Efficient RL for LLMs with Dynamic and Online Speculative Decoding

Chao Jin[§], Yinmin Zhong[§], Zili Zhang[§], Yimin Jiang[†], Yibo Zhu[‡]
[§]*School of Computer Science, Peking University* [†]*Anuttacon* [‡]*Stepfun*

Abstract

Reinforcement Learning (RL) has emerged as a core post-training paradigm for LLMs, which involves two main stages: rollout and training. The LLMs first generate new samples in the rollout stage, which are then used to update the model in the training stage. Scaling RL is challenging due to two primary factors: (*i*) The long-tail distribution of output lengths during the rollout stage, which causes GPU idling across parallel rollout instances; (*ii*) the memory-bandwidth-bounded nature of decoding iterations, which results in low GPU utilization for long-tail samples—a bottleneck that does not scale linearly with additional computational resources.

We propose to address these challenges by applying speculative decoding within the rollout stage. Applying speculative decoding to RL remains non-trivial. Algorithmically, RL for LLMs is highly sensitive to the quality of samples generated during rollouts. From a systems perspective, and in contrast to inference serving, the RL rollout stage must contend with long-tailed sample distributions and operate on a target model that is continuously being updated. We extend speculative decoding into a dynamic, online approach that allows for real-time adjustments based on the evolving state of the rollout process. Our method dynamically adapts the speculative decoding strategy in response to the current system load and periodically updates the draft model to maintain alignment with the target model's evolving parameters. We validate that leveraging speculative decoding with our proposed techniques does not compromise the quality of the generated samples and effectively mitigates the aforementioned challenges.

1 Introduction

The adoption of reinforcement learning (RL) represents a pivotal advancement in the methodology for optimizing large language models (LLMs). RL has yielded marked improvements in their capacity for reasoning and has uncovered a previously unrecognized scaling property applicable at inference time [5, 10]. The effectiveness of this approach is demonstrated by its integration into frontier LLMs, including OpenAI's o1 [1] and o3 [2], Claude 3.7 Sonnet [4], and DeepSeek-R1 [5]. These models all leverage RL to establish state-of-the-art capabilities in complex domains such as code generation and quantitative reasoning.

RL for LLMs typically involves two key stages: the rollout stage and the training stage. The RL workflow consists of iteratively executing these two stages. Specifically, during the rollout stage, the LLM model auto-regressively generates multiple samples for each input prompt. These samples are then evaluated using a reward function, which may be based on human feedback or rule-based verifiers. The training stage follows by updating the model parameters according to the reward signals of the generated samples.

Due to the inherent long-tail distribution of output lengths during the rollout stage, RL for LLMs experiences significant resource under-utilization. Towards the end of the rollout stage, as only a small fraction of long-tail samples remain active, a twofold resource inefficiency emerges: the GPUs of replicated workers are left idle, while the GPU of the worker still generating becomes severely under-utilized. This inefficiency is further exacerbated by the long chain-of-thought reasoning process, which is a common practice to enhance the reasoning capabilities of LLMs [5].

Unfortunately, scaling up the number of GPUs for the rollout stage does not efficiently translate into faster sample generation. This is because for the long-tail samples, the auto-regressive decoding generation process with a very small batch size falls short in fully utilizing the compute capabilities of modern GPUs. In each decoding iteration, the generation of the next token requires to load the intermediate states (i.e., key-value cache) of all previous tokens and the model parameters from High-Bandwidth Memory (HBM) to the GPUs' shared memory. This memory-bandwidth-bound operation dominates the computation time of each decoding iteration. The generation time for the long-tail samples is thus limited by the aggregated memory bandwidth of the GPUs. Existing parallelism strategies for LLM inference do not effectively address this challenge. Tensor parallelism parallelizes the computation of a single forward pass across multiple GPUs but incurs significant communication overhead on the critical path. Therefore, it is typically applied within a single GPU node. Data parallelism replicates the model and dispatches samples to the replicas. However, it provides the same aggregated memory bandwidth for each sample, thereby failing to further reduce the generation time when scaling out resources.

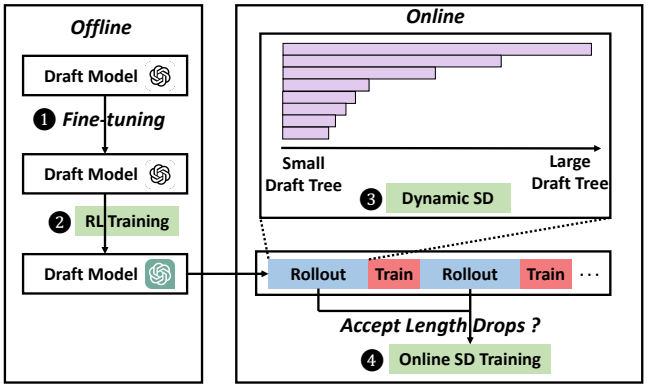

Figure 1: Overview of SpecRL.

Hyperparameter	Value
Batch size	128
Number of samples per prompt	32
Maximum output length	4096
Optimizer	AdamW
Learning rate	10^{-6}
Betas (β_1, β_2)	$[0.9, 0.95]$
Rollout sampling	
`top-k`	-1 (all tokens)
`top-p`	1.0
`temperature`	1.0
Training method	On-policy

Table 1: Training hyperparameters and configurations.

Speculative Decoding (SD) is a natural fit for accelerating the decoding iteration with a small batch size. It employs a small "draft" model to generate a tree of candidate future tokens. These candidates are then validated in a single, parallel forward pass by the original, larger "target" model. By replacing numerous sequential forward passes of the target model with a single verification step and accessing only the small draft models' KV cache and parameters in the decoding iteration, SD yields significant speedups.

However, directly applying existing speculative decoding methods to RL rollout is suboptimal. These methods are primarily designed for online LLM inference serving, a scenario that differs from RL rollout in two critical aspects. First, online inference typically avoids a long-tail distribution of output lengths because batches can be continually filled with incoming requests to maintain high throughput. In contrast, an RL rollout begins with a fixed batch of prompts, and the number of active samples inevitably diminishes as generation progresses. Second, the LLM parameters for online inference are static during serving. The RL rollout process, however, is inherently dynamic, involving continuous updates to the target model's parameters. This dynamic nature introduces a significant challenge for speculative decoding, as the draft model must closely track the evolving target model to ensure a high acceptance rate and overall decoding efficiency.

To address these challenges, we present SpecRL, a system that employs dynamic and online speculative decoding to accelerate the rollout stage of RL for LLMs (Figure 1). Based on a detailed analysis of the performance speedup of speculative decoding, we optimize RL rollout with speculative decoding from both the systems and algorithmic levels.

From a systems perspective, we design a dynamic SD configuration strategy that adapts to the generation batch size. This strategy considers the trade-off between the idle compute capabilities for the draft model and the benefit from more aggressive speculation. We also apply an online draft training approach to continuously align the draft model with the evolving target model. From an algorithmic perspective, we validate the use of speculative decoding does not alter the key properties of RL rollout, including the distributions of output lengths and rewards. We also develop a novel domain-specific RL algorithm to enhance the speedup from speculative decoding.

We show some preliminary results of SpecRL on RL training of Qwen2.5-7B-Instruct [3]. The experimental results demonstrate that SpecRL reduces the generation time by 1.3-2.5× compared to the original RL rollout process under different batch sizes. Furthermore, SpecRL achieves better rollout efficiency than naive speculative decoding by dynamically adjusting the SD configuration, continuously training the draft model, and employing the domain-specific RL training algorithm for the draft model.

2 SpecRL Design

2.1 Impact of SD on RL rollout

Prior work has theoretically proved that, through a modified rejection sampling method, the token distribution of SD given a context is identical to the distribution produced by the target model. Due to the sensitivity of RL training to rollout trajectories, we first carefully examine the distributions of output lengths and rewards after replacing original auto-regressive decoding with SD.

We first conduct RL training on Qwen2.5-7B-Instruct using GRPO [5] and a dataset from Open-Reasoner-Zero [6]. Table 1 lists the detailed experimental settings. Then, we train a draft model using the EAGLE-2 [8] approach. For our evaluation, we use checkpoints from the 50th and 450th steps of a 600-step RL training process. We benchmark four distinct rollout strategies:

- **Origin** is the baseline method using original auto-regressive decoding.

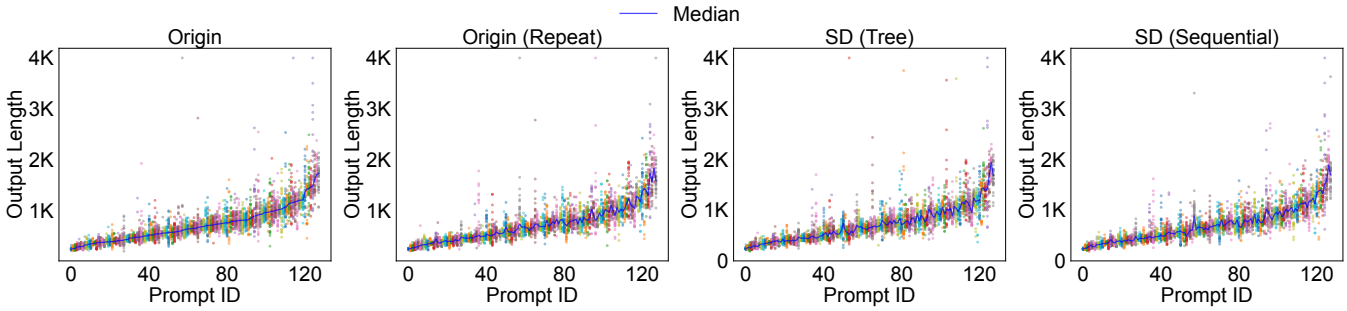

Figure 2: Output lengths of different rollout strategies at the 50th RL training step.

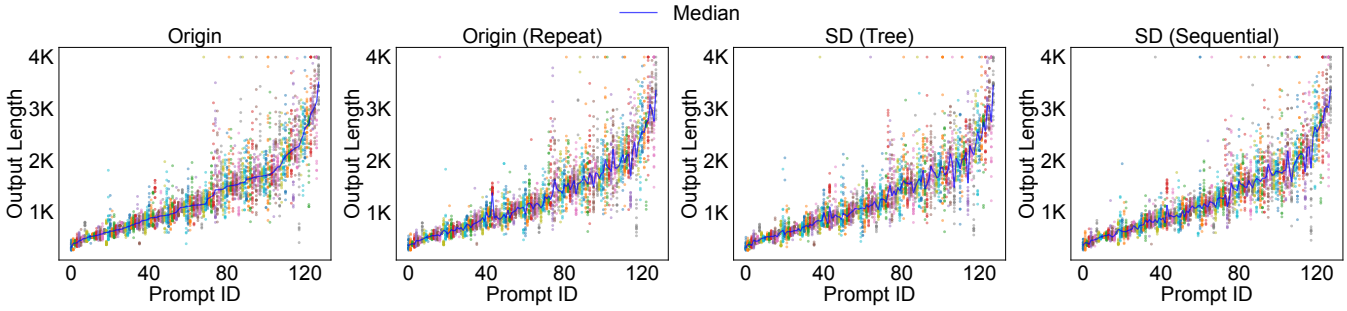

Figure 3: Output lengths of different rollout strategies at the 450th RL training step.

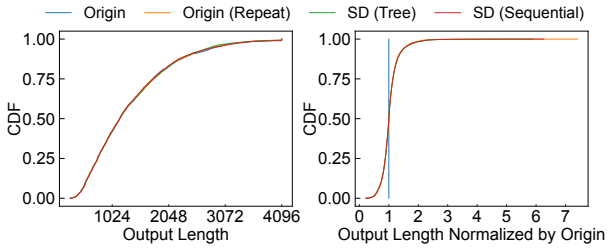

Figure 4: The output length CDF at the 450th RL training step.

- **Origin (Repeat)** is a second, independent run of the auto-regressive decoding baseline. RL typically encourages model exploration, which is often implemented by selecting larger values for top-k and top-p during sampling. Consequently, multiple independent and repeated rollouts may produce non-identical results. We, therefore, evaluate the effect of these independent rollouts via Origin (Repeat).
- **SD (Tree)** is the SD-assisted rollout strategy based on the EAGLE-2 speculative decoding method. At each step, it generates a candidate token tree with a depth of 5 and a width of 8. The 32 highest-scoring tokens from the tree are then selected for verification.
- **SD (Sequential)** also employs SD, but it speculatively generates a linear sequence of 4 future tokens at each step, all of which are subsequently verified.

Figure 2 and Figure 3 illustrates the output lengths for 128 prompts, each sampled 32 times, using model checkpoints from the 50th and 450th RL training steps. To facilitate a clear comparison across the different rollout strategies, the prompts (represented on the x-axis) are sorted in ascending order based on the median output length generated by the baseline "Origin" method. This same prompt ordering is maintained when presenting the results for the other three strategies. A key observation is that **for any given prompt, its output lengths tend to occupy a similar rank within the overall distribution, regardless of which of the four rollout strategies is used.**

Figure 4 further analyzes the output length distribution from the 450th training step by showing both the overall CDF and the per-sample output lengths normalized by the baseline ("Origin") output length. It reveals that the output length distributions of all four rollout methods are nearly identical. Furthermore, it demonstrates that **the effect of SD on an individual sample's output length is consistent with the natural variance observed when simply performing a second, independent rollout with the target model.**

We employ a similar methodology to assess the impact of SD on the model's mathematical problem-solving capabilities. Specifically, we examine the distribution of the average reward per prompt for the four rollout methods at different RL training steps. A binary reward scheme is used: a reward of 1 is assigned if and only if the model's output contains the

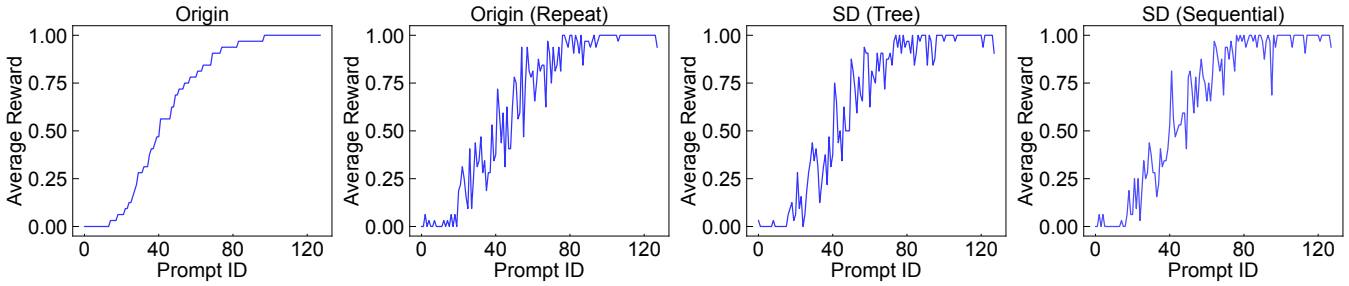

Figure 5: The distribution of average rewards with different rollout strategies at the 450th RL training step.

Symbol	Meaning
$T_T(b)$	Target model decoding time with batch size b
$T_D(b)$	Draft model decoding time with batch size b
$T_V(b)$	Target model verification time for b tokens
L	Output length
R	Number of Draft-Verify rounds
γ	Number of draft decoding steps per round
t	Number of tokens per draft decoding step
k	Number of tokens per verification step

Table 2: Key notations.

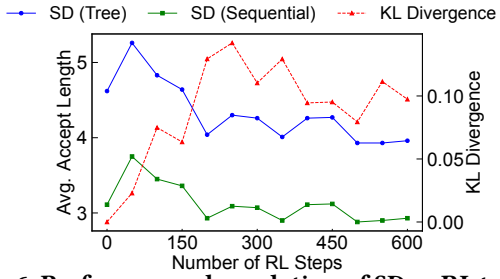

Figure 6: Performance degradation of SD as RL training progresses.

correct answer according to the ground truth, and 0 otherwise. As shown in Figure 5, which sorts the prompts on the x-axis with the same method as Figure 3, the distribution of average rewards is highly similar across all methods. This indicates that **the integration of SD does not cause performance regression**; we observe no cases where problems that are solvable using original auto-regressive decoding become unsolvable.

Therefore, applying speculative decoding in the RL rollout stage can provide system acceleration without negatively impacting algorithmic performance. This conclusion holds provided that signals from the draft model's inference are not incorporated into the loss function (in contrast to MTP used in DeepSeek-V3's pre-training). However, key differences between the RL rollout and online inference serving necessitate the use of dynamic and online SD.

2.2 Dynamic SD Configurations

The first distinction between RL rollout and online serving is the long-tail output length distribution, which causes the batch size to decrease over time within each RL step's rollout. This leads to a gradual shift in the process from being compute-bounded to memory-bandwidth-bounded. Given that SD primarily accelerates memory-bound decoding by reducing parameter access, its utility changes throughout the rollout. In the early, compute-bounded iterations, there is limited surplus compute capabilities for speculation. Consequently, the application of SD in RL rollout requires its configuration to be adjusted based on the decoding batch size.

The dynamic SD configuration strategy is based on the analysis of the speedup from SD with different configurations (which is similar to prior work [7]). Table 2 lists the key concepts. For a single prompt, the total time for auto-regressive generation by the target model is $T_{AR} = L \times T_T(b)$. The time for generation with speculative decoding is $T_{SD} = R \times (\gamma \times T_D(bt) + T_V(bk))$, where $k \leq \gamma \times t$. The speedup is calculated as

$$\frac{T_{AR}}{T_{SD}} = \frac{L \times T_T(b)}{R \times (\gamma \times T_D(bt) + T_V(bk))} \tag{1}$$

$$= \frac{L}{R} \times \frac{T_T(b)}{\gamma \times T_D(bt) + T_V(bk)} \tag{2}$$

Note that L/R is the accept length of the draft model, which can be obtained through offline or online metrics. The relationships between T_T, T_D, T_V and the batch size can be profiled offline as prior work does [11].

2.3 Online Draft Model Training

A second challenge arises from the fact that RL training continuously updates the target model's parameters, causing the token distribution generated by the draft model to progressively diverge from that of the target model. Figure 6 illustrates this issue. In the figure, "KL Divergence" measures the divergence between the current target model (updated by RL) and the original reference model; a higher value signifies a greater shift from the initial state. "Accept Length" measures the efficiency of the draft model's speculation—specifically, it is the number of tokens in a drafted sequence that are accepted by the target model in a single

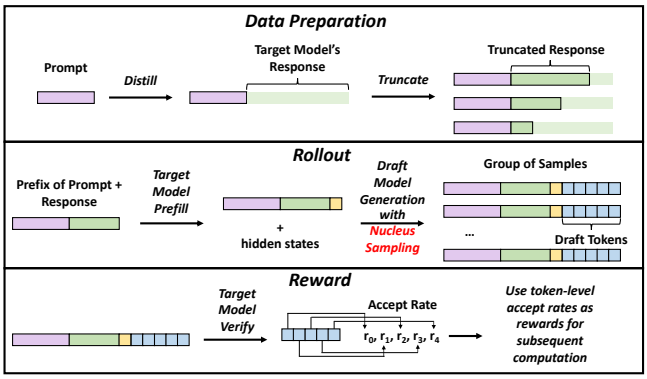

Figure 7: RL training for draft model.

step. A longer accept length directly corresponds to greater acceleration from speculative decoding. The figure clearly reveals a negative correlation between these two metrics, indicating that as RL training causes the target model's distribution to shift, the performance of speculative decoding degrades.

This necessitates online training of the draft model, which has been proposed by Online Speculative Decoding [9]. Typically, to preserve the model's general capabilities across various domains, the distributional shift caused by RL training is kept relatively small. This allows for a straightforward solution: we can counteract the performance degradation by periodically tuning the draft model every N training steps.

2.4 RL Training for Draft Model

Throughout our research, we observed that the draft model appears to have an intrinsic performance ceiling. Specifically, beyond a certain threshold, further expanding the depth and width of the candidate token tree yields no further increase in the accept length.

After initially training the draft model with the pre-training paradigm to the point where its accept length saturates, we explore the use of RL to further enhance its capabilities. This approach is predicated on the hypothesis that RL for LLMs is typically aimed at improving performance on specific tasks or domains, such as math and coding. Therefore, we assume that it is also possible to leverage RL to specialize the draft model for improved performance in a target domain. Figure 7 shows the overview of the RL training process for the draft model. The details are as follows.

- **Dataset.** We construct the training set by extracting multiple, distinct prefixes from each full sample (which consists of a prompt and the target model's response). These prefixes then serve as the individual prompts for the draft model's RL training.
- **Rollout.** Each rollout consists of generating a 5-token sequence from a given prompt. Diverging from standard SD inference, which often uses greedy sampling, our draft model's rollout utilizes nucleus sampling.
- **Reward.** Since the acceleration from Speculative Decoding is directly proportional to the accept length, we use the average accept length per speculation as the outcome reward. (This is equivalent to treating the acceptance rate of each individual token as a process reward).

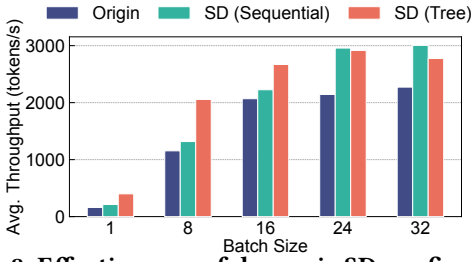

Figure 8: Effectiveness of dynamic SD configurations.

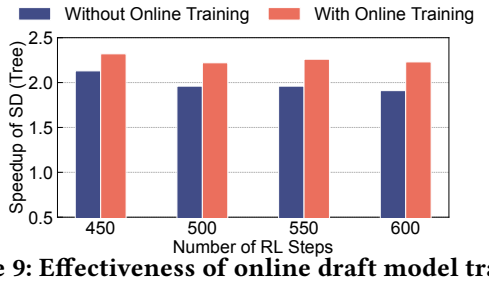

Figure 9: Effectiveness of online draft model training.

3 Evaluation

In this section, we show the preliminary evaluation results of SpecRL.

Figure 8 compares the performance of the Origin, SD (Tree), and SD (Sequential) methods across batch sizes ranging from 1 to 32. In this experiment, the SD (Sequential) configuration predicts one subsequent token per iteration. For small batch sizes (e.g., 1 to 8), SD (Tree) provides a greater speedup than SD (Sequential). However, as the batch size increases, the performance of SD (Sequential) gradually approaches and eventually surpasses that of SD (Tree). Notably, as the batch size increases from 24 to 32, the throughput of SD (Tree) declines. This is because at larger batch sizes, the complex tree-based approach expends excessive computational resources on speculation and verification, diminishing its returns. SpecRL adopts the optimal configuration according to the batch size.

Figure 9 shows the SD speedup with/without online draft model training. We train the draft model for one epoch using the 4096 rollout samples collected at the 450th RL training step. The updated model demonstrated a significant performance improvement in subsequent RL training.

As illustrated in Figure 10, our RL training paradigm for the draft model yields a substantial increase in the mean

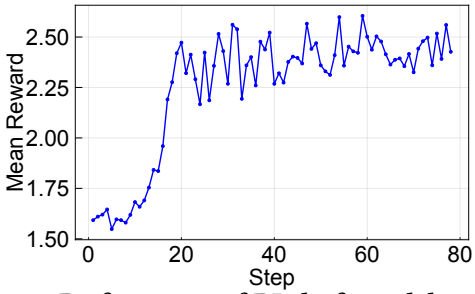

Figure 10: Performance of RL draft model training.

reward, raising it from approximately 1.6 to 2.4. In this context, the mean reward is defined as the average accept length under nucleus sampling. When the mean reward is 1.6, the average accept length is 4.27. Consequently, RL training for the draft model brings a significant improvement in the accept length.

4 Conclusion

We present SpecRL, a system that accelerates the long-tail rollout stage in RL for LLMs. SpecRL introduces a dynamic SD configuration strategy that adapts to fluctuating batch sizes, an online training approach to align the draft model with the evolving target model, and a novel domain-specific RL algorithm to maximize decoding speedup. Experimental results demonstrate that SpecRL achieves up to a 2.5× reduction in generation time compared to the conventional RL rollout process.

References

[1] 2024. Introducing OpenAI o1. https://openai.com/o1/. (2024).
[2] 2024. OpenAI o3-mini: Pushing the frontier of cost-effective reasoning. https://openai.com/index/openai-o3-mini/. (2024).
[3] 2024. Qwen2.5-7B-Instruct. https://huggingface.co/Qwen/Qwen2.5-7B-Instruct. (2024).
[4] 2025. Claude 3.7 Sonnet and Claude Code. https://www.anthropic.com/news/claude-3-7-sonnet/. (2025).
[5] Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, et al. 2025. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948* (2025).
[6] Jingcheng Hu, Yinmin Zhang, Qi Han, Daxin Jiang, Xiangyu Zhang, and Heung-Yeung Shum. 2025. Open-reasoner-zero: An open source approach to scaling up reinforcement learning on the base model. *arXiv preprint arXiv:2503.24290* (2025).
[7] Zongle Huang, Lei Zhu, Zongyuan Zhan, Ting Hu, Weikai Mao, Xianzhi Yu, Yongpan Liu, and Tianyu Zhang. 2025. MoESD: Unveil Speculative Decoding's Potential for Accelerating Sparse MoE. *arXiv preprint arXiv:2505.19645* (2025).
[8] Yuhui Li, Fangyun Wei, Chao Zhang, and Hongyang Zhang. 2024. Eagle-2: Faster inference of language models with dynamic draft trees. *arXiv preprint arXiv:2406.16858* (2024).
[9] Xiaoxuan Liu, Lanxiang Hu, Peter Bailis, Alvin Cheung, Zhijie Deng, Ion Stoica, and Hao Zhang. 2023. Online speculative decoding. *arXiv preprint arXiv:2310.07177* (2023).
[10] ByteDance Seed, Jiaze Chen, Tiantian Fan, Xin Liu, Lingjun Liu, Zhiqi Lin, Mingxuan Wang, Chengyi Wang, Xiangpeng Wei, Wenyuan Xu, et al. 2025. Seed1. 5-thinking: Advancing superb reasoning models with reinforcement learning. *arXiv preprint arXiv:2504.13914* (2025).
[11] Yinmin Zhong, Shengyu Liu, Junda Chen, Jianbo Hu, Yibo Zhu, Xuanzhe Liu, Xin Jin, and Hao Zhang. 2024. DistServe: Disaggregating Prefill and Decoding for Goodput-optimized Large Language Model Serving. In *USENIX OSDI*.