Accelerated Test-Time Scaling with Model-Free Speculative Sampling

Woomin Song[1,†], **Saket Dingliwal**[2], **Sai Muralidhar Jayanthi**[2],
Bhavana Ganesh[3,‡], **Jinwoo Shin**[1], **Aram Galstyan**[2], **Sravan Babu Bodapati**[2]
[1]KAIST [2]Amazon AGI [3]AirSignal

Abstract

Language models have demonstrated remarkable capabilities in reasoning tasks through test-time scaling techniques like best-of-N sampling and tree search. However, these approaches often demand substantial computational resources, creating a critical trade-off between performance and efficiency. We introduce STAND (STochastic Adaptive N-gram Drafting), a novel model-free speculative decoding approach that exploits the inherent redundancy in reasoning trajectories to achieve significant acceleration without compromising accuracy. Our analysis shows that reasoning paths frequently reuse similar reasoning patterns, enabling efficient model-free token prediction without requiring separate draft models. By introducing stochastic drafting and preserving probabilistic information through a memory-efficient logit-based N-gram module, combined with optimized Gumbel-Top-K sampling and data-driven tree construction, STAND significantly improves token acceptance rates. Extensive evaluations across multiple models and reasoning tasks (AIME-2024, GPQA-Diamond, and LiveCodeBench) demonstrate that STAND reduces inference latency by 60-65% compared to standard autoregressive decoding while maintaining accuracy. Furthermore, STAND consistently outperforms state-of-the-art speculative decoding methods across diverse inference patterns, including single-trajectory decoding, batch decoding, and test-time tree search. As a model-free approach, STAND can be applied to any existing language model without additional training, making it a powerful plug-and-play solution for accelerating language model reasoning.

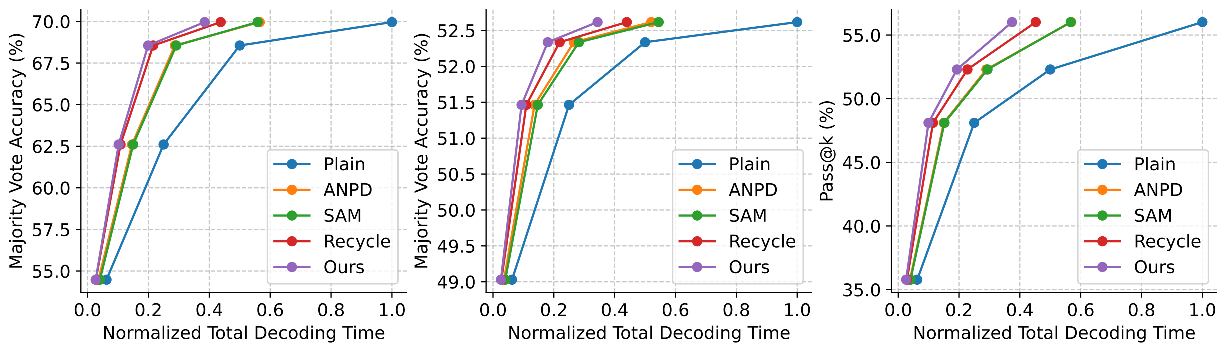
1 Introduction

Test-time scaling has emerged as a prominent paradigm for enhancing the performance of language models by allocating additional computational resources during inference (Snell et al., 2024). This includes generating long sequences of thoughts though Large Reasoning Models (LRMs) (Muennighoff et al., 2025), multi-sampling approaches like best-of-N sampling and majority voting that generate multiple independent outputs to select the most promising one (Wang et al., 2022), as well as iterative methods like tree search and sequential refinement that allow models to progressively improve their reasoning process (Uesato et al., 2022). While these methods demonstrate the potential for significant accuracy improvements through increased computation, they often demand substantial computational resources due to the large number of tokens that need to be generated.

Recent research has focused on reducing the high computational costs of test-time scaling and reasoning approaches (Sui et al., 2025). Some work has explored training with length-based rewards to generate more concise outputs (Aggarwal and Welleck, 2025; Qu et al., 2025), while other approaches use combinations of small and large models to distribute the workload efficiently (Liao et al., 2025; Yang et al., 2025). However, these efficiency-focused methods typically face a fundamental trade-off. While they reduce computational costs, they tend to sacrifice some accuracy compared to more exhaustive approaches, as using fewer samples or cutting short the exploration process often leads to lower performance.

This raises a crucial question: How can we improve the efficiency of test-time scaling and reasoning methods without compromising their accuracy? To address this challenge, we turn our attention to speculative decoding (SD), which offers a promising solution for lossless acceleration of language model inference. Speculative decoding accelerates language model inference by using a smaller "draft" model to predict tokens, which are then verified by the larger target model (Leviathan

[†] Work done during an internship at Amazon. [‡] Work done at Amazon.



(a) AIME-2024 (b) GPQA-Diamond (c) LiveCodeBench

Figure 1: **Scaling curve with speculative decoding.** We report the scaling curve describing how the task performance improves with respect to the total decoding time. Keeping simple auto-regressive decoding total time as 1, we also report the scaling curves for different model-free SD methods. We report the reward-weighted majority voting accuracy for AIME-2024 and GPQA-Diamond, and pass@k for LiveCodeBench, where k is the total number of generated sequences generated at a given point. All measurements are made on a single A100 GPU with DeepSeek-R1-Distill-Qwen-7B.

et al., 2023). With appropriate verification strategies (Chen et al., 2023a), SD can speed up the auto-regressive decoding process of large language models while preserving their output distribution.

A key observation in LRMs is the significant repetition of token sequences across different reasoning paths. When models are performing chain-of-thought reasoning (Snell et al., 2024) or exploring multiple solutions (Wang et al., 2022; Xie et al., 2024), they frequently reuse similar expressions, logical deductions, and reasoning patterns.

This redundancy presents an opportunity for model-free speculative decoding (Saxena, 2023; Ou et al., 2024). Unlike model-based approaches that rely on neural networks as drafters (Li et al., 2024c; Cai et al., 2024), model-free methods can leverage patterns from previous generations to construct drafts. This makes them particularly well-suited for exploiting cross-trajectory information. Our experiments confirm this approach's effectiveness, demonstrating improved efficiency as the number of reasoning trajectories increases.

To fully leverage the power of model-free speculative decoding for reasoning tasks, we propose **STAND (STochastic Adaptive N-gram Drafting)**. Our approach is motivated by two key observations: First, existing model-free approaches have primarily focused on greedy decoding, leaving the potential benefits of sampling largely unexplored. Second, our experimental analysis demonstrates that stochastic drafting (i.e. sampling draft tokens from the draft probability distribution) significantly improves token acceptance rates. Building on these insights, STAND introduces three key innovations: (1) a memory-efficient logit-based N-gram module that preserves probabilistic information for better stochastic drafting, (2) an optimized sampling strategy using Gumbel-Top-K for efficient token selection, and (3) a data-driven approach to draft tree construction that balances efficiency with effectiveness. Combined, these techniques significantly enhance the speculative decoding performance in the context of test-time scaling, where sampling and diverse trajectory exploration are crucial.

Our extensive evaluations demonstrate STAND's effectiveness across diverse reasoning tasks (math, science, and coding) and model scales. As highlighted in Figure 1, STAND's benefits become more pronounced as the number of reasoning trajectories increases. With best-of-16 sampling for optimal accuracy, STAND reduces inference latency by 60–65% compared to standard autoregressive decoding while maintaining performance. Moreover, STAND outperforms state-of-the-art speculative decoding methods by 14–28% in throughput, establishing an efficient drafting strategy for reasoning tasks.

Furthermore, STAND consistently achieves the best throughput across multiple inference patterns, reducing the inference latency by 58% in single-trajectory decoding, 30% in batch decoding, and 61% in test-time tree search, compared to standard autoregressive decoding. As a model-free speculative decoding approach, STAND accomplishes all these achievements without requiring any additional drafter model, or fine-tuning the target model, being able to be used in plug-and-play manner to any existing LRMs.

2 Related Works

Test-time scaling and efficiency. Test-Time Scaling (TTS) has emerged as a prominent strategy to enhance problem-solving capabilities during inference without model retraining (Snell et al., 2024; Muennighoff et al., 2025; Wang et al., 2022; Uesato et al., 2022; Xie et al., 2024). Generating long chain-of-thoughts or sampling multiple sequences have consistently showcased higher accuracy in many complex tasks like math, science and coding (Wei et al., 2022; Cobbe et al., 2021; Chen et al., 2023b). However, the computational cost of TTS remains a critical bottleneck for their practical use. Recent work has explored optimizing inference using adaptive thinking lengths, cascading models of different sizes, length penalties during training, and budget-constrained decoding (Aggarwal and Welleck, 2025; Qu et al., 2025; Liao et al., 2025; Li et al., 2024b; Wan et al., 2024), yet the fundamental trade-off between accuracy gains and costs persists. Our method aims to accelerate reasoning while ensuring no performance degradation.

Speculative decoding. SD have been shown to accelerate Large Language Model (LLM) inference without any loss in the accuracy (Leviathan et al., 2023). The approach typically involves a smaller "draft" model proposing candidate token sequences for parallel verification by the larger "target" model. If the tokens align with the target model's output distribution, they are "accepted", resulting in more than one token being produced in a single forward pass of the LLM. Various compute-efficient drafting strategies have been proposed in the literature to increase the chances of acceptance. Neural draft architectures have evolved from simple, smaller LMs (Leviathan et al., 2023; Choi et al., 2025) to sophisticated self-drafting approaches (Cai et al., 2024; Li et al., 2024d; Cheng et al., 2024)). Although the use of light-weight model-free drafters based on n-grams (Li et al., 2024a; Somasundaram et al., 2024; Hu et al., 2024; Oliaro et al., 2024; Geva et al., 2023; Ou et al., 2024; Saxena, 2023) has been explored previously for generic tasks, we revisit them in the context of LRMs. While these approaches limit themselves to deterministic n-gram based lookups as draft sequences, we highlight the significance of stochastic drafting with logit information of previously generated n-grams for reasoning in our proposed method. To further boost SD performance, tree drafting was proposed where multiple draft token predictions are organized in a tree structure, enabling efficient parallel verification through a specialized tree attention mask (Miao et al., 2023; Li et al., 2024d). Methods like Eagle-2 (Li et al., 2024c) even used dynamic tree layout choices for SD. Extending these existing methods, we additionally propose a computationally efficient data-driven offline tree optimization method for our lightweight model-free drafting method for LRMs.

Other approaches in literature that tie SD with LRMs include Speculative Thinking (Hu et al., 2025), SpecReason (Pan et al., 2025), Reward-guided SD (Liao et al., 2025). However, they do not maintain the lossless nature of SD and hence can also be used in combination with our work. A contemporary work (Li et al., 2025) have explored the importance of model-free n-gram based drafting for multi-sample inference, but did not showcase any practical speedup. We extend their findings with our novel model-free stochastic drafting, and showcase a comparative analysis with existing methods through our extensive experimentation.

3 Motivation

3.1 N-gram overlap analysis

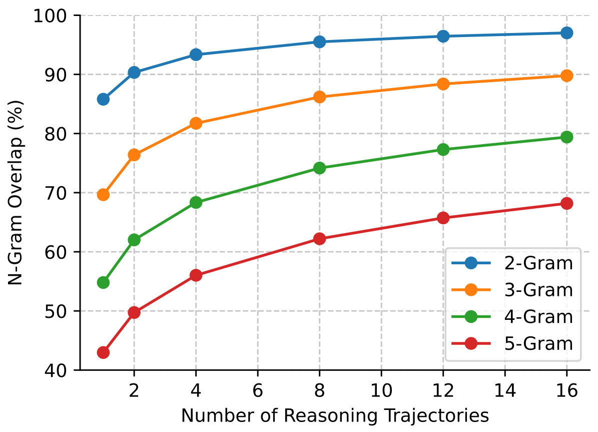


Figure 2: **N-gram overlaps across reasoning trajectories.** We report the N-gram overlaps across different number of reasoning trajectories, generated by DeepSeek-R1-Distill-Qwen-7B on AIME-2024. The overlap is defined as the percentage of the N-grams that appear twice or more in the k reasoning trajectories, counting duplicates multiple times. We observe high n-gram overlaps across reasoning paths, presenting an opportunity for faster drafting.

To assess the degree of redundancy in reasoning trajectories, we conducted a comprehensive analysis of n-gram overlap patterns across multiple solutions generated by the DeepSeek-R1-Distill-

Qwen-7B model on the AIME-2024 dataset. Figure 2 illustrates our findings, depicting the overlap rates for n-grams ranging from bigrams to 5-grams across varying numbers of reasoning trajectories.

The results reveal a substantial level of repetition in token sequences. Notably, we observed that up to 97% of bigrams and 80% of 4-grams recur across 16 distinct reasoning trajectories. Even when considering only two trajectories, over 90% of bigrams are repeated. This high degree of overlap suggests a significant probability that any given n-gram generated by the model has likely appeared in a previous trajectory.

These findings present a compelling opportunity for the development of an efficient drafting strategy. By leveraging this inherent redundancy, we can implement a straightforward approach where previously generated n-grams are proposed as draft sequences, potentially leading to significant improvements in computational efficiency without compromising the chance of acceptance of the generated draft. This presents a key motivation for our proposed method STAND.

3.2 Effectiveness of stochastic drafting

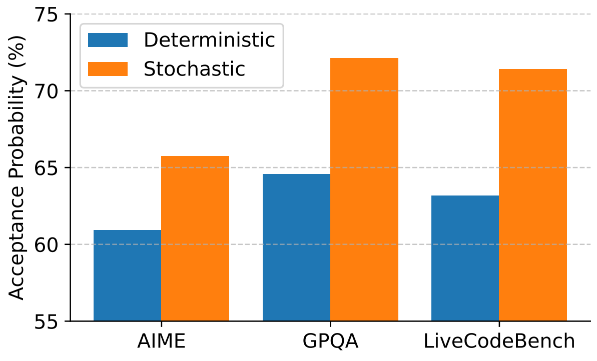


Figure 3: **Deterministic vs. stochastic drafting.** We report the acceptance probability of a token, given a draft tree with depth 1 and width 3. Measurements are done using DeepSeek-R1-Distill-Qwen-7B model, and the draft tree is constructed using the N-gram module in STAND.

In contrast to traditional generation approaches that rely on greedy decoding, LRMs typically employ sampling-based generation strategies to produce multiple diverse solution trajectories, making the choice of drafting strategy particularly crucial. In speculative sampling (Chen et al., 2023a), given a target distribution $p(x)$ and draft distribution $q(x)$, the speculative sampling procedure operates by first sampling $x \sim q(x)$. The sampled token is accepted if $q(x) \leq p(x)$. Otherwise, when $q(x) > p(x)$, the token is rejected with probability $1 - \frac{p(x)}{q(x)}$ and resampled from an adjusted distribution $p'(x) = \text{norm}(\max(0, p(x) - q(x)))$. This procedure guarantees that the final output distribution matches the target distribution $p(x)$, for any drafting distribution $q(x)$.

One can choose the drafting strategy to be deterministic or stochastic. In the former, $q(x)$ is treated as a one-hot vector where $q(x_{\text{draft}}) = 1$ for the most probable token x_{draft} and $q(x) = 0$ for all other x. For speculative sampling, this means the drafted token is accepted with $p(x_{\text{draft}})$, which can be particularly low when the target model is uncertain about its prediction. In contrast, stochastic drafting generates drafts through sampling from a probability distribution. Aligning this draft distribution with the target can significantly boost the chances of acceptance.

In generic greedy decoding setups where this choice does not matter, existing model-free SD methods (Ou et al., 2024; Hu et al., 2024; Saxena, 2023) do not store any probability distribution with the n-gram lookup-based drafters. Eagle-2 (Li et al., 2024c) also uses deterministic drafting for better compatibility with their dynamic tree construction logic. However, for LRMs where sampling plays a key role in generation, we showcase that this choice plays a pivotal role in acceptance probability of the draft sequence. As shown in Figure 3, this fundamental difference leads to 5%, 7% and 8% higher acceptance probabilities for stochastic drafting compared to deterministic drafting across different reasoning tasks i.e. AIME, GPQA and LiveCodeBench respectively. These experimental findings motivated us to find effective ways to compute draft model probabilities in STAND, that aligns well with the probability distributions of LRMs from which the multiple trajectories are sampled.

4 STAND

In this section, we present the details of STAND. In Section 4.1, we propose a memory- and compute-efficient approach to construct the logit-based N-gram module. Then in Section 4.2, we illustrate how to use the N-gram module as a drafter for stochastic sampling, together with several optimization techniques that further improve performance.

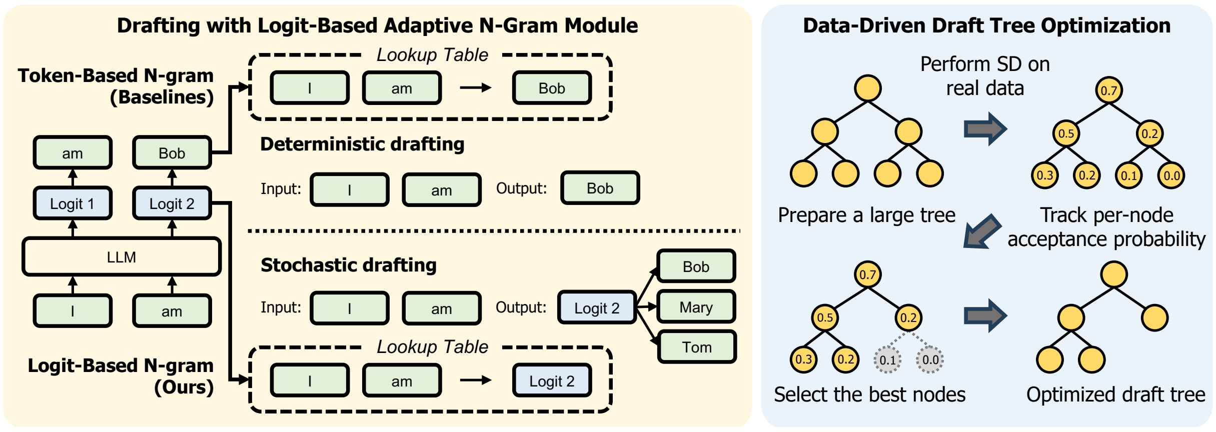


Figure 4: **Overview of STAND.** (Left) The N-gram module stores logits instead of discrete tokens, enabling stochastic drafting. When the language model generates "I am Bob", we store the probability distribution over the next token rather than just the sampled token. (Right) Data-driven draft tree optimization: We start with an initial large draft tree, measure node-wise acceptance rates during speculative decoding on real data, and prune to retain the most successful paths.

4.1 Logit-based adaptive N-gram module

Traditional N-gram modules for speculative decoding typically store pairs of N-grams and their corresponding next tokens (Ou et al., 2024). We improve this approach by instead storing the logit distribution from which the next token is sampled. This modification preserves the rich probabilistic information of potential next tokens, enabling more sophisticated stochastic drafting strategies. While existing methods like Token Recycle partially utilize logit information by storing top-k token IDs, they discard valuable probability information that are crucial for stochastic drafting. Like previous works (Saxena, 2023; Ou et al., 2024), we maintain separate lookup tables from unigrams to 4-grams.

Efficient logit approximation. To address the memory overhead associated with storing full logit distributions, particularly for models with large vocabularies, we implement a compressed representation scheme. Our approach maintains only the top-k indices and their corresponding probabilities. When encountering repeated n-grams, we merge distributions by treating non-stored indices as having zero probability and computing a weighted average: for an n-gram seen k times previously, the existing distribution (representing the mean of k occurrences) is weighted by $k/(k+1)$ and the new distribution by $1/(k+1)$. The resulting averaged distribution is then truncated to retain only the top-10 most probable tokens, ensuring constant memory usage while preserving the most relevant probability information for future speculation.

4.2 Drafting with STAND

Stochastic tree drafting. For each position in the draft tree, we predict the next tokens using a multi-level N-gram approach. Following previous works (Saxena, 2023; Ou et al., 2024), we search for matching N-grams in decreasing order of length, from 4-grams down to unigrams, using the first successful match. This lookup returns the top-10 candidate tokens and their corresponding probabilities from our stored distributions. Based on the number of children required at each tree node, we sample k tokens without replacement from these candidates. These sampled tokens then undergo standard speculative sampling verification to ensure draft quality.

Gumbel-Top-K sampling. For efficient stochastic drafting, we replace traditional sequential sampling with a parallel sampling approach based on the Gumbel-Top-K trick (Kool et al., 2019). For each candidate token's log probability ϕ_i, we add Gumbel noise to create a perturbed distribution:

$$\phi_i' = \phi_i - \log(-\log U_i), \quad U_i \sim \text{Uniform}(0,1)$$

Taking the top-k indices from these perturbed values ϕ_i' effectively samples k tokens without replacement in parallel, significantly reducing sampling latency compared to sequential methods.

To further optimize performance, we precompute and cache the Gumbel noise terms rather than generating them during drafting. This cached noise is periodically refreshed when depleted, effectively separating the sampling overhead from drafting. These optimizations further enhance the performance of our stochastic drafting approach.

Draft tree optimization. Tree-based speculative decoding typically uses either dynamic trees constructed during inference or static trees built using heuristics. While dynamic trees offer context-adaptability, they add computational overhead. Conversely, static trees are computationally efficient but may underperform if constructed through heuristics alone.

We address this limitation through a data-driven approach to static tree construction. Our method begins by initializing a large tree with 625 nodes and performing speculative decoding on 30 data samples. During this process, we track which nodes are frequently part of successful speculation paths. We then select the top-80 most effective nodes and reorganize them into a compact tree structure. This empirical approach maintains the computational efficiency of static trees while ensuring the tree structure is optimized based on real-world performance data.

5 Experiments

This section highlights the effectiveness of STAND through extensive experiments. In Section 5.1, we showcase that STAND can significantly speed up generation in multi-trajectory inference. Section 5.2, we highlight that STAND can also be used in single-trajectory inference. In Section 5.3, we extend our evaluations to more diverse inference patterns, namely batch decoding and test-time tree search. Finally in Section 5.4, we perform an ablation study of the components that make STAND effective, followed by an additional analysis of the optimized tree structure.

Experimental setup and baselines. Throughout the experiments, we evaluate the effectiveness of our approach on diverse tasks, including math reasoning (AIME-2024), STEM QA (GPQA-Diamond), and coding (LiveCodeBench). We perform evaluations across different model scales, including DeepSeek-R1-Distill-Qwen-7B and 14B. For all tasks, we generate maximum 32k tokens for the 7B model, and 24k tokens for the 14B model. All sampling is done with temperature 0.6. All measurements are done on a single A100 GPU, with 30 samples per task for efficient experiments.

For model-free baselines, we compare STAND against Prompt Lookup Decoding (PLD, (Saxena, 2023)), Adaptive N-gram Parallel Decoding (ANPD, (Ou et al., 2024)), Token Recycle (Recycle, (Luo et al., 2024)), SAM-Decoding (SAM, (Hu et al., 2024)) and a combination of SAM decoding and Token Recycle, also proposed in the SAM paper. For Static SAM (which is a component of SAM that uses a pre-constructed suffix automation from a datastore), we construct the datastore using 4k samples from the OpenThoughts-114k (Team, 2025) dataset. For all methods involving static draft trees, we apply our tree optimization algorithm using 30 samples from the AIME-2024 dataset unless otherwise stated. We also compare against Eagle-2, as a representative model-based baseline. To enable long context inference, we trained all Eagle models using OpenThoughts-114k dataset, where long samples exceeding 32k tokens were truncated.

To ensure fair comparison, we conducted all experiments on a unified codebase, adapted from Spec-Bench (Xia et al., 2024). In this implementation, all components such as model forwarding and draft verification are shared, and the only differences lie in the drafting algorithms themselves. Furthermore, the draft length was fixed at 80 tokens for all methods, including our proposed approach and the baselines.

Evaluation metrics. We adopt throughput and acceptance length as our main evaluation metrics. Throughput measures the number of tokens generated per second, computed as the total number of generated tokens divided by the total drafting time. The acceptance length quantifies the average number of tokens generated per speculation step. For both metrics, higher values indicate better performance.

5.1 Evaluation on multi-trajectory decoding

In Figure 1 and Table 1, we evaluate STAND's performance in multi-trajectory inference, where we generate multiple candidate answers by sequentially producing k independent reasoning traces and then aggregate the results.

As shown in Figure 1, STAND significantly improves decoding efficiency, achieving equivalent performance to plain decoding in less than 40% the time. Table 1 provides detailed throughput and acceptance length comparisons across methods. STAND not only achieves the highest throughput but also maintains longer acceptance lengths compared to baselines. Importantly, both metrics improve as we increase the number of trajectories, making STAND's speedup advantage more pronounced with increased compute scaling.

Notably, Token Recycle's performance remains

Table 1: **Speculative decoding performance in multi-trajectory reasoning.** We report the average throughput (T) and acceptance length (A) for multi-trajectory test-time scaling scenarios, with different number of reasoning trajectories per problem. We evaluate each model on AIME-2024 (AIME), GPQA-Diamond (GPQA), and LiveCodeBench (LCB). Best results are shown in **bold**.

		4 Trajectories				8 Trajectories				16 Trajectories			
		AIME	GPQA	LCB	Avg.	AIME	GPQA	LCB	Avg.	AIME	GPQA	LCB	Avg.
DeepSeek-R1-Distill-Qwen-7B													
Plain	T	26.63	31.34	27.75	28.57	26.63	31.34	27.75	28.57	26.63	31.34	27.75	28.57
Eagle-2	T	29.91 (x1.12)	31.69 (x1.01)	27.61 (x0.99)	29.74 (x1.04)	29.91 (x1.12)	31.69 (x1.01)	27.61 (x0.99)	29.74 (x1.04)	29.91 (x1.12)	31.69 (x1.01)	27.61 (x0.99)	29.74 (x1.04)
	A	2.21	1.99	2.13	2.11	2.21	1.99	2.13	2.11	2.21	1.99	2.13	2.11
PLD	T	43.93 (x1.65)	50.49 (x1.61)	44.01 (x1.59)	46.14 (x1.61)	44.95 (x1.69)	53.04 (x1.69)	45.08 (x1.62)	47.69 (x1.67)	46.60 (x1.75)	53.47 (x1.71)	46.02 (x1.66)	48.70 (x1.70)
	A	1.78	1.81	1.73	1.77	1.84	1.89	1.79	1.84	1.89	1.96	1.85	1.90
ANPD	T	45.52 (x1.71)	57.39 (x1.83)	46.30 (x1.67)	49.74 (x1.74)	46.40 (x1.74)	58.97 (x1.88)	47.86 (x1.72)	51.08 (x1.79)	47.06 (x1.77)	60.25 (x1.92)	48.81 (x1.76)	52.04 (x1.82)
	A	1.89	1.97	1.88	1.91	1.92	2.03	1.91	1.95	1.96	2.11	1.96	2.01
SAM	T	44.35 (x1.67)	53.21 (x1.70)	45.63 (x1.64)	47.73 (x1.67)	45.64 (x1.71)	55.47 (x1.77)	47.24 (x1.70)	49.45 (x1.73)	47.64 (x1.79)	57.53 (x1.84)	48.92 (x1.76)	51.36 (x1.80)
	A	1.81	1.87	1.85	1.84	1.89	1.96	1.89	1.91	1.97	2.03	1.95	1.98
Recycle	T	61.38 (x2.30)	71.51 (x2.28)	60.62 (x2.18)	64.50 (x2.26)	61.70 (x2.32)	71.55 (x2.28)	60.93 (x2.20)	64.73 (x2.27)	60.86 (x2.29)	71.23 (x2.27)	61.36 (x2.21)	64.48 (x2.26)
	A	2.76	2.73	2.73	2.74	2.77	2.73	2.73	2.74	2.77	2.73	2.74	2.75
SAM+Recycle	T	61.11 (x2.29)	70.43 (x2.25)	62.20 (x2.24)	64.58 (x2.26)	60.66 (x2.28)	69.98 (x2.23)	63.41 (x2.29)	64.68 (x2.26)	60.63 (x2.28)	69.85 (x2.23)	63.39 (x2.28)	64.62 (x2.26)
	A	2.71	2.73	2.68	2.71	2.69	2.74	2.69	2.71	2.68	2.71	2.67	2.69
STAND (Ours)	T	**64.99 (x2.44)**	**83.47 (x2.66)**	**69.70 (x2.51)**	**72.72 (x2.55)**	**66.88 (x2.51)**	**87.02 (x2.78)**	**71.83 (x2.59)**	**75.24 (x2.63)**	**69.15 (x2.60)**	**91.17 (x2.91)**	**74.14 (x2.67)**	**78.15 (x2.74)**
	A	**3.21**	**3.48**	**3.30**	**3.33**	**3.35**	**3.70**	**3.47**	**3.51**	**3.46**	**3.90**	**3.64**	**3.67**
DeepSeek-R1-Distill-Qwen-14B													
Plain	T	17.76	18.16	17.43	17.78	17.76	18.16	17.43	17.78	17.76	18.16	17.43	17.78
Eagle-2	T	25.38 (x1.43)	24.86 (x1.37)	21.89 (x1.26)	24.04 (x1.35)	25.38 (x1.43)	24.86 (x1.37)	21.89 (x1.26)	24.04 (x1.35)	25.38 (x1.43)	24.86 (x1.37)	21.89 (x1.26)	24.04 (x1.35)
	A	2.72	2.44	2.51	2.56	2.72	2.44	2.51	2.56	2.72	2.44	2.51	2.56
PLD	T	24.37 (x1.37)	26.6 (x1.46)	23.36 (x1.34)	24.78 (x1.39)	25.44 (x1.43)	27.36 (x1.51)	23.96 (x1.37)	25.59 (x1.44)	26.35 (x1.48)	28.43 (x1.57)	24.97 (x1.43)	26.58 (x1.49)
	A	1.74	1.82	1.74	1.77	1.84	1.91	1.81	1.85	1.92	2.00	1.88	1.93
ANPD	T	25.74 (x1.45)	28.21 (x1.55)	24.78 (x1.42)	26.24 (x1.48)	26.12 (x1.47)	29.51 (x1.63)	25.63 (x1.47)	27.09 (x1.52)	26.49 (x1.49)	30.62 (x1.69)	26.32 (x1.51)	27.81 (x1.56)
	A	1.87	1.97	1.87	1.90	1.91	2.04	1.93	1.96	1.96	2.13	1.99	2.03
SAM	T	25.22 (x1.42)	28.03 (x1.54)	24.41 (x1.40)	25.89 (x1.46)	26.11 (x1.47)	29.39 (x1.62)	25.37 (x1.46)	26.96 (x1.52)	27.25 (x1.53)	30.59 (x1.68)	26.67 (x1.53)	28.17 (x1.58)
	A	1.78	1.85	1.79	1.81	1.88	1.95	1.87	1.90	1.98	2.06	1.96	2.00
Recycle	T	34.97 (x1.97)	38.99 (x2.15)	34.05 (x1.95)	36.00 (x2.02)	35.06 (x1.97)	38.89 (x2.14)	33.98 (x1.95)	35.98 (x2.02)	35.31 (x1.99)	38.81 (x2.14)	33.96 (x1.95)	36.03 (x2.03)
	A	2.78	2.73	2.72	2.74	2.77	2.73	2.72	2.74	2.77	2.74	2.72	2.74
SAM+Recycle	T	34.81 (x1.96)	38.24 (x2.11)	34.15 (x1.96)	35.73 (x2.01)	35.16 (x1.98)	38.57 (x2.12)	34.19 (x1.96)	35.97 (x2.02)	35.53 (x2.00)	38.99 (x2.15)	34.31 (x1.97)	36.28 (x2.04)
	A	2.70	2.71	2.65	2.69	2.71	2.71	2.66	2.69	2.72	2.71	2.65	2.69
STAND (Ours)	T	**37.56 (x2.11)**	**43.71 (x2.41)**	**38.71 (x2.22)**	**39.99 (x2.25)**	**39.13 (x2.20)**	**46.81 (x2.58)**	**40.45 (x2.32)**	**42.13 (x2.37)**	**40.76 (x2.30)**	**49.11 (x2.70)**	**42.72 (x2.45)**	**44.20 (x2.49)**
	A	**3.16**	**3.42**	**3.29**	**3.29**	**3.28**	**3.63**	**3.47**	**3.46**	**3.42**	**3.86**	**3.65**	**3.64**

flat despite increasing trajectories, unlike other model-free approaches. This limitation likely comes from its lookup table update strategy, which replaces rather than aggregates information from new trajectories. While this approach may offer some drafting speed benefits, STAND's superior and scaling-dependent performance suggests that aggregating historical information is more beneficial than harmful for test-time scaling.

5.2 Evaluation on single-trajectory decoding

While STAND is primarily designed to leverage information across multiple reasoning trajectories, we also evaluate its performance on single-trajectory generation, where the model only produces one long reasoning chain. As shown in Table 2, STAND achieves both the highest acceptance length and throughput in most scenarios, demonstrating its effectiveness even when generating individual solutions.

Table 2: **Single-trajectory evaluations.** We report the throughput (T) and acceptance length (A) for generating a single sequence with DeepSeek-R1-Distill-Qwen-7B and 14B. Best results are shown in **bold**.

		AIME	GPQA	LCB	Avg.
		DeepSeek-R1-Distill-Qwen-7B			
Plain	T	26.63	31.34	27.75	28.57
Eagle-2	T	29.91 (x1.12)	31.69 (x1.01)	27.61 (x0.99)	29.74 (x1.04)
	A	2.21	1.99	2.13	2.11
PLD	T	44.34 (x1.67)	42.84 (x1.37)	43.40 (x1.56)	43.53 (x1.52)
	A	1.72	1.64	1.59	1.65
ANPD	T	46.18 (x1.73)	54.05 (x1.72)	44.79 (x1.61)	48.34 (x1.69)
	A	1.88	1.82	1.80	1.83
SAM	T	40.85 (x1.53)	48.45 (x1.55)	42.92 (x1.55)	44.07 (x1.54)
	A	1.69	1.69	1.80	1.73
Recycle	T	60.61 (x2.28)	71.00 (x2.27)	60.12 (x2.17)	63.91 (x2.24)
	A	2.73	2.71	2.73	2.72
SAM+Recycle	T	61.15 (x2.30)	71.51 (x2.28)	62.78 (x2.26)	65.15 (x2.28)
	A	2.70	2.81	2.69	2.73
STAND (Ours)	T	**61.79 (x2.32)**	**75.39 (x2.41)**	**66.41 (x2.39)**	**67.86 (x2.38)**
	A	**3.07**	**3.05**	**3.01**	**3.04**
		DeepSeek-R1-Distill-Qwen-14B			
Plain	T	17.76	18.16	17.43	17.78
Eagle-2	T	25.38 (x1.43)	24.86 (x1.37)	21.89 (x1.26)	24.04 (x1.35)
	A	2.72	2.44	2.51	2.56
PLD	T	21.82 (x1.23)	24.97 (x1.38)	21.76 (x1.25)	22.85 (x1.28)
	A	1.61	1.64	1.58	1.61
ANPD	T	25.60 (x1.44)	26.40 (x1.45)	23.16 (x1.33)	25.05 (x1.41)
	A	1.76	1.79	1.76	1.77
SAM	T	23.26 (x1.31)	25.38 (x1.40)	22.36 (x1.28)	23.67 (x1.33)
	A	1.63	1.65	1.63	1.64
Recycle	T	33.71 (x1.90)	**38.91 (x2.14)**	33.85 (x1.94)	35.49 (x2.00)
	A	2.77	2.73	2.71	2.74
SAM+Recycle	T	34.35 (x1.93)	37.53 (x2.07)	34.45 (x1.98)	35.44 (x1.99)
	A	2.67	2.72	2.70	2.70
STAND (Ours)	T	**34.52 (x1.94)**	38.71 (x2.13)	**34.86 (x2.00)**	**36.03 (x2.03)**
	A	**2.91**	**3.00**	**2.93**	**2.95**

5.3 Evaluation on diverse inference patterns

We extend our evaluation to more diverse and practical inference patterns, focusing on batch decoding and test-time tree search. In the batch decoding setup, multiple reasoning traces are generated simultaneously, and the N-gram drafters can only leverage the content produced up to the current decoding step. As shown in Table 3, STAND consistently achieves significant speedups even with a batch size of 8, whereas Token Recycle provides only marginal improvements at batch size 4 and even degrades performance at batch size 8.

Table 3: **Batch decoding evaluations.** We report the throughput (T) and acceptance length (A) for batch decoding with DeepSeek-R1-Distill-Qwen-7B. Both speculative decoding approaches use trees optimized with OpenThoughts-114k. Best results are shown in **bold**.

		AIME	GPQA	LCB	Avg.
		Batch Size 4			
Plain	T	89.04	88.32	92.64	90.00
Recycle	T	90.51 (x1.02)	98.41 (x1.11)	89.55 (x0.97)	92.82 (x1.03)
	A	1.77	1.83	1.72	1.77
STAND (Ours)	T	**127.58 (x1.43)**	**134.64 (x1.52)**	**122.09 (x1.32)**	**128.10 (x1.42)**
	A	**2.57**	**2.74**	**2.57**	**2.63**
		Batch Size 8			
Plain	T	111.62	106.23	114.73	110.86
Recycle	T	99.18 (x0.89)	105.69 (x0.99)	99.2 (x0.86)	101.36 (x0.91)
	A	1.66	1.75	1.63	1.68
STAND (Ours)	T	**148.08 (x1.33)**	**154.68 (x1.46)**	**149.41 (x1.30)**	**150.72 (x1.36)**
	A	**2.70**	**2.86**	**2.69**	**2.75**

Table 4: **Test-time tree search evaluations.** We report the throughput (T) and acceptance length (A) for performing Diverse Verifier Tree Search (DVTS) with DeepSeek-R1-Distill-Qwen-7B. Both speculative decoding approaches use trees optimized with OpenThoughts-114k. Best results are shown in **bold**.

		AIME	GPQA	LCB	Avg.
Plain	T	33.35	33.12	32.22	32.90
Recycle	T	71.52 (x2.14)	70.64 (x2.13)	69.47 (x2.16)	70.54 (x2.14)
	A	2.73	2.69	2.74	2.72
STAND (Ours)	T	**82.62 (x2.48)**	**86.93 (x2.62)**	**80.97 (x2.51)**	**83.51 (x2.54)**
	A	**3.55**	**3.71**	**3.51**	**3.59**

Test-time tree search represents another widely used pattern for scaling inference, where the model generates multiple candidate reasoning steps and dynamically selects among them. In Table 4, we evaluate STAND within the Diverse Verifier Tree Search (DVTS, (Beeching et al., 2024)) framework. As the results show, STAND consistently outperforms Token Recycle, achieving an average speedup of 2.54×.

Table 5: **Effect of Stochastic Drafting.** We report the throughput (T) and acceptance length (A) for generating 4 sequences with DeepSeek-R1-Distill-Qwen-7B.

		AIME	GPQA	LCB	Avg.
Plain	T	26.63	31.34	27.75	28.57
Deterministic	T	62.13 (x2.33)	73.67 (x2.35)	63.44 (x2.29)	66.41 (x2.32)
	A	2.94	2.98	2.90	2.94
Stochastic	T	63.44 (x2.38)	81.20 (x2.59)	65.90 (x2.37)	70.18 (x2.46)
	A	3.24	3.56	3.29	3.36
+ Gumbel-Top-K	T	64.99 (x2.44)	83.47 (x2.66)	69.70 (x2.51)	72.72 (x2.55)
	A	3.21	3.48	3.30	3.33

Table 6: **Effect of tree optimization.** Comparison of throughput and acceptance length when generating 4 sequences with DeepSeek-R1-Distill-Qwen-7B on two datasets: AIME-2024 and GPQA-Diamond. We compare two types of static trees: the heuristic trees from Token Recycle and our data-optimized trees, optimized on AIME-2024. Best results are shown in **bold**.

	AIME		GPQA (OOD)	
	Heuristic	Optimized	Heuristic	Optimized
Throughput	59.96	**64.99**	77.32	**83.47**
Acc. Lens	3.17	**3.21**	3.35	**3.48**

5.4 Ablations and analysis

We evaluate key components of STAND through ablation studies and further analysis. Our ablation studies examine the impact of stochastic drafting and the Gumbel-Top-K optimization trick, followed by an investigation of our tree optimization approach. We then analyze the structural characteristics of the optimized trees to better understand the patterns that emerge from our method.

Effect of stochastic drafting. In Table 5, we compare three drafting approaches: deterministic drafting, basic stochastic drafting (using PyTorch's multinomial sampling), and our optimized stochastic drafting with Gumbel-Top-K. For fair comparison, we separately perform tree optimization for determinisic drafting and stochastic drafting. Stochastic drafting consistently achieves higher acceptance lengths across all tasks, resulting in improved throughput compared to deterministic drafting. Our Gumbel-Top-K optimization further improves performance by maintaining similar acceptance lengths while significantly reducing latency, leading to even higher throughput.

Effect of tree optimization. In Table 6, we showcase the effectiveness of our tree optimization technique. We compare the performance of a heuristic tree originally used by Token Recycle (Luo et al., 2024) with our tree, optimized on the AIME-2024 dataset. The results demonstrates that the optimized tree improves performance on both AIME-2024 and GPQA-Diamond, showcasing that the optimization not only works within the same dataset, but also generatlizes to out-of-domain (OOD) tasks.

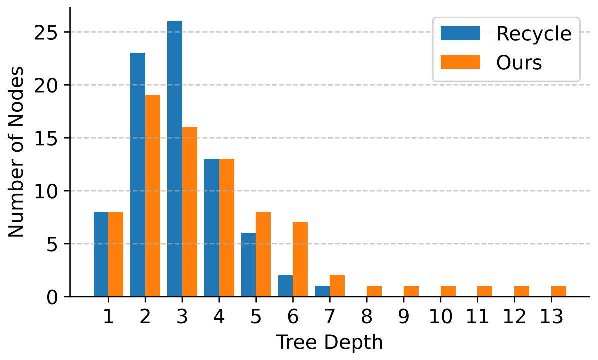


Figure 5: **Structure of the Optimized Tree.** We report the number of nodes at specific tree depths for draft trees optimized for each Token Recycle and STAND. Both trees are optimized on AIME-2024 dataset with DeepSeek-R1-Distill-Qwen-7B.

Tree structure analysis. We analyze how different drafting approaches lead to different optimal tree structures by comparing trees optimized for STAND versus Token Recycle. As shown in Figure 5, the tree optimized for STAND reaches greater depths, extending to 13 levels compared to 7 levels in the Token Recycle-optimized tree. This difference likely stems from STAND's higher acceptance rate, which favors deeper, narrower tree structures under the same tree size budget.

A distinctive feature of STAND's optimized tree is its long tail structure, with single nodes at depths 8 through 13. This pattern suggests the presence of occasional long, deterministic sequences, possibly arising from consistent patterns found across multiple reasoning trajectories.

6 Conclusion

In this work, we introduced STAND, a model-free speculative decoding approach that accelerates language model reasoning while maintaining accuracy. By utilizing reasoning trajectory redundancy and historical logit information, STAND significantly improves throughput over standard auto-regressive decoding and existing alternatives, offering an efficient solution for scaling AI reasoning systems.

Limitations

While STAND demonstrates strong performance across diverse inference patterns, all measurements were conducted using the HuggingFace implementation, which is less optimized than popular serving frameworks such as vLLM or SGLang. Although we expect the benefits of STAND to extend to these optimized frameworks, this has not yet been verified. In addition, the current N-gram lookup operation is implemented in Python, which may introduce a slowdown. Latency could be further reduced with a more optimized implementation of the N-gram module.

Acknowledgements

For WS and JS: This work was supported by Institute for Information & communications Technology Promotion(IITP) grant funded by the Korea government(MSIT) (No.RS-2019-II190075 Artificial Intelligence Graduate School Program (KAIST); No. RS-2024-00509279, Global AI Frontier Lab). This work was also supported by the NIPA(National IT Industry Promotion Agency), through the Ministry of Science and ICT (Hyperscale AI flagship project).

References

Pranjal Aggarwal and Sean Welleck. 2025. L1: Controlling how long a reasoning model thinks with reinforcement learning. *arXiv preprint arXiv:2503.04697*.

Edward Beeching, Lewis Tunstall, and Sasha Rush. 2024. Open models. https://huggingface.co/spaces/HuggingFaceH4/blogpost-scaling-test-time-compute. Hugging Face Blog, Accessed: 2025-09-20.

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

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

Xinyun Chen, Renat Aksitov, Uri Alon, Jie Ren, Kefan Xiao, Pengcheng Yin, Sushant Prakash, Charles Sutton, Xuezhi Wang, and Denny Zhou. 2023b. Universal self-consistency for large language model generation. *arXiv preprint arXiv:2311.17311*.

Yunfei Cheng, Aonan Zhang, Xuanyu Zhang, Chong Wang, and Yi Wang. 2024. Recurrent drafter for fast speculative decoding in large language models. *arXiv preprint arXiv:2403.09919*.

Daewon Choi, Seunghyuk Oh, Saket Dingliwal, Jihoon Tack, Kyuyoung Kim, Woomin Song, Seojin Kim, Insu Han, Jinwoo Shin, Aram Galstyan, and 1 others. 2025. Mamba drafters for speculative decoding. *arXiv preprint arXiv:2506.01206*.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, and 1 others. 2021. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*.

Mor Geva, Tal Schuster, Jonathan Berant, and Omer Levy. 2023. Token recycling: Making llms faster and more data-efficient. *arXiv preprint arXiv:2310.02548*.

Yunhai Hu, Zining Liu, Zhenyuan Dong, Tianfan Peng, Bradley McDanel, and Sai Qian Zhang. 2025. Speculative decoding and beyond: An in-depth survey of techniques. *arXiv preprint arXiv:2502.19732*.

Yuxuan Hu, Ke Wang, Xiaokang Zhang, Fanjin Zhang, Cuiping Li, Hong Chen, and Jing Zhang. 2024. Sam decoding: Speculative decoding via suffix automaton. *arXiv preprint arXiv:2411.10666*.

Langlin Huang, Chengsong Huang, Jixuan Leng, Di Huang, and Jiaxin Huang. 2025. Poss: Position specialist generates better draft for speculative decoding. *arXiv preprint arXiv:2506.03566*.

Wouter Kool, Herke Van Hoof, and Max Welling. 2019. Stochastic beams and where to find them: The gumbel-top-k trick for sampling sequences without replacement. In *International Conference on Machine Learning*, pages 3499–3508. PMLR.

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.

Minghan Li, Xilun Chen, Ari Holtzman, Beidi Chen, Jimmy Lin, Scott Yih, and Victoria Lin. 2024a. Nearest neighbor speculative decoding for llm generation and attribution. *Advances in Neural Information Processing Systems*, 37:80987–81015.

Yiwei Li, Jiayi Shi, Shaoxiong Feng, Peiwen Yuan, Xinglin Wang, Yueqi Zhang, Ji Zhang, Chuyi Tan, Boyuan Pan, Yao Hu, and Kan Li. 2025. Speculative decoding for multi-sample inference. *arXiv preprint arXiv:2503.05330*.

Yiwei Li, Peiwen Yuan, Shaoxiong Feng, Boyuan Pan, Xinglin Wang, Bin Sun, Heda Wang, and Kan Li. 2024b. Escape sky-high cost: Early-stopping self-consistency for multi-step reasoning. *arXiv preprint arXiv:2401.10480*.

Yuhui Li, Fangyun Wei, Chao Zhang, and Hongyang Zhang. 2024c. Eagle-2: Faster inference of language models with dynamic draft trees. *arXiv preprint arXiv:2406.16858*.

Yuhui Li, Fangyun Wei, Chao Zhang, and Hongyang Zhang. 2024d. Eagle: Speculative sampling requires rethinking feature uncertainty. *arXiv preprint arXiv:2401.15077*.

Baohao Liao, Yuhui Xu, Hanze Dong, Junnan Li, Christof Monz, Silvio Savarese, Doyen Sahoo, and Caiming Xiong. 2025. Reward-guided speculative decoding for efficient llm reasoning. *arXiv preprint arXiv:2501.19324*.

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

Xiang Miao, Gabriele Oliaro, Zhen Zhang, Xinyun Cheng, Zeyu Wang, Zheng Zhang, Ruijie Yan, Alvin Zhu, Lei Yang, Xipeng Shi, and 1 others. 2023. Specinfer: Accelerating generative large language model serving with tree-based speculative inference and verification. *arXiv preprint arXiv:2305.09781*.

Niklas Muennighoff, Zitong Yang, Weijia Shi, Xiang Lisa Li, Li Fei-Fei, Hannaneh Hajishirzi, Luke Zettlemoyer, Percy Liang, Emmanuel Candès, and Tatsunori Hashimoto. 2025. s1: Simple test-time scaling. *arXiv preprint arXiv:2501.19393*.

Gabriele Oliaro, Zhihao Jia, Daniel Campos, and Aurick Qiao. 2024. Suffixdecoding: A model-free approach to speeding up large language model inference. *arXiv preprint arXiv:2411.04975*.

Jie Ou, Yueming Chen, and Wenhong Tian. 2024. Lossless acceleration of large language model via adaptive n-gram parallel decoding. *arXiv preprint arXiv:2404.08698*.

Rui Pan, Yinwei Dai, Zhihao Zhang, Gabriele Oliaro, Zhihao Jia, and Ravi Netravali. 2025. Specreason: Fast and accurate inference-time compute via speculative reasoning. *arXiv preprint arXiv:2504.07891*.

Yuxiao Qu, Matthew YR Yang, Amrith Setlur, Lewis Tunstall, Edward Emanuel Beeching, Ruslan Salakhutdinov, and Aviral Kumar. 2025. Optimizing test-time compute via meta reinforcement fine-tuning. *arXiv preprint arXiv:2503.07572*.

Apoorv Saxena. 2023. Prompt lookup decoding.

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*.

Shwetha Somasundaram, Anirudh Phukan, and Apoorv Saxena. 2024. Pld+: Accelerating llm inference by leveraging language model artifacts. *arXiv preprint arXiv:2412.01447*.

Yang Sui, Yu-Neng Chuang, Guanchu Wang, Jiamu Zhang, Tianyi Zhang, Jiayi Yuan, Hongyi Liu, Andrew Wen, Shaochen Zhong, Hanjie Chen, and 1 others. 2025. Stop overthinking: A survey on efficient reasoning for large language models. *arXiv preprint arXiv:2503.16419*.

OpenThoughts Team. 2025. Open Thoughts. https://open-thoughts.ai.

Jonathan Uesato, Nate Kushman, Ramana Kumar, Francis Song, Noah Siegel, Lisa Wang, Antonia Creswell, Geoffrey Irving, and Irina Higgins. 2022. Solving math word problems with process-and outcome-based feedback. *arXiv preprint arXiv:2211.14275*.

Guangya Wan, Yuqi Wu, Jie Chen, and Sheng Li. 2024. Dynamic self-consistency: Leveraging reasoning paths for efficient llm sampling. *arXiv preprint arXiv:2408.17017*.

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc Le, Ed H. Chi, and Denny Zhou. 2022. Self-consistency improves chain of thought reasoning in language models. *ArXiv*, abs/2203.11171.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, and 1 others. 2022. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35:24824–24837.

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*.

Wang Yang, Xiang Yue, Vipin Chaudhary, and Xiaotian Han. 2025. Speculative thinking: Enhancing small-model reasoning with large model guidance at inference time. *arXiv preprint arXiv:2504.12329*.

A Experimental Details

A.1 Initial tree for optimization

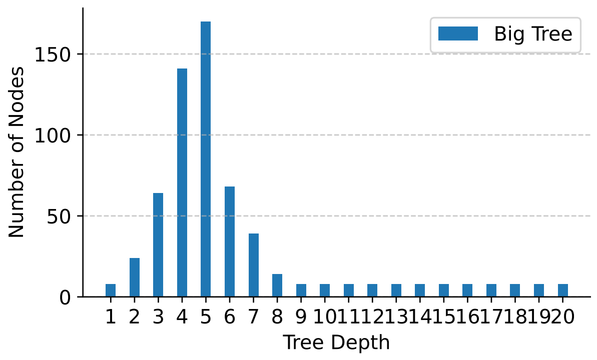


Figure 6: **Structure of the initial tree.** We report the number of nodes at specific tree depths for the initial tree used for tree optimization.

To initialize data-driven tree optimization, we heuristically applied predefined rules that assign the number of child nodes based on node depth and position. In particular, leftmost nodes at each level receive more children, as they are more likely to yield higher acceptance probabilities. The pseudocode for this initialization is shown in Algorithm 1.

As illustrated in Figure 6, the resulting tree has a maximum depth of 20 and 625 total nodes.

A.2 Experiment Details

For the batch decoding experiments, we modified the single-batch inference code to sequentially construct draft trees for all sequences in a batch, and then verify them in parallel. This design choice was made because, unlike model-based drafters that benefit substantially from forwarding multiple samples in parallel (even during the drafting stage), N-gram–based drafters inherently rely on sequential memory lookups. For the DVTS experiments, we adopted the self-evaluation strategy from SpecReason (Pan et al., 2025) to determine which reasoning step to accept.

B Further discussions

B.1 Effect of tree optimization dataset

In this section, we analyze the impact of the dataset used for tree optimization. To evaluate this effect, we optimized the draft tree on the OpenThoughts-114k dataset and then evaluated STAND on AIME-2024, rather than optimizing directly on AIME-2024. As shown in Table 9, STAND maintains high throughput and acceptance length under both configurations, in some cases even outperforming the tree optimized on AIME-2024. These results suggest that STAND is robust to the choice of initialization dataset.

B.2 Remarks on Eagle-2 performance

Table 7: **Eagle-2 acceptance lengths.** We report the acceptance lengths of the Eagle-2 model across different context lengths, using DeepSeek-R1-Distill-Qwen-14B.

Input Length	0-2k	2k-4k	4k-8k	8k-16k	16k-32k
AIME-2024	2.89	2.79	2.68	2.61	2.47
GPQA-Diamond	2.60	2.50	2.38	2.31	2.28
LiveCodeBench	2.76	2.66	2.50	2.38	2.14

Somewhat surprisingly, we observed lower acceptance lengths for Eagle-2 compared to those originally reported in the paper. One potential explanation is the long-context setup used in our evaluation. Whereas most prior Eagle-2 benchmarks focused on short inputs (fewer than 2k tokens), our experiments trained Eagle drafters to handle up to 32k tokens in order to support long-form reasoning. This broader context window may reduce acceptance length as a trade-off.

We also observed that acceptance lengths tend to degrade with longer inputs, as shown in Table 7, which provides another possible explanation for the lower average acceptance lengths in our experiments.

B.3 Performance on non-reasoning model

Table 8: **Non-reasoning model evaluation.** We report the throughput (T) and acceptance length (A) for generating multiple sequences with Qwen2.5-7B-Instruct with batch decoding.

	Batch Size	AIME (T / A)	GPQA (T / A)
Plain	4	96.64 / –	130.00 / –
Recycle	4	115.75 / 2.06	133.53 / 1.99
STAND (Ours)	4	**152.59 / 2.65**	**152.07 / 2.39**
Plain	8	119.32 / –	166.05 / –
Recycle	8	134.40 / 1.95	155.41 / 1.94
STAND (Ours)	8	**187.43 / 2.82**	**198.09 / 2.62**

While our analysis in the main text focused on reasoning models, namely the DeepSeek-R1-Distill-Qwen family, STAND can also be applied to test-time scaling with non-reasoning models. As shown in Table 8, STAND significantly outperforms Token Recycle in both throughput and acceptance length at batch sizes 4 and 8 with Qwen2.5-7B-Instruct.

Algorithm 1 Initial Tree Construction

```
1: Initialize active_list ← {root_node}
2: Initialize children_list ← $\emptyset$
3: num_nodes ← 1
4: for depth = 0 to 19 do
5:     for each node in active_list with index i do
6:         if node.depth = 0 then
7:             n_children ← 8
8:         else if node.depth = 1 then
9:             n_children ← $\max(8 - 2 \cdot \text{node.order}, 1)$
10:        else if node.depth = 2 then
11:            n_children ← $\max\left(\left\lceil \frac{|\text{node.parent.children}|-1}{\text{node.order}\cdot 0.7+1} \right\rceil, 2\right)$
12:        else if node.depth = 3 then
13:            n_children ← $\max\left(\left\lceil \frac{|\text{node.parent.children}|-1}{\text{node.order}\cdot 0.7+1} \right\rceil, 2\right)$
14:        else
15:            n_children ← $\max\left(\left\lceil \frac{|\text{node.parent.children}|-1}{\text{node.order}\cdot 0.7+1} \right\rceil, 0\right)$
16:        end if
17:        if i = 0 then
18:            n_children ← $\max(\text{n\_children}, 3)$
19:        end if
20:        for i_child = 0 to n_children − 1 do
21:            Create new child_node with:
22:                id = num_nodes,
23:                depth = node.depth + 1,
24:                order = i_child,
25:                parent = node
26:            num_nodes ← num_nodes + 1
27:            Append child_node to node.children and children_list
28:        end for
29:    end for
30:    active_list ← children_list
31:    children_list ← $\emptyset$
32: end for
```

Table 9: **Effect of tree optimization dataset.** We report the average throughput (T) and acceptance length (A) for the best baseline and STAND with trees optimized from AIME-2025 and OpenThoughts-114k. We evaluate each model on AIME-2024.

	Single Trajectory		4 Trajectories		8 Trajectories		16 Trajectories	
	T	A	T	A	T	A	T	A
DeepSeek-R1-Distill-Qwen-7B								
Best Baseline	61.15	2.73	61.38	2.76	61.70	2.77	60.86	2.77
STAND w/ AIME Tree	61.79	3.07	64.99	3.21	66.88	3.35	69.15	3.46
STAND w/ OpenThoughts Tree	62.75	3.04	65.33	3.24	68.58	3.35	70.63	3.49
DeepSeek-R1-Distill-Qwen-14B								
Best Baseline	34.35	2.77	34.97	2.78	35.16	2.71	35.53	2.72
STAND w/ AIME Tree	34.52	2.91	37.56	3.16	39.13	3.28	40.76	3.42
STAND w/ OpenThoughts Tree	35.26	2.87	37.41	3.10	39.22	3.26	41.38	3.41

B.4 Effect of tree depth on speed-up

Prior work on model-based speculative decoding has investigated the trade-off between tree depth and speed-up. For example, POSS (Huang et al., 2025) reports that deeper draft trees can increase acceptance length, but at the cost of higher drafting latency, since each additional depth requires an extra forward pass of the draft model. This trade-off directly affects the overall speed-up ratio in model-based methods such as Eagle-2.

By contrast, STAND uses a static, pre-computed tree and performs efficient dictionary lookups rather than model forward passes. As it sequentially performs the dictionary lookup for each tree node, the drafting time does not necessarily increase with the tree depth. Even if lookup times increase slightly, the cost remains negligible compared to model-based approaches, thanks to the efficiency of dictionary lookups.