 2023. Batchprompt: Accomplish more with less. *arXiv preprint arXiv:2309.00384*.

Nelson F Liu, Kevin Lin, John Hewitt, Ashwin Paranjape, Michele Bevilacqua, Fabio Petroni, and Percy Liang. 2024. Lost in the middle: How language models use long contexts. *Transactions of the Association for Computational Linguistics*, 12:157–173.

Bo-Ru Lu, Nikita Haduong, Chien-Yu Lin, Hao Cheng, Noah A Smith, and Mari Ostendorf. 2024. Encode once and decode in parallel: Efficient transformer decoding. *arXiv preprint arXiv:2403.13112*.

Xupeng Miao, Gabriele Oliaro, Zhihao Zhang, Xinhao Cheng, Hongyi Jin, Tianqi Chen, and Zhihao Jia. 2023a. Towards efficient generative large language model serving: A survey from algorithms to systems. *arXiv preprint arXiv:2312.15234*.

Xupeng Miao, Gabriele Oliaro, Zhihao Zhang, Xinhao Cheng, Zeyu Wang, Rae Ying Yee Wong, Zhuoming Chen, Daiyaan Arfeen, Reyna Abhyankar, and Zhihao Jia. 2023b. Specinfer: Accelerating generative llm serving with speculative inference and token tree verification.

Xupeng Miao, Gabriele Oliaro, Zhihao Zhang, Xinhao Cheng, Zeyu Wang, Zhengxin Zhang, Rae Ying Yee Wong, Alan Zhu, Lijie Yang, Xiaoxiang Shi, et al. 2024. Specinfer: Accelerating large language model serving with tree-based speculative inference and verification. In *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 3*, pages 932–949.

Xuefei Ning, Zinan Lin, Zixuan Zhou, Zifu Wang, Huazhong Yang, and Yu Wang. 2024. Skeleton-of-thought: Prompting llms for efficient parallel generation. In *The Twelfth International Conference on Learning Representations*.

Rizwan Qureshi, Muhammad Irfan, Hazrat Ali, Arshad Khan, Aditya Shekhar Nittala, Shawkat Ali, Abbas Shah, Taimoor Muzaffar Gondal, Ferhat Sadak, Zubair Shah, et al. 2023. Artificial intelligence and biosensors in healthcare and its clinical relevance: A review. *IEEE Access*, 11:61600–61620.

Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. 2016. Squad: 100,000+ questions for machine comprehension of text.

Baptiste Rozière, Jonas Gehring, Fabian Gloeckle, Sten Sootla, Itai Gat, Xiaoqing Ellen Tan, Yossi Adi, Jingyu Liu, Romain Sauvestre, Tal Remez, Jérémy Rapin, Artyom Kozhevnikov, Ivan Evtimov, Joanna Bitton, Manish Bhatt, Cristian Canton Ferrer, Aaron Grattafiori, Wenhan Xiong, Alexandre Défossez, Jade Copet, Faisal Azhar, Hugo Touvron, Louis Martin, Nicolas Usunier, Thomas Scialom, and Gabriel Synnaeve. 2024. Code llama: Open foundation models for code.

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

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

Zhongxiang Sun. 2023. A short survey of viewing large language models in legal aspect. *arXiv preprint arXiv:2303.09136*.

Leo Tolstoy. 1869. *War and Peace*.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2023. Attention is all you need.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, et al. 2020. Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 conference on empirical methods in natural language processing: system demonstrations*, pages 38–45.

Pengfei Wu, Jiahao Liu, Zhuocheng Gong, Qifan Wang, Jinpeng Li, Jingang Wang, Xunliang Cai, and Dongyan Zhao. 2024. Parallel decoding via hidden transfer for lossless large language model acceleration. *arXiv preprint arXiv:2404.12022*.

Shijie Wu, Ozan Irsoy, Steven Lu, Vadim Dabravolski, Mark Dredze, Sebastian Gehrmann, Prabhanjan Kambadur, David Rosenberg, and Gideon Mann. 2023. Bloomberggpt: A large language model for finance. *arXiv preprint arXiv:2303.17564*.

Mengzhou Xia, Tianyu Gao, Zhiyuan Zeng, and Danqi Chen. 2024. Sheared llama: Accelerating language model pre-training via structured pruning.

Zhihang Yuan, Yuzhang Shang, Yang Zhou, Zhen Dong, Zhe Zhou, Chenhao Xue, Bingzhe Wu, Zhikai Li, Qingyi Gu, Yong Jae Lee, et al. 2024. Llm inference unveiled: Survey and roofline model insights. *arXiv preprint arXiv:2402.16363*.

Siyan Zhao, Daniel Israel, Guy Van den Broeck, and Aditya Grover. 2024. Prepacking: A simple method for fast prefilling and increased throughput in large language models. *arXiv preprint arXiv:2404.09529*.

Lianmin Zheng, Liangsheng Yin, Zhiqiang Xie, Chuyue Sun, Jeff Huang, Cody Hao Yu, Shiyi Cao, Christos Kozyrakis, Ion Stoica, Joseph E Gonzalez, et al. 2024. Sglang: Efficient execution of structured language model programs. *arXiv preprint arXiv:2312.07104*.

Zixuan Zhou, Xuefei Ning, Ke Hong, Tianyu Fu, Jiaming Xu, Shiyao Li, Yuming Lou, Luning Wang, Zhihang Yuan, Xiuhong Li, et al. 2024. A survey on efficient inference for large language models. *arXiv preprint arXiv:2404.14294*.

Lei Zhu, Xinjiang Wang, Wayne Zhang, and Rynson WH Lau. 2024. Relayattention for efficient large language model serving with long system prompts. *arXiv preprint arXiv:2402.14808*.

A Example Appendix

In this section we show the affect of number of unique prefixes content for different methods on the parallel generation. The performance of RelayAttention method has a huge decline, since it dose not support the hybrid batching in its current implementation. Our methods performs well under a small number of quesitons be asked fore each shared content. As the number of questions becomes bigger(over 100), the computation of attention will be slower since our packed sequence length is much longer than other methods, and the efficiency of the generation process will be affected.

$\#queries$	$BatchSize$	$Throughput1B(tokens/second)$	$Throughput7B(tokens/second)$
128	1	4283	1931
	2	4625	1843
	4	3654	1468
	8	2850	1018
256	1	5911	2115
	2	6384	2250
	4	5748	2071
	8	4959	1615
512	1	5419	1850
	2	6845	2214
	4	7725	2382
	8	7181	2146

Table 3: Comparing the throughput using parallel Batching with different Batch sizes of parallel generation on 1B and 7B Llama model when the $doc\_len = 512 \| q\_len = 12 \| ans\_len = 5$.

doc_len	$Throughput(1B)(tokens/second)$	$Throughput(7B)(tokens/second)$
256	9512	2750
512	8199	2430
1024	6591	1924

Table 4: Comparing the throughput using parallel Batching with 7B and 1B Llama model with different lengths of doc length when $q\_len = 12 \| q\_num = 128 \| ans\_len = 5$ and the number of unique doc content equals 8. As the content length increases, the degradation of throughput performance becomes severe.

Method	**model**	**New Tokens**	**Batch Size**	**Parallel Size**	**Latency(s)**	**Peak Memory(MB)**
Our	llama-160m	10	1	128	10.1	23770
Our	llama-160m	10	2	64	7.3	24066
Our	llama-160m	10	4	32	2.4	24781
Our	llama-160m	10	8	16	2.9	26258
Our	llama-160m	10	16	8	3.9	29247
Our	llama-160m	10	1	64	6.3	12331
Our	llama-160m	10	2	32	3.4	12692
Our	llama-160m	10	4	16	3.9	13428
Our	llama-160m	10	8	8	4.8	14921
Our	llama-160m	10	16	4	6.9	17917
Our	llama-160m	10	1	32	5.1	6665
Our	llama-160m	10	2	16	5.4	7034
Our	llama-160m	10	4	8	6.4	7780
Our	llama-160m	10	8	4	8.7	9276
Our	llama-160m	10	16	2	13.0	12275

Table 5: Comparing the end-to-end NVIDIA-A100-SXM4-80GB inference latency of parallel generation with baseline method. Numbers in parenthesis show the length of document, length of each question and number of all questions for prompting each LLM.($len_{doc} = 512, len_q = 10, num_q = 1024$). Results averaged over 50 runs.

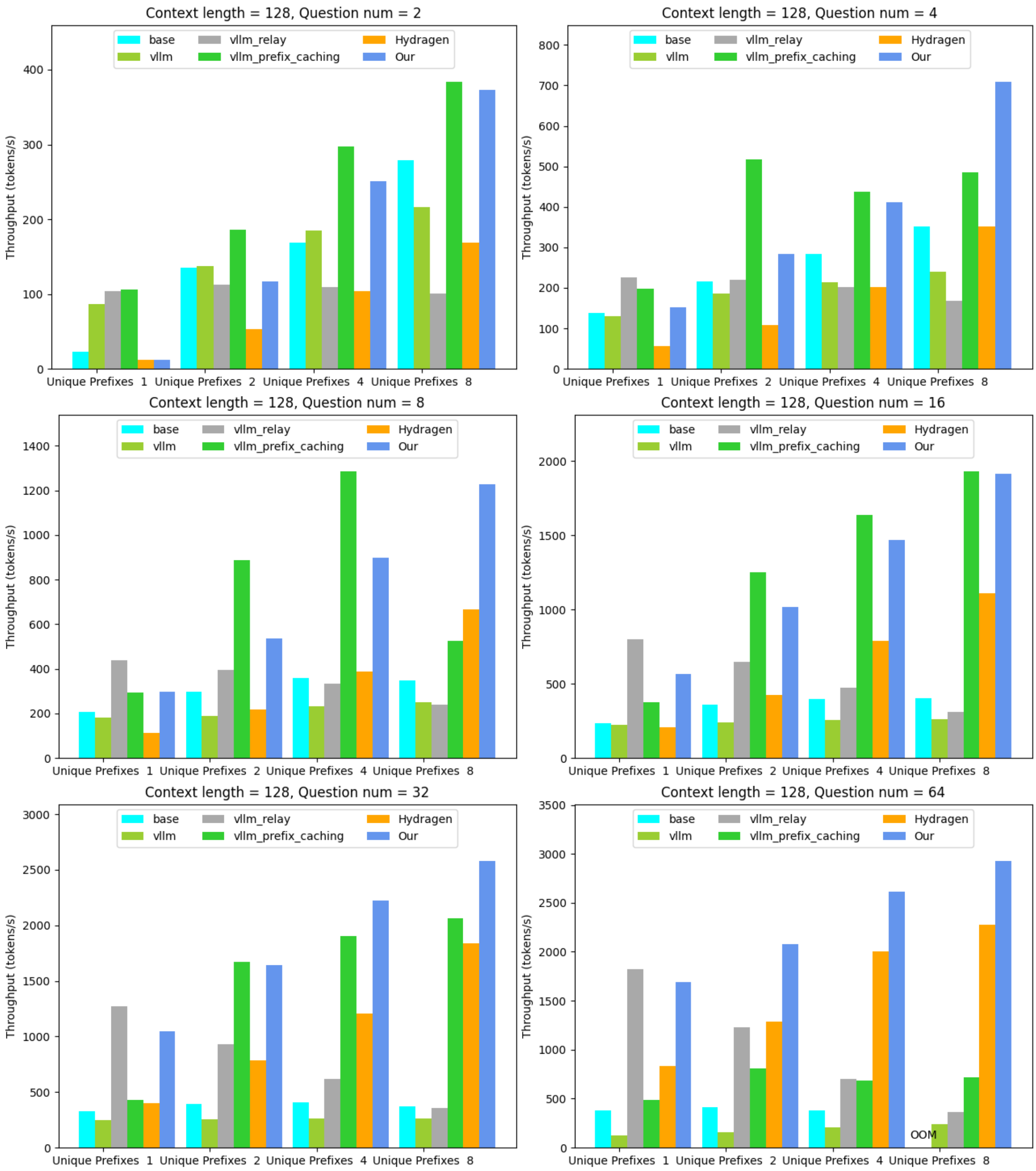

Figure 5: Throughputs of different methods when the number of unique documents changes in the LLM inference.CodeLlama-7b-Instruct attention inference Throughput w.r.t. number of unique documents (A100-SXM4-80GB GPU). We set the length of content to 128, the number queries for each context sweeps over the list of [2,4,8,16,32,64] , the length of each query to 12, the length of generated token to 5.

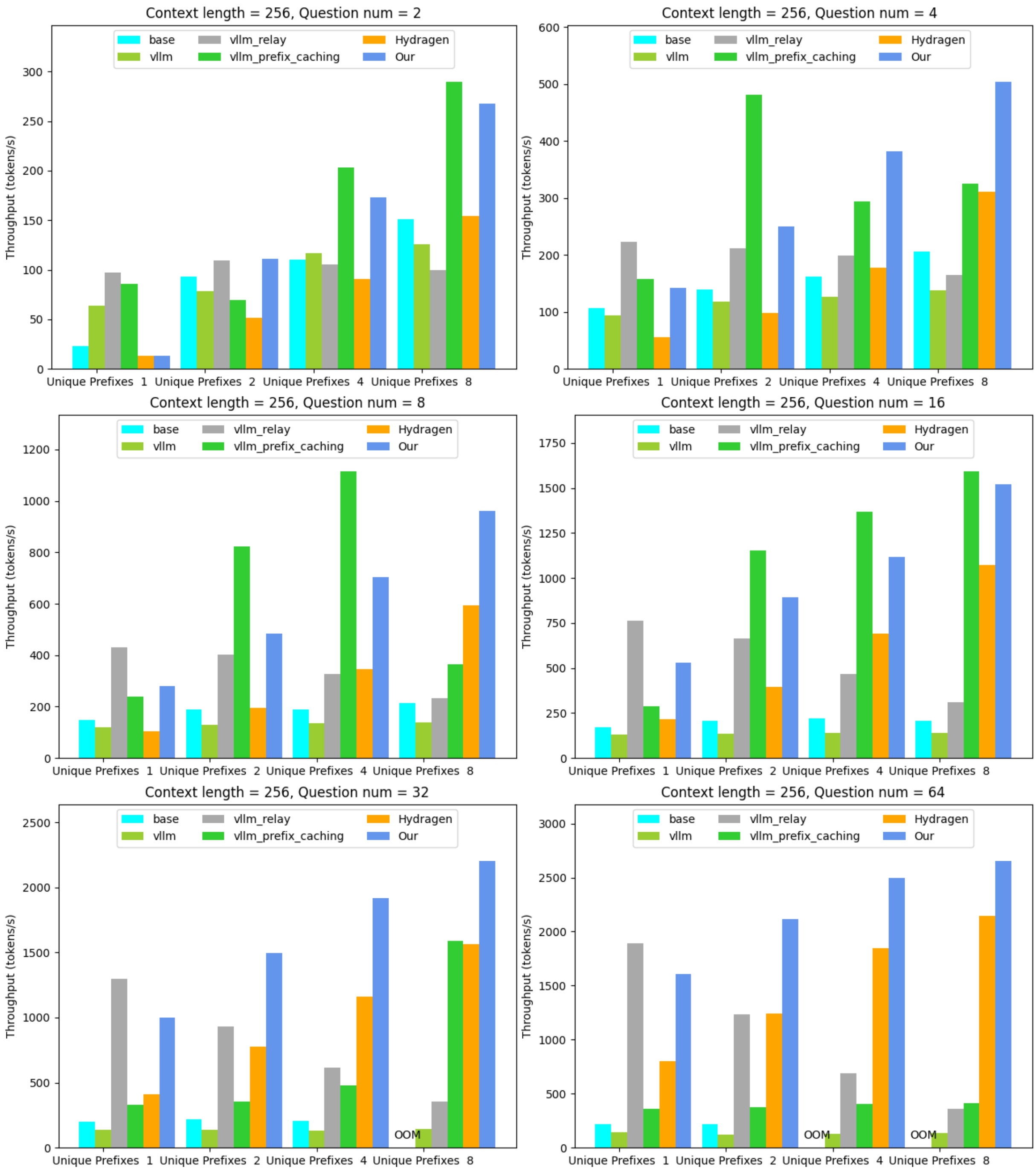

Figure 6: Throughputs of different methods when the number of unique documents changes in the LLM inference.CodeLlama-7b-Instruct attention inference Throughput w.r.t. number of unique documents (A100-SXM4-80GB GPU). We set the length of content to 256, the number queries for each context sweeps over the list of [2,4,8,16,32,64] , the length of each query to 12, the length of generated token to 5.

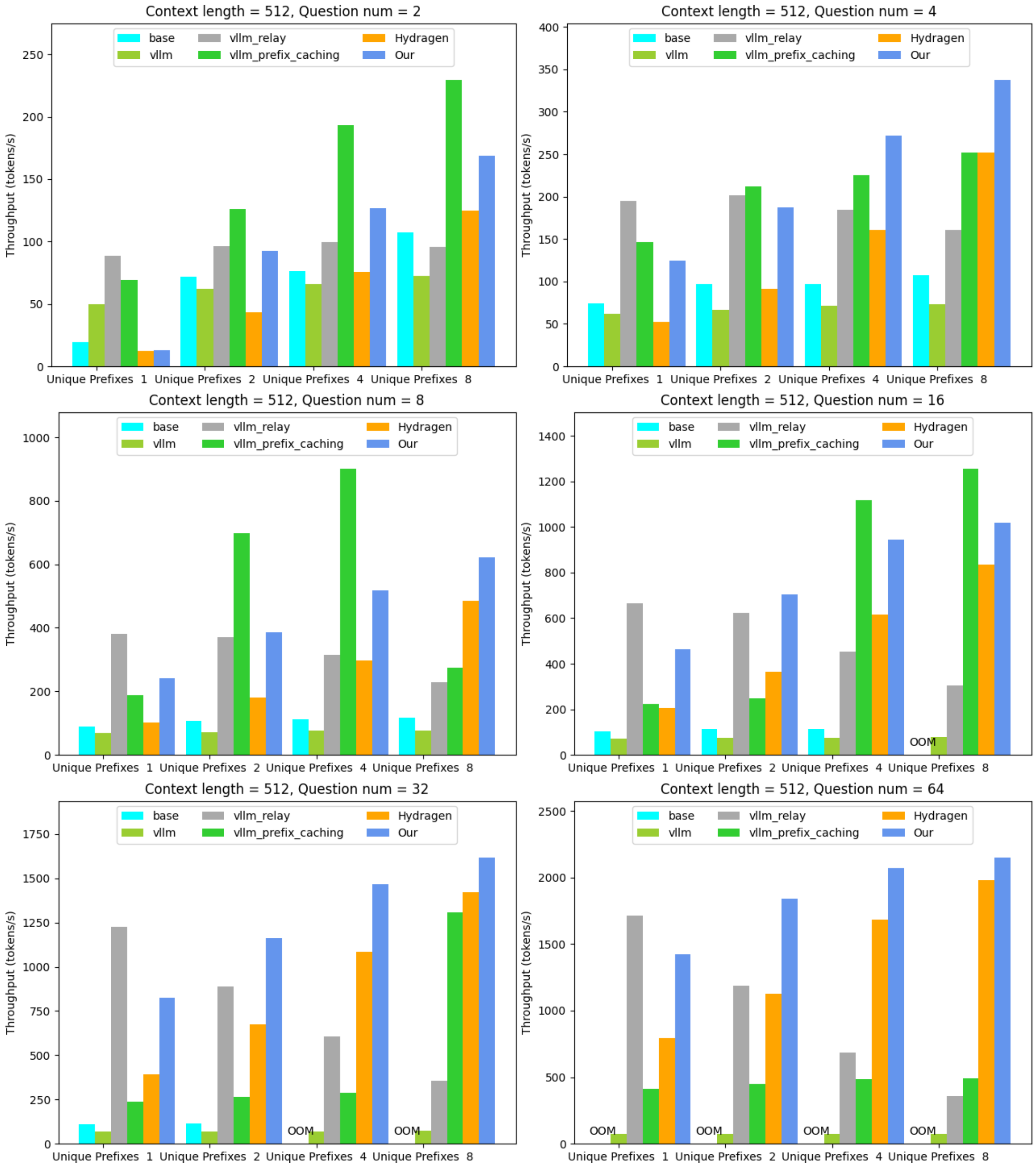

Figure 7: Throughputs of different methods when the number of unique documents changes in the LLM inference.CodeLlama-7b-Instruct attention inference Throughput w.r.t. number of unique documents (A100-SXM4-80GB GPU). We set the length of content to 512, the number queries for each context sweeps over the list of [2,4,8,16,32,64] , the length of each query to 12, the length of generated token to 5.

Method	model	New Tokens	Batch Size	Parallel Size	Latency(s)	Peak Memory(MB)
Our	llama-160m	10	1	128	38.9	3135
Our	llama-160m	10	2	64	31.2	3434
Our	llama-160m	10	4	32	4.7	4146
Our	llama-160m	10	8	16	8.0	5623
Our	llama-160m	10	16	8	9.9	8610
Our	llama-160m	10	1	64	59.5	12363
Our	llama-160m	10	2	32	11.8	12726
Our	llama-160m	10	4	16	12.9	13464
Our	llama-160m	10	8	8	13.9	14960
Our	llama-160m	10	16	4	17.5	17951
Our	llama-160m	10	1	32	21.9	6667
Our	llama-160m	10	2	16	22.1	7036
Our	llama-160m	10	4	8	27.8	7781
Our	llama-160m	10	8	4	32.1	9278
Our	llama-160m	10	16	2	37.7	12275

Table 6: Comparing the end-to-end NVIDIA-GeForce-RTX-3090 inference latency of parallel generation with baseline method. Numbers in parenthesis show the length of document, length of each question and number of all questions for prompting each LLM.($len_{doc} = 512, len_q = 10, num_q = 1024$). Results averaged over 50 runs.