SEED: Accelerating Reasoning Tree Construction via Scheduled Speculative Decoding

Anonymous ACL submission

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 surpassing the capabilities of chain-of-thought prompting, encouraging exploration of intermediate steps. However, such methods introduce significant inference latency due to the systematic exploration and evaluation of multiple thought paths. This paper introduces SEED, a novel and efficient inference framework to optimize runtime speed and GPU memory management concurrently. By employing a scheduled speculative execution, SEED efficiently handles multiple iterations for the thought generation and the state evaluation, leveraging a rounds-scheduled strategy to manage draft model dispatching. Extensive experimental evaluations on three reasoning datasets demonstrate superior speedup performance of SEED, providing a viable path for batched inference in training-free speculative decoding.[1]

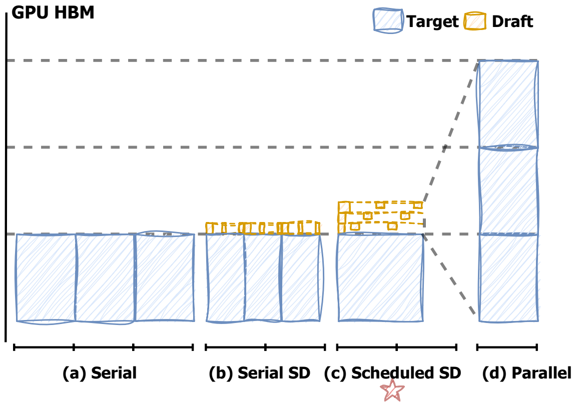

Figure 1: Illustration of four LLM execution strategies for generating $n = 3$ sequences in Reasoning Tree constructing: (a) *Serial*, where executions are operated one after another, simplifying resource management but increasing overall execution time; (b) *Seiral SD*, where speculative decoding is used for each execution; (c) *Scheduled*, 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.

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 in complex reasoning and planning tasks remains suboptimal. Traditional or simple prompting techniques (Wei et al., 2022; Kojima et al., 2022), which have been widely leveraged, are insufficient for 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 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 fully leverage the capabilities of LLMs to generate and evaluate intermediate thoughts and then integrate them with search algorithms to improve problem-solving efficiency.

[1] The code of this paper will be publicly available upon the acceptance of the paper.

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 efficacy of 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 crafting, which is the focus of this paper. Tree construction involves two components that directly impact the inference time of LLMs: the Thought Generator and the State Evaluator. The Thought Generator is responsible for creating multiple distinct paths from the same prompt, whereas the State Evaluator evaluates these paths to determine the optimal one, utilizing different prompts for each evaluation.

Traditional *Sequential* execution of LLMs necessitates repeated executions by both components, leading to long execution time, as shown in 1 (a). For instance, when applying ToT prompting to execute a single sample in the GSM8K dataset, the average total runtime is approximately 80 seconds using *sequential* processing with a 7B model on a consumer GPU. If the execution of LLMs shifts from *sequential* to *parallel* processing, it could pose challenges for end-users or researchers with access only to consumer GPUs, as illustrated in 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, 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 speed performance that is comparable to the scenarios illustrated in Figure 1 (d). Moreover, speculative decoding employs a *draft-then-verify* two-stage paradigm, 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 full potential of a single target model can be effectively harnessed, ensuring its capacity is maximally 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. Specifically, we introduce a novel execution strategy, Speculative Scheduled Execution, inspired by the use of speculative decoding in parallel drafting, 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 operating system management of process scheduling (Zhao and Stankovic, 1989; Siahaan, 2016). To this end, the Rounds-Scheduled strategy that uses a Fist-Come-Fist-Serve (FCFS) deque 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. Our framework also provides a viable path for conducting *batched inference* in training-free speculative decoding.

Our contribution can be summarized as follows:

- An efficient inference framework, SEED, is proposed to accelerate two components in reasoning tree construction.
- We propose the Speculative Scheduled Execution that integrates parallel drafting with speculative decoding, employing an effective Rounds-Scheduled strategy to manage parallel drafting devoid of verification conflicts.
- Empirically, extensive experiments and ablation studies are conducted to demonstrate the effectiveness of SEED. We show that SEED achieves an average speedup of up to 1.5× across three reasoning datasets.

2 Related Works

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, crafting or searching 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 crafting. Instead, we focus on the tree-crafting process, leveraging speculative decoding to accelerate the crafting 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). SD accelerates LLMs inference by employing a faster draft model for generating multiple tokens, which are then verified in parallel by a larger target model, resulting in the text generated according to the target model distribution (Xia et al., 2023; Leviathan et al., 2023). 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., 2024; 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. Recent SD works focus on designing diversity strategies for the single drafting or verifying process (Chen et al., 2023b; Yang et al., 2024), and entirely different training and inference mechanisms (Li et al., 2024; Kou et al., 2024; Zhong and Bharadwaj, 2024). In contrast, this paper explores a scheduled SD execution to speed up parallel inference further. As far as we know, we are the first to integrate multiple parallel prompts with the TSB reasoning task, without modifying LLM architecture or requiring additional training.

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 and M_d and M_t be the draft and the target model respectively, k be the number of draft tokens generated per step. Speculative decoding is a *Draft-then-Verify* [2] two-stage decoding paradigm. In the draft stage, M_d samples a draft sequence of tokens autoregressively, denoted as $\hat{x}_1, \dots, \hat{x}_k$, where $\hat{x}_i \sim p_d(x|\hat{x}_1, \dots, \hat{x}_{i-1}, c)$. In the verification stage, the draft tokens along with c, are passed to M_t to obtain their output distribution $p_t(x|\hat{x}_1, \dots, \hat{x}_{i-1}, c)$ in parallel, and then verified from $\hat{x}_1$ to $\hat{x}_k$. The draft token $\hat{x}_i$ is accepted with probability $\min(1, \frac{p_t(x|\hat{x}_1,\dots,\hat{x}_{i-1},c)}{p_d(x|\hat{x}_1,\dots,\hat{x}_{i-1},c)})$. 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 endpoint following the accepted tokens. It has been proven to maintain the same output as sampling autoregressively using the target model alone (Leviathan et al., 2023).

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). Within tree attention, a unique attention mask is applied to prevent information contamination among candidates and preserve causal relationships between tokens. Specifically, in a drafting phase, consider a scenario where the number of draft tokens is 3, with the multiple sampling configured as $k_{\text{config}} = (2, 2, 1)$ [3]. In this scenario,

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

[3] The length k of the k_{config} is 3, and each element represents the number of candidate tokens sampled at the corresponding position.

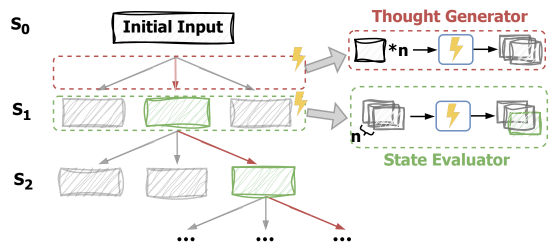

Figure 2: Two main components in reasoning tree construction, which are the Thought Generator and the State Evaluator, respectively.

M_d samples 2 candidate tokens in the first two positions and 1 candidate token in the third position per step. We denote $\hat{x}_{ij}$ as the j-th token generated by the M_d at position i. In the draft phase: At position 1, the candidates $\hat{x}_{11}$ and $\hat{x}_{12}$ are sampled. At position 2, with $\hat{x}_{11}$ as the predecessor, the $\hat{x}_{21}$ and $\hat{x}_{22}$ are sampled, and with $\hat{x}_{12}$ as the predecessor, $\hat{x}_{23}$ and $\hat{x}_{24}$ are sampled. At position 3, with $\hat{x}_{21}, \hat{x}_{22}, \hat{x}_{23}$ and $\hat{x}_{24}$ as the predecessors respectively, $\hat{x}_{31}, \hat{x}_{32}, \hat{x}_{33}$ and $\hat{x}_{34}$ are sampled respectively. We illustrate the tree attention mask strategy in Appendix B. For instance, we let $\hat{x}_{31}$ only attention to its ancestors $\hat{x}_{11}$ and $\hat{x}_{21}$ on the same continuation, while $\hat{x}_{22}$ is masked due to situate in different continuation with $\hat{x}_{31}$. This method, along with the KV-Cache (Park et al., 2020), enhances verification efficiency while introducing negligible computational overhead, making a practical solution for optimizing the latency of speculative decoding (Cai et al., 2024; Yang et al., 2024).

4 Method

Our proposed SEED is an efficient inference framework designed to accelerate the construction of a reasoning tree. 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. Finally, the combined approach is elaborated in §4.3.

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 exploration.

Algorithm 1 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_θ, 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**
  $S'_i \leftarrow \{[c, z_i] \mid c \leftarrow S_{i-1},$
    $z_i \in \mathrm{G}(p_\theta, c, n)\}$ ▷ Propose in Parallel
  $E_i \leftarrow E(p_\theta, S'_i)$ ▷ Evaluate in Parallel
  $S_i \leftarrow \arg\max_{S \subset S'_i, |S|=b} \sum_{s \in S} E_i(s)$
end for
return $\mathrm{G}(p_\theta, \arg\max_{s \in S_\mathcal{T}} E_\mathcal{T}(s), 1)$

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.

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 1.

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

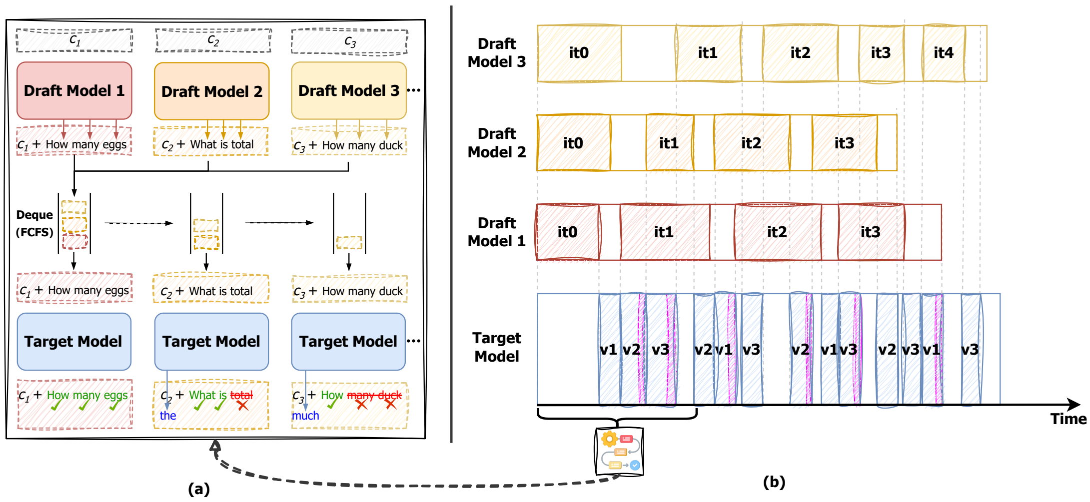

Figure 3: (a) The scenario where the target model manages the verification of target models at the beginning; (b) Overall scheduling diagram for one target model and three draft models. , , represent Draft Model 1, Draft Model 2, Draft Model 3, respectively. , , denotes the execution times of drafting 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.

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 shown in Figure 3 (a), three draft models initiate simultaneously sampling when the queue Q is initially empty. In the subsequent stage, 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 . While the target model M_t is verifying the tokens of other draft models, each draft model is generating its own tokens. In this way, we can fully 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 our proposed framework. 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 maximizing the utilization of the target model and minimizing the overall execution latency.

Inspired by the operating system management of process scheduling, which utilizes the First-Come-First-Serve (FCFS) scheduling policy for all requests, 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 deque.

As depicted in Figure 3 (a), when the deque Q is not empty, a sequence of draft tokens is dequeued in a 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. This approach ensures fairness and prevents starvation for all small draft models, avoiding prolonged wait times for those who complete the drafting phase earlier. 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.

The technical principle of SEED is inspired by the operation system schedule. We present the detailed analogy between the operation system scheduling with SEED in Appendix A.4.

4.3 Algorithm

The core acceleration mechanisms of SEED, which combines speculative scheduled execution with the rounds-scheduled strategy, is presented in Algorithm 2.

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_D, ensures each draft process waits for verification before proceeding with new drafts. At the initial stage, each element in the draft label map γ_D is set to 1, indicating all draft models can perform drafting. After completing the verification of a draft model, the corresponding label in γ_D 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

All experiments are conducted on a single NVIDIA RTX A100 80GB GPU.

5.1 Datasets

Three widely used reasoning and planning datasets are chosen for our experiments to validate the speedup performance of our proposed framework. For mathematical reasoning, GSM8K (Cobbe et al., 2021) is a dataset comprising high-quality grade-school math word problems that require multi-step reasoning. To assess the effectiveness of creativity and planning task, we leverage the Creative Writing dataset (Yao et al., 2024), a task 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. This task is open-ended and exploratory, posing significant challenges to creative thinking and high-level planning. To better demonstrate the speedup performance of our proposed SEED in solving more complex planning problems, we select the Blocksworld dataset (Valmeekam et al., 2023).

Specifically, we utilize 1319 samples from the GSM8K test set, 100 random samples from the Creative Writing dataset following (Yao et al., 2024), and 145 samples from the Blocksworld 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. We consider AR, SD, MCSD 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 different 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.

The selection of baselines will be discussed in Appendix A.1.

5.3 Setup

For comparison with standard draft-target speculative decoding (Leviathan et al., 2023) and MCSD,

Dataset	Methods	Tree Depth	Base k_{config}	Base Speedup	Tree Attention k_{config}	Tree Attention Speedup
Creative Writing	AR	2	-	1×	-	1×
	SD	2	(1,1,1)	1.05×	-	-
	MCSD	2	(1,1,1)	1.16×	(2,2,1)	1.40×
	SEED(ours)	2	(1,1,1)	1.18×	(2,2,1)	1.66×
	SD	2	(1,1,1,1)	1.11×	-	-
	MCSD	2	(1,1,1,1)	1.13×	(4,2,1,1)	1.47×
	SEED(ours)	2	(1,1,1,1)	**1.26**×	(4,2,1,1)	**1.71**×
GSM8K	AR	4	-	1×	-	1×
	SD	4	(1,1,1)	1.05×	-	-
	MCSD	4	(1,1,1)	1.09×	(2,2,1)	1.14×
	SEED(ours)	3	(1,1,1)	1.13×	(2,2,1)	1.21×
	SD	4	(1,1,1,1)	1.17×	-	-
	MCSD	4	(1,1,1,1)	1.20×	(4,2,1,1)	1.27×
	SEED(ours)	4	(1,1,1,1)	**1.24**×	(4,2,1,1)	**1.43**×
Blocksworld	AR	7	-	1×	-	1×
	SD	7	(1,1,1,1)	1.06×	-	-
	MCSD	7	(1,1,1,1)	1.10×	(2,2,1,1)	1.16×
	SEED(ours)	7	(1,1,1,1)	1.13×	(2,2,1,1)	1.25×
	SD	7	(1,1,1,1,1)	1.12×	-	-
	MCSD	7	(1,1,1,1,1)	1.17×	(8,2,1,1,1)	1.36×
	SEED(ours)	7	(1,1,1,1,1)	**1.19**×	(8,2,1,1,1)	**1.39**×

Table 1: Speedup performance of our proposed SEED and baselines. All speedups are relative to the vanilla AR. The best results among all methods are in **bolded**.

we conduct speculative decoding with tree attention using LLaMA-2-Chat-7B[4] as the target model following Chen et al. (2023b). Since there is no official release of a smaller model in the LLaMA suite, we use a pre-trained 160M model LLaMA-160M-Chat[5] with the same tokenizer as the draft model. To validate the extensibility of our framework, we also conducted experiments using the QWen2 suite (Bai et al., 2023). Detailed information can be found in Appendix A.2. We perform a BFS algorithm as the search strategy for all tasks. For Creative Writing, following the ToT setup (Yao et al., 2024), the tree depth is 2. For GSM8K, we simplify by setting the tree depth to 4. For the more complex Blocksworld, we set the tree depth to 7 to allow for more iterations. The detailed prompts for the Thought Generator and the State Evaluator, along with the ToT setup for each task are provided in Appendix C.

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: Creative Writing, GSM8K, and Blocksworld. The Tree Depth suggests that the operations with varying levels of complexity or iterations, with deeper trees potentially representing more complex calculations or decision-making processes. The Base setting indicates traditional single sampling at each position of the draft sequence, while the Tree Attention represents sample multiple candidate tokens at each position and verifying leveraging tree attention which details in Section 3.2. For instance, when k_{config} is set to (2,2,1), it indicates the Tree Attention method: during each draft phase, a group of $k = 3$ tokens is generated, with the first two positions each sampling 2 candidates, and the third position sampling 1. The illustration of this configuration is presented in Figure 6. If each element in k_{config} is 1, the Base setting is applied. A greater number at each position in k_{config} signifies that more candidates, generally yield higher speedups.

In the Creative Writing dataset with a reasoning tree depth of 2, the best performance was achieved with a speedup performance of 1.26× in the base setting and 1.71× using tree attention. This remarkable improvement may be attributed to the fine-tuning of the draft model LLaMA-160M-Chat

[4] https://huggingface.co/meta-llama/Llama-2-7b-chat-hf
[5] https://huggingface.co/Felladrin/Llama-160M-Chat-v1

Component	Tree Attention	α	Speedup
Thought Generator	✗	0.37	1.32×
	✓	0.41	1.51×
State Evaluator	✗	0.23	1.10×
	✓	0.35	1.35×

Table 2: The speedup performance on GSM8K of the two main components of SEED. The average **acceptance rate** is represented as α.

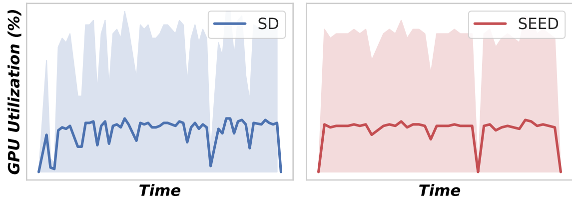

Figure 4: The comparison visualization of GPU utilization between the vanilla SD (on the left part) and the proposed SEED (on the right part) over the 120-second period.

on this specific corpus (Felladrin, 2024), resulting in a higher acceptance rate and improved speedup performance.

Across all datasets, SEED, consistently outperforms the other methods across different settings and configurations in terms of speedup, achieving the highest speedup. Specifically, it achieves an average speedup of 1.2× in the base setting and 1.5× in the candidate setting, respectively. This indicates that SEED is more efficient in inferencing these tasks.

6.2 Ablation Study

SEED accelerate two components in reasoning tree construction, which are the Thought Generator (TG) and the State Evaluator (SE). Table 2 presents the speedup performance of two main components of the SEED method on the GSM8K dataset. For both components, the application of the tree attention leads to higher acceptance rates and greater speedup. When the tree attention is not applied, the TG component has an acceptance rate (α) of 0.37 and a speedup of 1.32×. With the tree attention, both the acceptance rate and the speedup increase, to 0.41 and 1.51× respectively. Similar to TG, the SE component shows improved performance with the tree attention. Without it, α is 0.23 and the speedup is 1.10×; with it, these values rise to 0.35 and 1.35×, respectively. 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 TG is relatively simpler compared to the SE component. The proficiency between the target model and draft model may be more closely aligned in the proposal of thoughts, compared to decision-making capability.

6.3 Analysis of GPU Utilization

In the paradigm of speculative decoding, all model parameters, including those of both target and draft models, are initially moved to GPU memory. When the draft model is in drafting processing, the target model remains idle. The utilization rate of the target model is low when the acceptance rate is relatively high. To address this limitation, SEED introduces parallel draft models to fully involve the target model in the verification phase.

We recorded GPU utilization over the same durations for the SD and the proposed SEED to visualize the effectiveness of parallel drafting. As depicted in Figure 4, the left part illustrates the GPU utilization of SD shows intermittent fluctuations, primarily due to the target model being idle when the drafting process. In contrast, the SEED process, shown in the right part, exhibits more stable GPU utilization, attributed to the continuous engagement of the target model in the verification phase. This demonstrates that our method SEED effectively leverages the GPU resources by continuously interacting operations between the pre-loaded target model and smaller draft models.

7 Conclusion

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 maximizes the utilization of available computational resources. Our extensive experimental evaluation across three reasoning demonstrates that SEED achieves significant improvements in inference speed, achieving an average speedup of 1.5×.

Limitations

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

KV-cache has emerged as a critical bottleneck by growing linearly in size with the sequence length. Our frameworks introduce parallel drafting, involving $n-1$ additional drafting models, which inherently necessitates the addition of an equivalent number of KV caches. Given the increase attributed to small draft models (168M) is relatively minimal, we do not optimize the management of the KV cache in this work. 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).

This study focuses solely on optimizing the inference speed of the tree-crafting process for the TSB reasoning task and does not optimize the search speed for these tasks.

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.

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. *arXiv e-prints*, pages arXiv–2401.

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*.

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, et al. 2024. Layer skip: Enabling early exit inference and self-speculative decoding. *arXiv e-prints*, pages arXiv–2404.

Felladrin. 2024. Llama-160m-chat-v1.

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.

Siqi Kou, Lanxiang Hu, Zhezhi He, Zhijie Deng, and Hao Zhang. 2024. Cllms: Consistency large language models. *arXiv preprint arXiv:2403.00835*.

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. *arXiv e-prints*, pages arXiv–2401.

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.

Junki Park, Hyunsung Yoon, Daehyun Ahn, Jungwook Choi, and Jae-Joon Kim. 2020. Optimus: Optimized matrix multiplication structure for transformer neural network accelerator. *Proceedings of Machine Learning and Systems*, 2:363–378.

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.

Ziteng Sun, Ananda Theertha Suresh, Jae Hun Ro, Ahmad Beirami, Himanshu Jain, and Felix Yu. 2024. 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.

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.

Heming Xia, Tao Ge, Peiyi Wang, Si-Qing Chen, Furu Wei, and Zhifang Sui. 2023. Speculative decoding: Exploiting speculative execution for accelerating seq2seq generation. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, pages 3909–3925.

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.

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.

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.

Wei Zhong and Manasa Bharadwaj. 2024. S3d: A simple and cost-effective self-speculative decoding scheme for low-memory gpus. *arXiv e-prints*, pages arXiv–2405.

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*.

M_t	**Methods**	k_{config}	**Speedup**
QWen2-1.5B	AR	-	1×
	SD	(1,1,1,1)	1.19×
	MCSD	(1,1,1,1)	1.20×
	SEED	(1,1,1,1)	**1.25**×
QWen2-7B	AR	-	1×
	SD	(1,1,1)	1.32×
	MCSD	(1,1,1)	-
	SEED	(1,1,1)	**1.40**×

Table 3: The speedup performance on Creative Writing dataset of SEED within using QWen2-0.5B as M_d. The result of MCSD using QWen2-7B as M_t is not reported because QWen2-0.5B and QWen2-7B do not have the same tokenizer, making speculative sampling with a consistent vocabulary impossible. The results of SD and SEED using Qwen2-7B as M_t employ naive sampling.

A.2 Extensibility

LLM 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 QWen2 suite[6]. Specifically, we use QWen2-0.5B-Instruct[7] as the draft model and use QWen2-1.5B-Instruct[8] or QWen2-7B-Instruct[9]as the target model. The results are presented in Table. 3. 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).

Search Algorithm in ToT 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.

A.3 Task Performance

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

[6] https://qwenlm.github.io/zh/blog/qwen2/
[7] https://huggingface.co/Qwen/Qwen2-0.5B-Instruct
[8] https://huggingface.co/Qwen/Qwen2-1.5B-Instruct
[9] https://huggingface.co/Qwen/Qwen2-7B-Instruct

Methods	Training-free	Lossless	SD Type	Extra-knowledge-free	Speedup
Vanilla AR	✓	✓	-	✓	✗
Speculative Decoding (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.

the AR with QWen2-7B and SEED with the aforementioned QWen2 suite using QWen2-0.5B and QWen2-7B, and found that the performance difference was within $\pm 1.5\%$, which is acceptable and substantiates that the performance is effectively **lossless**.

A.4 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 5, each component in SEED can be mapped to a corresponding component in the operating system scheduler. Next, we will elaborate on each component individually.

- The rounds-scheduled execution in SEED corresponds to the process scheduling in OS. Both use an FCFS deque 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.

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.

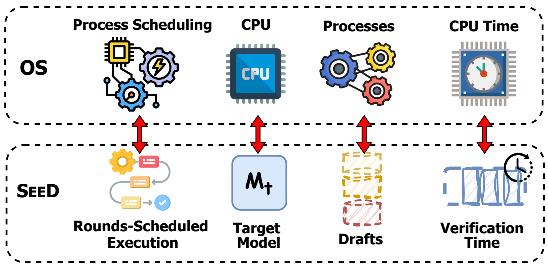

Figure 5: Analogy between the Operation System scheduler with our proposed SEED.

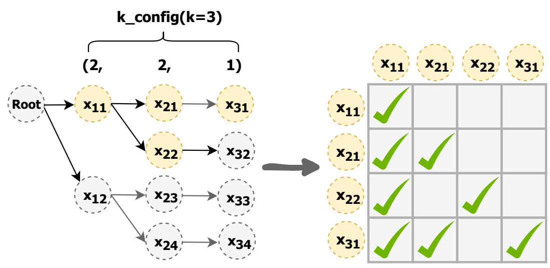

Figure 6: The tree attention used in SEED, multiple tokens in single sequence concurrently are processed. *Root* indicates previous tokens. ✓ indicates where attention is present, while the rest are masked. For simplicity, we only visualize the tree attention mask of tokens in yellow colors.

B Details of Tree Attention

Figure 6 illustrates a case of tree attention with a configuration of $k_{\text{config}} = (2, 2, 1)$.

C 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 encourage diverse thought generation in all tasks, we set the generation temperature as 1 for the LLaMA2 and QWen2 suite models.

The tot setup of the three tasks SEED utilized is as follows:

- **Creative Writing**: We build a reasoning tree with a depth 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 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 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 parts in prompts are required for LLM completion.

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.

Algorithm 2 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, verify phase $\mathcal{E}$ in verification, resampling phase $\mathcal{R}$ in verification, 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 deque Q and a draft label map γ[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**
    $\hat{\mathcal{X}}_i[1:k] \leftarrow p_{d_i}(M_{d_i}, c_i, \hat{\mathcal{X}}_i[1:\mathcal{L}_i], k)$ ▷ Generate k draft tokens
    $Q \leftarrow \hat{\mathcal{X}}_i[1:k]$ ▷ 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**
    $\hat{\mathcal{X}}_i[1:k] \leftarrow$ deque(Q) ▷ Dequeue a group of draft tokens (FCFS)
    $t_1, \cdots, t_k \leftarrow \mathcal{E}(M_t, c_i, \hat{\mathcal{X}}_i[1:k])$ ▷ Verify a group of draft tokens
    for $j = 1$ **to** k **do**
      if t_j is acceptance **then**
        $\hat{\mathcal{X}}_i[\mathcal{L}_i + 1] \leftarrow \hat{x}_j$ and $\mathcal{L}_i \leftarrow \mathcal{L}_i + 1$
      else
        $\hat{\mathcal{X}}[\mathcal{L}_i + 1] \leftarrow \mathcal{R}(M_t, c_i, \hat{\mathcal{X}}_i[1:\mathcal{L}_i])$ and $\mathcal{L}_i \leftarrow \mathcal{L}_i + 1$
        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 $[response_1, \ldots, response_n]$