Speculative Sampling with Reinforcement Learning

Chenan Wang[1], **Daniel H. Shi**[1], **Haipeng Chen**[1]

[1]William & Mary
{cwang33,dshi01,hchen23}@wm.edu

Abstract

Inference time latency has remained an open challenge for real world applications of large language models (LLMs). State-of-the-art (SOTA) speculative sampling (SpS) methods for LLMs, like EAGLE-3, use tree-based drafting to explore multiple candidate continuations in parallel. However, the hyperparameters controlling the tree structure are static, which limits flexibility and efficiency across diverse contexts and domains. We introduce **Re**inforcement learning for **Sp**eculative **S**ampling (**Re-SpS**), the first reinforcement learning (RL)-based framework for draft tree hyperparameter optimization. Re-SpS dynamically adjusts draft tree hyperparameters in real-time, learning context-aware policies that maximize generation speed by balancing speculative aggression with computational overhead. It leverages efficient state representations from target model hidden states and introduces multi-step action persistence for better context modeling. Evaluation results across five diverse benchmarks demonstrate consistent improvements over the SOTA method EAGLE-3, achieving up to 5.45× speedup over the backbone LLM and up to 1.12× speedup compared to EAGLE-3 across five diverse benchmarks, with no loss in output fidelity.

Code — https://github.com/wmd3i/ReSpS.git

Introduction

Inference time latency has been an open challenge in deploying large language models (LLMs) (OpenAI 2023; Touvron et al. 2023; Anthropic 2024; Dubey et al. 2024). This latency, which stems from the sequential nature of autoregressive decoding and large model sizes, severely restricts the viability of LLMs in many time-sensitive applications (Kaplan et al. 2020; Li et al. 2024a; Urlana et al. 2024). To address this, speculative sampling (SpS) techniques aim to accelerate generation by parallelizing the process without compromising correctness, thereby enabling more efficient deployment of these massive models (Chen et al. 2023; Leviathan, Kalman, and Matias 2023; Sun et al. 2023; Jeon et al. 2024; Miao et al. 2024).

SpS accelerates language model inference by first using a lightweight draft model to generate candidate tokens, which are then jointly verified by the larger target model in a single pass. If any candidate is rejected, the process resumes from the last accepted token; if all are accepted, the target model generates the next token and the cycle repeats. This approach reduces the number of full model forward passes, enabling faster generation without sacrificing accuracy. This fundamental concept has been extended, first by SpecInfer (Miao et al. 2024) and subsequently by other methods, which replace the linear chain of candidates with a "draft tree," enabling the exploration of multiple potential continuations simultaneously (Cai et al. 2024; Li et al. 2024b, 2025; Wang et al. 2025; Zheng and Wang 2025). For example, EAGLE-2 (Li et al. 2024b) and EAGLE-3 (Li et al. 2025), the SOTA methods, adopt dynamic draft tree shaping, with EAGLE-3 taking a leap forward by recurrently feeding the draft model's unverified predictions as input for making further predictions, alongside feature sequences of the target model's hidden layers. Figure 1 (upper half) illustrates the sampling procedure of EAGLE-2 and EAGLE-3. While these works allow the nodes in the draft tree to take on different orientations, the hyperparameters governing overall structure—such as depth and branching factor—remain fixed and hand-tuned.

Since reinforcement learning (RL) (Sutton, Barto et al. 1998; Schulman et al. 2017) is a natural fit for sequential optimization problems, we take an RL-based approach that can adjust SpS draft tree hyperparameters dynamically. While promising, the actual implementation of a vanilla RL algorithm poses a central challenge: frequent RL policy calls incur substantial computational overhead that is comparable with the computation gains of improved SpS.

We identify two distinct sources of this computational overhead. First, *the state representation overhead*: generating rich, context-sensitive state embeddings at every decoding step—such as SentenceBERT (Reimers and Gurevych 2019)—incurs significant latency. In many cases, this encoding cost (often several milliseconds per step) can exceed the time saved by speculative sampling itself, especially when applied to long sequences where embedding computation is invoked repeatedly. Second, *the policy inference overhead*: each adaptation of the speculative sampling hyperparameters requires a forward pass through the RL policy network. When policy inference is performed at every generation step, the cumulative runtime cost quickly adds up over long sequences, creating a fundamental trade-off: while more frequent adaptation enables more context-aware and potentially

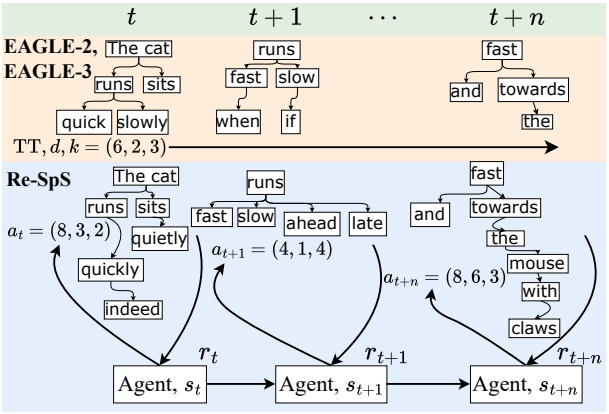

Figure 1: Re-SpS vs. EAGLE-2 & 3: Comparison of speculative sampling tree structures through steps $1 \cdots n$ of a generation task. **Top:** EAGLE-2 & 3 uses static hyperparameters (upper limits for the total token TT, depth d, and expansion factor k). **Bottom:** Re-SpS dynamically adapts the draft tree hyperparameters based on the context. Step $t+1$ exemplifies a more cautious draft tree, tuning top-k up and depth down to safely capture more possible tokens without risking costly rejections. Step $t+n$ shows a more aggressive configuration, with depth adjusted to allow for confident drafting. Draft tokens and branching structures were manually chosen to provide a readable, illustrative comparison. Actual model outputs may vary, but the figure faithfully reflects the parameter and adaptivity constraints of each approach.

optimal draft tree control, it also results in prohibitively high computational expense.

In this paper, we propose **Re**inforcement learning for **Sp**eculative **S**ampling (**Re-SpS**), the first framework to formulate the control of SpS draft tree hyperparameters as a reinforcement learning problem. Re-SpS introduces two key innovations to address computational overhead: (1) an *efficient feature reuse* mechanism, which leverages the target model's internal hidden states (specifically, a concatenated multi-layer feature vector from the draft model) as rich, low-cost state representations—eliminating the need for a separate encoder; and (2) a *multi-step action persistence* strategy, where a selected hyperparameter configuration is cached and reused across multiple decoding steps, thereby amortizing the cost of policy inference. As illustrated in Figure 1, this enables the Re-SpS policy to observe the generation context and dynamically select draft tree upper limits for the total tokens, depth, and expansion factor at each draft-verify run, optimizing tree-based SpS hyperparameters more efficiently than the static approaches.

Our key contributions include: 1) We propose Re-SpS, the first RL-based framework for SpS draft tree hyperparameter optimization. 2) We identify sources of the key technical challenge of computational overhead from RL policy calls, and introduce two technical innovations—efficient feature reuse and multi-step action persistence—to address them. 3) We conduct extensive empirical evaluations across five diverse benchmarks and three model sizes, demonstrating consistent speedup improvements (up to 5.45× over the backbone LLM and up to 1.12× over the SOTA method EAGLE-3) while maintaining exact output fidelity. Ablation studies further validate the effectiveness of our technical designs.

Related Work

This section outlines two methodological themes for SpS.

Chain and Tree-based Verification. The early SpS methods focus on a linear chain-based draft sequence (Chen et al. 2023; Leviathan, Kalman, and Matias 2023). Recent works, starting with SpecInfer (Miao et al. 2024), have leveraged tree-based structures to systematically explore and optimize token candidate selection (Zhang, Liu, and Song 2024). Many architectural innovations have emerged to optimize tree-based speculation (Sun et al. 2023; He et al. 2024; Cai et al. 2024; Li et al. 2024c; Jeon et al. 2024; Chen et al. 2024). SpecInfer (Miao et al. 2024) introduced parallel token verification using tree attention (Vaswani et al. 2017) and fixed expansion hyperparameters. Medusa (Cai et al. 2024) integrated drafting into the target model via MLP heads, while EAGLE (Li et al. 2024c) combined tree-based drafting with feature-level autoregression. C2T (Huo et al. 2025) employed lightweight classifiers for tree management. However, all these methods share a common limitation: they use static hyperparameters for draft tree generation.

Adaptive Control of Draft Tree Structures. A major thrust of recent research has been toward heuristic-driven adaptivity (Zhang, Liu, and Song 2024). EAGLE-2 (Li et al. 2024b) introduces a context-aware dynamic draft tree, which uses the confidence of the draft model as a proxy for the final acceptance rate to intelligently prune the unlikely branches of the speculative tree and re-rank all nodes to select the most promising candidates for verification (Li et al. 2024b). The most recent iteration, EAGLE-3, abandons feature prediction for direct token prediction and incorporates target model hidden states as well as past draft outputs into draft model inputs, but retains static draft tree hyperparameters (Li et al. 2025). Complementary adaptive strategies have also gained traction, such as SpecDec++ (Huang, Guo, and Wang 2024), DySpec (Xiong et al. 2025), OPT-Tree (Wang et al. 2025), and ProPD (Zhong et al. 2024). The latter two enlist early pruning and dynamic tree generation strategies– however, these approaches remain limited: OPT-Tree optimizes node allocation within fixed budgets, while ProPD uses regression-based tree sizing. There have been some initial explorations on data-driven and learning-based frameworks for SpS hyperparameter optimization. For example, HASS uses distillation-based supervised learning to better align the draft model's predictions with the target's (Zhang et al. 2025), and MetaSD and BanditSpec use multi-armed bandit algorithms to select drafters (MetaSD) or distinct speculative sampling strategies (Kim, Jung, and Yun 2024; Hou et al. 2025). While these methods focus on selecting between pre-defined configurations or models, none explore learning adaptive policies that can dynamically adjust draft tree structure hyperparameters based on generation context.

Preliminary

Auto-Regressive Decoding in LLMs

Large Language Models (LLMs) generate text one token at a time using auto-regressive decoding (Bengio et al. 2003). Given a discrete token sequence $x = (x_1, x_2, \ldots, x_s) \in \mathcal{T}^s$ of length s over a token set $\mathcal{T}$, we define a slice of this sequence at decoding round t as $x_{1:m}^t = (x_1, x_2, \ldots, x_m)$. The LLM outputs a probability distribution over the next token conditioned on all previous tokens. Specifically, the probability of token x_t is given by $P_{\text{LLM}}(x_t \mid x_1, \ldots, x_{t-1})$.

To generate the next token, a sampling method (e.g., greedy, top-k sampling (Fan, Lewis, and Dauphin 2018)) is applied to this distribution. We define x_0 as the prompt or query tokens and let the model generate an output sequence of m tokens, denoted by $y_1, \ldots, y_m$. In this work, we use greedy decoding for reproducibility, where each token is deterministically chosen as:

$$y_i = \arg\max_y P_{\text{LLM}}(y \mid x_0, y_1, \ldots, y_{i-1}), \quad i = 1, \ldots, m.$$

Speculative Sampling

Speculative Sampling (Leviathan, Kalman, and Matias 2023; Chen et al. 2023) introduces a *draft-then-verification* paradigm to accelerate auto-regressive decoding. We formalize this process mathematically to establish notation for subsequent analysis.

Draft Phase. At decoding step t, given context $\mathbf{c}_t = (x_0, y_1, \ldots, y_{t-1})$ consisting of prompt x_0 and accepted tokens, a draft model M_d generates a candidate sequence $\hat{\mathbf{y}}_t = (\hat{y}_t, \hat{y}_{t+1}, \ldots, \hat{y}_{t+n-1})$ of length n:

$$\hat{\mathbf{y}}_t \sim M_d(\cdot \mid \mathbf{c}_t)$$

Verification Phase. The target model evaluates the extended sequence $(\mathbf{c}_t, \hat{\mathbf{y}}_t)$ in a single forward pass, yielding conditional distributions:

$$P_T(y_{t+i} \mid \mathbf{c}_t, \hat{y}_t, \ldots, \hat{y}_{t+i-1}), \quad i = 0, \ldots, n-1$$

Acceptance Criterion. Each draft token $\hat{y}_{t+i}$ is accepted with probability determined by the acceptance function α:

$$\alpha(\hat{y}_{t+i}) = \min\left(1, \frac{P_T(\hat{y}_{t+i} \mid \mathbf{c}_t, \hat{y}_t, \ldots, \hat{y}_{t+i-1})}{P_d(\hat{y}_{t+i} \mid \mathbf{c}_t, \hat{y}_t, \ldots, \hat{y}_{t+i-1})}\right)$$

where P_T and P_d represent the probability distributions from the target model and draft model, respectively The acceptance length ℓ_t is defined as the number of consecutively accepted tokens:

$$\ell_t = \min\{j \in \{0, 1, \ldots, n\} : \alpha(\hat{y}_{t+j}) = 0\} \cup \{n\}$$

Draft Tree Expansion. EAGLE-2 (Li et al. 2024b) introduces a two-phase dynamic construction process that moves beyond EAGLE's (Li et al. 2024c) static draft tree approach. Unlike EAGLE's fixed tree structure, EAGLE-2 dynamically adjusts the draft tree based on context-dependent acceptance rates rather than assuming acceptance depends solely on token position. EAGLE-2 constructs the tree through selective layer-wise expansion. At each layer, the algorithm selects the top-k tokens with highest global acceptance probabilities from the entire current layer for expansion to the next layer. The selection is based on global acceptance probability V_i, calculated as the cumulative confidence score along the path from root to each token:

$$V_i = \prod_{i=1}^{m} c(y_i \mid y_1, \ldots, y_{i-1})$$

where $c(\cdot)$ represents the draft model's confidence in token acceptance.

Confidence-Based Candidate Reranking. EAGLE-2 (Li et al. 2024b) introduces reranking based on V_i: after draft tree expansion is complete, *all* tokens in the tree are reranked before being used as candidates for verification.

Methodology

Despite notable progress from recent tree-based SpS methods such as EAGLE-2 and EAGLE-3 (Li et al. 2024b, 2025), a critical limitation remains: these approaches rely on static draft tree hyperparameters throughout the decoding process. As a result, their efficiency and adaptability are inherently restricted, especially when handling diverse or dynamically changing contexts. To overcome these obstacles, we introduce Re-SpS, the first RL-based approach for draft tree hyperparameter optimization, alongside innovative designs that address the computational overhead of RL policy calls.

MDP Formulation and Vanilla RL Solution

We first formulate the problem of dynamically selecting speculative sampling hyperparameters as a Markov Decision Process (MDP), defined by the tuple $(\mathcal{S}, \mathcal{A}, R, \xi)$.

State Space $\mathcal{S}$ At each decision point t, the agent is in a given state $s_t \in \mathcal{S}$ that captures the current context of the text generation process. This state should contain sufficient information for the agent to make an informed decision about the optimal speculative strategy. A naive approach would be to use a rich embedding of the entire context (prompt, question, and all previously generated tokens) using a powerful model like SentenceBERT (Reimers and Gurevych 2019):

$$s_t = [e(\text{prompt}, y_{<t})], \tag{1}$$

where $e(\cdot)$ denotes the embedding function.

Action Space $\mathcal{A}$ The action space consists of discrete hyperparameter combinations drawn from predefined sets of values:

$$\mathcal{A} = \left\{ (\texttt{TT}, d, k) \,\middle|\, \begin{array}{l} \texttt{TT} \in \mathcal{S}_{\texttt{TT}} \\ d \in \mathcal{S}_d \\ k \in \mathcal{S}_k \end{array} \right\} \tag{2}$$

where TT, d, and k represent the upper limits for the total number of tokens, tree depth, and top-k expansion factor per layer, respectively. The sets $\mathcal{S}_{TT}$, $\mathcal{S}d$, and $\mathcal{S}_k$ are finite collections of pre-selected integer values for each hyperparameter. This action space allows the agent to select a specific draft tree configuration at each decision point, enabling dynamic adaptation to the current context (see Figure 2).

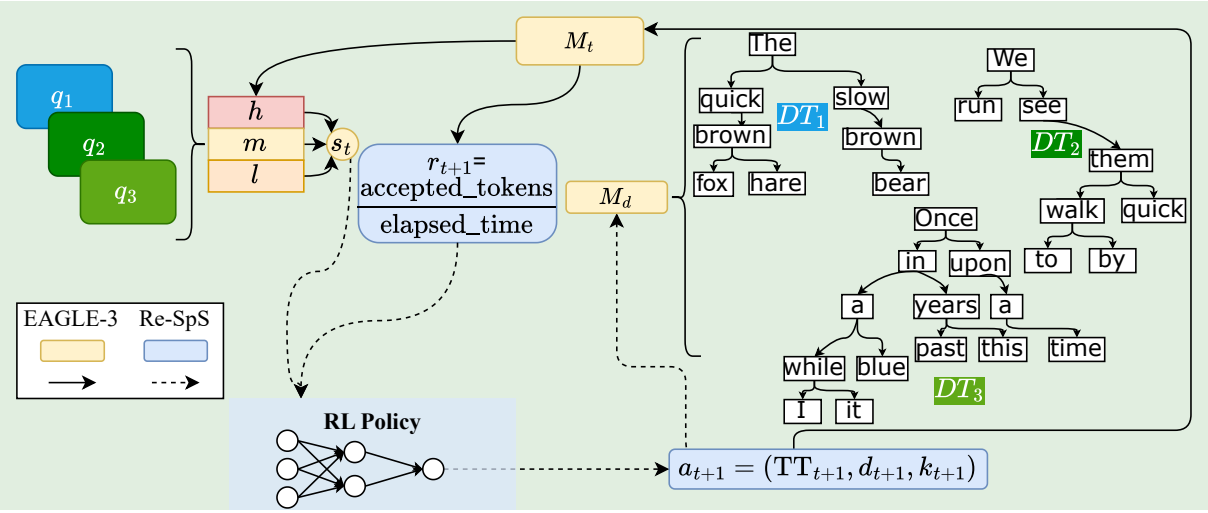

Figure 2: Architecture of Re-SpS: The diagram illustrates the Re-SpS framework for SpS in LLMs. For each new input prefix (from question tasks $q_1, q_2, q_3, \ldots$), hidden state vector (h, m, l) from the target model (M_t) is aggregated into s_t and passed to a reinforcement learning (RL) policy. The RL agent outputs draft tree hyperparameters $(TT_{t+1}, d_{t+1}, k_{t+1})$ for the next generation step. The draft model (M_d) constructs tree-structured speculative candidates (DT_1, DT_2, DT_3), which are verified by the target model. The number of accepted tokens and elapsed time are recorded, and the generation speed $r_{t+1} = \frac{\text{accepted tokens}}{\text{elapsed time}}$ is used as the RL reward. Solid arrows show EAGLE-3's static pipeline; dashed arrows highlight Re-SpS's adaptive, RL-driven control. Note: For clarity, the figure is simplified; in practice, the draft trees DT are redefined multiple times for each new prefix within a question task until a maximum sequence length or end-of-sequence is reached.

Reward Function R Upon executing an action, the agent receives a reward $r_t = R(s_t, a_t)$. We define the immediate reward as the generation speed, measured in accepted tokens per second calculated by dividing the number of accepted tokens by the elapsed time for the drafting/verification step, which directly aligns the agent's objective with our goal of minimizing latency.

$$r_t = \frac{\text{accepted tokens}}{\text{elapsed time (seconds)}} \tag{3}$$

This reward function captures the efficiency of the speculative sampling process, incentivizing the agent to select hyperparameters that maximize the number of accepted tokens while minimizing the time taken for generation.

Transition Function ξ The transition function $\xi(s_{t+1} \mid s_t, a_t)$ defines how the state evolves based on the selected action. In our case, it is implicitly and deterministically defined by the draft tree construction process, which generates a new state based on the current state (i.e., feature vector) and selected hyperparameters, and by the speculative decoding process itself, where the state evolves as new tokens are generated and accepted.

Vanilla RL Implementation We use PPO (Schulman et al. 2017) as our backbone reinforcement learning algorithm due to its stability and effectiveness in sequential decision-making tasks. PPO updates the policy by maximizing the following clipped surrogate objective:

$$L^{\text{PPO}}(\theta) = \mathbb{E}_t \left[\min \left(r_t(\theta)\hat{A}_t,\ \text{clip}\big(r_t(\theta), 1-\epsilon, 1+\epsilon\big)\hat{A}_t \right) \right]$$

where $r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)}$ is the probability ratio and $\hat{A}_t$ is the estimated advantage.

We also explore the maximum entropy variant, which augments the standard PPO objective with an entropy regularization term. This addition encourages the policy to maintain high exploration by favoring more stochastic action distributions, thereby mitigating premature convergence to suboptimal policies. The overall objective is given by:

$$L^{\text{MAX-ENT}}(\theta) = L^{\text{PPO}}(\theta) + \beta_H \, \mathbb{E}_t \big[H(\pi_\theta(\cdot|s_t)) \big]$$

where $H(\pi_\theta(\cdot|s_t))$ is the entropy of the policy at state s_t, and β_H is a weight factor (Haarnoja et al. 2018; Schulman et al. 2017).

Lossless Output Fidelity Re-SpS is built upon EAGLE-3 (Li et al. 2025) and inherits its lossless property: it uses SpS with target-model verification (Leviathan, Kalman, and Matias 2023; Chen et al. 2023), preserving the output distribution of standard autoregressive decoding. In contrast, methods like MEDUSA (Cai et al. 2024) relax the acceptance conditions of SpS and lack rejection-based correction, thereby failing to guarantee distributional equivalence under non-greedy decoding (Li et al. 2025).

Motivation: Challenges in a Naive RL Implementation

While the MDP formulation is straightforward, a naive implementation RL implementation (e.g., based on vanilla PPO or Max-Entropy PPO), where the agent makes a decision at every single decoding step, presents significant practical challenges. Our initial explorations revealed that this

approach fails because the computational overhead introduced by the RL agent can easily outweigh the latency gains from improved speculation. This overhead stems from two primary sources.

First, generating a rich state representation for the RL agent is prohibitively expensive. A robust policy requires a detailed understanding of the current generation context. A naive approach would be to encode the entire context (prompt, question, and all previously generated tokens) at each step using a powerful model like SentenceBERT (Reimers and Gurevych 2019) via Equation 1. However, this would introduce significant inference latency (approximately 5–15 ms per call) and memory overhead even with a small model, likely negating any speedup from the speculative decoding itself, rendering the entire process slower than the baseline it is meant to accelerate. In addition to the computational latency, maintaining and processing 384-dimensional embedding vectors for multiple contexts also increases the memory footprint considerably. Especially when decisions may be needed every decoding step, this overhead accumulates rapidly, potentially negating any speedup gained through speculative decoding.

The second major source of overhead arises from the frequency of RL policy network inference. Making hyperparameter decisions at every decoding step enables fine-grained, highly adaptive control via Equation 3 over the draft tree structure, but it also creates prohibitive computational costs. Specifically, each policy call requires a forward pass through the neural network, which typically consists of 2–4 layers with 64–512 hidden units. While more frequent decisions can provide better adaptation to the evolving context, this increased decision frequency also raises the inference overhead proportionally. Over the course of generating a single response—which can involve 50–100 or more decoding steps per turn and 2–5 turns per question—the cumulative cost of repeated policy calls can quickly outweigh any computational savings achieved through a more optimized draft tree or improved speculative sampling. Thus, the compounded inference time from frequent policy queries may become a major bottleneck, ultimately negating the efficiency benefits of the approach.

Together, these factors make it likely that this method would render the overall process slower than the baseline it is intended to accelerate.

Addressing the Challenges with Re-SpS

Our Re-SpS framework is designed to solve this MDP while explicitly addressing the computational overheads identified above through two key innovations.

Efficient State Representation via Feature Reuse To resolve the challenge of state representation overhead, we redefine the state s_t not with a costly external embedding, but by reusing the internal features already computed by the EAGLE-3 (Li et al. 2025) draft model. Specifically, the state is a concatenation of hidden states from three strategically selected layers of the target language model:

$$s_t = [h_{\text{LM}}^{(h,m,l)}] \tag{4}$$

where $h_{\text{LM}}^{(h,m,l)}$ is the fused feature vector, shown as s_t in Figure 2. This approach provides a rich, multi-level representation of the generation context—capturing syntactic, semantic, and task-specific information—without introducing any additional inference cost, as these features are an integral part of the EAGLE-3 architecture. In EAGLE-3, these features serve as hidden states for the draft model, enabling it to generate draft tokens; similarly, we utilize them to construct the state representation for our RL agent. The distinction is that, while EAGLE-3 passes these hidden states through a fully connected layer to obtain a single fused feature vector, our approach instead concatenates the hidden states from the three layers directly, thereby reducing the computational cost.

Amortizing Policy Inference with Multi-Step Action Persistence To mitigate the overhead of frequent policy calls, we introduce a multi-step action persistence mechanism, or action caching. Instead of querying the policy network at every step, a selected action (TT, d, k) is cached and reused for N consecutive decoding steps—that is, the *cache interval* has length N ($N = 10$ during training and $N = 30$ during inference). This cache interval amortizes the cost of a single policy inference over multiple decoding steps, striking a balance between adaptivity and efficiency. This approach leverages the Markov property, as the reward signal, which we average over the N decoding steps in the cache interval, naturally captures the temporal dynamics and performance impact of the chosen action without requiring a complex, multi-step state history. To complement this, we compute the reward as an average across the cache duration:

$$r_{\text{avg}} = \frac{1}{N} \sum_{i=1}^{N} \frac{\text{accepted\_tokens}_i}{\text{elapsed\_time}_i(\text{seconds})} \tag{5}$$

This allows that the reward signal reflects the cumulative performance impact of the cached hyperparameter decision. The core logic of this process is detailed in Algorithm 1. The overall architecture of Re-SpS is illustrated in Figure 2 and the overall algorithm in Algorithm 2 in the Appendix B. The algorithm is designed to be efficient and adaptive to all tree-based speculative sampling methods, such as Medusa (Cai et al. 2024), EAGLE-2 (Li et al. 2024b), and EAGLE-3 (Li et al. 2025).

Experiments

We evaluate Re-SpS against SOTA SpS method, EAGLE-3, across multiple benchmarks and model configurations. Our experimental setup follows established protocols while introducing comprehensive ablation studies to validate our technical contributions.

Experimental Setup

Models and Hardware Experiments use three LLM backbones: LLaMA 3.1-8B, Vicuna-13B, and LLaMA 3.3-70B, with their pretrained EAGLE-3 draft models. LLaMA 3.1-8B and Vicuna-13B experiments are conducted on a single NVIDIA A40 GPU (40GB), while LLaMA 3.3-70B requires 4 NVIDIA H100 GPUs (80GB each).

Backbone	Method	MT-Bench	HumanEval	GSM8K	Alpaca	CNN/DM	Mean	p-value
LLaMA 3.1-8B	Medusa	2.07×	2.50×	2.23×	2.08×	1.71×	2.12×	–
	Hydra	2.88×	3.28×	2.93×	2.86×	2.05×	2.80×	–
	EAGLE-3	3.39×	3.65×	3.52×	3.67×	**2.96**×	3.44×	–
	Re-SpS	**3.43**×	**3.89**×	**3.62**×	**3.90**×	2.87×	**3.54**×	$< 10^{-4}$
Vicuna-13B	Medusa	2.07×	2.50×	2.23×	2.08×	1.71×	2.12×	–
	Hydra	2.88×	3.28×	2.93×	2.86×	2.05×	2.80×	–
	EAGLE-3	3.75×	4.28×	3.85×	3.76×	**3.35**×	3.80×	–
	Re-SpS	**3.76**×	**4.64**×	**3.99**×	**3.99**×	3.24×	**3.92**×	$< 10^{-9}$
LLaMA 3.3-70B	EAGLE-3	4.35×	4.87×	4.74×	4.77×	**4.09**×	4.46×	–
	Re-SpS	**4.47**×	**5.45**×	**5.13**×	**5.34**×	4.03×	**4.88**×	$< 10^{-29}$

Table 1: Speedup ratios (accepted tokens per second; higher is better) across five benchmarks for LLaMA 3.1-8B, Vicuna-13B, and LLaMA 3.3-70B, comparing Re-SpS with EAGLE-3, Medusa, and Hydra. The p-values are derived from the Wilcoxon signed-rank test, indicating a highly statistically significant difference between the speeds of Re-SpS and EAGLE-3 in all cases. All results are reported at temperature 0. Re-SpS and EAGLE-3 are tested on our machine, while Medusa and Hydra results are from the EAGLE-2 & 3 papers (Li et al. 2024b, 2025).

Algorithm 1: Re-SpS with Action Caching

Require: Target model M_t, draft model M_d, policy π_θ, cache interval length N, cache step $c \in [0, N]$
Ensure: Draft tree DT

$c \leftarrow 0$ ▷ Initialize cache step within interval
$s_t \leftarrow \text{concat}(h, m, l)$ from M_t
if $cache\_step = 0$ or no cached action **then**
  $(TT, d, k) \leftarrow \pi_\theta(s_t)$
  $cache\_step \leftarrow 1$
else
  Use cached (TT, d, k)
  $cache\_step \leftarrow cache\_step + 1$
end if
Construct draft tree DT using (TT, d, k)
Verify DT with M_t
Compute reward $r(s_t, a_t)$ using Eq. 5
Store trajectory and update policy
Reset $cache\_step$ if $cache\_step \geq N$

Experimental Setup The RL policy is trained using PPO with maximum entropy regularization (entropy coefficient $\beta_H = 0.1$) for enhanced exploration. Training is conducted on a diverse subset of ShareGPT (Wang et al. 2024) and UltraChat200K (Ding et al. 2023) datasets containing 4,000 questions across multiple domains. In terms of model architecture, we use a two-layer MLP with 128 hidden units for both the actor and critic networks. The detailed experimental settings are in Appendix A.

Evaluation Benchmarks Following EAGLE-3, we evaluate our policy on five common tasks, using the same weights for all tasks without fine-tuning on the respective tasks,

For multi-turn conversation, code generation, mathematical reasoning, instruction following, and summarization we chose the MT-bench (Zheng et al. 2023), HumanEval (Chen et al. 2021), GSM8K (Cobbe et al. 2021), Alpaca (Taori et al. 2023), and CNN/DailyMail (Nallapati et al. 2016) datasets.

Main Results

Table 1 presents the comprehensive performance comparison across all benchmarks. Re-SpS achieves consistent speedup improvements over EAGLE-3 baselines:

- **LLaMA 3.1-8B**: Average speedup of $1.03\times$ over EAGLE-3, with notable improvements on HumanEval ($1.07\times$) and Alpaca ($1.07\times$).
- **Vicuna-13B**: Average speedup of $1.04\times$ over EAGLE-3, with strongest gains on HumanEval ($1.09\times$) and Alpaca ($1.08\times$).
- **LLaMA 3.3-70B**: Average speedup of $1.06\times$ over EAGLE-3, with significant improvements on HumanEval ($1.12\times$) and Alpaca ($1.12\times$).

The results demonstrate that dynamic hyperparameter adaptation provides measurable efficiency gains across diverse task domains while maintaining exact output fidelity (byte-for-byte identical to greedy decoding). Statistical significance is assessed with the paired Wilcoxon signed-rank test, with tiny p-values showing strong evidence that the difference in inference speed is not noise.

CNN/DailyMail Performance Note The slight performance degradation on CNN/DailyMail ($0.98\times$ and $0.97\times$ respectively) results from a necessary evaluation setup modification. To prevent KV cache overflow errors with longer documents, we increased the maximum sequence length from 2048 to 2200 tokens for Re-SpS evaluation, while baselines used the standard 2048-token limit. This difference creates additional computational overhead not present in baseline evaluations.

Ablation Study

To validate the contributions of our framework's key components, we conduct a series of ablation studies. We analyze the effects of the following components:

1) Feature Representation: We compare two state representations. The first is a *Text Embedding* where the entire

Model	Policy Configuration	Avg. Speedup vs. EAGLE-3	Unique Actions
LLaMA 3.1-8B	Standard PPO + Text Embedding	1.044×	3
	Standard PPO + Feature Vector	**1.049**×	5
	Max-Entropy PPO + Text Embedding	1.017×	8
	Max-Entropy PPO + Feature Vector	1.025×	**18**
Vicuna-13B	Standard PPO + Text Embedding	1.006×	8
	Standard PPO + Feature Vector	1.028×	3
	Max-Entropy PPO + Text Embedding	1.015×	14
	Max-Entropy PPO + Feature Vector	**1.033**×	**15**

Table 2: Ablation results for policies with a [128,128] hidden layer architecture for both actor and critic networks.

context (prompt, question, and previously generated tokens) is encoded using SentenceBERT (Reimers and Gurevych 2019) to provide rich semantic information. The second is a *Feature Vector* representation, which uses the draft model's internal hidden states, avoiding the computational overhead of an external encoder. The results in Table 2 show that the Feature Vector representation consistently outperforms the Text Embedding approach across both LLaMA 3.1-8B and Vicuna-13B backbones, achieving speedups of 1.049× and 1.033×, respectively. This validates our design choice to leverage internal model features for efficient state representation, as it provides sufficient context without incurring additional inference costs.

2) Cache Interval Length: We evaluate the impact of the cache interval length, which determines how many decoding steps the RL policy's action is cached and reused. Figure 3 shows that increasing the cache interval length from 1 to 50 decoding steps leads to a significant reduction in inference time for LLaMA 3.1-8B, while increasing the generated speed (tokens per second). This verifies our assumption that amortizing the cost of policy inference over multiple steps can yield substantial efficiency gains.

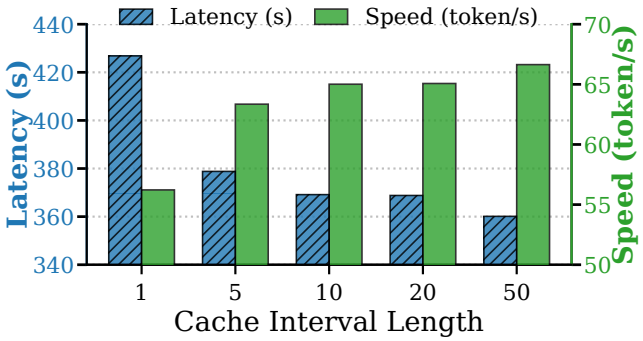

Figure 3: Cache interval length vs. inference latency in seconds and generated speed in tokens per second for LLaMA 3.1-8B. The cache interval length is the number of decoding steps over which the RL policy's action is cached and reused. Tested with a fresh SpS RL policy on random 80 questions from the training dataset.

3) RL Algorithm: We compare a *standard PPO* variant against *Max-Entropy PPO*, which includes an entropy regularization term in its objective function. This term encourages the policy to explore a more diverse set of actions rather than converging to a single strategy prematurely. The results in Table 2 show that Max-Entropy PPO achieves slightly lower speedups than Standard PPO for LLaMA 3.1-8B (1.025× vs. 1.049×) but outperforms it for Vicuna-13B (1.033× vs. 1.028×). However, the Max-Entropy PPO policy exhibits greater action diversity, with 18 unique actions for LLaMA 3.1-8B and 15 for Vicuna-13B, compared to only 5 and 3 for Standard PPO, respectively. This suggests that while Max-Entropy PPO may not always yield the highest speedup, it promotes a more adaptive and robust policy that can better handle diverse contexts. Other ablations are in Appendix C.

Overall, these studies show a complex interplay between network capacity, RL algorithm, and the underlying target model. While Max-Entropy PPO consistently promotes action diversity, the optimal configuration for performance depends on the specific model architecture and the capacity of the policy network to leverage either broad exploration or focused, context-driven decision-making.

Conclusion

We have presented Re-SpS, the first framework to formulate speculative sampling hyperparameter selection as a Markov Decision Process solved through online reinforcement learning. Through evaluation across five diverse benchmarks, Re-SpS consistently outperforms static baselines while preserving exact output fidelity. Our key contributions demonstrate that dynamic hyperparameter adjustment significantly outperforms static configurations, with our RL agent learning to balance speculative aggression and computational overhead across varying contexts. The framework validates that optimal draft tree structures vary substantially across domains, enabling principled optimization of complex trade-offs between the upper limits for the draft tokens, tree depth, and expansion factor.

Future work will extend the framework to other speculative sampling architectures, incorporate sophisticated contextual state representations, and explore multi-objective optimization for throughput, latency, and memory efficiency. The success of RL-based adaptive control suggests broad potential for intelligent optimization across LLM inference acceleration techniques.

References

Anthropic. 2024. Introducing the next generation of Claude.

Bengio, Y.; Ducharme, R.; Vincent, P.; and Jauvin, C. 2003. A neural probabilistic language model. In *Journal of machine learning research*, volume 3, 1137–1155.

Cai, T.; Li, Y.; Geng, Z.; Peng, H.; Lee, J. D.; Chen, D.; and Dao, T. 2024. Medusa: Simple LLM Inference Acceleration Framework with Multiple Decoding Heads. In *Forty-first International Conference on Machine Learning*.

Chen, C.; Borgeaud, S.; Irving, G.; Lespiau, J.-B.; Sifre, L.; and Jumper, J. 2023. Accelerating large language model decoding with speculative sampling. *arXiv preprint arXiv:2302.01318*.

Chen, M.; Tworek, J.; Jun, H.; Yuan, Q.; de Oliveira Pinto, H.; Kaplan, J.; Edwards, H.; Burda, Y.; Joseph, N.; Brockman, G.; et al. 2021. Evaluating large language models trained on code.

Chen, Z.; May, A.; Svirschevski, R.; Huang, Y.-H.; Ryabinin, M.; Jia, Z.; and Chen, B. 2024. Sequoia: Scalable and Robust Speculative Decoding. In *NeurIPS*.

Cobbe, K.; Kosaraju, V.; Bavarian, M.; Chen, M.; Jun, H.; Hilton, J.; Nakano, R.; Schulman, J.; Knight, M.; Kaplan, J.; et al. 2021. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*.

Ding, N.; Chen, Y.; Xu, B.; Qin, Y.; Hu, S.; Liu, Z.; Sun, M.; and Zhou, B. 2023. Enhancing Chat Language Models by Scaling High-quality Instructional Conversations. In *The 2023 Conference on Empirical Methods in Natural Language Processing*.

Dubey, A.; Jauhri, A.; Pandey, A.; Kadian, A.; Al-Dahle, A.; Letman, A.; Mathur, A.; Schelten, A.; Yang, A.; Fan, A.; et al. 2024. The llama 3 herd of models. *arXiv e-prints*, arXiv–2407.

Fan, A.; Lewis, M.; and Dauphin, Y. 2018. Hierarchical Neural Story Generation. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (ACL)*, 889–898.

Haarnoja, T.; Zhou, A.; Abbeel, P.; and Levine, S. 2018. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *International conference on machine learning*, 1861–1870.

He, Z.; Zhong, Z.; Cai, T.; Lee, J. D.; and He, D. 2024. REST: Retrieval-Based Speculative Decoding. In *2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL 2024*, 1582–1595.

Hou, Y.; Zhang, F.; Du, C.; Zhang, X.; Pan, J.; Pang, T.; Du, C.; Tan, V. Y. F.; and Yang, Z. 2025. BanditSpec: Adaptive Speculative Decoding via Bandit Algorithms. In *Forty-second International Conference on Machine Learning*.

Huang, K.; Guo, X.; and Wang, M. 2024. SpecDec++: Boosting Speculative Decoding via Adaptive Candidate Lengths. In *Workshop on Efficient Systems for Foundation Models II@ ICML2024*.

Huo, F.; Tan, J.; Zhang, K.; Cai, X.; and Sun, S. 2025. C2T: A Classifier-Based Tree Construction Method in Speculative Decoding. *arXiv preprint arXiv:2502.13652*.

Jeon, W.; Gagrani, M.; Goel, R.; Park, J.; Lee, M.; and Lott, C. 2024. Recursive Speculative Decoding: Accelerating LLM Inference via Sampling Without Replacement. In *ICLR 2024 Workshop on Large Language Model (LLM) Agents*.

Kaplan, J.; McCandlish, S.; Henighan, T.; Brown, T. B.; Chess, B.; Child, R.; Gray, S.; Radford, A.; Wu, J.; and Amodei, D. 2020. Scaling laws for neural language models. *arXiv preprint arXiv:2001.08361*.

Kim, T.; Jung, H.; and Yun, S.-Y. 2024. A unified framework for speculative decoding with multiple drafters as a bandit. *arXiv preprint arXiv:2409.09316*.

Leviathan, Y.; Kalman, M.; and Matias, Y. 2023. Fast inference from transformers via speculative decoding. In *International Conference on Machine Learning*, 19274–19286.

Li, J.; Xu, J.; Huang, S.; Chen, Y.; Li, W.; Liu, J.; Lian, Y.; Pan, J.; Ding, L.; Zhou, H.; et al. 2024a. Large language model inference acceleration: A comprehensive hardware perspective. *arXiv preprint arXiv:2410.04466*.

Li, Y.; Wei, F.; Zhang, C.; and Zhang, H. 2024b. EAGLE-2: Faster Inference of Language Models with Dynamic Draft Trees. In *Empirical Methods in Natural Language Processing*.

Li, Y.; Wei, F.; Zhang, C.; and Zhang, H. 2024c. EAGLE: Speculative Sampling Requires Rethinking Feature Uncertainty. In *International Conference on Machine Learning*.

Li, Y.; Wei, F.; Zhang, C.; and Zhang, H. 2025. EAGLE-3: Scaling up Inference Acceleration of Large Language Models via Training-Time Test.

Miao, X.; Oliaro, G.; Zhang, Z.; Cheng, X.; Wang, Z.; Zhang, Z.; Wong, R. Y. Y.; Zhu, A.; Yang, L.; Shi, X.; et al. 2024. Specinfer: Accelerating large language model serving with tree-based speculative inference and verification. In *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 3*, 932–949.

Nallapati, R.; Zhou, B.; dos Santos, C.; Gulcehre, C.; and Xiang, B. 2016. Abstractive Text Summarization using Sequence-to-sequence RNNs and Beyond. In *Proceedings of The 20th SIGNLL Conference on Computational Natural Language Learning*, 280.

OpenAI. 2023. GPT-4 Technical Report. *arXiv preprint arXiv:2303.08774*.

Reimers, N.; and Gurevych, I. 2019. Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks. 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)*, 3982–3992.

Schulman, J.; Wolski, F.; Dhariwal, P.; Radford, A.; and Klimov, O. 2017. Proximal Policy Optimization Algorithms.

Sun, Z.; Suresh, A. T.; Ro, J. H.; Beirami, A.; Jain, H.; and Yu, F. 2023. Spectr: Fast speculative decoding via optimal

transport. In *Advances in Neural Information Processing Systems*, volume 36, 30222–30242.

Sutton, R. S.; Barto, A. G.; et al. 1998. *Reinforcement learning: An introduction*, volume 1. MIT press Cambridge.

Taori, R.; Gulati, I.; Zhang, T.; Dubois, Y.; Guestrin, C.; Liang, P.; et al. 2023. Stanford Alpaca: An Instruction-following LLaMA model.

Touvron, H.; Lavril, T.; Izacard, G.; Martinet, X.; Lachaux, M.-A.; Lacroix, T.; Rozière, B.; Goyal, N.; Hambro, E.; Azhar, F.; et al. 2023. LLaMA: Open and Efficient Foundation Language Models. *arXiv preprint arXiv:2302.13971*.

Urlana, A.; Kumar, C. V.; Singh, A. K.; Garlapati, B. M.; Chalamala, S. R.; and Mishra, R. 2024. LLMs with Industrial Lens: Deciphering the Challenges and Prospects–A Survey. *arXiv preprint arXiv:2402.14558*.

Vaswani, A.; Shazeer, N.; Parmar, N.; Uszkoreit, J.; Jones, L.; Gomez, A. N.; Kaiser, Ł.; and Polosukhin, I. 2017. Attention is all you need. In *Advances in neural information processing systems*, volume 30.

Wang, G.; Cheng, S.; Zhan, X.; Li, X.; Song, S.; and Liu, Y. 2024. OpenChat: Advancing Open-source Language Models with Mixed-Quality Data. In *The Twelfth International Conference on Learning Representations*.

Wang, J.; Su, Y.; Li, J.; Xia, Q.; Ye, Z.; Duan, X.; Wang, Z.; and Zhang, M. 2025. OPT-Tree: Speculative Decoding with Adaptive Draft Tree Structure. In *Transactions of the Association for Computational Linguistics*, volume 13, 188–199.

Xiong, Y.; Zhang, R.; Li, Y.; and Zou, L. 2025. DySpec: Faster speculative decoding with dynamic token tree structure. *World Wide Web*, 28: 36.

Zhang, C.; Liu, Z.; and Song, D. 2024. Beyond the Speculative Game: A Survey of Speculative Execution in Large Language Models.

Zhang, L.; Wang, X.; Huang, Y.; and Xu, R. 2025. Learning Harmonized Representations for Speculative Sampling. In *The Thirteenth International Conference on Learning Representations*.

Zheng, H.; and Wang, X. 2025. Faster Speculative Decoding via Effective Draft Decoder with Pruned Candidate Tree. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, 9856–9868.

Zheng, L.; Chiang, W.-L.; Sheng, Y.; Zhuang, S.; Wu, Z.; Zhuang, Y.; Lin, Z.; Li, Z.; Li, D.; Xing, E.; et al. 2023. Judging llm-as-a-judge with mt-bench and chatbot arena. In *Advances in neural information processing systems*, volume 36, 46595–46623.

Zhong, S.; Yang, Z.; Gong, R.; Wang, R.; Huang, R.; and Li, M. 2024. Propd: Dynamic token tree pruning and generation for llm parallel decoding. In *Proceedings of the 43rd IEEE/ACM International Conference on Computer-Aided Design*, 1–8.