SEED: Accelerating Reasoning Tree Construction via Scheduled Speculative Decoding

Zhenglin Wang*, **Jialong Wu***, **Yilong Lai**, **Congzhi Zhang**, **Deyu Zhou**†
School of Computer Science and Engineering, Key Laboratory of Computer Network
and Information Integration, Ministry of Education, Southeast University, China
{zhenglin, jialongwu, yilong.lai, zhangcongzhi, d.zhou}@seu.edu.cn

Abstract

Large Language Models (LLMs) demonstrate remarkable emergent abilities across various tasks, yet fall short of complex reasoning and planning tasks. The tree-search-based reasoning methods address this by encouraging the exploration of intermediate steps, surpassing the capabilities of chain-of-thought prompting. However, significant inference latency is introduced due to the systematic exploration and evaluation of multiple thought paths. This paper introduces SEED, a novel and efficient inference framework to improve both runtime speed and GPU memory management concurrently. Based on a scheduled speculative execution, SEED efficiently handles multiple iterations for thought generation and state evaluation, leveraging a rounds-scheduled strategy to manage draft model dispatching. Extensive experimental evaluations on three reasoning datasets demonstrate the superior speedup performance of SEED[1].

1 Introduction

Despite Large Language Models (LLMs) have shown remarkable emergent abilities across a variety of tasks (Ouyang et al., 2022; OpenAI, 2022; Touvron et al., 2023a,b; Achiam et al., 2023), their performance on the complex reasoning and planning tasks remains suboptimal (Zhang et al., 2024b). Traditional or simple prompting techniques (Wei et al., 2022; Kojima et al., 2022), which have been widely leveraged, are insufficient for the tasks that require exploratory actions or strategic lookahead (Liao et al., 2024).

Tree-Search-Based (TSB) reasoning methods effectively harness the planning and reasoning capabilities of LLMs by decomposing the problems and subsequently orchestrating a structured plan (Hui et al., 2024). These methods not only leverage the inherent strengths of LLMs in processing vast datasets but also address their limitations in dynamic problem-solving scenarios (Hao et al., 2023; Guan et al., 2023). For example, Yao et al. (2024) introduced Tree-of-Thoughts (ToT) prompting, which generalizes beyond Chain-of-Thought (CoT) prompting by fostering the exploration of intermediate thoughts that serve as crucial steps in general problem-solving with LLMs. Following this way, subsequent works, such as Reasoning via Planning (RAP) (Hao et al., 2023) and Refection on search Trees (RoT) are proposed (Hui et al., 2024). These approaches leverage the capabilities of LLMs to generate and evaluate the intermediate thoughts and then integrate them with search algorithms to improve the problem-solving efficiency.

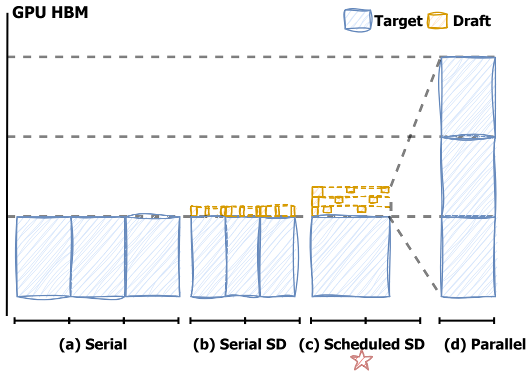

Figure 1: Illustration of four LLM execution strategies for generating 3 sequences in Reasoning Tree construction: (a) *Serial*, where executions are operated one after another, simplifying resource management but increasing overall execution time; (b) *Seiral SD*, where speculative decoding is used for each execution; (c) *Scheduled SD*, which involves several parallel draft models and one target model; (d) *Parallel*, where multiple executions run concurrently, reducing completion time but increasing GPU HBM. ▨ refers to a large target model, □ signifies a smaller draft model, ├─┤ represents a `unit length` of execution time.

* Equal Contribution.

† Corresponding Author.

[1] https://github.com/Linking-ai/SEED

Proceedings of the 31st International Conference on Computational Linguistics, pages 4920–4937
January 19–24, 2025. ©2025 Association for Computational Linguistics

However, such methods introduce a serious issue of inference latency due to the requirement for systematic exploration of thoughts with lookahead and backtracking. TSB reasoning methods primarily consist of two key parts, tree construction and the search algorithm. Recent studies have enhanced the efficiency of the search algorithms by incorporating diversity rewards or pruning techniques (Yan et al., 2024; Hui et al., 2024). To the best of our knowledge, no prior work explored the acceleration of tree construction, which is the focus of this paper.

Traditional *Sequential* execution of LLMs necessitates repeated executions, leading to long execution time, as shown in Figure 1 (a). For instance, when applying ToT prompting to execute a single sample in the GSM8K dataset, the average total runtime is approximately 100 seconds using *sequential* processing with a 7B model on consumer GPUs. If the execution of LLMs shifts from *sequential* to *parallel* processing, it could pose challenges for end-users or researchers only with consumer GPUs, as illustrated in Figure 1 (d). Such condition typically exacerbates the issues related to hardware limitations, necessitating strategies for efficient resource management and optimization. Speculative decoding is now widely used to accelerate inference (Xia et al., 2024), which involves employing a small draft model with a larger target model, as depicted in Figure 1 (b). Intuitively, these draft models achieve rapid inference speeds owing to their small size. If they are executed in parallel, concerns about the GPU memory constraints become negligible, allowing for the speed performance comparable to the scenarios illustrated in Figure 1 (d). Moreover, speculative decoding employs a *draft-then-verify* two-stage paradigm, and the target model is not fully utilized when the acceptance rate of drafted tokens is relatively high. By increasing the number of draft models, the potential of a single target model can be effectively harnessed, ensuring its capacity is optimally utilized.

Therefore, we propose a novel and efficient inference framework, SEED, to address both runtime speed and GPU memory resource management concurrently in reasoning tree construction. SEED effectively handles two scenarios: (1) executing multiple iterations with the same prompt; (2) evaluating multiple iterations with different prompts. We utilize scheduled speculative decoding to manage the scheduling of parallel draft models. As depicted in Figure 1 (c), given that there is only one shared target model, which can not simultaneously verify multiple draft models, we address this limitation by drawing inspiration from process scheduling in operating system management (Zhao and Stankovic, 1989; Siahaan, 2016). To this end, the Rounds-Scheduled strategy which uses a First-Come-First-Serve (FCFS) queue, is employed to control and maintain the overall execution flow.

SEED achieves excellent speed performance on three reasoning and planning datasets: GSM8K, Creative Writing and Blocksworld. It also provides a viable path for conducting *batched inference* in training-free speculative decoding while preserving the original distribution, ensuring a **lossless** outcome.

Our contribution can be summarized as follows:

- An efficient inference framework, SEED, is proposed to accelerate the both Thought Generator and State Evaluator in reasoning tree construction.
- Speculative Scheduled Execution that integrates parallel drafting with speculative decoding is proposed, employing an effective Rounds-Scheduled strategy to manage parallel drafting devoid of verification conflicts.
- Empirically, extensive experiments and analysis studies are conducted to demonstrate the effectiveness of SEED. SEED achieves $1.1-1.5\times$ speedups, generating up to **20 additional tokens per second** across three reasoning datasets.

2 Related Work

2.1 Tree-Search-Based Reasoning

Recently, TSB reasoning methods have been widely leveraged to augment the reasoning capabilities of LLMs such as RAP (Hao et al., 2023), ToT (Yao et al., 2024), RoT (Hui et al., 2024). These methods craft a reasoning tree allowing consider multiple reasoning paths and self-evaluate the choices to determine the next course of action. At each reasoning step, the popular tree search algorithms such as Breadth-First Search (BFS) (Bundy and Wallen, 1984) and Monte-Carlo Tree Search (MCTS) (Kocsis and Szepesvári, 2006) are integrated to explore the tree in search of an optimal state. Also, the construction or search of the tree requires more iterations than single sampling methods (*e.g.*, Input-output prompting and CoT (Wei et al., 2022)), leading to higher inference latency.

To address this, some studies introduce diversity rewards (Yan et al., 2024) or pruning techniques (Hui et al., 2024) to mitigate inefficient searches during iterations, improving search efficiency. However, these methods still overlook the inference latency caused by the iterative process of tree construction. Instead, we focus on tree construction, leveraging speculative scheduled decoding to accelerate the process and reduce inference latency.

2.2 Parallel Decoding

The inference latency of LLMs has emerged as a substantial obstacle, restricting their remarkable reasoning capabilities in downstream tasks (Xia et al., 2024). One major factor contributing to the high inference latency is the sequential decoding strategy for token generation adopted by almost all LLMs (Lu et al., 2024b). There are numerous studies have explored this challenge through parallel decoding strategies, such as Speculative Decoding (SD) (Zhou et al., 2023; Cai et al., 2024), Early Exiting (EE) (Del Corro et al., 2023; Elhoushi et al., 2024), and Non-AutoRegressive (NAR) (Ghazvininejad et al., 2019; Lu et al., 2024a). In this paper, we focus on the study of Speculative Decoding. Within SD, one line of work falls into the training-free category (Sun et al., 2024b; Liu et al., 2023). This plug-and-play approach seamlessly integrates with other modular inference methods (*e.g.*, CoT, TSB), significantly enabling direct inference acceleration and reducing inference latency on open-source models. As far as we know, we are the **first** to explore a scheduled SD execution to integrate with the TSB framework, without modifying LLM architecture or requiring additional training and maintaining lossless output.

3 Preliminaries

3.1 Speculative Decoding

The core technique of speculative decoding involves using a small draft model to generate tokens sequentially, with a larger target model validating these tokens (Leviathan et al., 2023). Specifically, let c be the input tokens, M_d and M_t be the draft and the target model respectively, and k be the number of draft tokens generated per step. Speculative decoding is a *Draft-then-Verify* two-stage decoding paradigm. [2] In the draft stage, M_d samples a draft sequence of tokens autoregressively, denoted as $\hat{x}_1, \ldots, \hat{x}_k$, where $\hat{x}_i \sim p_d(x|\hat{x}_1, \ldots, \hat{x}_{i-1}, c)$ for $i = 1, \ldots, k$. In the verification stage, the draft sequence of tokens along with c, are passed to M_t to obtain their output distribution $p_t(x|\hat{x}_1, \ldots, \hat{x}_{i-1}, c)$ in parallel, and then verified from $\hat{x}_1$ to $\hat{x}_k$. The draft token $\hat{x}_i$ is accepted with the probability $\min(1, \frac{p_t(x|\hat{x}_1,\ldots,\hat{x}_{i-1},c)}{p_d(x|\hat{x}_1,\ldots,\hat{x}_{i-1},c)})$. Once a token is rejected, the verifying terminates and a resampling phase follows to return a new token by M_t. This new token is then used as the end-generated point following the accepted tokens. As is proven in Leviathan et al. (2023), this method is equivalent to sampling directly from the target LLM. SEED adopts this method, ensuring that the distribution of the generated text **remains unchanged** for both the greedy and non-greedy settings.

3.2 Tree Attention

Current speculative decoding studies have demonstrated that when the draft model samples multiple candidates per position in the draft sequence, the expected acceptance length per step can be enhanced during the verification stage (Chen et al., 2023a). Additionally, the tree attention technique enables multiple candidate draft sequences to share the caches of generated tokens, further improving the efficiency of the verification stage (Cai et al., 2024). By utilizing tree attention, the verification acceptance of speculative decoding is increased. We illustrate the detailed tree attention mask strategy in Appendix B. Our proposed SEED can leverage this approach to achieve further speedup.

3.3 TSB Task Formulation

Given an initial input question $\mathcal{I}$, a reasoning tree is constructed with the relatively common search algorithm BFS following Yao et al. (2024), as shown in Figure 2. In the constructed reasoning tree, each node represents a distinct state S_i, which includes a partial solution with the input c and the progressively elaborated thoughts proposal $z_1, \cdots, z_n$. During the expansion of each node, the Thought Generator $\mathrm{G}(\cdot)$ produces multiple reasoning paths to decompose the intermediate process from the current state. Once these thoughts are generated, the State Evaluator $\mathrm{E}(\cdot)$ assesses the contribution of each path toward solving the problem, serving as a heuristic for guiding the search algorithm. This evaluation aids in determining which states to continue exploring and in establishing the order of ex-

[2] In the following paper, we define "Verification" as the "*Verify*" mentioned here, which includes both the verify and resampling phases.

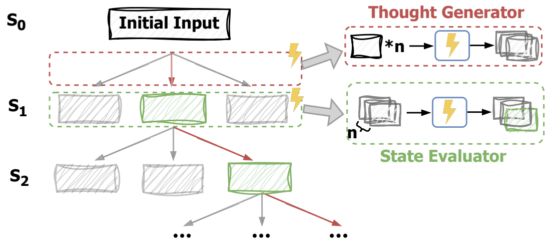

Figure 2: Two main components in reasoning tree construction, which are Thought Generator and State Evaluator, respectively.

ploration. Taking the root node S_0 as an example in Figure 2, it first generates n reasoning paths based on the same input c, which is the initial prompt $\mathcal{I}$ and subsequently selects the middle path by the State Evaluator for these n paths.

4 Method

Our proposed SEED is an efficient inference framework designed to accelerate the construction of a reasoning tree. Different generation executions in the Thought Generator or the State Evaluator are conducted in distinct branches, ensuring that they do not interfere with each other. Consequently, the Speculative Scheduled Execution is implemented in both the Thought Generator and the State Evaluator, enabling parallel processing to accelerate the overall reasoning tree construction, as detailed in Algorithm 2.

We first introduce two phases in the Speculative Scheduled Execution in §4.1. Subsequently, we depict the Rounds-Scheduled Strategy designed to effectively manage parallel drafting without conflicts in §4.2. The technical principle of SEED is inspired by the operation system schedule. The detailed analogy between the operation system scheduling with SEED is presented in §4.3. Finally, the combined algorithm is elaborated in §4.4.

4.1 Speculative Scheduled Execution

We further detail the speculative scheduled execution algorithm within SEED. To enhance clarity, we delve the algorithm into two phases: the *parallel drafting phase* and the *sequential verification phase*.

Parallel Drafting Phase The model size significantly impacts memory usage and inference time. In light of this, given the small size and rapid inference speed of the draft models, we can directly initialize multiple draft models corresponding to the number of thoughts, enabling parallel processes. To be specific, if the number of thoughts N_t is set to n, the draft models $M_{d_1}, M_{d_2}, \cdots, M_{d_n}$ take $c_1, c_2, \cdots, c_n$ as input tokens respectively in the drafting phase. Note that, during the Thought Generation, the input instructions are the same, *i.e.*, $c_1 = c_2 = \cdots = c_n$; during the State Evaluation, they may differ, denoted as $c_1 \neq c_2 \neq \cdots \neq c_n$.

As illustrated in Figure 3 (a), three draft models initiate sampling simultaneously when the queue Q is initially empty. In the subsequent stage, the draft models enter the queue according to which completes the generation first. In Figure 3 (a), Draft Model 🟥 first completes the drafting process and is the first to enter the queue Q, followed by Draft Model 🟧 and Draft Model 🟨. Each draft model is generating its own tokens while the target model M_t is verifying the tokens of other draft models. In this way, we can leverage the potential of small draft models to complete their drafting processes simultaneously, while the larger target model only needs to verify them sequentially.

Sequential Verification Phase Only one single target model is employed for the sequential verification of multiple draft sequences in SEED. The target model first verifies the tokens generated by the draft model at the front of the queue. During the verification phase, two scenarios may occur: acceptance and rejection. If the tokens generated by the draft model are accepted by the target model, they are retained, as exemplified by Draft Model 🟥 in Fugure 3 (a). If rejected, one new token is resampled by the target model, as demonstrated by Draft Model 🟧 and Draft Model 🟨. Taking Draft Model 🟨 as an example, it drafts two tokens, "*many*" and "*duch*", which are rejected by the target model. Target Model 🟦 then resamples a new token "*much*". Furthermore, when accepted, the target model only requires the execution time □, when rejected, it incurs additional time for resampling ▨.

4.2 Rounds-Scheduled Strategy

With the integration of parallel drafting and sequential verification, it is crucial to optimize the scheduling to ensure the correctness of speculative execution while effectively utilizing the target model and reducing the overall execution latency.

Inspired by process scheduling in operating system management, which utilizes the First-Come-First-Serve (FCFS) scheduling policy for all re-

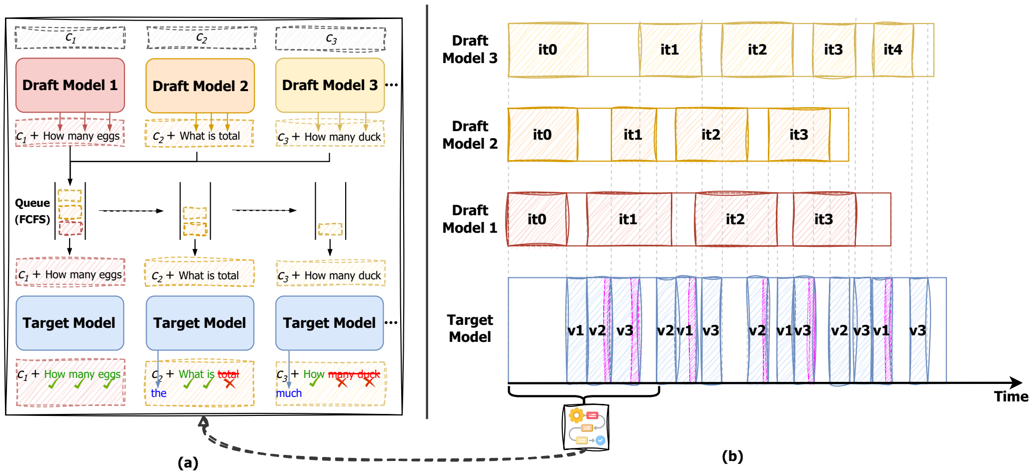

Figure 3: (a) The scenario where the target model manages the verification of target models at the beginning; (b) Overall scheduling diagram for one target model and three draft models. 🟥 , 🟧 , 🟨 represent Draft Model 1, Draft Model 2, Draft Model 3, respectively. ▯ , ▯ , ▯ denotes the execution times of drafting for each corresponding draft model. 🟦 refers to Target Model. ▯ represents the execution time of the verification phase, while ▨ specifies the resampling time in cases of rejection.

quests, ensuring fairness and preventing starvation (Zhao and Stankovic, 1989; Siahaan, 2016), we leverage a Rounds-Scheduled Strategy integrated with the FCFS scheduling policy to manage the verification process efficiently. When a draft model completes its drafting phase and is ready for verification, the draft sequences along with c are placed into a queue.

As depicted in Figure 3 (a), when the queue Q is not empty, a sequence of draft tokens is dequeued in the FCFS manner. Target Model 🟦 first verifies the tokens generated by Draft Model 🟥 , followed sequentially by tokens generated by Draft Model 🟧 and Draft Model 🟨 , adhering to FCFS. Upon completion of the verification of a draft sequence associated with a draft model, the draft model proceeds to the drafting process in the next iteration.

The overall scheduling diagram is shown in Figure 3 (b), each draft model displays a series of iterations to complete the overall drafting progress for the Thought Generator or the State Evaluator. The target model is consistently active across the overall scheduling timeline. This continuous activity ensures that the target model is utilized efficiently, addressing issues related to idle time when acceptance rates are relatively high. Once all drafting and verification processes are completed, the entire execution concludes, resulting in the generation of n sequences.

4.3 Technical Principle

Previous research has adapted the principle of the operating system (OS) scheduler for efficient process management (Kwon et al., 2023). As shown in Figure 4, each component in SEED can be mapped to a corresponding component in the operating system scheduler. We elaborate on each component individually as below:

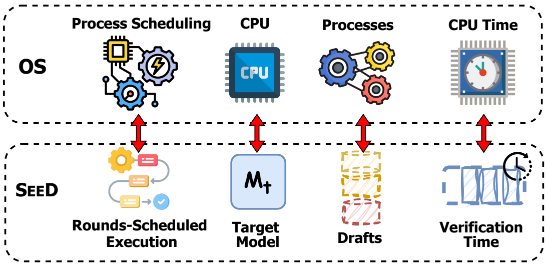

Figure 4: Analogy between the Operation System scheduler with our proposed SEED.

- The rounds-scheduled execution in SEED corresponds to the process scheduling in OS. Both use an FCFS queue to control and maintain the overall execution flow. A key distinction exists: in SEED, after the drafting tokens are processed by the verification phase, the draft model is returned to the queue, *i.e.*, "*rounds*". In contrast, in OS scheduling, a process that has been handled by the CPU is marked as completed.
- The verification of draft tokens $\hat{\mathcal{X}}$ mirrors an operating process in OS scheduling.
- The target model serves M_t analogously to the CPU.
- The total verification time of M_t resembles the CPU time in OS process scheduling.

Algorithm 1 Speculative Scheduled Execution with a Rounds-Scheduled Strategy

```
**Input:** Draft models $\{M_{d_1}, \cdots, M_{d_n}\}$, prefixes $\{c_1, \cdots, c_n\}$, target model $M_t$, max new length $l$, draft length $k$, auto-regressive drafting $p_{d_i}$ and length of current validated token $\mathcal{L}_i$ of the $i$-th draft model $M_{d_i}$, $i \in [1, n]$;
**Initialize:** Prefill $\{M_{d_1}, \cdots, M_{d_n}\}$ with prefixes; Create a verify queue $Q$ and a draft label map $\gamma$[i] of length $n$, with each element set to 1, $i \in [1, n]$; $\mathcal{L}_i \leftarrow 1$ , $i \in [1, n]$; Define $\hat{\mathcal{X}}_i[1:k]$ represents $\hat{x}_1, \ldots, \hat{x}_k$ the sequence of draft tokens generated from $p_{d_i}$, $i \in [1, n]$; Start $n$ draft processes $\mathcal{D}$(n) and 1 verification process $\mathcal{V}$ *Synchronously*;
**Processes** $\mathcal{D}$**(n):**                                   ▷ Prallel Drafting
while $\exists i \in [1, n] : \mathcal{L}_i < l$ do
    if $\gamma(i)$ then
        $\hat{\mathcal{X}}_i[1:k] \leftarrow p_{d_i}(M_{d_i}, c_i, \hat{\mathcal{X}}_i[1:\mathcal{L}_i], k)$
        $Q \leftarrow \hat{\mathcal{X}}_i[1:k]$            ▷ Add draft tokens to the queue
        $\gamma[i] \leftarrow 0$                           ▷ Draft Process D(i) wait
    end if
end while
**Process** $\mathcal{V}$**:**                              ▷ Sequential Verification
while $\exists i \in [1, n] : \mathcal{L}_i < l$ do
    if Q is not empty then
        $\hat{\mathcal{X}}_i[1:k] \leftarrow$ queue$(Q)$                    ▷ FCFS
        $t_1, \cdots, t_k \leftarrow \mathcal{E}(M_t, c_i, \hat{\mathcal{X}}_i[1:k])$
        for $j = 1$ to $k$ do
            if $t_j$ is acceptance then
                $\hat{\mathcal{X}}_i[\mathcal{L}_i + 1] \leftarrow \hat{x}_j$
                $\mathcal{L}_i \leftarrow \mathcal{L}_i + 1$
            else
                $\hat{\mathcal{X}}[\mathcal{L}_i + 1] \leftarrow \mathcal{R}(M_t, c_i, \hat{\mathcal{X}}_i[1:\mathcal{L}_i])$
                $\mathcal{L}_i \leftarrow \mathcal{L}_i + 1$
                Break
            end if
        end for
        $\gamma[i] \leftarrow 1$                    ▷ Draft Process D(i) continue
    end if
end while
Wait for all $\mathcal{D}$(n) and $\mathcal{V}$ to finish
return $[response_1, \ldots, response_n]$
```

4.4 Algorithm

The core acceleration mechanisms of SEED, which combines speculative scheduled execution with the rounds-scheduled strategy, is presented in Algorithm 1.

At its essence, the parallel drafting is realized by multiple parallel processes $\mathcal{D}(n)$, while the sequential verification is realized by a verification process $\mathcal{V}$ that cyclically verifies from the verify queue $\mathcal{Q}$. The verification process has two phases, which are the verify phase $\mathcal{E}$ and the resampling phase $\mathcal{R}$. To maintain the asynchronous nature of the *draft-then-verify* event loop, leveraging a draft label map γ ensures each draft process waits for verification before proceeding with new drafts. At the initial stage, each element in the draft label map γ is set to 1, indicating all draft models can perform drafting. After completing the verification of a draft model, the corresponding label in γ changes to 0, awaiting for re-drafting. Notably, $\mathcal{D}(n)$ and $\mathcal{V}$ are *synchronized*. The termination condition for both process $\mathcal{D}(n)$ and process $\mathcal{V}$ is that all current validated token $\mathcal{L}_i, i \in [1, n]$ equals the max new length l. When all the processes are finished, we can obtain a list containing n response.

5 Experiments

5.1 Datasets

Three widely used reasoning and planning datasets are chosen for our experiments. To assess the effectiveness of creativity and planning tasks, we leverage the Creative Writing dataset (CW) (Yao et al., 2024), where the input is four random sentences and the output should be a coherent passage with four paragraphs that end in the four input sentences respectively, with a ToT tree depth $\mathcal{T}$ of 2. For mathematical reasoning, GSM8K (Cobbe et al., 2021) is a dataset comprising high-quality grade-school math word problems that require multi-step reasoning, with a tree depth $\mathcal{T}$ of 4. This task is open-ended and exploratory, posing significant challenges to creative thinking and high-level planning. To better demonstrate the speedup performance in solving more complex planning problems, we select the Blocksworld dataset (BW) (Valmeekam et al., 2023). We set the tree depth $\mathcal{T}$ to 7 for this task to allow for more iterations. Specifically, we utilize 1319 samples from the GSM8K test set, 100 random samples from the CW dataset following (Yao et al., 2024), and 145 samples from the BW step-6 dataset.

5.2 Baselines

This study focuses on accelerating the reasoning tree construction process rather than the search algorithm or advanced prompting methods. The selection of baselines will be discussed in Appendix A.1. We consider the following decoding paradigms as our baselines: (1) **AR** denotes the original ToT (Yao et al., 2024) that employing standard autoregressive generation as shown in Figure 1 (a); (2) **SD** presents the application of speculative sampling which is detailed in 3.2 on the basis of ToT as shown in Figure 1 (b); (3) **MCSD** utilizes multi-candidate sampling and employs a advanced verifying algorithm to improve the acceptance rate and enhance the speed of SD (Yang et al., 2024). Similar to SD, it adheres to only one single-sample serial execution process. Notably, both SD and

Temp.	k_{config}	Methods	CW($\mathcal{T}=2$) Tokens/s	CW($\mathcal{T}=2$) Speedup	GSM8K($\mathcal{T}=4$) Tokens/s	GSM8K($\mathcal{T}=4$) Speedup	BW($\mathcal{T}=7$) Tokens/s	BW($\mathcal{T}=7$) Speedup
0.2	-	AR	38.42	1.000×	42.31	1.000×	34.19	1.000×
	(1,1,1)	SD	39.96	1.040×	51.11	1.208×	36.28	1.061×
		w. SEED	**41.53**	**1.081**×	**53.14**	**1.256**×	**36.93**	**1.080**×
		MCSD	40.19	1.046×	52.42	1.239×	36.04	1.054×
		w. SEED	**41.46**	**1.079**×	**53.78**	**1.271**×	**36.96**	**1.081**×
	(2,2,1)	SD	46.22	1.203×	60.63	1.433×	40.04	1.171×
		w. SEED	**48.60**	**1.265**×	**65.24**	**1.542**×	**44.24**	**1.294**×
		MCSD	46.80	1.218×	60.88	1.439×	40.79	1.193×
		w. SEED	**48.79**	**1.270**×	**65.58**	**1.550**×	**44.75**	**1.309**×
1.0	-	AR	39.47	1.000×	47.81	1.000×	34.62	1.000×
	(1,1,1)	SD	45.90	1.163×	55.32	1.157×	35.14	1.015×
		w. SEED	**46.77**	**1.185**×	**61.01**	**1.276**×	**38.94**	**1.125**×
		MCSD	45.63	1.156×	58.47	1.223×	38.05	1.099×
		w. SEED	**46.54**	**1.179**×	**65.50**	**1.370**×	**40.02**	**1.156**×
	(2,2,1)	SD	57.39	1.454×	66.74	1.396×	45.98	1.328×
		w. SEED	**58.89**	**1.492**×	**72.62**	**1.519**×	**47.22**	**1.364**×
		MCSD	56.24	1.425×	67.36	1.409×	46.18	1.334×
		w. SEED	**59.76**	**1.514**×	**74.44**	**1.557**×	**47.71**	**1.378**×

Table 1: The speedup performance of our proposed SEED and baselines, with settings of SEED for M_d and M_t being LLaMA-68M and LLaMA2-7B, respectively. The illustration of k_{config}=(2,2,1) is presented in Appendix B. All speedups are relative to the vanilla AR. The best results among all methods are in **bolded**.

MCSD are **orthogonal** to our proposed SEED. We apply our framework within these two decoding approaches to validate SEED's effectiveness across different accetance rates.

5.3 Setup

Model Suite Our evaluation is based on the publicly available LLaMA Chat suite (Touvron et al., 2023b), which has shown strong performance in executing instructions and in TSB scenarios. We utilize (M_d, M_t) following previous work (Chen et al., 2023b; Yang et al., 2024): (LLaMA-68M-Chat[3] , LLaMA-2-Chat-7B[4]) and (LLaMA-160M-Chat[5] , LLaMA-2-Chat-13B[6]). To validate the extensibility of our framework, we also conducted experiments using the QWen suite (Bai et al., 2023). Detailed information and results for both the other LLaMA pair and QWen suite can be found in Appendix A.2.

Hyperparameters We perform a BFS algorithm as the search strategy. Temperatures are set to 0.2 and 1.0 to evaluate under different conditions.[7] The detailed prompts for the Thought Generator and the State Evaluator, along with the ToT setup for each task are provided in Appendix D.

Environment The experiments are conducted on a single NVIDIA RTX A100-80G or a single node which is equipped with four NVIDIA RTX 3090-24GB GPUs. Subtle differences in hardware performance between these platforms are discussed in Appendix A.4.

6 Results and Analysis

6.1 Main Results

Table 1 presents a comprehensive analysis of our proposed SEED and baselines applied to three reasoning datasets. If each element in k_{config} is 1, we use the traditional single sampling at each position of the draft sequence. Otherwise, we employ tree attention, which represents sample multiple candidate tokens at each position and verify in parallel (details in Section 3.2). A greater number at each position in k_{config} signifies that more candidates,

[3] https://huggingface.co/Felladrin/Llama-68M-Chat-v1

[4] https://huggingface.co/meta-llama/Llama-2-7b-chat-hf

[5] https://huggingface.co/Felladrin/Llama-160M-Chat-v1

[6] https://huggingface.co/meta-llama/Llama-2-13b-chat-hf

[7] We avoid the temperature 0 because greedy decoding is not meaningful in Thought Generator.

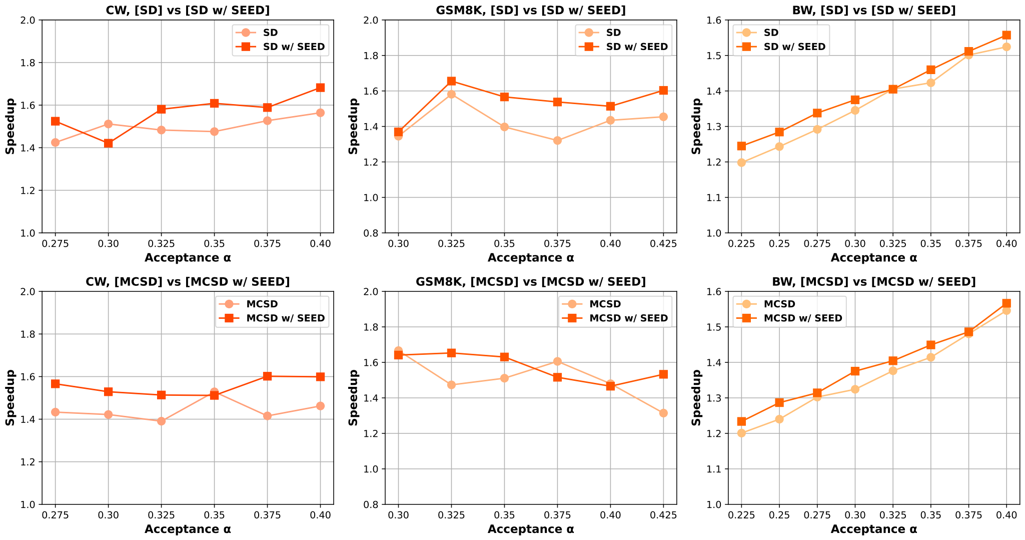

Figure 5: The variation of speedup performance across three datasets at different acceptance rates α.

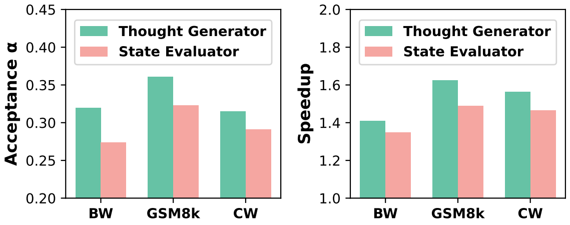

Figure 6: The acceptance rate α and the speedup performance of the Thought Generator and the State Evaluator.

generally yield higher speedups. MCSD achieves better speedup than SD by using an advanced verifying algorithm that results in higher acceptance rates. With our SEED, the performance of these two baselines is further improved, demonstrating its effectiveness across different acceptance rates. Across all datasets across various reasoning depths $\mathcal{T}$, our framework, consistently outperforms the baselines across different settings and configurations, including temperature and k_{config}, in terms of speedup, achieving the further speedup. Specifically, on the GSM8K dataset, using tree attention, MCSD in our proposed SEED framework achieves up to 1.5$\times$ speedup compared to AR, generating nearly 30 additional tokens per second.

6.2 Analysis

We use the SEED (with MCSD) to conduct the following analytical experiment to answer the following research question (*RQ*) using under the condition k_{config} = (2,2,1) and temperature = 1.0.

RQ1: How does* SEED *perform at different acceptance rates? We sampled data points from three datasets within different acceptance rate ranges, we separately reported the speedup achieved by SEED and the baseline for these samples in Figure 5. It is evident that under the same acceptance rate, SEED outperforms the baseline in terms of speedup. This improvement is attributed to our framework, which achieves speedup not by increasing the acceptance rate but by scheduling draft models. Additionally, as the acceptance rate increases, both SEED and the baseline exhibit a noticeable upward trend in speedup, which is the inherent characteristic of the speculative decoding method.

RQ2: Does* SEED *exhibit different acceleration effects on different components of ToT? SEED accelerate two components in reasoning tree construction, which are the TG and the SE. Figure 6 presents the acceptance rate α and the speedup performance of two main components of the SEED method on the GSM8K dataset, confirming that the answer to the *RQ2* is **Yes**. The TG executes multiple iterations with the same prompt while the SE refers to evaluates multiple iterations with different prompts. The TG component consistently outperforms the SE component in terms of both α and speedup, possibly because the SE is relatively harder compared to the TG. The proficiency between the target model and draft model may be more closely aligned in the proposal of thoughts, compared to decision-making capability.

RQ3: How does the speedup and GPU utilization scale with the number of thoughts? In speculative decoding, both the target and draft model pa-

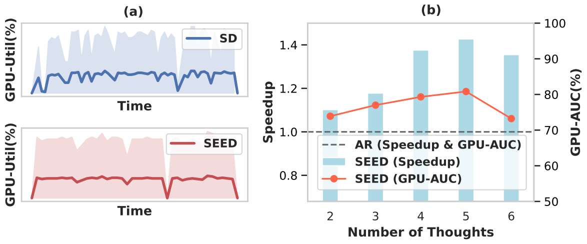

Figure 7: (a) The comparison visualization of GPU utilization between SD and SEED over the 120 seconds under n = 3. (b) The variation of speedup and acceptance rate α with the number of reasoning paths n.

rameters are loaded into GPU memory. We record the GPU utilization over the same durations for the SD and SEED on a GSM8K instance to visualize the effectiveness of parallel drafting in Figure 7 (a). The upper part illustrates the GPU utilization of SD fluctuates intermittently, primarily due to the target model being idle during drafting, while the lower part shows SEED exhibits stable utilization, attributed to the active engagement of the target model in the verification phase. As the number of thoughts n increases within a certain range, the idle time of the target model decreases, leading to higher GPU utilization and speedup, as shown in Figure 7 (b). However, when the number of thoughts becomes too large (*e.g.*, n=6), the target model's fixed verification capacity leads to SEED speedup saturation. This manifests as more draft models being placed in a waiting state, reducing draft parallelism and causing bottlenecks that lower utilization and acceleration.

7 Conclusion and Discussion

In this paper, we introduce SEED, a novel inference framework designed to optimize the runtime speed and manage GPU memory usage effectively during the reasoning tree construction for complex reasoning and planning tasks. SEED employs scheduled speculative execution to enhance the performance of LLMs by integrating the management of multiple draft models and a single target model, based on principles similar to operating system process scheduling. This strategy not only mitigates the inference latency inherent in tree-search-based reasoning methods but also efficiently utilizes the available computational resources. Our extensive experimental evaluation across three reasoning demonstrates that SEED achieves significant improvements in inference speed, generating up to 20 additional tokens per second.

Our work accelerates the development of ToT, offering the potential for **seamless** extension to further advance the test-time scaling of LLMs (Zhang et al., 2024a; Xie et al., 2024; Zhang et al., 2024c; Snell et al., 2024; Wu et al., 2024). This framework and vision represent a promising direction for improving the efficiency of LLM reasoning in real-world applications.

Limitations

Although SEED already achieves exceptional speedup performance in the experiments, our work also has the following limitations.

- Our frameworks introduce parallel drafting, involving $n-1$ additional drafting models, which inherently necessitates the addition of an equivalent number of KV-Cache. Given the increase attributed to small draft models (68M/160M) is relatively minimal, we do not optimize the management of the KV-Cache in this work.
- This study focuses solely on optimizing the inference speed of the tree construction for the TSB reasoning task and does not optimize the search speed for these tasks. Our framework uses the relatively simple search algorithm BFS. In fact, SEED can seamlessly integrate more advanced search algorithms, such as A^* (Hart et al., 1968) and MCTS (Kocsis and Szepesvári, 2006), *etc.*, which we leave for future research.
- We employ the most widely used scheduling algorithm, FCFS. Future work may explore the integration of more advanced scheduling algorithms, such as those used in real-time systems, to further enhance the responsiveness and efficiency of SEED.

In the future, SEED can be compatible with vLLM (Kwon et al., 2023) and FlashAttention-2 (Dao, 2024), enabling more memory-efficient inference on longer sequences. Additionally, the extra KV-Cache could be reduced by caching the common prefix during reasoning tree construction, which would lower the parallel overhead in later iterations.

Moreover, our method offers a potential implementation of batched speculative decoding from the execution scheduling perspective, which could be integrated with other KV-Cache based batch speculative decoding methods (Ni et al., 2024), as further discussed in Appendix A.5.

Acknowledgement

The authors would like to thank the anonymous reviewers for their insightful comments. This work is funded by the National Natural Science Foundation of China (Grant No.62176053). This work is supported by the Big Data Computing Center of Southeast University.

References

Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. 2023. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*.

Jinze Bai, Shuai Bai, Yunfei Chu, Zeyu Cui, Kai Dang, Xiaodong Deng, Yang Fan, Wenbin Ge, Yu Han, Fei Huang, et al. 2023. Qwen technical report. *arXiv e-prints*, pages arXiv–2309.

Nikhil Bhendawade, Irina Belousova, Qichen Fu, Henry Mason, Mohammad Rastegari, and Mahyar Najibi. 2024. Speculative streaming: Fast llm inference without auxiliary models. *arXiv e-prints*, pages arXiv–2402.

Ondřej Bojar, Christian Buck, Christian Federmann, Barry Haddow, Philipp Koehn, Johannes Leveling, Christof Monz, Pavel Pecina, Matt Post, Herve Saint-Amand, Radu Soricut, Lucia Specia, and Aleš Tamchyna. 2014. Findings of the 2014 workshop on statistical machine translation. In *Proceedings of the Ninth Workshop on Statistical Machine Translation*, pages 12–58, Baltimore, Maryland, USA. Association for Computational Linguistics.

Alan Bundy and Lincoln Wallen. 1984. Breadth-first search. *Catalogue of artificial intelligence tools*, pages 13–13.

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. In *Forty-first International Conference on Machine Learning*.

Charlie Chen, Sebastian Borgeaud, Geoffrey Irving, Jean-Baptiste Lespiau, Laurent Sifre, and John Jumper. 2023a. Accelerating large language model decoding with speculative sampling. *arXiv e-prints*, pages arXiv–2302.

Ziyi Chen, Xiaocong Yang, Jiacheng Lin, Chenkai Sun, Jie Huang, and Kevin Chen-Chuan Chang. 2023b. Cascade speculative drafting for even faster llm inference. *arXiv preprint arXiv:2312.11462*.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, et al. 2021. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*.

Tri Dao. 2024. Flashattention-2: Faster attention with better parallelism and work partitioning. In *The Twelfth International Conference on Learning Representations*.

Luciano Del Corro, Allison Del Giorno, Sahaj Agarwal, Bin Yu, Ahmed Hassan Awadallah, and Subhabrata Mukherjee. 2023. Skipdecode: Autoregressive skip decoding with batching and caching for efficient llm inference.

Mostafa Elhoushi, Akshat Shrivastava, Diana Liskovich, Basil Hosmer, Bram Wasti, Liangzhen Lai, Anas Mahmoud, Bilge Acun, Saurabh Agarwal, Ahmed Roman, Ahmed Aly, Beidi Chen, and Carole-Jean Wu. 2024. LayerSkip: Enabling early exit inference and self-speculative decoding. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 12622–12642, Bangkok, Thailand. Association for Computational Linguistics.

Marjan Ghazvininejad, Omer Levy, Yinhan Liu, and Luke Zettlemoyer. 2019. Mask-predict: Parallel decoding of conditional masked language models. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 6112–6121.

Lin Guan, Karthik Valmeekam, Sarath Sreedharan, and Subbarao Kambhampati. 2023. Leveraging pretrained large language models to construct and utilize world models for model-based task planning. *Advances in Neural Information Processing Systems*, 36:79081–79094.

Shibo Hao, Yi Gu, Haodi Ma, Joshua Hong, Zhen Wang, Daisy Wang, and Zhiting Hu. 2023. Reasoning with language model is planning with world model. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 8154–8173.

Peter E Hart, Nils J Nilsson, and Bertram Raphael. 1968. A formal basis for the heuristic determination of minimum cost paths. *IEEE transactions on Systems Science and Cybernetics*, 4(2):100–107.

Zhenyu He, Zexuan Zhong, Tianle Cai, Jason D Lee, and Di He. 2023. Rest: Retrieval-based speculative decoding. *arXiv e-prints*, pages arXiv–2311.

Wenyang Hui, Yan Wang, Kewei Tu, and Chengyue Jiang. 2024. Rot: Enhancing large language models with reflection on search trees. *arXiv preprint arXiv:2404.05449*.

Levente Kocsis and Csaba Szepesvári. 2006. Bandit based monte-carlo planning. In *European conference on machine learning*, pages 282–293. Springer.

Takeshi Kojima, Shixiang Shane Gu, Machel Reid, Yutaka Matsuo, and Yusuke Iwasawa. 2022. Large language models are zero-shot reasoners. *Advances in neural information processing systems*, 35:22199–22213.

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

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

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

Haoran Liao, Jidong Tian, Shaohua Hu, Hao He, and Yaohui Jin. 2024. Look before you leap: Problem elaboration prompting improves mathematical reasoning in large language models. *arXiv e-prints*, pages arXiv–2402.

Xiaoxuan Liu, Lanxiang Hu, Peter Bailis, Ion Stoica, Zhijie Deng, Alvin Cheung, and Hao Zhang. 2023. Online speculative decoding.

Bo-Ru Lu, Nikita Haduong, Chien-Yu Lin, Hao Cheng, Noah A Smith, and Mari Ostendorf. 2024a. Encode once and decode in parallel: Efficient transformer decoding. *arXiv e-prints*, pages arXiv–2403.

Jinghui Lu, Ziwei Yang, Yanjie Wang, Xuejing Liu, and Can Huang. 2024b. Padellm-ner: Parallel decoding in large language models for named entity recognition. *arXiv e-prints*, pages arXiv–2402.

Yunsheng Ni, Chuanjian Liu, Yehui Tang, Kai Han, and Yunhe Wang. 2024. Ems-sd: Efficient multi-sample speculative decoding for accelerating large language models. *arXiv e-prints*, pages arXiv–2405.

OpenAI. 2022. Introducing ChatGPT.

Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, et al. 2022. Training language models to follow instructions with human feedback. *Advances in neural information processing systems*, 35:27730–27744.

Andysah Putera Utama Siahaan. 2016. Comparison analysis of cpu scheduling: Fcfs, sjf and round robin. *International Journal of Engineering Development and Research*, 4(3):124–132.

Charlie Snell, Jaehoon Lee, Kelvin Xu, and Aviral Kumar. 2024. Scaling llm test-time compute optimally can be more effective than scaling model parameters. *arXiv preprint arXiv:2408.03314*.

Hanshi Sun, Zhuoming Chen, Xinyu Yang, Yuandong Tian, and Beidi Chen. 2024a. Triforce: Lossless acceleration of long sequence generation with hierarchical speculative decoding. In *First Conference on Language Modeling*.

Ziteng Sun, Ananda Theertha Suresh, Jae Hun Ro, Ahmad Beirami, Himanshu Jain, and Felix Yu. 2024b. Spectr: Fast speculative decoding via optimal transport. *Advances in Neural Information Processing Systems*, 36.

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. 2023a. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*.

Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, et al. 2023b. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*.

Karthik Valmeekam, Matthew Marquez, Sarath Sreedharan, and Subbarao Kambhampati. 2023. On the planning abilities of large language models-a critical investigation. *Advances in Neural Information Processing Systems*, 36:75993–76005.

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc V Le, Ed H. Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. 2023. Self-consistency improves chain of thought reasoning in language models. In *The Eleventh International Conference on Learning Representations*.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. 2022. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35:24824–24837.

Siwei Wu, Zhongyuan Peng, Xinrun Du, Tuney Zheng, Minghao Liu, Jialong Wu, Jiachen Ma, Yizhi Li, Jian Yang, Wangchunshu Zhou, et al. 2024. A comparative study on reasoning patterns of openai's o1 model. *arXiv preprint arXiv:2410.13639*.

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.

Yuxi Xie, Anirudh Goyal, Wenyue Zheng, Min-Yen Kan, Timothy P Lillicrap, Kenji Kawaguchi, and Michael Shieh. 2024. Monte carlo tree search boosts reasoning via iterative preference learning. *arXiv preprint arXiv:2405.00451*.

Hanqi Yan, Qinglin Zhu, Xinyu Wang, Lin Gui, and Yulan He. 2024. Mirror: A multiple-perspective self-reflection method for knowledge-rich reasoning. *arXiv preprint arXiv:2402.14963*.

Sen Yang, Shujian Huang, Xinyu Dai, and Jiajun Chen. 2024. Multi-candidate speculative decoding. *arXiv e-prints*, pages arXiv–2401.

Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Tom Griffiths, Yuan Cao, and Karthik Narasimhan. 2024. Tree of thoughts: Deliberate problem solving with large language models. *Advances in Neural Information Processing Systems*, 36.

Di Zhang, Jianbo Wu, Jingdi Lei, Tong Che, Jiatong Li, Tong Xie, Xiaoshui Huang, Shufei Zhang, Marco Pavone, Yuqiang Li, et al. 2024a. Llama-berry: Pairwise optimization for o1-like olympiad-level mathematical reasoning. *arXiv preprint arXiv:2410.02884*.

Linhai Zhang, Jialong Wu, Deyu Zhou, and Guoqiang Xu. 2024b. STAR: Constraint LoRA with dynamic active learning for data-efficient fine-tuning of large language models. In *Findings of the Association for Computational Linguistics ACL 2024*, pages 3519–3532, Bangkok, Thailand and virtual meeting. Association for Computational Linguistics.

Xuan Zhang, Chao Du, Tianyu Pang, Qian Liu, Wei Gao, and Min Lin. 2024c. Chain of preference optimization: Improving chain-of-thought reasoning in LLMs. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*.

Wei Zhao and John A Stankovic. 1989. Performance analysis of fcfs and improved fcfs scheduling algorithms for dynamic real-time computer systems. In *1989 Real-Time Systems Symposium*, pages 156–157. IEEE Computer Society.

Yongchao Zhou, Kaifeng Lyu, Ankit Singh Rawat, Aditya Krishna Menon, Afshin Rostamizadeh, Sanjiv Kumar, Jean-François Kagy, and Rishabh Agarwal. 2023. Distillspec: Improving speculative decoding via knowledge distillation. In *The Twelfth International Conference on Learning Representations*.

A Discussions

A.1 Selection of Baselines

See Section 5.2, where we list all the baselines used to compare with our proposed SEED in this study. However, several other speculative decoding strategies have not been explored as baselines. We do not conclude these strategies based on the following considerations as shown in Table 4:

(1) Training-free indicates whether the method requires training.

- ∗ **Medusa** (Cai et al., 2024) adds extra FFN heads atop the Transformer decoder, allowing for parallel token generation at each step;
- ∗ **Eagle** (Li et al., 2024) performs the drafting process autoregressively at a more structured level, specifically the second-to-top layer of features;
- ∗ **SS** (Bhendawade et al., 2024) integrates drafting phase into the target model by modifying the fine-tuning objective from the next token to future n-gram predictions.

These methods all **require training and are not plug-and-play**, since they train the LLM to serve as both the target model and the draft model, which classifies them as self-drafting ■ according to Xia et al. (2024); in contrast, our method employs independent drafting ▲ (draft-and-target), placing it in a different SD type. Therefore, we do not consider them as baselines.

(2) Extra-knowledge-free indicates whether the SD process uses additional knowledge modules.

- ∗ **CS-drafting** (Chen et al., 2023b) resorts to a bigram model based on the probability distribution of Wikipedia as the draft model at a more basic level.
- ∗ **REST** (He et al., 2023) retrieve from extensive code and conversation data stores to generate draft tokens.

The two approaches introduce external knowledge modules, making it significantly dependent on the effectiveness of the external knowledge modules and unfair to compare us with draft-and-target models.

(3) Lossless indicates whether the method generates the same output distribution as AR decoding does in the backbone model.

SS (Bhendawade et al., 2024) and **Medusa** (Cai et al., 2024), which are inherently not lossless, are unsuitable for comparison with **our proposed SEED, which maintains losslessness consistent with SD in a single *draft-then-verify***.

Temp.	k_{config}	**Methods**	**Tokens/s**	**Speedup**
0.2	-	AR	31.22	1.000×
	(1,1,1,1)	SD	32.91	1.054×
		w. SEED	**34.62**	**1.109×**
0.6	-	AR	37.93	1.000×
	(1,1,1,1)	SD	39.22	1.034×
		w. SEED	**41.91**	**1.105×**
1	-	AR	33.86	1.000×
	(1,1,1,1)	SD	34.91	1.031×
		w. SEED	**39.35**	**1.162×**

Table 2: Speedup performance on Creative Writing dataset of SEED within using QWen1.5-0.5B-Chat as M_d and QWen1.5-7B-Chat as M_t. The vocabularies of these two models are identical, allowing for speculative sampling.

A.2 Scalability and Extensibility

LLaMA Suite Table 3 shows the performance of each method when using LLaMA-160M-Chat[8] as draft model M_d and LLaMA-2-Chat-13B[9] as target model M_t.

QWen Suite Our framework is based on speculative decoding, so the model setup of the draft model and the target model can be consistent with it. Consequently, any LLM suite can be integrated into our framework. We also conducted experiments using the QWen1.5 suite.[10] Specifically, we use QWen1.5-0.5B-Chat[11] as the draft model M_d and use QWen1.5-7B-Chat[12] as the target model M_t. The results are presented in Table 2. The results align with the findings presented in Section 6.1, demonstrating the superior performance of our framework. It also highlights the scalability of our framework to the LLM suite (Bai et al., 2023).

A.3 Task Performance

Accuracy Leviathan et al. (2023) has proved the outputs of AR and SD are the same. We separately evaluated the performance of the GSM8K dataset using the AR with QWen1.5-7B and SEED with

[8] https://huggingface.co/Felladrin/Llama-160M-Chat-v1

[9] https://huggingface.co/meta-llama/Llama-2-13b-chat-hf

[10] https://qwenlm.github.io/zh/blog/qwen1.5/

[11] https://huggingface.co/Qwen/Qwen1.5-0.5B-Chat

[12] https://huggingface.co/Qwen/Qwen1.5-7B-Chat

Temp.	k_{config}	Methods	**CW**($\mathcal{T} = 2$)		**GSM8K**($\mathcal{T} = 4$)		**BW**($\mathcal{T} = 7$)	
			Tokens/s	Speedup	Tokens/s	Speedup	Tokens/s	Speedup
0.2	-	AR	32.33	1.000×	32.08	1.000×	32.91	1.000×
	(2,1,1)	SD	33.14	1.025×	34.97	1.090×	33.17	1.008×
		w. SEED	**33.82**	**1.046**×	**36.80**	**1.147**×	**33.54**	**1.019**×
		MCSD	33.27	1.029×	35.71	1.113×	33.37	1.014×
		w. SEED	**36.18**	**1.119**×	**36.28**	**1.131**×	**34.36**	**1.044**×
	(4,2,1)	SD	34.23	1.059×	38.95	1.214×	36.04	1.095×
		w. SEED	**38.57**	**1.193**×	**41.06**	**1.280**×	**36.76**	**1.117**×
		MCSD	35.56	1.100×	41.09	1.281×	37.58	1.142×
		w. SEED	**40.28**	**1.246**×	**44.11**	**1.375**×	**38.70**	**1.176**×
1.0	-	AR	39.57	1.000×	31.54	1.000×	32.87	1.000×
	(2,1,1)	SD	40.28	1.018×	35.23	1.117×	34.32	1.044×
		w. SEED	**42.74**	**1.080**×	**36.71**	**1.164**×	**35.37**	**1.076**×
		MCSD	40.68	1.028×	35.26	1.118×	35.01	1.065×
		w. SEED	**43.37**	**1.096**×	**37.15**	**1.178**×	**35.86**	**1.091**×
	(4,2,1)	SD	43.69	1.104×	36.87	1.169×	37.83	1.151×
		w. SEED	**47.25**	**1.194**×	**40.66**	**1.289**×	**38.56**	**1.173**×
		MCSD	45.19	1.142×	36.90	1.170×	39.28	1.195×
		w. SEED	**49.74**	**1.257**×	**41.54**	**1.317**×	**40.43**	**1.230**×

Table 3: Speedup performance of our proposed SEED and baselines, with settings of SEED for M_d and M_t being LLaMA-160M and LLaMA2-13B, respectively. All speedups are relative to the vanilla AR. The best results among all methods are in **bolded**.

the aforementioned QWen1.5 suite using QWen1.5-0.5B and QWen1.5-7B, and found that the performance difference was within $\pm 1.5\%$, which is acceptable and substantiates that the performance is effectively **lossless**.

Performance on Non-Reasoning Tasks SEED is a versatile method that can be applied not only in reasoning tasks involving TSB but also in non-reasoning tasks. Its general applicability makes it a robust solution for various scenarios. We specifically applied SEED to the TSB in reasoning tasks based on several key considerations:

- **Practicality of TSB:** The TSB method allows the generation of multiple sequences simultaneously in both identical and varied input scenarios. This makes it a practical choice for efficient processing.
- **Efficiency on Consumer-Grade GPUs:** Typically, TSB involves generating 2-6 reasoning paths concurrently, which can be handled by consumer-grade GPUs. By contrast, promtping methods like Self-Consistency (Wang et al., 2023) often require generating 10-20 sequences, parallelly placing a greater strain on hardware resources.
- **Relevance to Task Difficulty:** Reasoning tasks are challenging benchmarks for evaluating LLMs. If our framework achieves effective acceleration under acceptance in these tasks, it is likely to perform well on simpler tasks, like translation, where the alignment between the target and draft models is better. In early exploratory experiments, SEED achieved a **1.31x speedup** over AR on the WMT dataset (Bojar et al., 2014), demonstrating its efficacy.

A.4 Hardware Dependency

The experiments was conducted on a 4×3090 server in the earlier exploratory. From the experiments on different hardware shown in Figure 5, our method is still effective compared with SD with the same setting. The speedup performance on 4×3090 is lower than on 1×A100, likely due to the increased communication time between multiple GPUs (Sun et al., 2024a). This is also evident from the Qwen suite results, where SD performs worse than AR on 4×3090.

A.5 Batch Inference

Batch inference processes multiple sequences of varying lengths. In SD, each sequence in the same batch requires extra padding due to different accep-

Methods	**Training-free**	**Lossless**	**SD Type**	**Extra-knowledge-free**	**Speedup**
Vanilla AR	✓	✓	-	✓	✗
SD (Leviathan et al., 2023)	✓	✓	▲	✓	✓
CS-Drafting (Chen et al., 2023b)	✓	✓	▲	✗	✓
REST (He et al., 2023)	✓	✓	▲	✗	✓
Medusa (Cai et al., 2024)	✗	✗	■	✓	✓
Eagle (Li et al., 2024)	✗	✓	■	✓	✓
SS (Bhendawade et al., 2024)	✗	✗	■	✓	✓
MCSD (Yang et al., 2024)	✓	✓	▲	✓	✓
SEED (Ours)	✓	✓	▲	✓	✓

Table 4: The comprehensive comparison of the listed methods and SEED. ■ represents draft-and-target SD method, while ▲ represents self-draft SD method.

LLM Suite	**GPUs**	**Methods**	**Tokens/s**	**Speedup**
LLaMA2 160M/13B	**4×RTX 3090s**	AR	38.77	1.000×
		SD	42.18	1.088×
		w. SEED	**44.93**	**1.159×**
	1×RTX A100	AR	39.57	1.000×
		SD	43.69	1.104×
		w. SEED	**47.25**	**1.194×**
Qwen1.5 0.5B/7B	**4×RTX 3090s**	AR	27.51	1.000×
		SD	27.43	0.997×
		w. SEED	**29.57**	**1.075×**
	1×RTX A100	AR	33.86	1.000×
		SD	34.91	1.031×
		w. SEED	**39.35**	**1.162×**

Table 5: Speed performance of LLaMA2 suite on Creative Writing dataset under different hardware environments with temperture = 1.0 and k_{config} = (1,1,1,1), as well as the performance of Qwen1.5 suite suite on GSM8K dataset across different hardware environments with temperture = 1.0 and k_{config} = (4,2,1).

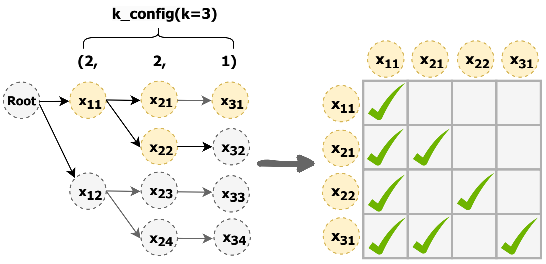

Figure 8: The tree attention used in SEED, multiple tokens in single sequence concurrently are processed. *Root* indicates previous tokens. ✓ indicates where attention is present, while the rest are masked. For simplicity, we only visualize the tree attention mask of tokens in yellow colors.

tance rates and sequence lengths, potentially leading to excessive storage and computation (Ni et al., 2024). This can result in an overly long KV-Cache, thereby slowing down the speedup effect due to inconsistent acceptance lengths. Our SEED maintains the original length of KV-Cache without the need for padding based on varying acceptance rates. Each verified draft sequence corresponds directly to a sequence in the batch (number of draft models n = batch size). Our parallel drafting approach ensures efficient batch implementation while preserving the acceleration benefits of SD.

B Details of Tree Attention

Setting k_{config} to (2,2,1) indicates that each draft phase generates a group of $k = 3$ tokens, with the first two positions each sampling 2 candidates, and the third position sampling 1. Figure 8 illustrates a case of tree attention with a configuration of $k_{\text{config}} = (2, 2, 1)$.

C Reasoning Tree Construction

The process of constructing a reasoning tree using the BFS Algorithm is outlined in Algorithm 2.

Algorithm 2 SEED$(x, p_\theta, \mathrm{G}, n, \mathrm{E}, s, b)$

```
Input: Initial prompt $\mathcal{I}$, speculative scheduled execution with a rounds-scheduled strategy $p_\theta$, thought generator $\mathrm{G}(\cdot)$ with a number of thought $n$, states evaluator $\mathrm{E}(\cdot)$, step limit $\mathcal{T}$, breadth limit $b$.
Initialize: States $S$; $S_0 \leftarrow \{\mathcal{I}\}$
for $i = 1, \cdots, \mathcal{T}$ do
    $S'_i \leftarrow \{[c, z_i] \mid c \leftarrow S_{i-1}, z_i \in \mathrm{G}(p_\theta, c, n)\}$    ▷ Generate thoughts in Parallel
    $E_i \leftarrow E(p_\theta, S'_i)$    ▷ Evaluate states in Parallel
    $S_i \leftarrow \arg\max_{S \subset S'_i, |S|=b} \sum_{s \in S} E_i(s)$
end for
return $\mathrm{G}(p_\theta, \arg\max_{s \in S_{\mathcal{T}}} E_{\mathcal{T}}(s), 1)$
```

D Detailed Setup and Prompts

We implemented a simple and generic ToT-BFS according to Yao et al. (2024). Within the Thought Generator, we leverage a sampling strategy to generate thoughts for the next thought step. Within the State Evaluator, we leverage a value strategy to evaluate the generated thoughts and output a scalar value (*e.g.*, "1-10") or a classification (*e.g.*, "*good/bad*") which can be heuristically converted into a value. To introduce diversity in thought generation across all tasks, we set the generation temperature as 0.2/1($>$0) for the LLaMA suite models and 0.2/0.6/1($>$0) for the QWen suite models. The tree depth $\mathcal{T}$ suggests that the operations with varying levels of complexity or iterations, with deeper trees potentially representing more complex calculations or decision-making processes. The ToT setup of the three tasks SEED utilized is as follows:

- **Creative Writing**: We build a reasoning tree with a depth $\mathcal{T}$ of 2 (with 1 intermediate thought step) that generates 3 plans and passages. The State Evaluator assesses the plans and outputs a coherency score with each plan and passage.
- **GSM8K**: We build a reasoning tree with a depth $\mathcal{T}$ of 4 (with 3 intermediate thought steps) that generates 3 sub-questions and corresponding sub-answers. This setup aligns with the findings from Hao et al. (2023), which indicated that three steps are generally sufficient to achieve a passable level of accuracy. The State Evaluator assesses them and outputs a number representing the helpfulness for answering the question. We select the one with the highest values and add it to the previous sub-question and sub-answers.
- **Blocksworld 6-step**: We build a reasoning tree with a depth $\mathcal{T}$ of 7 (with 6 intermediate thought steps) that generates 3 thoughts, including action plans and current actions. Due to the complexity of this task, demonstrations are provided in the prompt, labeled as "*good/bad*", to assist the State Evaluator in its assessment.

The prompts for the tasks described above are presented below. The orange parts in prompts are required for LLM completion. During the evaluation, we require the LLM to generate both a score and an explanation (a context with 128 new tokens), **rather than just a score**. This approach promotes the speedup in generation and makes the evaluation of ToT more reasonable.

Prompts for GSM8K

The Thought Generator

Given a question: {initial_prompt}, the previous sub−question and sub−answer is:
{state_text}
Please output the next sub−question to further reason the question.
The sub−question is: {sub-question}
−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−
Given a question: {initial_prompt}, the sub−question is: {sub_question}
Please answer the sub−question based on the question.
The sub−answer is: {sub_answer}

The State Evaluator

Given a question: {initial_prompt}, the sub−question is: {sub_question}, the sub−answer is: {sub_answer}
Please output a number between 1 and 10 to evaluate the answer. The higher the number, the more help there is in answering the question.

The number is: {value}

Prompts for Creative Writing

The Thought Generator

Write a coherent passage of 4 short paragraphs. The end sentence of each paragraph must be:
{initial_prompt}
Make a plan then write. Your output should be of the following format:

Plan:
Your plan here.

Passage:
Your passage here.

The output is:
{Plan}
{Passage}

The State Evaluator

Analyze the passage: {Passage}, then at the last line conclude "Thus the coherency score is [s]", where [s] is an integer from 1 to 10.
The coherency score is: {value}

Prompts for Blocksworld

The Thought Generator

I am playing with a set of blocks where I need to arrange the blocks into stacks. Here are the actions I can do:

Pick up a block
Unstack a block from on top of another block
Put down a block
Stack a block on top of another block

I have the following restrictions on my actions:
##Restrictions on Action##

<—Omit demonstrations—>

[STATEMENT]
{initial_prompt}

My plan is as follows:
{state_text}
The current action is:
{action}

The State Evaluator

I am playing with a set of blocks where I need to arrange the blocks into stacks. Here are the actions I can do:

Pick up a block
Unstack a block from on top of another block
Put down a block
Stack a block on top of another block

I have the following restrictions on my actions:
##Restrictions on Action##

<—Omit demonstrations—>

Please evaluate whether the given action is a good one under certain conditions.

[STATEMENT]
{initial_prompt}
[ACTION]
{state_text}
[EVALUATION]
The evaluation is:
{evaluation}

Restrictions on Action for Blocksworld

I have the following restrictions on my actions:
I can only pick up or unstack one block at a time.
I can only pick up or unstack a block if my hand is empty.
I can only pick up a block if the block is on the table and the block is clear. A block is clear if the block has no other blocks on top of it and if the block is not picked up.
I can only unstack a block from on top of another block if the block I am unstacking was really on top of the other block.
I can only unstack a block from on top of another block if the block I am unstacking is clear.
Once I pick up or unstack a block, I am holding the block.
I can only put down a block that I am holding.
I can only stack a block on top of another block if I am holding the block being stacked.
I can only stack a block on top of another block if the block onto which I am stacking the block is clear.
Once I put down or stack a block, my hand becomes empty.