Multi-Branch Self-Drafting for LLM Inference Acceleration

Zipeng Gao[1], Qingrong Xia[2], Tong Xu[1*], Xinyu Duan[2], Zhi Zheng[1*], Zhefeng Wang[2*], Enhong Chen[1]

[1]School of Computer Science and Technology, University of Science and Technology of China
[2]Huawei Cloud
gaozp619@mail.ustc.edu.cn, xiaqingrong@huawei.com, tongxu@ustc.edu.cn, duanxinyu@huawei.com, zhengzhi97@mail.ustc.edu.cn, wangzhefeng@huawei.com, cheneh@ustc.edu.cn

Abstract

The autoregressive decoding paradigm endows large language models (LLMs) with superior language generation capabilities; however, its step-by-step decoding process inherently limits decoding speed. To mitigate these constraints, the prevalent "draft and validation" strategy enables parallel validation of candidate drafts, allowing LLMs to decode multiple tokens simultaneously during one model forward propagation. However, existing methodologies for obtaining drafts often incur additional overhead in communication or training process, or statistical biases from the corpus. To this end, we propose an innovative draft generation and maintenance approach that leverages the capabilities of LLM itself. Specifically, we extend the autoregressive decoding paradigm to a multi-branch drafting procedure, which can efficiently generate draft sequences without any additional models or training process, while preserving the quality of the generated content by maintaining LLM parameters. Experiments across various open-source benchmarks show that our method generates 2.0 to 3.2 tokens per forward step and achieves around 2 times improvement of end-to-end throughput compared to the autoregressive decoding strategy.

Code — https://github.com/ZipECHO/Self-Draft

Introduction

Large Language Models (LLMs) have demonstrated their outstanding capabilities in natural language understanding and generation, rendering them indispensable tools across a broad spectrum of disciplines, including linguistics (Jipeng et al. 2023; Chen et al. 2024a), cognitive psychology (Hagendorff 2023), social science (Lei et al. 2024), and recommendation system (Zheng et al. 2024; Zhao et al. 2024c; Wu et al. 2024). Nonetheless, it is essential to note that their auto-regressive decoding process, loading billion-parameters of LLM from the memory to the on-chip cache to merely generate one token, leading to a notable constraint on decoding speed and a subsequent increase in the computational cost of inference in practice.

Many efforts are focusing on enhancing the efficiency of LLM inference from various angles. Model compression techniques like model distillation (Liang et al. 2021; Wu, Wu, and Huang 2021), pruning (Wang, Wohlwend, and Lei 2020; Ma, Fang, and Wang 2023), and quantization (Xiao et al. 2023; Bai et al. 2022) aim to reduce the number of parameters loaded at each model forward step. Additionally, optimizing resource management and utilization through model and data parallelization (Cai et al. 2024; Chen et al. 2024b; Ghazvininejad et al. 2019), caching mechanisms (He et al. 2023; Zhao et al. 2024b), and prefetching strategies seeks to maximize the potential of different hardware resources while minimizing redundant resource consumption. Recently, there has been a surge of interest in the draft-and-verify paradigm among researchers (He et al. 2023; Zhao et al. 2024b; Zhang et al. 2023; Cai et al. 2024; Leviathan, Kalman, and Matias 2023a; Chen et al. 2023; Sato et al. 2021). This paradigm typically consists of two components: draft and verification modules. The draft module generates draft sequences, and then LLMs verify them in parallel. This pipeline allows LLMs to decode several tokens in a single forward step, enhancing decoding speed while maintaining the output quality.

Typically, draft sequences are produced using smaller, more efficient LLMs as **draft models** to expedite the drafting process (Sato et al. 2021; Zhao et al. 2024a; Zhang et al. 2023; Chen et al. 2023). However, despite LLMs being able to verify and accept more tokens in one forward step, the substantial computational and serial communication overhead incurred by the draft model results in only a marginal improvement in end-to-end decoding speed. Besides, the alignment between the draft model and the LLM also requires additional training. Certain studies have modified the architecture of LLMs to enable the generation of multiple tokens in a single forward pass (Cai et al. 2024; Monea, Joulin, and Grave 2023; Stern, Shazeer, and Uszkoreit 2018). However, these methods typically require additional fine-tuning to mitigate the impact of such modifications on the quality of the generated output.

In an effort to mitigate the computational demands at the draft stage, some efforts have tried to extract candidate sequences from some **pre-built caches**, followed by verification through larger models (He et al. 2023; Zhao et al. 2024b). This strategy is more cost-effective for acquiring candidate sequences, and there is almost no additional computational overhead in the reasoning process. Nevertheless,

*Corresponding author

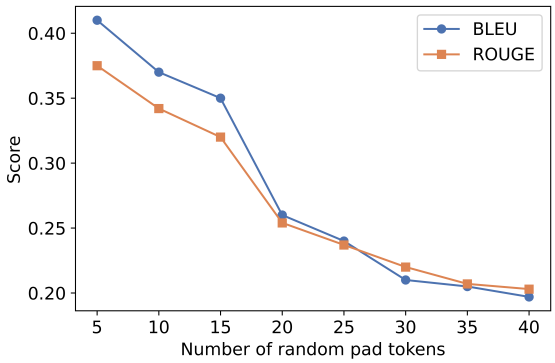


Figure 1: BLEU and ROUGE score for the different number of padding tokens reference to the autoregressive decoding.

the adaptability of this method is limited due to discrepancies between the datasets used for cache construction, model preference, and the actual inference scenarios. For instance, the throughput of the PIA (Zhao et al. 2024b) method will decrease more than 30% if we utilize a GSM-8K cache but perform inference on the Dolly-15K dataset. This result demonstrates that the pre-built cache methods are not suitable for practical complex reasoning scenarios.

Therefore, we believe that the pre-built cache is better suited for describing generalized linguistic expressions. To bridge the gap between the cache and the context, more contextualized drafts should be used to update the cache during the actual inference process. Fortunately, we observed that even when the inputs contain some noise, the outputs of the LLM still demonstrate a significant proportion of overlap with the content of normal decoding outputs. As shown in Figure 1, we randomly pad several tokens to the end of prompt, and use BLEU and ROUGE score to depict the overlap grams between the pad decoding result and vanilla result. We can see that as the number of padding tokens increases, both BLEU and ROUGE scores gradually decline. Nevertheless, these scores still remain above 0.2, indicating that at least 20% of the grams in the padded decoding results are consistent with those in the vanilla results..

Based on this, we extend the autoregressive decoding process into multi-branch drafting, and develop our in-context drafting method, **Self-Draft**. Self-Draft does not need additional draft models to generate draft sequences but generates the draft sequences by LLM itself with additional draft branches and could be executed in parallel with the autoregressive decoding process. Furthermore, these additional draft branches leverage the robustness of the LLM and thus do not require fine-tuning. By appropriately setting the attention mask for each branch, they can produce contextually relevant draft sequences. Finally, introducing these branches does not change the structure or parameters of the LLM and therefore do not have any effect on the quality of the inference content. Compared with the ordinary draft model methods, Self-Draft avoids the need to select and align the draft model with LLMs, thus eliminating additional computational and communication overheads between the draft model and the target model. Moreover, as the draft sequences are produced by the LLM within the current contextual setting, there is no need for additional training to acquire draft sequences that conform to both the contextual demands and the intrinsic preferences of LLM.

Overall, our contributions are as follows:

- We proposed a novel LLM decoding strategy, Self-Draft, which could generate and verify the drafts by additional training free draft branches that could execute with autoregressive decoding in parallel.
- We achieved more comprehensive draft cache maintenance by considering both the common and frequently used expression drafts from the corpus and the contextual drafts extracted from the drafting branches.
- Experiments conducted on various open-source LLMs and benchmarks demonstrated that Self-Draft could achieve approximately a two-fold increase in end-to-end speed and decode more than two tokens per forward step.

Related Works

Model Inference Acceleration

Numerous research efforts are focused on enhancing the inference speed of autoregressive large language models during deployment from various dimensions (Yuan et al. 2024). These efforts include model compression, aimed at reducing the model parameter size, which involves decreasing the precision of parameters (through model quantization) and reducing the number of parameters (through model pruning and distillation) to alleviate computational load during inference (Ma, Fang, and Wang 2023; Wu, Wu, and Huang 2021; Bai et al. 2022). System scheduling optimization seeks more effective allocation of system resources, covering strategies like memory management (Kwon et al. 2023), operator fusion (Aminabadi et al. 2022), and parallel services (Yu et al. 2022). Additionally, efficient decoding algorithms mainly accelerate model inference in two ways: first, reducing the amount of parameters needed for inferring each token, such as Mixture of Experts (MoE) (Fedus, Dean, and Zoph 2022), early exit (Chen et al. 2024b), and KV-cache; and second, maximizing the number of tokens decoded during each forward, such as parallel decoding and speculative decoding (Sato et al. 2021; Song et al. 2021; Santilli et al. 2023; Chen et al. 2023).

Speculative Decoding

Speculative decoding (Chen et al. 2023; Sato et al. 2021) is widely employed in scenarios that demand low latency and lossless inference due to its exceptional decoding speed. It is primarily composed of two stages: draft and verification. The draft stage aims to create high-quality preliminary text drafts, while the verification stage involves extracting relevant candidate sequences from the drafts and validating them in parallel. Naturally, the superior quality of the draft is intricately linked to the number of tokens that a large language model can decode in a single forward pass. Thus, the pursuit of crafting high-quality drafts is the focal point of research.

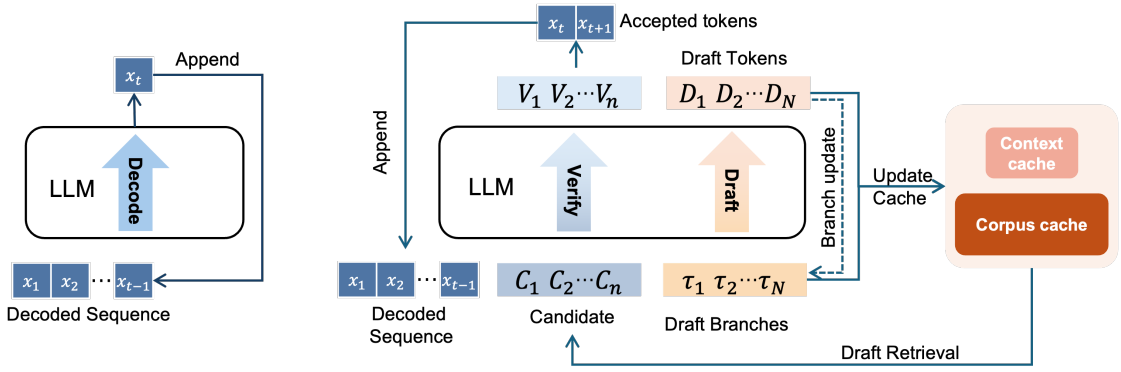


Figure 2: Overview of autoregressive decoding (left) and Self-Draft decoding (right).

Drafts are typically generated by a smaller model (Chen et al. 2023; Leviathan, Kalman, and Matias 2023b), but since the inference processes of the smaller and larger models are often sequential, careful selection of the model—both in terms of size and architecture is crucial to balance the quality of generation with speed. To reduce the time spent on retrieving candidate sequences, some researchers (He et al. 2023; Zhao et al. 2024b) have attempted to extract suitable candidates from a pre-built cache directly. However, this approach requires appropriate data to construct the caches, and due to potential disparities between the cache and actual inference scenarios—particularly in cross-domain situations—the effectiveness of this method is often suboptimal.

Parallel Decoding

Although there is some overlap between parallel decoding and speculative decoding, the focus of parallel decoding is not on efficiently generating high-quality drafts but on how to utilize the LLM itself to generate drafts and avoid serial communication between models. To achieve this, some studies have attempted to change the architecture of autoregressive models during the training or SFT phase, transforming them into non-autoregressive models to facilitate the parallel generation of subsequent tokens (Cai et al. 2024; Stern, Shazeer, and Uszkoreit 2018; Monea, Joulin, and Grave 2023). On the other hand, some researchers noticed that LLMs are robust to noise, so they leverage a parallel decoding method such as Jacobi decoding to predict several forthcoming tokens (Santilli et al. 2023; Song et al. 2021; Fu et al. 2024; Zhang et al. 2023). These methods do not require any modifications to the model or auxiliary model which is similar to our method and we chose a SOTA work, LADE (Fu et al. 2024) as the baseline.

Methods

Overview of Self-Draft

As illustrated in Figure 2, our decoding strategy comprises two modules: the model parallel forward module and the cache update and maintenance module. The former encompasses candidate drafts verification and contextual drafts generation. Notably, these operations can be executed in parallel within a single model forward pass. The latter module is responsible for managing a comprehensive n-grams cache, housing statistical information derived from both general corpus data and multi-branch draft sequences. In the rest of this section, we will first introduce the cache initialization by corpus; then, we will introduce the parallel drafting and verification together with the context cache maintenance.

Notations	Descriptions
$\mathbf{X} = [x_1, x_2, ..., x_{t-1}]$	Input prompt
$\mathcal{T} = \{\tau_0, \tau_1, ...\tau_N\}$	Draft branches
$\mathbb{D} = \{\mathcal{D}_i \mid 0 < i \leq N\}$	Draft results for each draft branch
$\mathcal{C} = [\mathcal{C}_1, \mathcal{C}_2, ...]$	Candidate drafts
$\mathbb{V} = [V_1, V_2, ...]$	Verification results

Table 1: Notations

Draft Cache Initialization

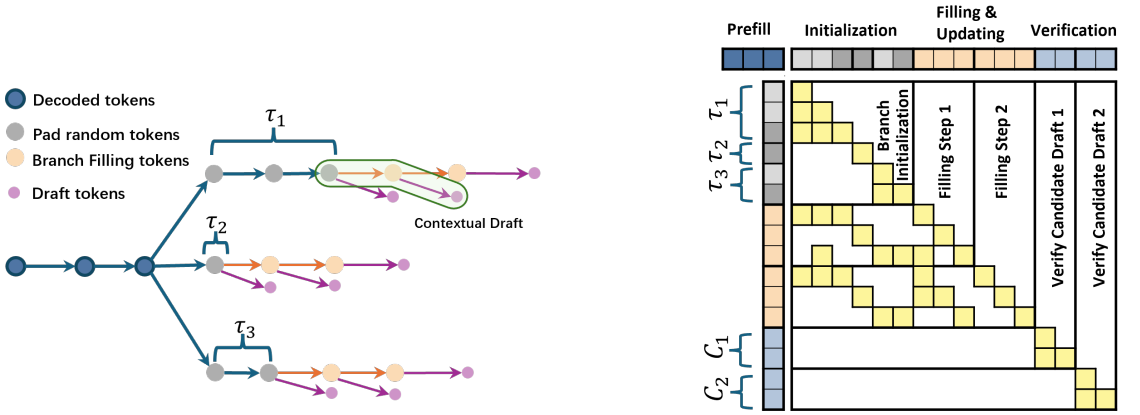
The cache is constructed from two sources, the general corpus data and the contextual drafts of the outputs from each draft branch of the LLMs, which are employed to generate general and frequent expression and contextual drafts, respectively. In this section, we will explain how to initialize a corpus cache by corpus data. In the following section, in conjunction with model forwarding, we will describe how to obtain and update this context cache based on multiple parallel drafting branches.

We obtain common expressions through simple statistical methods. Specifically, we chose an open-source general corpus, OANC[1], which includes both dialogues and writing scenarios, containing 14 million words. Based on OANC, we extract the co-occurrence relationships among all grams in the OANC dataset. To ensure comprehensive matching, we assign different priorities to different prefix lengths and frequencies, which means longer prefixes and higher frequencies receive higher priority when retrieving.

Self-Draft Model Forward

The model forward procedure of Self-Draft decoding comprises two parts: multi-branch drafting and candidate drafts

[1]https://anc.org/data/oanc/



(a) Multi-branch drafting.

(b) Attention masks for drafting branches and draft verification.

Figure 3: Illustration of multi-branch drafting and attention mask setting.

verification. The former is employed to generate contextual draft sequences that could be applied to update the context cache, while the latter verifies candidate drafts retrieved from the caches.

Multi-branch drafting. As shown in Figure 2. The conventional process of autoregressive decoding can be formulated as follows. Given a prompt with $t-1$ tokens, $\mathbf{X} = [x_1, x_2, ..., x_{t-1}]$ as input, LLM M will calculate the distribution of the next token, denoted as $P_M(y|\mathbf{X})$. Then, the next token x_t will be obtained from the distribution P by different sample strategies $\mathcal{S}$ (e.g., greedy, Top-K, and Top-P). In the next iteration, x_t will append to $\mathbf{X}$ as input and continue the above processes. Generally, we can formulate above processes as the following equation:

$$x_t = \mathcal{S}(P_M(y|\mathbf{X}))$$

Evidently, the next token can only be determined after all preceding tokens have been decoded, and the decoding of each token requires loading all model parameters onto the computational unit. This characteristic of sequential decoding necessitates frequent and substantial communication between the computational unit and memory, which is precisely the bottleneck of decoding speed for autoregressive models.

Thus, we propose a novel approach that enables the model to generate draft sequences with random padding tokens. This strategy is rooted in the robustness of the model, allowing it to predict potential future draft sequences while performing standard decoding operations.

Figure 3a illustrates the architecture of the draft branches. We extend the autoregressive decoding process into a multi-branch draft process that could decode and generate the draft sequences in parallel. Specifically, for each branch i, we first pad several random tokens τ_i to the prompt as branch initialization. As the model inference progresses, all continuous sub-sequences in each branch will generate a draft token. That is:

$$x_t, \mathcal{D}_{i,j} = \mathcal{S}\left(P_M(y|[\mathbf{X}; \tau_{i,:j-1}])\right)$$

where $[.;.]$ means tensor concatenate. $\tau_{i,:j-1}$ means the preceding pad tokens before j-th token in draft branch i. Leveraging the parallel character of transformers, we can integrate all draft process of drafting branches and accomplish them in a single forward step, which can formulate this process as follows:

$$x_t, \mathbb{D} = \mathcal{S}\left(P_M(\mathbf{Y}|[\mathbf{X}; \mathcal{T}])\right)$$

where, $\mathbb{D} = \{\mathcal{D}_i \mid 0 < i \leq N\}$, and $\mathcal{T} = \{\tau_i \mid 0 < i \leq N\}$, and N denotes the number of draft branches.

It is crucial to recognize that tokens across different drafting branches are independent. Consequently, it is necessary to strategically design their attention mask matrix to ensure the following restriction during a single forward pass: within a branch, each token is only aware of its preceding tokens, and across different branches, there is no inter-token awareness. So, as shown in Figure 3b, when we initialize the drafting branches with random tokens, each branch is set only aware of its preceding tokens by the lower triangular attention masks, which is the same as ordinary LLM causal attention masks. Tokens from distinct branches are set with separate attention mask matrices to ensure they do not interfere with each other, represented as several lower triangular matrices distributed along the diagonal of the attention mask matrix.

As model forwarding, the consecutive sub-sequences of each draft branch will generate corresponding draft tokens D based on the current context, and we can use these draft tokens to fill and update draft branches and extract contextual draft sequences.

Branch updating is similar to the standard decoding process. We append the last draft tokens $D_{i,|\tau_i|}$ of each branch to its preceding tokens and then perform iterative forward propagation. That is,

$$\tau_i = [\tau_i; D_{i,|\tau_i|}].$$

Notably, we only update the draft branch with the last draft token of each branch rather all draft tokens. The intermediate draft tokens $D_{i,:|\tau_i|-1}$ are not used for branch

updating. This is because frequent updates will reduce the coherence of each draft branch and will compromise the semantic coherence of contextual drafts.

It is evident that when the length of a branch exceeds a certain limit, the draft result can deviate significantly from the actual output. Consequently, we set the maximum length of each branch to L. Once the length of each branch reaches L, we discard the foremost token of each draft branch. We will discuss how to strike a balance between the quality of the draft and the efficiency by appropriately setting L and N in the Experiments Section.

Besides branch updating, we can extract draft sequences by the draft tokens. As shown in Figure 3a, every subsequence of the draft branches produces draft tokens in the current context. Thus, we compile all these sub-sequences along with their respective successor draft tokens as draft sequences and the first token of the sub-sequence as the key to update our context cache, which can be formulated as:

$$d = [X;\ \tau_{i,k-G:k};\ \mathcal{D}_{i,k}]\,,$$

where G is the global gram length, we set the global gram length to 4 based on the overall decoding efficiency discussed in the Experiments Section.

We propose an LRU (Least Recently Used) queue to maintain the contextual draft sequences for each prefix token, which are designed to maintain information that closely reflects the current contextual environment. Branch filling and updating also follow the attention restrictions as we mentioned before; thus we designed the attention mask as shown in Figure 3b.

Candidate drafts verification. While performing normal decoding and drafting, we can verify each draft sequences C_i in parallel:

$$x_t, V_i = \mathcal{S}\left(P_M(\mathbf{Y}|[\mathbf{X}; C_i])\right),$$

the accepted tokens of C_i are the common prefix tokens between $[x_t; V_i]$ and C_i. Similarly, by setting the appropriate attention masks for each candidate sequence that ensure the sequences do not aware with each other, we can verify them in parallel.

Finally, we can formulate the Self-Draft decoding process as follows:

$$x_t, \mathbb{D}, \mathbb{V} = \mathcal{S}\left(P_M(\mathbf{Y}|[\mathbf{X}; \mathcal{T}; \mathcal{C}])\right),$$

where $\mathcal{C}$ represents all candidate drafts. The decoded tokens for each model forward are the longest common prefix tokens for all $([x_t; V_i], C_i)$ pairs.

Algorithm 1: Self-Draft Algorithm

Input: Prompt $\mathbf{X} = [x_1, x_2, ..., x_{t-1}]$; Maximum branch length:$L$; Maximum number of branches:N; randomly initialized draft branches $\mathcal{T} = \{\tau_0, \tau_1, ...\tau_N\}$;
output: O;
while True **do**
  if $size(O) < L$ **then**
    $x, \mathbb{D} = \mathcal{S}\left(P_M(\mathbf{y}|[\mathbf{X}; \tau_0; \tau_1; ...; \tau_N])\right)$
    for $\mathcal{D}_i$ in $\mathbb{D}$ **do**
      $\tau_i = [\tau_i; D_{i,|\tau_i|}]$
    end for
    $o \leftarrow x$
  else
    $\mathcal{C} \leftarrow$ retrieve from context and corpus cache
    $x, \mathbb{D}, \mathbb{V} = \mathcal{S}\left(P_M(\mathbf{Y}|[\mathbf{X}; \mathcal{T}; \mathcal{C}])\right)$
    for $\mathcal{D}_i$ in $\mathbb{D}$ **do**
      for k = G **to** L **do**
        $d = [X; \tau_{i,k-G:k}; \mathcal{D}_{i,k}]$
        updating context cache using d
      end for
      $\tau_i = [\tau_{i,1:}; D_{i,|\tau_i|}]$
    end for
    $o \leftarrow$ longest common prefix sequences between $[x; V_i]$ and C_i
  end if
  O.append(o)
end while

Experiments

Settings

Models. We selected two different sizes of open source LLMs: Llama-7B, Llama-13B, CodeLlama-7B, and CodeLlama-13B (Touvron et al. 2023) to evaluate our decoding strategy in different models and scenarios.

Evaluation benchmarks. We evaluated the Self-Draft decoding strategy on different benchmarks. MT-Bench (Zheng et al. 2023) is a diverse set of multi-turn dialogue that covers 8 kinds of tasks and 10 questions for each task. GSM-8K (Cobbe et al. 2021) is a high-quality grade school math problems dataset. We randomly sampled 100 questions from the test set and constructed GSM-100. For code completion tasks, we apply the whole HumanEval (Chen et al. 2021) dataset and randomly sample 100 questions from the test set of MBPP (Austin et al. 2021) and constructed MBPP-100.

Metric. We applied two metrics to evaluate the decoding strategies: throughput (TP) and decoding efficiency (DE). The former is the average number of tokens generated per second, and the latter is the average number of tokens per forward step.

All of our experiments were conducted on an NVIDIA A100 GPU, equipped with 40GB of RAM, and capitalized on mixed precision (FP16) to enhance the computational efficiency. Throughout, the inference was carried out using a batch size of one.

Results

Main results. Table 2 shows the main results of Self-Draft compared with autoregressive decoding, speculative decoding (Chen et al. 2023) (denoted as SpeDe) and LADE (Fu et al. 2024). For speculative decoding, we employed the assisted decoding strategy available in the Transformers library with Llama-68M (Miao et al. 2024) as the draft model and all parameters set to default. For the LADE method, we adopted the configuration described in their paper. For Self-Draft, we set both the draft length and the number of draft

Model	Benchmark	AR	SpeDe		Self-Draft			LADE		
			TP	Imp.	TP	Imp.	DE	TP	Imp.	DE
llama-7b	GSM-100	49.0	58.7	20%	88.2	80%	2.31	82.8	69%	2.23
llama-13b	GSM-100	33.5	47.2	41%	57.2	71%	2.28	53.8	60%	2.05
llama-7b	MT-bench	47.1	57.5	22%	73.6	56%	2.01	70.2	49%	1.97
llama-13b	MT-bench	33.6	48.2	43%	48.7	45%	2.00	45.4	35%	1.82
code-llama-7b	MBPP-100	42.3	80.3	90%	107.3	154%	2.98	98.8	134%	3.09
code-llama-13b	MBPP-100	32.2	65.4	103%	71.5	122%	3.03	67.6	110%	2.95
code-llama-7b	HumanEval	47.4	80.7	70%	111.2	135%	3.22	95.1	101%	2.90
code-llama-13b	HumanEval	32.4	62.3	92%	73.3	126%	3.22	67.4	108%	2.86

Table 2: Self-Draft and LADE performance across models and benchmarks

branches to 6 across all models, and we will discuss the selection of the length and number of draft branches in the next section.

The experimental results show that our method outperforms the basic speculative decoding and the LADE method across various models and datasets. On the MT-Bench dataset, our approach achieved more than 56% times and 45% times speed-up for Llama-7b and 13b models, respectively. In the mathematical problem-solving task on the GSM-100 dataset, our approach achieved 80% and 71% decoding throughput improvements for Llama-7B and 13B models, respectively. In the code completion tasks across two datasets and models, our method achieved over 2 times improvements in throughput across the board. Regarding decoding efficiency, our method has realized an improvement that exceeds 2 tokens per forward step across all evaluation datasets and models in general, and reaches up to 3.22 tokens per forward step on HumanEval, thus we established a global gram length of 4.

We find that our method performs better on smaller models in terms of throughput. That is because the overhead of drafting and verification for smaller models is less than the larger model, thus, the improvement of throughput brought by draft sequences will be more pronounced. Furthermore, the improvement in code generation tasks is greater than that in open-ended generation tasks. This is due to the lower content diversity in code generation tasks. In such scenarios, our method can effectively capture common expressions as draft sequences.

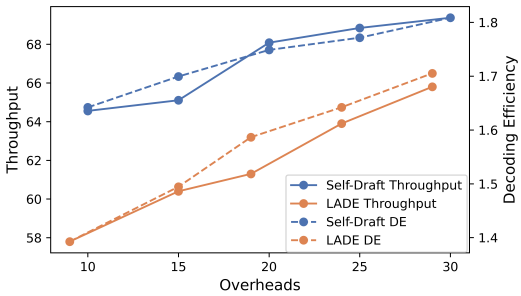


Figure 4: Draft efficiency for Llama-7b on MT-bench.

Draft efficiency. As we mentioned before, our approach will update the draft tokens for each prefix sequence of the draft branch during each model forward, allowing efficient production of numerous draft sequences with less additional computational overhead. We use the number of additional draft tokens to represent the extra computational overhead incurred during the draft phase. We removed the corpus cache module from our method and also included the randomly initialized tokens for fair comparison. Figure 4 shows the throughput and decoding efficiency comparison between our method and LADE with different additional drafting overheads. We can see that the results of our method are better than LADE overall, especially when the extra computational overhead is small. In fact, owing to the parallel computing capability of GPU, these additional computational overheads do not lead to a significant increase in GPU computation time (from 0.023s to 0.024s for each step in the Figure 4 conditions).

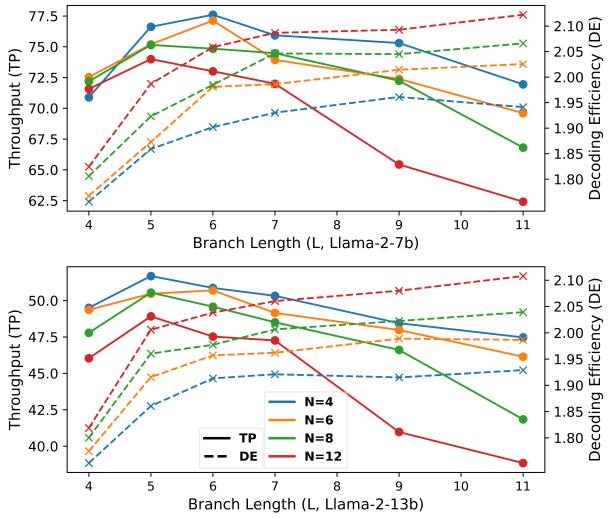


Figure 5: The effect of the length of draft branches.

Length and number of the draft branch for Self-Draft performance. By adjusting the length of draft branches, we get the curves of throughput and decoding efficiency

as shown in Figure 5. We can see that the decoding efficiency of Self-Draft increases as the branches become longer, and the throughput decreases after the length of the branches exceeds 6. This is because as the draft branch length extends, the quantity of drafts escalates. Initially, an increase in draft sequences correlates with a positive enhancement in throughput. However, an excessive number of draft sequences incurs a significant overhead for the verification process, ultimately resulting in a reduction of the final throughput.

Similarly, we analyze how the number of draft branches affects Self-Draft's performance. As shown in Figure 6. Similarly, it is observed that the decoding efficiency of the model improves incrementally with an increase in the number of branches. For throughput, when the branch length is small, specifically between 5 and 7, the throughput initially increases and then decreases. When the branch length exceeds 7, the throughput exhibits a gradual decline. This pattern is similar to the one previously observed: increasing the number of short branches enable quick replenishment of drafts, but the advantage diminishes with longer branches due to increased draft overhead. After reviewing the two scenarios discussed above, we set both the length and the number of draft branches to 6.

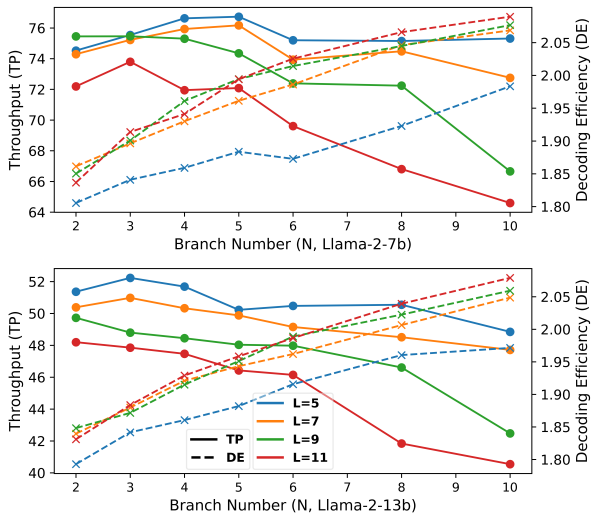


Figure 6: The effect of the number of draft branches.

In addition, we found that smaller models showed greater adaptability to the draft branch setting. As shown in Figure 7, we tested the throughput in the extreme condition, where both the length and number of draft branches range from 2 to 20, and the Llama-7b model shows speedup in a larger interval compared to the Llama-13b model.

Ablation study. To assess the impact of the *corpus cache* and *context cache* for the decoding throughput, we conduct several ablation experiments on the different sub-tasks of the MT-Bench dataset. Specifically, we removed the drafts from the corpus cache (denoted as w/o-Corpus) and the context cache (denoted as w/o-Context) to evaluate the functions of

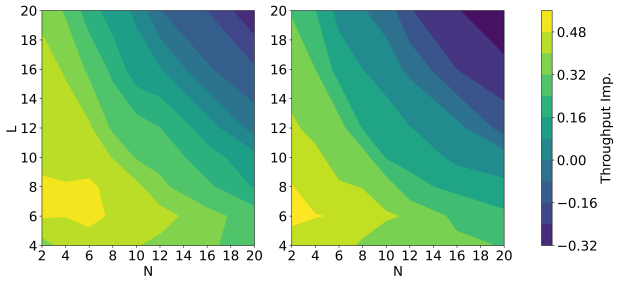


Figure 7: Decoding throughput in extreme conditions for Llama-7b(left) and Llama-13b(right).

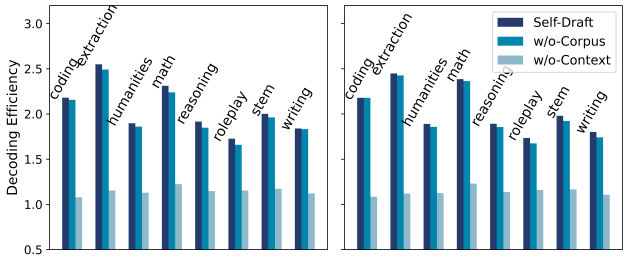


Figure 8: Ablation study on different sub-tasks of MT-bench of Llama-7B(left) and Llama-13B(right).

the corpus cache and draft branches, respectively. The results are shown in Figure 8.

We can see that removing drafts from the context brings much more decline than the corpus cache drafts. In specialized domains such as code completion, mathematical problem solving, and information extraction, the removal of drafts from the context cache leads to a more significant decline compared to more open-ending generation tasks like writing, role-playing, and humanities.

Conclusion

In this study, we proposed a novel decoding strategy that leveraged additional draft branches, achieving approximately a twofold increase in end-to-end throughput and decoding efficiency without any auxiliary models or additional training. However, some issues remain for future work. First, while string matching efficiently retrieves candidate drafts, it fails to capture semantic information accurately. Optimizing the use of semantic information from both decoded and draft sentences to achieve better matches will be a key focus. Second, although our approach improves decoding throughput, the use of grid search to determine optimal branch lengths and counts is inefficient. Future efforts will aim to better align branch dimensions with GPU hardware features.

Acknowledgments

This work was supported in part by the grants from National Natural Science Foundation of China (No.62222213, 62072423).

References

Aminabadi, R. Y.; Rajbhandari, S.; Awan, A. A.; Li, C.; Li, D.; Zheng, E.; Ruwase, O.; Smith, S.; Zhang, M.; Rasley, J.; and He, Y. 2022. DeepSpeed- Inference: Enabling Efficient Inference of Transformer Models at Unprecedented Scale. In *SC22: International Conference for High Performance Computing, Networking, Storage and Analysis*, 1–15.

Austin, J.; Odena, A.; Nye, M.; Bosma, M.; Michalewski, H.; Dohan, D.; Jiang, E.; Cai, C.; Terry, M.; Le, Q.; et al. 2021. Program Synthesis with Large Language Models. *arXiv preprint arXiv:2108.07732*.

Bai, H.; Hou, L.; Shang, L.; Jiang, X.; King, I.; and Lyu, M. R. 2022. Towards Efficient Post-training Quantization of Pre-trained Language Models. In Koyejo, S.; Mohamed, S.; Agarwal, A.; Belgrave, D.; Cho, K.; and Oh, A., eds., *Advances in Neural Information Processing Systems*, volume 35, 1405–1418. Curran Associates, Inc.

Cai, T.; Li, Y.; Geng, Z.; Peng, H.; Lee, J. D.; Chen, D.; and Dao, T. 2024. Medusa: Simple LLM Inference Acceleration Framework with Multiple Decoding Heads. arXiv:2401.10774.

Chen, C.; Borgeaud, S.; Irving, G.; Lespiau, J.-B.; Sifre, L.; and Jumper, J. 2023. Accelerating Large Language Model Decoding with Speculative Sampling. arXiv:2302.01318.

Chen, M.; Tworek, J.; Jun, H.; Yuan, Q.; de Oliveira Pinto, H. P.; Kaplan, J.; Edwards, H.; Burda, Y.; Joseph, N.; Brockman, G.; Ray, A.; Puri, R.; Krueger, G.; Petrov, M.; Khlaaf, H.; Sastry, G.; Mishkin, P.; Chan, B.; Gray, S.; Ryder, N.; Pavlov, M.; Power, A.; Kaiser, L.; Bavarian, M.; Winter, C.; Tillet, P.; Such, F. P.; Cummings, D.; Plappert, M.; Chantzis, F.; Barnes, E.; Herbert-Voss, A.; Guss, W. H.; Nichol, A.; Paino, A.; Tezak, N.; Tang, J.; Babuschkin, I.; Balaji, S.; Jain, S.; Saunders, W.; Hesse, C.; Carr, A. N.; Leike, J.; Achiam, J.; Misra, V.; Morikawa, E.; Radford, A.; Knight, M.; Brundage, M.; Murati, M.; Mayer, K.; Welinder, P.; McGrew, B.; Amodei, D.; McCandlish, S.; Sutskever, I.; and Zaremba, W. 2021. Evaluating Large Language Models Trained on Code. arXiv:2107.03374.

Chen, W.; Zhao, L.; Zheng, Z.; Xu, T.; Wang, Y.; and Chen, E. 2024a. Double-Checker: Large Language Model as a Checker for Few-shot Named Entity Recognition. In Al-Onaizan, Y.; Bansal, M.; and Chen, Y.-N., eds., *Findings of the Association for Computational Linguistics: EMNLP 2024*, 3172–3181. Miami, Florida, USA: Association for Computational Linguistics.

Chen, Y.; Pan, X.; Li, Y.; Ding, B.; and Zhou, J. 2024b. EE-LLM: Large-Scale Training and Inference of Early-Exit Large Language Models with 3D Parallelism. arXiv:2312.04916.

Cobbe, K.; Kosaraju, V.; Bavarian, M.; Chen, M.; Jun, H.; Kaiser, L.; Plappert, M.; Tworek, J.; Hilton, J.; Nakano, R.; Hesse, C.; and Schulman, J. 2021. Training Verifiers to Solve Math Word Problems. *arXiv preprint arXiv:2110.14168*.

Fedus, W.; Dean, J.; and Zoph, B. 2022. A Review of Sparse Expert Models in Deep Learning. arXiv:2209.01667.

Fu, Y.; Bailis, P.; Stoica, I.; and Zhang, H. 2024. Break the Sequential Dependency of LLM Inference Using Lookahead Decoding. arXiv:2402.02057.

Ghazvininejad, M.; Levy, O.; Liu, Y.; and Zettlemoyer, L. 2019. Mask-Predict: Parallel Decoding of Conditional Masked Language Models. In Inui, K.; Jiang, J.; Ng, V.; and Wan, X., eds., *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, 6112–6121. Hong Kong, China: Association for Computational Linguistics.

Hagendorff, T. 2023. Machine psychology: Investigating emergent capabilities and behavior in large language models using psychological methods. *arXiv preprint arXiv:2303.13988*, 1.

He, Z.; Zhong, Z.; Cai, T.; Lee, J. D.; and He, D. 2023. REST: Retrieval-Based Speculative Decoding. arXiv:2311.08252.

Jipeng, Q.; Feng, Z.; Yun, L.; Yunhao, Y.; Yi, Z.; and WU, X. 2023. Unsupervised statistical text simplification using pre-trained language modeling for initialization. *Frontiers of Computer Science*, 17(1): 171303.

Kwon, W.; Li, Z.; Zhuang, S.; Sheng, Y.; Zheng, L.; Yu, C. H.; Gonzalez, J.; Zhang, H.; and Stoica, I. 2023. Efficient Memory Management for Large Language Model Serving with PagedAttention. In *Proceedings of the 29th Symposium on Operating Systems Principles*, SOSP '23, 611–626. New York, NY, USA: Association for Computing Machinery. ISBN 9798400702297.

Lei, W.; Chen, M.; Xueyang, F.; Zeyu, Z.; Hao, Y.; Jingsen, Z.; Zhiyuan, C.; Jiakai, T.; Xu, C.; Yankai, L.; Wayne Xin, Z.; Zhewei, W.; and WEN, J. 2024. A survey on large language model based autonomous agents. *Frontiers of Computer Science*, 18(6): 186345.

Leviathan, Y.; Kalman, M.; and Matias, Y. 2023a. Fast Inference from Transformers via Speculative Decoding. In Krause, A.; Brunskill, E.; Cho, K.; Engelhardt, B.; Sabato, S.; and Scarlett, J., eds., *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, 19274–19286. PMLR.

Leviathan, Y.; Kalman, M.; and Matias, Y. 2023b. Fast Inference from Transformers via Speculative Decoding. arXiv:2211.17192.

Liang, K. J.; Hao, W.; Shen, D.; Zhou, Y.; Chen, W.; Chen, C.; and Carin, L. 2021. MixKD: Towards Efficient Distillation of Large-scale Language Models. arXiv:2011.00593.

Ma, X.; Fang, G.; and Wang, X. 2023. LLM-Pruner: On the Structural Pruning of Large Language Models. In Oh, A.; Neumann, T.; Globerson, A.; Saenko, K.; Hardt, M.; and Levine, S., eds., *Advances in Neural Information Processing Systems*, volume 36, 21702–21720. Curran Associates, Inc.

Miao, X.; Oliaro, G.; Zhang, Z.; Cheng, X.; Wang, Z.; Zhang, Z.; Wong, R. Y. Y.; Zhu, A.; Yang, L.; Shi, X.; Shi, C.; Chen, Z.; Arfeen, D.; Abhyankar, R.; and Jia, Z. 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, 932–949. ACM.

Monea, G.; Joulin, A.; and Grave, E. 2023. PaSS: Parallel Speculative Sampling. arXiv:2311.13581.

Santilli, A.; Severino, S.; Postolache, E.; Maiorca, V.; Mancusi, M.; Marin, R.; and Rodola, E. 2023. Accelerating Transformer Inference for Translation via Parallel Decoding. In Rogers, A.; Boyd-Graber, J.; and Okazaki, N., eds., *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, 12336–12355. Toronto, Canada: Association for Computational Linguistics.

Sato, S.; Yoshinaga, N.; Toyoda, M.; and Kitsuregawa, M. 2021. Speculative Sampling in Variational Autoencoders for Dialogue Response Generation. In Moens, M.-F.; Huang, X.; Specia, L.; and Yih, S. W.-t., eds., *Findings of the Association for Computational Linguistics: EMNLP 2021*, 4739–4745. Punta Cana, Dominican Republic: Association for Computational Linguistics.

Song, Y.; Meng, C.; Liao, R.; and Ermon, S. 2021. Accelerating Feedforward Computation via Parallel Nonlinear Equation Solving. In Meila, M.; and Zhang, T., eds., *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, 9791–9800. PMLR.

Stern, M.; Shazeer, N.; and Uszkoreit, J. 2018. Blockwise Parallel Decoding for Deep Autoregressive Models. arXiv:1811.03115.

Touvron, H.; Martin, L.; Stone, K.; Albert, P.; Almahairi, A.; Babaei, Y.; Bashlykov, N.; Batra, S.; Bhargava, P.; Bhosale, S.; Bikel, D.; Blecher, L.; Ferrer, C. C.; Chen, M.; Cucurull, G.; Esiobu, D.; Fernandes, J.; Fu, J.; Fu, W.; Fuller, B.; Gao, C.; Goswami, V.; Goyal, N.; Hartshorn, A.; Hosseini, S.; Hou, R.; Inan, H.; Kardas, M.; Kerkez, V.; Khabsa, M.; Kloumann, I.; Korenev, A.; Koura, P. S.; Lachaux, M.-A.; Lavril, T.; Lee, J.; Liskovich, D.; Lu, Y.; Mao, Y.; Martinet, X.; Mihaylov, T.; Mishra, P.; Molybog, I.; Nie, Y.; Poulton, A.; Reizenstein, J.; Rungta, R.; Saladi, K.; Schelten, A.; Silva, R.; Smith, E. M.; Subramanian, R.; Tan, X. E.; Tang, B.; Taylor, R.; Williams, A.; Kuan, J. X.; Xu, P.; Yan, Z.; Zarov, I.; Zhang, Y.; Fan, A.; Kambadur, M.; Narang, S.; Rodriguez, A.; Stojnic, R.; Edunov, S.; and Scialom, T. 2023. Llama 2: Open Foundation and Fine-Tuned Chat Models. arXiv:2307.09288.

Wang, Z.; Wohlwend, J.; and Lei, T. 2020. Structured Pruning of Large Language Models. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Association for Computational Linguistics.

Wu, C.; Wu, F.; and Huang, Y. 2021. One Teacher is Enough? Pre-trained Language Model Distillation from Multiple Teachers. arXiv:2106.01023.

Wu, L.; Zheng, Z.; Qiu, Z.; Wang, H.; Gu, H.; Shen, T.; Qin, C.; Zhu, C.; Zhu, H.; Liu, Q.; Xiong, H.; and Chen, E. 2024. A Survey on Large Language Models for Recommendation. arXiv:2305.19860.

Xiao, G.; Lin, J.; Seznec, M.; Wu, H.; Demouth, J.; and Han, S. 2023. SmoothQuant: Accurate and Efficient Post-Training Quantization for Large Language Models. In Krause, A.; Brunskill, E.; Cho, K.; Engelhardt, B.; Sabato, S.; and Scarlett, J., eds., *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, 38087–38099. PMLR.

Yu, G.-I.; Jeong, J. S.; Kim, G.-W.; Kim, S.; and Chun, B.-G. 2022. Orca: A Distributed Serving System for Transformer-Based Generative Models. In *16th USENIX Symposium on Operating Systems Design and Implementation (OSDI 22)*, 521–538. Carlsbad, CA: USENIX Association. ISBN 978-1-939133-28-1.

Yuan, Z.; Shang, Y.; Zhou, Y.; Dong, Z.; Zhou, Z.; Xue, C.; Wu, B.; Li, Z.; Gu, Q.; Lee, Y. J.; Yan, Y.; Chen, B.; Sun, G.; and Keutzer, K. 2024. LLM Inference Unveiled: Survey and Roofline Model Insights. arXiv:2402.16363.

Zhang, J.; Wang, J.; Li, H.; Shou, L.; Chen, K.; Chen, G.; and Mehrotra, S. 2023. Draft & Verify: Lossless Large Language Model Acceleration via Self-Speculative Decoding. arXiv:2309.08168.

Zhao, W.; Huang, Y.; Han, X.; Xiao, C.; Liu, Z.; and Sun, M. 2024a. Ouroboros: Speculative Decoding with Large Model Enhanced Drafting. arXiv:2402.13720.

Zhao, Y.; Xie, Z.; Zhuang, C.; and Gu, J. 2024b. Lookahead: An Inference Acceleration Framework for Large Language Model with Lossless Generation Accuracy. arXiv:2312.12728.

Zhao, Z.; Lin, F.; Zhu, X.; Zheng, Z.; Xu, T.; Shen, S.; Li, X.; Yin, Z.; and Chen, E. 2024c. DynLLM: When Large Language Models Meet Dynamic Graph Recommendation. arXiv:2405.07580.

Zheng, L.; Chiang, W.-L.; Sheng, Y.; Zhuang, S.; Wu, Z.; Zhuang, Y.; Lin, Z.; Li, Z.; Li, D.; Xing, E. P.; Zhang, H.; Gonzalez, J. E.; and Stoica, I. 2023. Judging LLM-as-a-Judge with MT-Bench and Chatbot Arena. arXiv:2306.05685.

Zheng, Z.; Chao, W.; Qiu, Z.; Zhu, H.; and Xiong, H. 2024. Harnessing Large Language Models for Text-Rich Sequential Recommendation. In *Proceedings of the ACM Web Conference 2024*, WWW '24, 3207–3216. New York, NY, USA: Association for Computing Machinery. ISBN 9798400701719.