Faster In-Context Learning for LLMs via N-Gram Trie Speculative Decoding

Jinglin Chen[1†], Qiwei Li[2†], Zuchao Li[3*], Baoyuan Qi[4], Guoming Liu[4], Haojun Ai[1], Hai Zhao[5], Ping Wang[6]

[1]School of Cyber Science and Engineering, [2]School of Computer Science,
[3]School of Artificial Intelligence, [6]School of Information Management, Wuhan University,
[4]Xiaomi Inc. [5]School of Computer Science, Shanghai Jiao Tong University,
{aaa1234, qw-line, zcli-charlie}@whu.edu.cn,
{qibaoyuan, liuguoming}@xiaomi.com, aihj@whu.edu.cn,
zhaohai@cs.sjtu.edu.cn, wangping@whu.edu.cn

Abstract

As a crucial method in prompt engineering, In-Context Learning (ICL) enhances the generalization and knowledge utilization capabilities of Large Language Models (LLMs) (Dong et al., 2024). However, the lengthy retrieved contexts and limited token throughput in autoregressive models significantly constrain reasoning speed. To address this challenge, we propose N-Gram Trie Speculative Decoding, a novel approach that leverages the overlap between context and model output. This method constructs an n-gram trie from the context to generate drafts, accelerating token generation for LLMs. We evaluate our approach on summarization, Retrieval-Augmented Generation (RAG), and context-based Question Answering (QA) tasks. Experimental results on Vicuna-7B, Llama2-7B-Chat, and Llama3-8B-Instruct demonstrate substantial speed improvements without compromising accuracy. Compared with various strong baselines, our method achieves the highest mean speedup, showcasing its effectiveness and efficiency. Our implement code is available here: https://github.com/mrlife219/Ngram-Trie.

1 Introduction

In-Context Learning (ICL) has emerged as a transformative paradigm in the field of prompt engineering, fundamentally reshaping how Large Language Models (LLMs) adapt to and perform on diverse tasks. By leveraging context information provided within the input prompt, ICL enables LLMs to generalize across tasks and domains without requiring task-specific fine-tuning. This capability has profound implications for the scalability and versatility of LLMs, allowing them to excel in various applications, such as context question answering, summarization and Retrieval-Augmented Generation (RAG). The ability to dynamically incorporate contextual knowledge has made ICL a cornerstone of modern LLM deployment, driving advancements in both academic research and industrial applications.

Despite its remarkable success, ICL faces a significant challenge: the extensive length of retrieved contexts and the inherent limitations of autoregressive token generation will result in slow reasoning speeds. As the complexity and length of context information increase, the computational overhead grows substantially, leading to delays in token generation and reduced efficiency. This bottleneck is particularly problematic in real-time applications, such as interactive systems or large-scale retrieval-augmented tasks, where speed is critical. Addressing this issue is essential to unlocking the full potential of ICL and enabling its broader adoption in time-sensitive scenarios.

Nowadays, there are many approaches to enhance the speed of decoding, such as KV-Cache, Prompt compression and speculative decoding. KV-Cache (Yang et al., 2025; Zhao et al., 2025b; Shi et al., 2025, 2024) leverages the characteristic of LLMs in auto-regressive generation, where substantial redundant computations occur. It significantly reduces the computational load by storing the precomputed Key (K) and Value (V) vectors of the LLM. Prompt compression (Zhao et al., 2025a) refers to a technique that, on the premise of not significantly altering the output quality of large models, reduces the number of input tokens of large models by shortening the length of the input Prompt. Its purpose is to improve the response speed of large models and lower the cost of large model calls. Speculative decoding (Leviathan et al., 2023; Cai et al., 2024; Li et al., 2024; He et al., 2023; Luo et al., 2024) can effectively accelerate model inference. This approach employs a

*Corresponding author.

†Equal contribution. This work was supported by the National Natural Science Foundation of China (No. 62306216), National Social Science Fundation of China [No. 24&ZD186], the National Natural Science Foundation of China [No. 72374161] and Xiaomi Open-Competition Research Program.

Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing, pages 18040–18051
November 4-9, 2025 ©2025 Association for Computational Linguistics

smaller, faster draft model to predict potential token sequences, which are then verified by the larger target model in parallel. By reducing the number of sequential decoding steps required by the target model, speculative decoding achieves significant speedups while maintaining output quality. However, this method often requires additional computational resources and careful tuning to balance the trade-off between speed and accuracy. REST (He et al., 2023) employs an external corpus to generate draft tokens, where the output tokens serve as prefixes to search for matching suffixes within the corpus. However, the excessive reuse of nodes and the global corpus tire reduce the acceptance rate of draft tokens. Lookahead Decoding (Fu et al., 2024) utilizes *n*-gram token histories as drafts for verification. While this method shows promise, its utility is primarily confined to scenarios where output tokens exhibit repetitive patterns, restricting its applicability in more diverse or dynamic contexts.

We propose N-Gram-Trie, a novel approach designed to accelerate token generation by exploiting the overlap between the context and the model's output. Then a trie is constructed by using the set of prefixes and suffixes. Build a trie from the prefix and suffix sets. In the model prediction stage, the draft is constructed through the nodes in the trie, which significantly improves reasoning speed without compromising output quality.

We evaluate our approach on summarization (Nallapati et al., 2016), Retrieval-Augmented Generation (Xia et al., 2024; Joshi et al., 2017) and context Question Answering (context QA) (Kamalloo et al., 2023) tasks. Multiple base models including Vicuna-7B (Zheng et al., 2023), Llama2-7B-Chat (Touvron et al., 2023) and Llama3-8B-Instruct (AI@Meta, 2024) are selected to be tested. Experiment results show that our method exhibits remarkable speedups on multiple models (mean 2.27x on Vicuna-7B, 2.10x on Llama2-7B-Chat and 1.56x on Llama3-8B-Instruct). Through the experiment comparison of the inference effect of the model, we prove that our method can accelerate the model in the process of context prompt inference without affecting the inference ability of the base model. We also conduct many further experiments around the speedup effect. This work not only addresses a critical limitation of ICL but also provide a effective method for more efficient and scalable deployment of LLMs in real-world applications.

The contribution of this paper can be summarized as follows:

- We propose an n-gram trie speculative decoding method. It can effectively use the potential overlap of the context and output tokens to accelerate model inference speed.

- We design a novel n-gram trie construction method. The trie constructed by n-gram sampling can effectively improve the acceptance rate of the draft.

- We conduct extensive experiments on several models. It shows our excellent acceleration effect on summarization, RAG and context QA tasks.

2 Related work

2.1 In-Context Learning

In-Context Learning (ICL) is an approach which makes LLMs perform better on specific-domain task. By giving only a few examples or hints, LLMs can find the underlying patterns of the context and answer the question correctly. (Dong et al., 2024). There are many approaches that can be applied to ICL. (Gu et al., 2023) extract the context by pretraining in a large corpus that contains long context. (Wei et al., 2023) propose symbol tuning, which uses tagged symbols as fine-tuned data for LLMs to study. (Wei et al., 2022) leverages instruction tuning in LLMs to enhance the zero-shot learning in LLMs.

Also, there are also large variety of downstream applications in the In Context learning. Prompt engineering is one of them. We can write an accurate prompt to make LLMs easier to understand the downstream tasks and give a satisfying answer. Prompt engineering are widely used in downstream tasks, such as Context QA, RAG, Few-shot Learning and Summary. Context QA (Kamalloo et al., 2023) tasks need LLMs to read the context and find the potential answers. Concatenating the context and question as prompts, LLMs can read them and give an answer in a efficient way. Like Context QA, RAG (Li et al., 2023) also needs retrieved context to carry out user's query. In Few-shot Learning, some examples about downstream tasks are usually given. LLMs can study the potential patterns between them and complete the task based on the given pattern. Summarization also needs the ability of context-reading.

2.2 Speculative Decoding

Speculative decoding (Leviathan et al., 2023) has been first proposed to ease the problem of throughput in LLM generation. Using a small draft model to *explore the token way*, target LLM just need to verify in one step without calculating repeatedly for getting these tokens.

Now, many speculative methods are based on the *guess and verify* approach. For example, Medusa (Cai et al., 2024) uses some trained Medusa head to predict the next n-tokens, but the prediction is not continuous and it degrades accept rate. Based on Medusa (Cai et al., 2024), Hydra (Ankner et al., 2024) take the continuation of the draft into consideration. The draft head can predict tokens with However, both Medusa and Hydra need extra training cost for draft models. Also, some works focus on the reusing of the former tokens or the external corpus. For instance, REST (He et al., 2023) uses an external corpus as draft. The output tokens is used as prefix to search for the suffix in the corpus. Nonetheless REST simply store all the corpus into suffix arrays offline. When inference starts, REST will search for the suffix in the array corpus. Directly search from the corpus has disadvantage because LLMs can't see the given arrays during output stage. The generation is independent from the external corpus. Also, the smaller corpus decreases accept rate while the bigger corpus makes REST harder to find the right suffix. Lookahead decoding (Fu et al., 2024) uses n-gram token history as draft to verify. But it is useful only when the output token is repeatedly generated. PLD/LLMA (Yang et al., 2023) (Saxena, 2023) also try to use the overlap between the input and output, but they simply copy certain numbers of suffixes without matching all the potential suffixes in the prompt. Both of them don't fully make use of the given prompt. Besides, both of them don't use pre-bulit Trie approach to enhance the speedup of the LLM inference.

2.3 Tree Attention

Tree attention (Miao et al., 2023) is proposed to solve the problem how a tree-structured token sequences can be decoding in parallel. By using an attention mask, the drafts can be easily integrated in one mask in inference. In the attention mask, Now tree attention is widely used in multi-draft verification.

SpecInfer (Miao et al., 2024) uses some small draft models to independently predict the potential tokens sequences, the tokens will then be clipped and put in the attention masks. Medusa (Cai et al., 2024) uses some positional draft heads to predict the top-k tokens in the next i place. It uses attention mask to integrate the top-k tokens into token sequences for prediction. REST (He et al., 2023) retrieved many tokens in a big suffix-array datastore. After clipping the tokens, He et al. also use tree attention mask to make a trie tree for faster decoding.

3 Proposed Method

The structure of N-Gram-Trie is shown in Figure 1. In the in-context prompt tasks, we first build an n-gram trie based on the context. The tree records the dependencies between preceding and following tokens of context. Subsequently, in the process of model inference, the draft of model inference is constructed by speculative decoding through the dependencies of n-gram trie, which can accelerate model inference speed.

3.1 N-Gram Trie Construction

Trie is a tree structure used to store and retrieve strings efficiently by organizing tokens in a prefix-based hierarchy. Its key advantage is faster suffix finding, which makes it suitable for speculative decoding (He et al., 2023). However, traditional Trie relies on massive documents to build for higher acceptance rate. It is difficult to construct an effective retrieval scheme in the case of a small amount of corpus. To this end, we design n-gram trie, sampled by n-gram sliding window, and then used the sampling results to build the trie. This method can effectively improve the efficiency and accuracy of suffix retrieval by constructing additional dependency chains.

N-Gram Sampling Specifically, for the context token list $T = \{t_1, t_2, ..., t_l\}$, we set a sliding window of n-grams for sampling. The sampling length is n. The sliding window moves token by token from the beginning to the end over T. In the sliding window workspace, we set a maximum prefix length L_p to split tokens in the window. The split part will be the prefix part and the suffix part of the segment tokens. The prefix P_i and suffix S_i can be expressed as follows:

$$\left.\begin{array}{l} P_i = \{t_i, t_{i+1}, ..., t_{i+L_p-1}\} \\ S_i = \{t_{i+L_p}, t_{i+L_p+1}, ..., t_{i+n-1}\} \end{array}\right\} i \in [1, l], \tag{1}$$

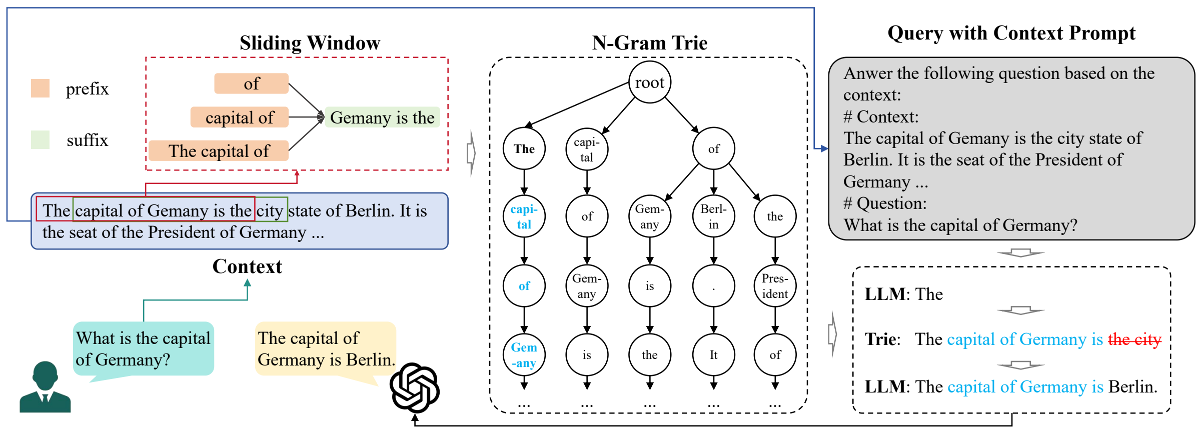

Figure 1: The structure of N-Gram-Trie. We sample through a sliding window of n-grams and get the prefixes and suffixes from the documents in that window. A trie can be constructed based on the set of prefixes obtained by window sliding sampling. In the process of model inference, the trie is used for speculative decoding to quickly predict the model output. The n in the n-gram sampling in the example of the figure is 6 and the maximum prefix length L_p is 3.

where i denotes the start index of the window. We establish the dependency between the prefix and suffix for each tokens group, and obtain the dependency set D by sliding window sampling. D can be defined in the following form:

$$D = \{< P_i, S_i, f > | i \in [1, l]\}, \tag{2}$$

where f is the frequency of dependency $< P_i, S_i >$ during the sampling process.

Trie Construction We build trie τ based on the sample results D and the construction process is as shown in Algorithm 1.

Specifically, for the prefix P_i in the sample set D, we traverse and split it according to the maximum prefix length to obtain its sub-prefixes SP_i. The process can be defined as:

$$\begin{aligned} SP_i &= \{SP_{i,j} | j \in (0, L_p)\} \\ &= \{P_i[j : L_p] | j \in (0, L_p)\}, i \in [1, l], \end{aligned} \tag{3}$$

where $j \in (0, L_p)$ denotes the cut length of the sub-prefix. By constructing additional prefix nodes, the corresponding prefix can be effectively found according to the model output in the retrieval process.

We take the dependency of each subprefix and its suffix as the basic unit for trie insertion. During insertion, the token t is used as the basic units of the tree nodes. We iterate from the root node, sharing a node for the same token. If there is no corresponding token in the current nodes, insert an additional token. The insertion logic is as follows:

$$node = \begin{cases} child, & if \quad t \in node.children \\ node_t, & if \quad t \notin node.children \end{cases}, \tag{4}$$

where child is the child of $node$ and $child.token = t$, $node_t$ is a new node built by t and inserted into the children of the original $node$. In this way, we let suffix nodes with the same prefix share the same prefix.

Note that we also record the frequency f of each node as it is inserted, in order to provide a priority reference for subsequent retrieval. Finally, by exploiting the samples in D, we can construct an efficient and accurate n-gram trie τ.

3.2 Draft Collecting and Matching

As shown in the gray area in Figure 1, in-context learning combines context with user query through templates in the prompt engineering. The query with context will serve as the reasoning basis for the target model. We define the tokens that have been generated by the s time step target model as $T_s = \{t_1, t_2, ..., t_k\}$. We will build the draft after s time step through the n-gram trie τ constructed in the former subsection that stores prefix and suffix dependency of context. Then, the target model will verify and revise the draft.

Draft Construction When searching for the draft, we firstly extract the suffix of new tokens T_s for prefix matching. At first, the length of the prefix token will be set to L_p. If T_s matches the prefix chain in τ, we can extract the suffix of this

Algorithm 1 Trie Generation

```
Input: T: Token list
       D: Collected n-gram sample results
       L_p: Maximum prefix length
Output: τ
Init root as τ                          ▷ an empty root node
for < P_i, S_i, f > ∈ D do
    for j ∈ (0, L_p) do
        subprefix ← P_i[j : L_p]
        key ← subprefix + S_i
        node ← root
        for t ∈ key do
            for child in node.children do
                if t = child.token then
                    node ← child
                    node.frequency.update(f)
                end if
            end for
            if t ∉ node.children then
                new ← Node(t, f)
                node.children.insert(new)
                node ← new
            end if
        end for
    end for
end for
return τ
```

prefix and break matching. If not found, we substract one token from prefix tokens until match the or prefix tokens length is 0. Then, we can obtain a suffix tree τ_s that matches the gernerated tokens T_s of the target model. To improve the acceptance rate of the draft, we prune the suffix tree according to the frequency f of nodes and extract nodes with lower f. The draft tree is not always very big, so sometimes the pruning is not used.

Specifically, refering to (He et al., 2023) and (Cai et al., 2024), we set a min-heap for storage of suffix chains. For each node v_k in τ_s, we build a draft d_k based on its path chain with the root node of τ_s. The priority of the draft is determined by the frequency of v_k. This process can be expressed as:

$$\begin{aligned} d_k &=< \text{Path}(v_r, v_k), f_k > \\ &=< \{v_r, v_1, ..., v_i, ..., v_k\}, f_k >, \\ & i \in [1, k], v_i \in \tau_s, \end{aligned} \tag{5}$$

where $\text{Path}(v_r, v_k)$ means the nodes from node v_r to node v_k. v_r is the root node of τ_s and f_k is the frequency of v_k.

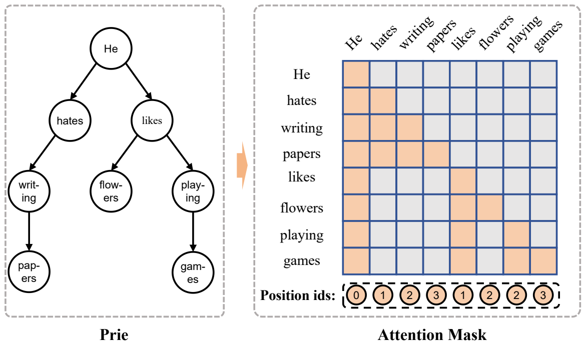

Figure 2: An Example of Tree Attention. The tokens in the orange part of the attention mask are visible to each other, and the tokens in the gray part are invisible to each other

Then, v_k will be placed in the min-heap in order of priority. Finally, alternative drafts are retained according to the length of min-heap. In this way, redundant nodes can be effectively removed and the pruning of suffix tree τ_s can be realized.

Model Verification Figure 2 shows an example of tree attention verifying the draft trie. For the draft trie τ_s, deep traverse it to obtain a linear list of tokens. In order to realize the tree attention, we set the same position id for the nodes of the same level. The specific form can be expressed as:

$$p_i = \text{Level}(v_i) + h, v_i \in \tau_s, \tag{6}$$

where p_i is the position id of t_i. $\text{Level}(v_i)$ is the level of v_i. h is the length of the preceding model tokens. This makes the tokens in each chain of the trie continuous.

Then, following tree attention meathod, we use attention mask to convert the draft tree into a 2-dimention mask m. For any tokens t_i and t_j, $m_{i,j}$ is 0 if there is a relationship between v_i and v_j in τ_s, otherwise it is 1. By matching the mask and position ids. The taget model can verify multiple branches of trie simultaneously.

4 Experiments

4.1 Experiment Setting

We implement all the experiments on one NVIDIA RTX 4090 with python version 3.9. All the experiments are run on greedy decoding. The pytorch version is 2.5.1 with CUDA version is 12.2.

4.1.1 Baselines

We choose the baselines provided on the Speculate Bench (Xia et al., 2024): vanilla inference

Model	Method	Spec-Bench Summary	Spec-Bench RAG	TriviaQA	Hagrid	Mean Speedup
Vicuna-7B	Vanilla	1.00× (1.00)	1.00×(1.00)	1.00×(1.00)	1.00×(1.00)	1.00x
	SpS	1.69×(<u>2.44</u>)	1.59×(2.30)	1.74×(<u>2.49</u>)	1.40×(2.46)	1.61x
	Medusa	1.48×(2.01)	1.45×(2.08)	1.45×(2.03)	<u>1.56×</u> (2.17)	1.49x
	SPACE	1.69×(2.26)	1.47×(1.91)	1.57×(2.26)	1.26×(2.05)	1.50x
	Hydra	**1.86×** (**2.70**)	<u>1.88×</u> (<u>2.90</u>)	<u>1.86×</u> (**2.84**)	1.52×(<u>2.98</u>)	<u>1.78×</u>
	Lookahead	1.29×(1.54)	1.19×(1.48)	1.27×(1.46)	0.95×(1.47)	1.18x
	REST	1.13×(1.65)	1.32×(1.89)	1.31×(1.71)	1.33×(1.82)	1.27x
	Ours	<u>1.75×</u> (2.39)	**3.48×** (**5.19**)	**1.92×** (2.36)	**1.94×** (**3.07**)	**2.27×**
Llama2-7B-Chat	Vanilla	1.00×(1.00)	1.00×(1.00)	1.00×(1.00)	1.00×(1.00)	1.00x
	SpS	1.25×(1.54)	<u>1.47×</u> (<u>1.91</u>)	1.35×(<u>1.86</u>)	1.38×(1.62)	1.36x
	Lookahead	**1.44×** (<u>1.59</u>)	1.40×(1.63)	<u>1.53×</u> (1.71)	1.38×(<u>1.97</u>)	<u>1.44×</u>
	REST	1.03×(1.54)	1.14×(<u>1.91</u>)	1.22×(1.68)	<u>1.42×</u> (1.47)	1.20x
	Ours	<u>1.28×</u> (**1.76**)	**3.62×** (**5.00**)	**1.89×** (**2.88**)	**1.61×** (**2.34**)	**2.10×**
Llama3-8B-Instruct	Vanilla	1.00×(1.00)	1.00×(1.00)	1.00×(1.00)	1.00×(1.00)	1.00x
	Lookahead	**1.25×** (**1.60**)	<u>1.18×</u> (1.51)	<u>1.58×</u> (1.54)	<u>1.38×</u> (<u>1.73</u>)	<u>1.50×</u>
	REST	0.93×(<u>1.54</u>)	1.14×(<u>1.91</u>)	1.13×(<u>1.61</u>)	1.02×(1.69)	1.05x
	Ours	<u>1.06×</u> (1.42)	**1.77×** (**2.11**)	**1.75×** (**1.86**)	**1.68×** (**2.34**)	**1.56×**

Table 1: Speedup Ratio and Accept Length Comparison. The data on the left means speedup and the data on the right means average accept length. The best performance for each metric is highlighted in **bold** font, while the second-best performance is indicated with an <u>underline</u>.

without any speculative methods, speculative Sampling (Chen et al., 2023), Medusa (Cai et al., 2024), SPACE (Yi et al., 2024), Hydra (Ankner et al., 2024), Lookahead (Fu et al., 2023) and REST (He et al., 2023). For Speculative Sampling, we used Llama-68m (Miao et al., 2024) as draft model to match Llama2-7b and use Vicuna-68m to match Vicuna-7b-v1.3. For Lookahead and REST, we simply use the same experiment setup in Spec-Bench(Xia et al., 2024).

4.1.2 Datasets

For datasets, we choose RAG, summary in Spec-bench (Xia et al., 2024). The RAG dataset contains 80 data from Natural Questions. Five retrieved documents from Wikipedia (Li et al., 2023) are concatenated. (Kwiatkowski et al., 2019) and the summary dataset is randomly chosen by CNN/Daily Mail (Nallapati et al., 2016). In addition, we make TriviaQA (Joshi et al., 2017) dataset for additional RAG task and make Hagrid (Kamalloo et al., 2023) dataset for context QA task. For TriviaQA task, we use bge-m3 (Chen et al., 2024a) and bge-reranker-v2-m3 (Chen et al., 2024b) to search for 5 relevant documents in Wikipedia corpus. For Hagrid task, we simply concatenate the given context and the question.

4.1.3 Base models

To conduct the experiments, we use three models for validation. One is Vicuna-7B-v1.3 (Zheng et al., 2023), One is Llama-2-7B-chat (Touvron et al., 2023) and the other is Llama-3-8B-Instruct (AI@Meta, 2024).

4.1.4 Hyperparameters

In the experiment, there are two hypermeter that need to be tuned: matched prefix L_p and gram-length n. So we conduct the experiment to test the efficiency. The details can be seen in table2. Also, we use FAISS (Douze et al., 2025) to store the embedding of the corpus using IVF-PQ method. The parameter of the number of clusters is 4096 and the parameter that the vector will be separated is 64. The clusters that will be searched is set to 16. We firstly encode all the corpus text using bge-m3 (Chen et al., 2024a), and search top-100 relevant texts for questions in triviaQA (Joshi et al., 2017). Then we rerank the texts using bge-reranker-v2-m3 (Chen et al., 2024b) to get the top-5 relevant contexts.

4.1.5 Metrics

Like other speculative decoding, we use *average accept length*, *mean speedup* in our evaluation. Average accept length shows the length that the

Hillary Clinton's security detail has added a second " Scooby " van to her motorcade , raising questions about the need for such an elaborate security measure . The second van , a GMC , is mechanically identical to the first van , a Chevrolet , but has different license plates. The Secret Service has employed de co y vehicles to confuse and disc ourage would-be a ttackers , but the use of two identical vans has raised e y eb rows . The vans were seen driving separately to Clinton's a ppointed location before leaving together in a seven-car motorcade. The Secret Service has decl ined to comment on the sec urity arrangements for dignitaries. The use of two Scooby vans has been observed in other instances , including when Presi dent Barack Obama returns to the White House after long trips . The Secret Service frequently deploys duplicates of aircra ft and cars it uses to transport VIPs , including Marine One, the president's customized hel icopter, which usually travel s with two deco ys . </s>

Figure 3: A Case on Summary Dataset. The inference exploration in one step. In the figure, the words are separated in tokens. Yellow text is generated by n-gram trie.

drafts are accepted in every decoding step. Usually the accept length is higher, the speedup can be higher. Mean speedup indicates the speedup of tokens throughput compared with decoding without any speculative method (baseline).

4.2 Main Results

The experiment results can be seen in Table 1. We can see that our method achieves optimal acceleration results compared to the baseline for all tasks except the summarization task. On average, the mean speedup of our method has achieved a meaningful improvement over the baselines (2.27× on Vicuna-7B, 2.10× on Llama2-7B-Chat and 1.56× on Llama3-8B-Instruct). It is worth noting that our method performs better than REST (He et al., 2023) on each model and task in the same speculative decoding with trie, demonstrating the superiority of our n-gram trie.

RAG Task. The experiment result on Spec-Bench RAG dataset show that the accept length of the drafts achieves 5.19 on Vicuna-7B, 5.00 on Llama2-7B-Chat and 2.11 on Llama3-8B-Instruct, making the speedup rate achieve 3.48×, 3.62× and 1.77×. Its acceleration performance is much better than that of the basic method REST. Experiment results in multiple models show that our method has the strongest speedup effect. It is far ahead of second place on all models.

On TriviaQA dataset, the speedup rate of our method achieves 1.92× on Vicuna-7B, 1.89× on Llama2-7B-Chat and 1.75× on Llama3-8B-Instruct. The speed-up performance is also the best. Even though the accept length of our approach is smaller than Hydra (Ankner et al., 2024) on Vicuna-7B, we still have a better throughput performance in this task.

Context QA Task. Our approach achieves the best results on all models (1.94× on Vicuna-7B, 1.61× on Llama2-7B-Chat and 1.68× on Llama3-8B-Instruct) on Hagrid datset, which outperforms other approaches by 0.19x-0.99×. Compared to the basic method REST, we have more speedup on all models. This fully demonstrates the advantages brought by n-gram trie.

Summary Task. The performance of our method on Spec-Bench Summary dataset is not the best. Global draft-getting method will result in wrongly draft clipping and unnecessarily draft-searching. In summarization tasks, input articles can be segmented into discrete text blocks, with most generated outputs demonstrating primary dependency on individual text units. The proposed methodology employs a global draft selection mechanism that may inadvertently incorporate non-essential drafts, potentially introducing redundant verification overhead. However, in RAG and multi-document QA scenarios, generated content exhibits stronger reliance on comprehensive document analysis, necessitating preservation and rigorous evaluation of multiple drafts. The experimental validation confirms that our method optimizes context utilization while maintaining computational efficiency through adaptive draft management. But our method still ranks second (1.75× on Vicuna-7B, 1.28× on Llama2-7B-Chat and 1.06× on Llama3-8B-Instruct) in terms of speedup and outperform REST.

4.3 Case Study

To fully demonstrate the speedup effect of our method, we conduct a case study on the Summary dataset. The example is shown in Figure 3. As can be seen from the figure, our speculative decoding method based on n-gram trie correctly predict the large model output many times. A large number of useful drafts provide an effective speedup scheme for in-context based model inference.

4.4 Hyperparameter Analysis

In this section, we will will conduct experiments on the hyperparameters n and L_p in our method to get the best hyperparameter configuration. We use the RAG task of the Spec-bench (Xia et al.,

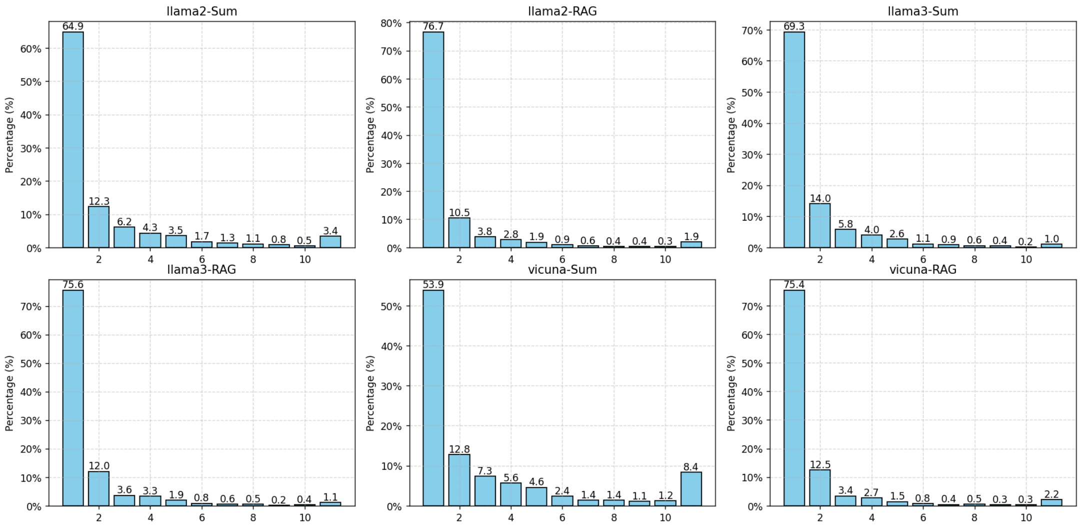

Figure 4: The accept length percentage in summary task and RAG tasks between models. The frequency of smaller accept length is usually larger except in the longest accept length

n/L_p	2	3	4	5
8	75.75	71.48	63.49	54.06
9	70.22	68.96	72.02	68.34
10	83.05	80.05	75.65	74.56
11	70.69	82.68	76.46	74.86
12	74.21	82.49	72.55	74.64
13	87.45	**92.46**	88.02	85.52
14	91.52	78.64	84.80	74.06
15	80.25	75.09	77.41	73.99
16	73.72	87.51	83.77	89.34

Table 2: Token Output Speed. The value in the table is token output number per second with different n and L_p

2024) on Llama2-7B-Chat to test the performance of N-Gram-Trie. We try the value between 8-16 for n-gram length n and 2-5 for maximum prefix length L_p. The performance of the N-Gram-Trie with different hyperparameter is shown in the Table 2. We can find that when n is small, the speedup effect will gradually deteriorate with the increase of L_p. We think this is because the excessively long L_p limits the length of the suffix, which in turn reduces the acceleration ability. When n is large, the token generation speed first speeds up and then slows down as L_p increases. This is mainly because when the suffix is not short, the longer prefix can better match the token of the model inference. Furthermore, it can be proved that when n value is large, appropriate redundant nodes can effectively improve the acceleration effect of speculative decoding. Statistically, we can see that the best choice of n is 13 and the maximum prefix length L_p is set to 3.

4.5 Time Analysis

The time expended on Trie search versus Tree construction during each step of draft retrieval from the Trie is illustrated in the figure 5. It can be seen that draft processing and draft tree making cost much more time than Trie searching.

Table 3: Speedup comparison across different num_draft values for Summarization and RAG tasks

num_draft	Sum. Speedup	RAG Speedup
8	1.5730x	1.9058x
16	1.5618x	1.9168x
32	1.5682x	1.9195x
64	1.5667x	1.7519x
128	1.4798x	1.8754x
256	1.4301x	1.7954x

4.6 Ablations on Retrieved Contexts

To evaluate the robustness of our method under noisy context conditions, we conducted an additional experiment comparing performance in both noisy and non-noisy retrieval settings across four representative tasks (RAG, Summarization, Trivia, and HAGRID). As shown in Table 4, our method still delivers consistent acceleration even under noisy retrieval, with speedup ratios such as 1.22× (RAG) and 2.27× (Trivia), although the overall gains are understandably reduced compared to

Table 4: Speedup comparison under noisy and clean (not noisy) retrieval contexts across four tasks on Qwen2-7b-Instruct

Context Quality	RAG (↑)	Summarization (↑)	Trivia (↑)	HAGRID (↑)
Noisy	1.22×	1.21×	2.27×	2.10×
Not noisy	1.87×	1.48×	1.71×	1.84×

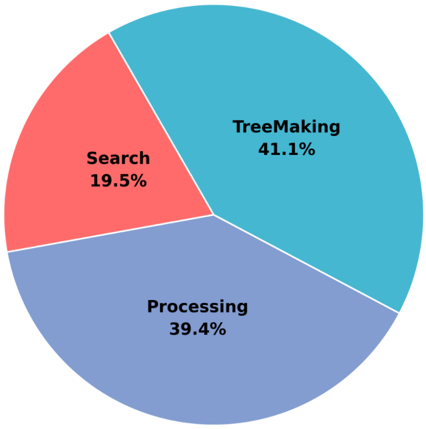

Figure 5: Time distribution in searching operation. The red opponent indicates time for searching in tree, the blue opponent means suffix processing and the cyan opponent means the attention tree making.

cleaner contexts (1.87× and 1.71× respectively).

We used the same hyperparameters as reported in the main paper $n = 3$ and $L_p = 10$ to ensure fair comparison. These results confirm that while performance is somewhat affected by noisy contexts, our method maintains its acceleration advantage and remains applicable in less ideal retrieval conditions.

4.7 Ablations on Num_Draft

We conducted the experiments using Qwen2-7b-instruct and different values of num_draft. The results are shown in the following table:

It can be seen that the speedup fluctuates with the number of drafts. and the speedup achieve the best when num_draft is 8. In summarization task, the speedup is 1.5730x, which is the best speedup we can get. For RAG task, the speedup is 1.9195x. In summarization task, the speedup doesn't change a lot, but in RAG task, the speedup become better when num_draft = 32. This means that num_draft shouldn't be too much larger than 32, otherwise the speedup will be affected.

4.8 Further Study

In order to explore the distribution of acceptance length of different models. We test Vicuna-7B (Zheng et al., 2023), Llama2-7B-Chat (Touvron et al., 2023), and Llama3-8B-Instruct (AI@Meta, 2024) on the Spec-Bench (Xia et al., 2024) dataset. The experimental results are shown in Figure 4. In this figure, it can be seen that the accept length concentrates in 1 (which means that no tokens are accepted). Besides, most of accept length is smaller than 4. And the percentage of the accept length decreases except in accept length = 11.

5 Conclusion

In this paper, we propose N-Gram Trie Speculative Decoding, a novel approach to accelerate in-context inference for large language models. By constructing an n-gram trie from the context through prefix and suffix dependencies, our method efficiently generates speculative decoding drafts, leveraging the overlap between context and model output. Extensive experiments on summarization, RAG, and context QA tasks demonstrate significant speedups—2.27x on Vicuna-7B, 2.10x on Llama2-7B-Chat, and 1.56x on Llama3-8B-Instruct—without compromising output quality. This work addresses a critical limitation of ICL, providing an effective and scalable solution for real-world LLM deployment.

6 Limitations

This approach presents several limitations. First, while the trie-based generation and search mechanism offers efficiency advantages, its current implementation has suboptimal aspects. A key issue arises when multiple suffix candidates share identical frequency scores, which may lead to the premature elimination of potentially useful draft outputs due to the fixed threshold imposed by the *num_draft* parameter. Second, the method exhibits strong dependency on the quality of external retrieved corpora - performance degradation becomes inevitable when processing noisy or irrelevant retrieval results. To address these challenges, our future work will focus on developing enhanced trie construction algorithms that incorporate more sophisticated frequency weighting schemes and context-aware candidate selection strategies.

References

AI@Meta. 2024. Llama 3 model card.

Zachary Ankner, Rishab Parthasarathy, Aniruddha Nrusimha, Christopher Rinard, Jonathan Ragan-Kelley, and William Brandon. 2024. Hydra: Sequentially-dependent draft heads for medusa decoding. *Preprint*, arXiv:2402.05109.

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. *Preprint*, arXiv:2302.01318.

Jianlv Chen, Shitao Xiao, Peitian Zhang, Kun Luo, Defu Lian, and Zheng Liu. 2024a. Bge m3-embedding: Multi-lingual, multi-functionality, multi-granularity text embeddings through self-knowledge distillation. *Preprint*, arXiv:2402.03216.

Jianlv Chen, Shitao Xiao, Peitian Zhang, Kun Luo, Defu Lian, and Zheng Liu. 2024b. Bge m3-embedding: Multi-lingual, multi-functionality, multi-granularity text embeddings through self-knowledge distillation. *Preprint*, arXiv:2402.03216.

Qingxiu Dong, Lei Li, Damai Dai, Ce Zheng, Jingyuan Ma, Rui Li, Heming Xia, Jingjing Xu, Zhiyong Wu, Tianyu Liu, Baobao Chang, Xu Sun, Lei Li, and Zhifang Sui. 2024. A survey on in-context learning. *Preprint*, arXiv:2301.00234.

Matthijs Douze, Alexandr Guzhva, Chengqi Deng, Jeff Johnson, Gergely Szilvasy, Pierre-Emmanuel Mazaré, Maria Lomeli, Lucas Hosseini, and Hervé Jégou. 2025. The faiss library. *Preprint*, arXiv:2401.08281.

Yichao Fu, Peter Bailis, Ion Stoica, and Hao Zhang. 2023. Breaking the sequential dependency of llm inference using lookahead decoding.

Yichao Fu, Peter Bailis, Ion Stoica, and Hao Zhang. 2024. Break the sequential dependency of llm inference using lookahead decoding. *arXiv preprint arXiv:2402.02057*.

Yuxian Gu, Li Dong, Furu Wei, and Minlie Huang. 2023. Pre-training to learn in context. *Preprint*, arXiv:2305.09137.

Zhenyu He, Zexuan Zhong, Tianle Cai, Jason D Lee, and Di He. 2023. Rest: Retrieval-based speculative decoding. *Preprint*, arXiv:2311.08252.

Mandar Joshi, Eunsol Choi, Daniel Weld, and Luke Zettlemoyer. 2017. triviaqa: A Large Scale Distantly Supervised Challenge Dataset for Reading Comprehension. *arXiv e-prints*, arXiv:1705.03551.

Ehsan Kamalloo, Aref Jafari, Xinyu Zhang, Nandan Thakur, and Jimmy Lin. 2023. HAGRID: A human-llm collaborative dataset for generative information-seeking with attribution. *arXiv:2307.16883*.

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

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.

Chaofan Li, Zheng Liu, Shitao Xiao, and Yingxia Shao. 2023. Making large language models a better foundation for dense retrieval. *Preprint*, arXiv:2312.15503.

Yuhui Li, Fangyun Wei, Chao Zhang, and Hongyang Zhang. 2024. EAGLE: Speculative sampling requires rethinking feature uncertainty. In *International Conference on Machine Learning*.

Xianzhen Luo, Yixuan Wang, Qingfu Zhu, Zhiming Zhang, Xuanyu Zhang, Qing Yang, Dongliang Xu, and Wanxiang Che. 2024. Turning trash into treasure: Accelerating inference of large language models with token recycling. *Preprint*, arXiv:2408.08696.

Xupeng Miao, Gabriele Oliaro, Zhihao Zhang, Xinhao Cheng, Zeyu Wang, Zhengxin Zhang, Rae Ying Yee Wong, Alan Zhu, Lijie Yang, Xiaoxiang Shi, Chunan Shi, Zhuoming Chen, Daiyaan Arfeen, Reyna Abhyankar, and Zhihao Jia. 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*, ASPLOS '24, page 932–949. ACM.

Xupeng Miao, Gabriele Oliaro, Zhihao Zhang, Xinhao Cheng, Zeyu Wang, Zhengxin Zhang, Rae Ying Yee Wong, Alan Zhu, Lijie Yang, Xiaoxiang Shi, et al. 2023. Specinfer: Accelerating generative large language model serving with tree-based speculative inference and verification. *arXiv preprint arXiv:2305.09781*.

Ramesh Nallapati, Bowen Zhou, Cícero Nogueira dos Santos, Çaglar Gülçehre, and Bing Xiang. 2016.

Abstractive text summarization using sequence-to-sequence rnns and beyond. In *Proceedings of the 20th SIGNLL Conference on Computational Natural Language Learning, CoNLL 2016, Berlin, Germany, August 11-12, 2016*, pages 280–290. ACL.

Apoorv Saxena. 2023. Prompt lookup decoding.

Luohe Shi, Zuchao Li, Lefei Zhang, Guoming Liu, Baoyuan Qi, and Hai Zhao. 2025. Kv-latent: Dimensional-level kv cache reduction with frequency-aware rotary positional embedding. *Preprint*, arXiv:2507.11273.

Luohe Shi, Hongyi Zhang, Yao Yao, Zuchao Li, and Hai Zhao. 2024. Keep the cost down: A review on methods to optimize llm' s kv-cache consumption. *Preprint*, arXiv:2407.18003.

Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, Dan Bikel, Lukas Blecher, Cristian Canton Ferrer, Moya Chen, Guillem Cucurull, David Esiobu, Jude Fernandes, Jeremy Fu, Wenyin Fu, Brian Fuller, Cynthia Gao, Vedanuj Goswami, Naman Goyal, Anthony Hartshorn, Saghar Hosseini, Rui Hou, Hakan Inan, Marcin Kardas, Viktor Kerkez, Madian Khabsa, Isabel Kloumann, Artem Korenev, Punit Singh Koura, Marie-Anne Lachaux, Thibaut Lavril, Jenya Lee, Diana Liskovich, Yinghai Lu, Yuning Mao, Xavier Martinet, Todor Mihaylov, Pushkar Mishra, Igor Molybog, Yixin Nie, Andrew Poulton, Jeremy Reizenstein, Rashi Rungta, Kalyan Saladi, Alan Schelten, Ruan Silva, Eric Michael Smith, Ranjan Subramanian, Xiaoqing Ellen Tan, Binh Tang, Ross Taylor, Adina Williams, Jian Xiang Kuan, Puxin Xu, Zheng Yan, Iliyan Zarov, Yuchen Zhang, Angela Fan, Melanie Kambadur, Sharan Narang, Aurelien Rodriguez, Robert Stojnic, Sergey Edunov, and Thomas Scialom. 2023. Llama 2: Open foundation and fine-tuned chat models. *Preprint*, arXiv:2307.09288.

Jason Wei, Maarten Bosma, Vincent Y. Zhao, Kelvin Guu, Adams Wei Yu, Brian Lester, Nan Du, Andrew M. Dai, and Quoc V. Le. 2022. Finetuned language models are zero-shot learners. *Preprint*, arXiv:2109.01652.

Jerry Wei, Le Hou, Andrew Lampinen, Xiangning Chen, Da Huang, Yi Tay, Xinyun Chen, Yifeng Lu, Denny Zhou, Tengyu Ma, and Quoc V. Le. 2023. Symbol tuning improves in-context learning in language models. *Preprint*, arXiv:2305.08298.

Heming Xia, Zhe Yang, Qingxiu Dong, Peiyi Wang, Yongqi Li, Tao Ge, Tianyu Liu, Wenjie Li, and Zhifang Sui. 2024. Unlocking efficiency in large language model inference: A comprehensive survey of speculative decoding. In *Findings of the Association for Computational Linguistics ACL 2024*, pages 7655–7671, Bangkok, Thailand and virtual meeting. Association for Computational Linguistics.

Haoqi Yang, Luohe Shi, Qiwei Li, Zuchao Li, Ping Wang, Bo Du, Mengjia Shen, and Hai Zhao. 2025. Faster moe llm inference for extremely large models. *Preprint*, arXiv:2505.03531.

Nan Yang, Tao Ge, Liang Wang, Binxing Jiao, Daxin Jiang, Linjun Yang, Rangan Majumder, and Furu Wei. 2023. Inference with reference: Lossless acceleration of large language models. *Preprint*, arXiv:2304.04487.

Hanling Yi, Feng Lin, Hongbin Li, Peiyang Ning, Xiaotian Yu, and Rong Xiao. 2024. Generation meets verification: Accelerating large language model inference with smart parallel auto-correct decoding. *arXiv preprint arXiv:2402.11809*.

Yi Zhao, Zuchao Li, Hai Zhao, Baoyuan Qi, and Guoming Liu. 2025a. Dac: A dynamic attention-aware approach for task-agnostic prompt compression. *Preprint*, arXiv:2507.11942.

Yi Zhao, Yajuan Peng, Cam-Tu Nguyen, Zuchao Li, Xiaoliang Wang, Hai Zhao, and Xiaoming Fu. 2025b. Smallkv: Small model assisted compensation of kv cache compression for efficient llm inference. *Preprint*, arXiv:2508.02751.

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric. P Xing, Hao Zhang, Joseph E. Gonzalez, and Ion Stoica. 2023. Judging llm-as-a-judge with mt-bench and chatbot arena. *Preprint*, arXiv:2306.05685.

Llama2 Summarization				
n/L_p	2	3	4	5
8	17.4	14.7	17.7	17.5
9	13.7	18.1	16.6	18.0
10	18.3	15.5	17.9	14.7
11	17.8	18.2	16.0	16.8
12	17.7	17.9	**18.6**	18.4
13	14.5	17.8	18.2	18.0
14	15.1	18.1	17.8	17.8
15	18.0	17.7	13.5	18.1

Llama2 RAG				
n/L_p	2	3	4	5
8	27.5	41.1	37.6	24.5
9	45.2	42.6	31.2	35.1
10	45.1	43.7	36.6	39.5
11	50.9	43.8	44.0	36.9
12	52.3	49.7	45.5	40.9
13	38.7	38.7	50.1	34.1
14	54.3	48.2	51.0	49.7
15	**55.3**	38.9	48.3	39.3

Vicuna Summarization				
n/L_p	2	3	4	5
8	18.7	23.1	22.5	18.8
9	25.7	22.1	23.1	17.7
10	25.5	25.9	25.6	23.1
11	24.3	25.6	25.8	23.8
12	25.1	25.1	**26.0**	25.9
13	24.7	24.5	24.5	25.5
14	18.5	24.5	25.1	24.9
15	25.1	25.1	23.9	24.2

Vicuna RAG				
n/L_p	2	3	4	5
8	35.7	42.8	29.5	25.2
9	50.3	47.6	33.2	29.9
10	38.2	50.5	36.1	44.0
11	53.2	48.9	37.9	35.9
12	54.8	52.1	50.8	50.8
13	55.1	54.0	52.8	50.9
14	55.3	41.8	54.5	53.8
15	**59.6**	54.1	41.2	54.6

Llama3 Summarization				
n/L_p	2	3	4	5
8	13.6	13.6	13.9	13.7
9	13.7	**13.8**	13.6	13.6
10	13.6	13.5	13.5	13.5
11	13.7	13.5	13.4	13.5
12	13.5	13.5	13.5	13.5
13	13.3	13.5	13.4	13.7
14	13.3	13.2	13.5	13.5
15	13.6	13.3	13.4	13.4

Llama3 RAG				
n/L_p	2	3	4	5
8	**20.9**	21.0	19.6	19.2
9	20.6	20.7	19.8	19.8
10	20.0	20.2	20.6	19.9
11	20.0	20.1	20.3	20.5
12	19.9	20.0	20.5	20.3
13	20.4	20.5	20.0	19.1
14	20.1	20.1	20.3	20.2
15	20.2	20.0	20.0	20.6

Table 5: Performance comparison of Llama2, Vicuna, and Llama3 across Summarization and RAG tasks

A More experiments about parameters

We conduct more experiments in summary and RAG tasks. The results of the experiment can be seen in Table 5.

There is a downgrade of the experiment result because the experiments are running with other processes using GPU. These results show that if the model and the dataset change, n and L_p also need to adjust for better speculative performance. What's more, the strategy of choosing the parameters may vary in GPU conditions. Also, we found that in Llama3, there is no significant degrade in the performance as the data changes. The choice of the n and L_p may need further discussion because now we haven't found the best way of selecting n and L_P. The best choices may be different due to many factors, which needs more experiments to check.

B Trie time between n and L_p

The Trie-construction can be used before inference. While we perform the prompt-guided tasks, the Trie will firstly be created before inference. However, we don't modify the Trie during an inference process. And we don't delete the Trie node at first. during inference, we can search for the Trie node to gain the smaller Trie for tree-attention as the path can be called prefix and the child node is the selected drafts. The draft-cutting process is done after getting the smaller Trie.

Also the trie can be built dynamically with data shifts. The overhead of the Trie-making can be seen on Table **??** (Llama2-Sum).

As the given data, it can be seen that the overhead of Trie generation is roughly 10ms, and the inference time varies from 82.3ms to 162.0ms. The Trie generation time is much smaller than the inference time. Besides, it can be seen that the inference time would vary due to the changes of n and L_p.

Table 6: Average Trie Time and Average Inference Time (unit: ms) under different n and L_p

Avg Trie Time (ms)				
n/L_p	2	3	4	5
8	11.55	10.31	11.00	12.62
9	9.13	9.45	10.54	11.35
10	8.23	9.46	10.07	11.27
11	8.24	10.11	11.15	11.88
12	9.06	9.58	10.52	11.54
13	9.06	9.54	11.46	11.72
14	10.83	9.71	10.57	14.26
15	8.93	9.95	12.75	11.75

Avg Inference Time (ms)				
n/L_p	2	3	4	5
8	162.5	89.5	94.4	129.0
9	86.7	91.2	110.2	92.8
10	91.6	89.0	130.3	86.9
11	82.9	91.5	87.3	129.4
12	82.3	86.1	86.6	87.3
13	104.7	97.7	82.3	105.6
14	86.5	104.8	81.5	83.3
15	84.6	103.7	107.1	97.0