An Adaptive Drop-In Solution for Real-Time Speculative Decoding in Large Language Models

Anonymous ACL submission

Abstract

Large Language Models (LLMs) are cutting-edge generative AI models built on transformer architecture, which tend to be highly memory-intensive when performing real-time inference. Various strategies have been developed to enhance the end-to-end inference speed for LLMs, one of which is speculative decoding. This technique involves running a smaller LLM (draft model) for inference over a defined window size, denoted as γ, while simultaneously being validated by the larger LLM (target model). Choosing the optimal γ value and the draft model is essential for unlocking the potential of speculative decoding. But it is difficult to do due to the complicated influence from various factors, including the nature of the task, the hardware in use, and the combination of the large and small models. This paper introduces *on-the-fly adaption of speculative decoding*, a solution that dynamically adapts the choices to maximize the efficiency of speculative decoding for LLM inferences. As a drop-in solution, it needs no offline benchmarking or training. Experiments show that the solution can lead to 3.55-16.48% speed improvement over the standard speculative decoding, and 1.2-3.4× over the default LLMs.

1 Introduction

Large Language Models (LLMs) are state-of-the-art generative AI models built on transformer-based blocks (Brown et al., 2020; Ouyang et al., 2022). LLMs have an enormous number of parameters, and recent research not only focuses on training them efficiently but also explores how to optimize inference performance. In fact, there is evidence indicating that even small improvement in LLM inference speeds can result in significant cost savings. For instance, Google's infrastructure optimizations have demonstrated that improving inference efficiency can lead to substantial reductions in operational expenses. In large-scale deployments, a 1% increase in speed can indeed translate into millions of dollars saved (AI, 2023; Cloud, 2023).

Due to the autoregressive and memory-intensive nature of LLMs, it is challenging to optimize its inference throughput. Sampling for a new token depends on the previously generated tokens. Researchers are exploring mainly two approaches to circumvent this sequential dependence for more efficient parallel executions. One is to change the model architecture thus sampling granularity to parallelize the decoding process. Medusa (Cai et al., 2024), for example, introduces multiple decoding heads to generate tokens in parallel; Lookahead Decoding (Jacobi Decoding) (Fu et al., 2024) generates multiple tokens in parallel using nonlinear systems. This approach changes the neural architecture and hence requires new training, the high costs of which makes them difficult to adopt in practice. The other approach is speculative decoding (Leviathan et al., 2023; Chen et al., 2023). This approach first runs inference with a smaller LLM M_q, called the *draft model*, to generate the next γ tokens (γ is called speculation *window size*). After generating one window of tokens (called a *speculation step*), a *verification step* uses the Large LLM M_p, called the *target model*, to validate those tokens *in parallel*. Upon finding the first incorrect token, the execution throws away the rest of the tokens speculated by the draft model in that window and corrects the first rejected token (or appends a new token when all of the tokens are accepted). From there, it continues the speculation-validation process. This approach allows direct use of the pretrained LLMs, making it easier for adoption.

What is crucial for unlocking the potential of speculative decoding is to choose the best speculation window length, γ, and the best draft model to use. The best choices depend on the nature of the inference task, target model, software stack, hardware, and resource availability or workload changes (if running in a cloud). Suboptimal choices

may not only substantially throttle the benefits but sometimes cause slowdowns to the inference (see Section 6). The standard approach (Leviathan et al., 2023; Chen et al., 2023) relies on offline trial-and-error-based search, which not only takes long time, but more importantly, cannot adapt to the changes in the tasks, target models, software stacks, hardware or other runtime changes. A recent study, SpecDec++ (Huang et al., 2024), attempts to improve it through a machine learning model. It trains a ResNet with many samples in offline data collection, and uses it to predict, at each generated token in actual inferences, whether the execution should stop speculation, so as to adapt the speculation window. Although the work shows some improvement in experiments, it requires hundreds or thousands of GPU-hours (Section 6.2) to train the model for one target-draft pair on one kind of task and one software and hardware configuration. Modern LLM servers often host many LLMs and their variants (e.g., different quantizations, with Lora or other fine-tuning models) and have various software and hardware configurations and task types, making the solution difficult to adopt in production systems.

This paper describes the first-known exploration of *on-the-fly adaptive speculation*, a drop-in solution that adapts speculative decoding at runtime without ahead-of-time training. Our exploration covers both speculation window size γ and the choice of draft models. It experiments with several agile online methods for the adaptation, including a state machine-based mechanism, a cache-enabled state machine-based method, a reinforcement learning-based approach, and a token accuracy-based online window size optimization method. It analyzes these methods and evaluates them on four LLMs across three GPU models and four types of inference tasks. The results show that *on-the-fly adaptive speculation*, especially the online window size optimization, can deliver similar or even better improvements than the prior method that uses extensive ahead-of-time trainings, leading to 3.55-16.48% speed improvement over the standard speculative decoding, and 1.2-3.4$\times$ over the default LLMs. As a drop-in solution, this new approach needs no model changes, ahead-of-time preparation, lengthy training, or extensive benchmarking. It automatically adapts the optimal window size and directs the requests to the appropriate draft models for speculation, especially suitable for large LLM service providers.

It is worth mentioning that besides adapting the speculation process, there are some other methods explored in recent studies to improve speculative decoding (Li et al., 2024; Yan et al., 2024; Spector and Re, 2023; Hooper et al., 2023). Online Speculative Decoding (Liu et al., 2023), for instance, uses knowledge distillation to continuously train the smaller draft model during inference, enhancing performance. SpecInfer (Miao et al., 2023) introduces a tree-based decoding algorithm that uses the draft model to speculate multiple possible token sequences in parallel and then validates each of these sequences by the target model to keep the longest validated one. The *on-the-fly adaptive speculation* proposed in this current paper is from a different angle. It is hence complementary to those studies in the sense that it can be integrated into the speculation process in those solutions to further improve their effectiveness.

2 Guess-and-Verify in LLMs

In LLM inference, the tokens generated later are dependent on the tokens generated earlier. This sequential dependency of autoregressive decoding in LLMs has led to the development of new techniques aimed at parallelizing the decoding process. Given that text is tokenized, some tokens can be easier or harder to predict by a lower-parameter LLM. This has sparked a new area of research known as “guess-and-verify" optimization (Li et al., 2024; Yan et al., 2024; Spector and Re, 2023; Hooper et al., 2023). In this approach, smaller draft models efficiently guess a number of tokens, which are then verified in parallel by a larger target model. It is a lossless optimization, maintaining the accuracy of the results.

Speculative decoding is one typical “guess-and-verify” approach in LLM optimization. In this technique, when an LLM samples logits, it essentially predicts the probabilities of the next token. Speculative decoding takes advantage of this by allowing a smaller model to guess the easier tokens based on its own sampling of the distribution. These tokens are then verified by a larger, more accurate model.

In speculative decoding, the process involves guessing a set of tokens using the smaller model M_q within a fixed window size, γ, and then verifying these γ tokens using the larger model M_p by sampling $\gamma + 1$ tokens in parallel. If all tokens are accepted, the $\gamma + 1$ tokens are appended to the generated sequence, and the process continues. If one token (say $(i + 1)th$) is rejected, the algo-

rithm accepts the i correct tokens, resample the $(i+1)th$ from an adjusted distribution in the validation, and continues the next round of guessing. The speculation and verification process is detailed in Algorithm 1.

Algorithm 1 Speculative Decoding (Leviathan et al., 2023)

function speculativeDecoding(M_p, M_q, $prefix$)
  ▷ Sample y guesses $x_{1,\cdots,y}$ from M_q autoregressively.
  for $i = 1$ **to** y **do**
    $q_i(x) \sim M_q(prefix + [x_1, \cdots, x_{i-1}])$
    $x_i \sim q_i(x)$
  ▷ Run M_p in parallel.
  $(p_1(x), \cdots, p_{y+1}(x)) \leftarrow$
    $M_p(prefix), \cdots, M_p(prefix + [x_1, \cdots, x_y])$
  ▷ Determine the number of accepted guesses n.
  $r_1 \sim U(0,1), \cdots, r_y \sim U(0,1)$
  $n \leftarrow \min(\{i-1 | 1 \le i \le y, r_i > \frac{p_i(x)}{q_i(x)}\} \cup \{y\})$
  ▷ Adjust the distribution from M_p if needed.
  $p'(x) \leftarrow p_{n+1}(x)$
  if $n < y$ **then**
    $p'(x) \leftarrow norm(\max(0, p_{n+1}(x) - q_{n+1}(x)))$
  ▷ Return one token from M_p and n tokens from M_q.
  $t \sim p'(x)$
  return $prefix + [x_1, \cdots, x_n, t]$

3 Overview

Our goal is to enable real-time adjustments in speculative decoding to achieve higher throughput without requiring extensive pre-training, making it a practical solution for large-scale LLM deployments. Figure 1 gives an overview of our solution. The workflow goes as follows. At the beginning, it sets up the target model and different draft model options. For each prompt, our solution as in Figure 1 involves two steps. First, it finds a proper draft model for the given prompt. This is done by extracting features of the prompt to estimate the single token accuracy. From there, the method approximates the acceptance rate and ultimately the throughput so it can choose a proper draft model. Second, it runs speculations, where γ is adapted on the fly with the given model pairing. In the following content, we will first introduce the adaptive window size selection (Section 4) followed by adaptive draft model selection (Section 5).

4 Adaptive Window Size Selection

In this section, we focus on how to determine the best window size for a given target-draft model pair. We first introduce the analytic model for capturing the relationship between the speculation setting and speculation benefits. With that, we present an analytical model-guided adaption (Section 4.1) and three other agile algorithms for adaptively changing γ during speculative decoding (Section 4.2). The agility of these algorithms is essential for minimizing the runtime overhead.

4.1 Method 1: Analytical Model-Guided Adaption

A speculation window size that is too large risks high overhead if verification fails early, while a size that is too small misses out on the full benefits. The optimal size varies depending on the language model, contexts, and speculation accuracy. We translate this trade-off into an objective function to adaptively determine the optimal window size across various configurations. For each prompt, we want to minimize the end-to-end latency in generating a response with a fixed number of tokens. We define our objective function as the expected number of tokens verified as correct per unit time, aiming to maximize this function by optimizing the window size γ:

Definition 1 (formulating objective)**.** *Let a_q represent the latency of generating one token by the draft model, and $b_p(\gamma)$ represent the latency of a verification step with window size γ. For $t = 1, 2, \cdots$, let $Acc(x_t|X_{<t})$ be the accuracy of the speculation of a single token given the current context $X_{<t} = \{x_1, \cdots, x_{t-1}\}$. The window size γ for the current speculation step can be determined by optimizing the objective*

$$\mathcal{G} = \max_{\gamma} \frac{1 - Acc(x_t|X_{<t})^{\gamma+1}}{(1 - Acc(x_t|X_{<t}))(\gamma a_q + b_p(\gamma))}. \quad (1)$$

Given the single token accuracy $\beta = Acc(x_t|X_{<t}) \in [0,1]$, the expected accepted number of tokens in a γ-long speculation window follows truncated geometric distribution, and is given as $\frac{1-\beta^{\gamma+1}}{1-\beta}$ (see Appendix B.1). The total latency of one speculation step and verification step is calculated as $\gamma a_q + b_p(\gamma)$. Therefore, the expected number of tokens verified as correct per unit time given a window size γ is given by $\frac{1-\beta^{\gamma+1}}{(1-\beta)(\gamma a_q + b_p(\gamma))}$, and thus objective (1).

Algorithm. To adaptively determine the optimal γ, we need to figure out the unknown terms $a_q, b_p(\gamma), Acc(x_t|X_{<t})$ in Equation 1. Using estimation for them, the algorithm goes as follows. At the start of each speculation step, it conducts the following two operations before it can solve the objective (1). First, it estimates a and b. These values

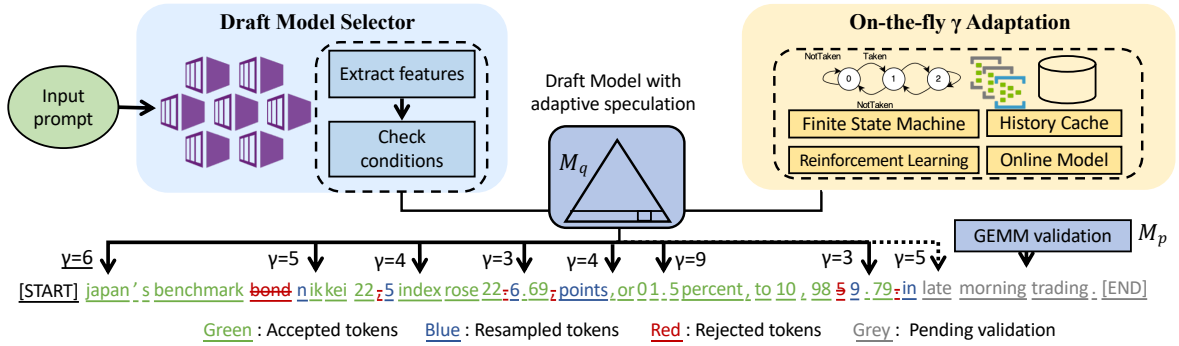

Figure 1: Our on-the-fly adaptive speculation framework. When a prompt arrives, our scheduler directs it to the draft model M_q. During speculation, our framework automatically adapts the right speculation window size γ. The speculation is then validated by the target model M_p.

are derived by observing the most recent steps. Second, it estimates $Acc(x_t|X_{<t})$ based on the recent history. We use maximum likelihood estimation over the last h speculations, ensuring the estimate $\widehat{Acc}$ reflects both locality and reduced variance (details in Appendix B.2). In our algorithm, we let $\gamma(j)$ be the speculation window size during the j-th most recent verification step, and $V(\gamma(j), X_{<t_j})$ the number of accepted tokens in this speculation window. We estimate $Acc(x_t|X_{<t})$ as

$$\frac{\sum_j V(\gamma(j), X_{<t_j})}{\sum_j V(\gamma(j), X_{<t_j}) + \sum_j \mathbf{1}(V(\gamma(j), X_{<t_j}) < \gamma(j))} \quad (2)$$

where $\mathbf{1}(\cdot)$ is the indicator function. To avoid overly optimistic estimates and potential division-by-zero error when $\widehat{Acc}$ gets close to 1, we set a fixed upper limit, $Acc_{\max}$ (e.g., 0.98), and cap $\widehat{Acc}$ at this value.

Analysis. Theorem 1 gives a direct comparison of the error bound of the analytical model-guided adaption and that of the fixed window size speculation, where the gamma value is determined from offline profiling data before real deployment, showing the superior theoretical results of our method in estimating the single token accuracy. The proofs are detailed in Section A.1.

Theorem 1. *Let β be the true acceptance probability of speculative decoding steps, and let $\widehat{\beta}_{adaptive}$ and $\widehat{\beta}_{fixed}$ be the estimators obtained from the analytical model-guided adaption and fixed window selection methods, respectively. Then the variance of the adaptive estimator satisfies:*

$$\mathrm{Var}(\widehat{\beta}_{adaptive}) \leq \mathrm{Var}(\widehat{\beta}_{fixed}).$$

Moreover, the expected absolute estimation error obeys:

$$\mathbb{E}\left[|\widehat{\beta}_{adaptive} - \beta|\right] \leq \mathbb{E}\left[|\widehat{\beta}_{fixed} - \beta|\right].$$

4.2 Other Drop-in Speculation Methods

Besides the analytic model-guided adaption, we have explored three other methods for on-the-fly γ adaption.

Method 2: Finite State Machine (FSM)-Based Speculation. A finite state machine-based predictor (Hennessy and Patterson, 2017) is similar to an n-bit saturating counter used in branch prediction. The mechanism works by decreasing γ by 1 if a token from the draft model is rejected, and increasing γ by 1 if all tokens are accepted. During benchmarking, we still select a value for γ, but it is considered an upper limit, $\gamma_{\max}$. If the draft and target models' distributions significantly differ, γ will remain low, potentially even at 0. Conversely, if the models align closely, γ should increase, approaching $\gamma_{\max}$. We consider this approach particularly effective for natural language processing because certain parts of a sentence—like common phrases or syntactically predictable structures—are easier for a smaller draft model to predict. In contrast, more unique or complex sub-sequences generated by the LLM might be harder to guess. By adaptively changing γ based on the previous token validations, we create a reward system that exploits patterns and predictable structures in autoregressive generation.

Method 3: Cache-Enabled FSM-Based Speculation. We adjust γ based on the context provided by the prompt and the history of generated tokens. In settings like question-answering, an LLM often reiterates or directly responds based on the context given by the user. Therefore, the user's input can inform predictions about the type of response the LLM will generate. Specifically, this approach includes a token cache that updates after every sam-

pling step. Initially, the cache is populated with tokens in the prompt, set up before the prefill stage. As new tokens are sampled and validated during speculation, the cache is updated with any previously unseen tokens. γ is then adjusted dynamically: It increases by one if a validated token is already in the cache, and by an additional one when all speculated tokens are accepted. Conversely, if none of the accepted tokens are in the cache, it decreases by one. We see that this approach is particularly effective for structured tasks like QA chatbot interactions or code completion, where context and history play a significant role. However, it may be less effective for short prompts expecting broad and diverse content, such as tasks that require informative or creative responses. In such cases, the lack of initial context or history means the cache is less informative, making γ adjustments less effective, potentially leading to performance similar to the more simplistic state-based adaptation.

Method 4: Reinforcement Learning-Based Speculation. We in addition explored a reinforcement learning-based approach. We use a Q-learning agent to choose γ. The modification to the algorithm is detailed in Algorithm 2 in Appendix B.4. The agent takes the previous states of γ as inputs and applies an action after each validation step.

5 Adaptive Draft Model Selection

Besides the speculation window size, the selection of the draft model also makes a difference: A smaller draft model can make faster inferences but at the risk of a low acceptance rate, while a larger draft model renders a longer latency. To dive deeper into the problem, we analyze the relationship between the throughput of the adaptive speculation and the acceptance rate in Theorem 2. Proofs are detailed in Appendix A.2.

Theorem 2. *Let L be the length of the answer to a prompt and is fixed, n be the total number of speculation steps. Let acceptance rate ρ_q be the number of accepted tokens divided by the total number of tokens sampled by M_q. The throughput (R) can be formulated as*

$$R = \frac{L}{b_p(\gamma)n + a_q\frac{L}{\rho_q}}. \tag{3}$$

As answer length L in Equation 3 is considered constant in our setting, the main influence for choosing a draft model comes from draft model latency a, target model latency $b(\gamma)$, the acceptance rate ρ, and the number of speculation steps n.

Influence of selecting a larger draft model. Let c represent the inference latency ratio between the draft model and the target model. Choosing a larger draft model increases the single token accuracy, $\alpha = \mathbb{E}(Acc(x_t|X_{<t}))$, and the draft latency a. We estimate $d = \mathbb{E}(\gamma)$ by finding the numerical integer solution in objective 1. With α and the corresponding d, the acceptance rate $\rho = \frac{1-\alpha^{\gamma+1}}{(1-\alpha)d}$ can be determined, as shown by scattered dots in Figure 2.

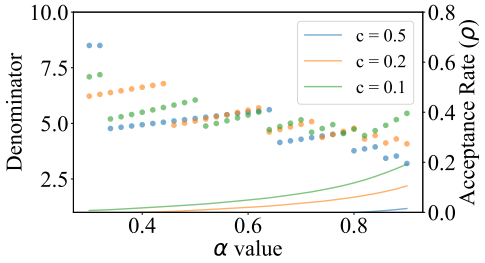

Figure 2: Results for the acceptance rate and the denominator in $n = \frac{L}{d\cdot\rho}$ across different single-token accuracy (α) and draft-to-target model size ratios (c).

We now analyze the influence on the number of speculation steps $n = \frac{L}{d\cdot\rho}$. The lines in Figure 2 illustrate how the denominator of n changes as α varies, reflecting the product of d and ρ.

Theorem 3. *Let Δn represent the reduction in speculation steps due to a larger draft model, Δc the increase in latency ratio, and $\Delta\rho$ the improvement in acceptance rate. As long as the following condition holds:*

$$\Delta n > \frac{\Delta c}{\Delta \rho} L, \tag{4}$$

the larger draft model would lead to a higher overall throughput than the smaller draft model.

A deeper look into formula 4 gives us that, when comparing two draft models, Δc can be easily determined using sample profiling results. If we are able to approximate the increase in α, $\Delta\rho$ and $\Delta n(\alpha, d)$ can also be determined because their relation to d, α, and c is deterministic. Therefore, to select a suitable draft model when a new prompt arrives, we need to approximate α and inspect whether condition 4 holds in order to determine whether to use a larger draft model.

Typically a prompt can be represented as a vector. We represent a prompt as a vector $\mathbf{u} \in \mathbb{R}^r$ with

$r > 0$ being the vector length. We model our goal $\alpha_c^{\mathbf{u}}$ of prompt $\mathbf{u}$ for a certain ratio c as

$$\alpha_c^{\mathbf{u}} = \mathbf{u}^\top \mathbf{Z}_c + \epsilon_c^{\mathbf{u}} \tag{5}$$

where $\mathbf{Z}_c$ is the parameter vector to determine and the random noise variable $\epsilon_c^{\mathbf{u}}$ is independent of $\mathbf{Z}_c$. For each $\mathbf{u} \in \mathbb{R}^r$, the random variables $\{\epsilon_c^{\mathbf{u}}\}$ are identically distributed with $\mathbb{E}(\epsilon_c^{\mathbf{u}}) = 0$ for all $\mathbf{u}$. The vector embedding is constructed as a concatenation of the prompt length, prompt perplexity and its TF-IDF score.

Algorithm. Based on the analysis, we devise the following algorithm to select draft model. Suppose there exist r linearly independent prompts $\mathbf{b}_1, \cdots, \mathbf{b}_r \in \mathbb{R}^r$. In the beginning, for each ratio c and these r prompts, the algorithm runs the speculative decoding and observes the single token accuracy $\alpha_c^{\mathbf{b}_p}$ and computes the ordinary least square estimate for $\mathbf{Z}_c$, given by

$$\widehat{\mathbf{Z}}_c = \left(\sum_{p=1}^{r} \mathbf{b}_p \mathbf{b}_p^\top \right)^{-1} \sum_{p=1}^{r} \mathbf{b}_p \alpha^{\mathbf{b}_p}.$$

For each newly arrived prompt $\mathbf{u}$, it computes the estimated $\hat{\alpha}_c^{\mathbf{u}}$ for potential draft-target model pairs and check Equation 4 to select the optimal draft model. In an LLM server center setting that has many machines hosting many LLMs, the selection of draft models can be implemented by redirecting requests to the appropriate nodes in the center equipped with the desired draft model and target model pair.

6 Evaluation

In this section, we present and analyze the experimental results gathered from testing our proposed algorithms and hypotheses.

6.1 Experimental Setups

This part outlines the configurations and setups used to collect the performance data.

Datasets and Models. we used three datasets to evaluate model performance and benchmark various implementations. These datasets were selected to reflect common tasks found in chatbot settings and other LLM applications. We employed system prompts to guide the LLMs for higher-quality outputs, particularly for tasks like coding and text summarization. See Appendix C.1 for more details. The datasets include OpenAI's HumanEval (Chen et al., 2021a) (CC-BY 4.0) for coding tasks, XSum for extreme text summarization (Chen et al., 2021b) (Apache 2.0), GSM8K (Cobbe et al., 2021) (MIT License) for mathematical reasoning, and Alpaca (Taori et al., 2023) (CC BY-NC 4.0) for complex advice queries. We include llama-2-chat 70B (Meta's Llama 2 Community License), Meta OPT 13B (MIT License), BigScience BLOOM 7B (RAIL License), and Dolly 12B (Databricks Open License) for target models. More details about the models we benchmarked are in Appendix C.1. Each dataset was sampled with 25 prompts in online predictive model construction, and evaluated with all remaining prompts across various settings. Note that when using speculative decoding, the draft model and the target model should have been trained on the same datasets to achieve good prediction accuracies, which limits the possible combinations in our experiments.

Platform. Table 4 in Appendix C.1 details the GPUs used, including memory bandwidth, capacity, and supported data types. For LLaMA 70B-7B and 70B-13B pairs, we use two NVIDIA A100 GPUs with 80GB memory each, distributing the 70B model across both GPUs. For other model pairs, we conduct our study using a single GPU, loading both the target and draft models on the same device.

6.2 Performance

We list in Table 1 the throughput results of adaptive window size selection for different model pairs on different hardware. The results of the online window optimization method are reported. We have the following observations. First, our method achieves a 2.07× speedup over autoregressive decoding and a 7.69% improvement over speculative baselines. Given that even a 1% speedup can save millions in large-scale LLM deployments (AI, 2023; Cloud, 2023), this improvement underscores the substantial impact of our approach. Second, our method achieves different speedups when benchmarking on different datasets. For the HumanEval dataset, speculative decoding has the potential to significantly accelerate performance due to the structured nature of programming languages, which follow stricter grammar and syntax compared to natural language. Repetitive patterns, such as `for` loops or `if-else` statements, are easier for the draft model to predict accurately. With adaptive speculation, the algorithm can adjust the parame-

ter γ dynamically to suit different sub-sequences. For instance, γ can be increased for predictable loops, whereas for more complex or less frequent constructs like API calls or high-level programming, γ can be reduced to improve the alignment between the draft and target models, minimizing token waste. Notably, conventional speculative decoding experiences a significant slowdown on the XSum dataset, highlighting a key limitation of speculative methods. In contrast, our approach dynamically adjusts the window size—sometimes reducing it to zero—effectively preventing slowdowns. As a result, we achieve a 70% throughput improvement on XSum, even though it provides no speedup over default LLMs without speculative decoding. Third, the ratio of model size matters when it comes to model pairing. Larger ratios generally lead to higher speedups while smaller target-draft parameter ratios such as BLOOM 7B-1B1 leave less room for improvement.

Table 1: Evaluation of adaptive window size selection. SPS denotes the throughput improvement our method achieves over the original speculative decoding. ARS denotes improvements over the default LLMs without speculative decoding. ("-" for not-runnable cases due to memory limit)

Model Pairing	Dataset	A100		V100		4090	
		SPS	ARS	SPS	ARS	SPS	ARS
LLaMA 70B/7B	finance-alpaca	6.43%	2.11×	-	-	-	-
LLaMA 70B/13B	finance-alpaca	4.89%	1.90×	-	-	-	-
BLOOM 7B/560M	finance-alpaca	4.28%	1.05×	7.69%	1.15×	3.70%	1.22×
BLOOM 7B/1B1	finance-alpaca	4.36%	1.04×	3.20%	1.15×	3.29%	1.17×
OPT 13B/125M	finance-alpaca	4.82%	2.32×	3.41%	3.4×	-	-
Dolly 12B/3B	finance-alpaca	9.11%	1.03×	-	-	-	-
LLaMA 70B/7B	humaneval	10.35%	2.41×	-	-	-	-
LLaMA 70B/13B	humaneval	8.53%	2.23×	-	-	-	-
BLOOM 7B/560M	humaneval	8.14%	1.04×	2.51%	1.09×	3.09%	1.25×
BLOOM 7B/1B1	humaneval	4.03%	1.1×	3.57%	1.16×	3.51%	1.3×
OPT 13B/125M	humaneval	11.40%	2.29×	2.15%	3.34×	-	-
Dolly 12B/3B	humaneval	15.20%	1.07×	-	-	-	-
LLaMA 70B/7B	gsm8k	7.13%	2.28×	-	-	-	-
LLaMA 70B/13B	gsm8k	9.66%	2.08×	-	-	-	-
BLOOM 7B/560M	gsm8k	15.03%	1.×	2.52%	1.01×	4.84%	1.18×
BLOOM 7B/1B1	gsm8k	10.70%	1.×	0.77%	1.02×	1.97%	1.19×
OPT 13B/125M	gsm8k	5.95%	2.24×	10.52%	3.36×	-	-
Dolly 12B/3B	gsm8k	16.92%	1.06×	-	-	-	-
LLaMA 70B/7B	xsum	2.94%	1.73×	-	-	-	-
LLaMA 70B/13B	xsum	0.14%	1.5×	-	-	-	-
BLOOM 7B/560M	xsum	77.50%	1.×	49.30%	1.×	54.63%	1.×
BLOOM 7B/1B1	xsum	70.91%	1.×	42.94%	1.×	54.17%	1.×
OPT 13B/125M	xsum	10.64%	1.02×	7.91%	2.43×	-	-

Next, we show the results of the draft model selection. This decision is made online for each prompt. Table 2 compares the speedups over the speculative decoding with and without draft model selection. For LLaMA 70B, the draft model currently includes LLaMA 7B and LLama 13B. For BLOOM 7B, the draft model includes BLOOM 560M, 1B1, and 1B7. The overall throughput speedups range from 3.55% to 16.48% using adaptive draft model selection.

We compare our online adaptive window size selection with SpecDec++ (Huang et al., 2024) in Table 3. SpecDec++ uses a ResNet to determine whether to stop speculation during speculative sampling at the current word predicted from the draft model. It employs this method based on its prediction of whether the next draft token will be accepted. Training this ResNet model requires conducting offline profiling runs and collecting data on the hardware (for example, 500 hours on A100-80G GPUs for training dataset generation, 400 hours for training, and 500 hours for evaluation set). To ensure a fair comparison, we employ the same setup from its original paper, using LLaMA-2-chat models (Touvron et al., 2023b). Specifically, we select the 7B version as the draft model and the 70B version as the target model for the A100 platform and BigScience BLOOM 560m version as the draft model and the 7B version as the target model for GTX 4090. To optimize memory usage, the models are implemented in the bfloat16 format. The tok/s speedups comparison is as follows on both the A100 and 4090 devices. We find that although our method uses no ahead-of-time training while SpecDec++ uses hundreds of GPU-hours to do that, our method outperforms SpecDec++ consistently, with an average of 5.7% improvement in latency. Part of the time savings come from selecting the γ value before each speculation instead of running a neural network each time the draft model produces a new token. Our approach further shows advancement by adaptively choosing γ on the fly without arduous data collecting and training.

Table 2: Throughput performance improvement over speculative decoding.

Target Model	finance-alpaca			humaneval			gsm8k		
	A100	V100	4090	A100	V100	4090	A100	V100	4090
LLaMA 70B (w/o draft selection)	6.43%	-	-	10.35%	-	-	9.66%	-	-
LLaMA 70B (w/ draft selection)	6.46%	-	-	11.11%	-	-	9.66%	-	-
BLOOM 7B (w/o draft selection)	4.36%	7.69%	3.70%	8.14%	3.57%	3.51%	9.76%	2.52%	4.84%
BLOOM 7B (w draft selection)	4.94%	16.48%	8.15%	8.57%	4.96%	4.17%	9.76%	3.55%	6.83%

Table 3: Comparison of Tok/s speedups (v.s. autoregressive) and productivity of SpecDec++ and our method (without draft model selection).

Dataset	A100 (LLaMA 70B/7B)		4090 (BLOOM 7B/560m)	
	SpecDec++	Ours	SpecDec++	Ours
Alpaca	2.04×	2.11×	1.21×	1.26×
HumanEval	2.23×	2.41×	1.22×	1.23×
GSM8K	2.26×	2.28×	1.17×	1.18×
Profile & Prepare	1000h	0	100h	0
Offline Training	400h		400h	

6.3 Detailed Analysis

We compare the throughput and acceptance rate for different adaptive speculation methods in Figure 3.

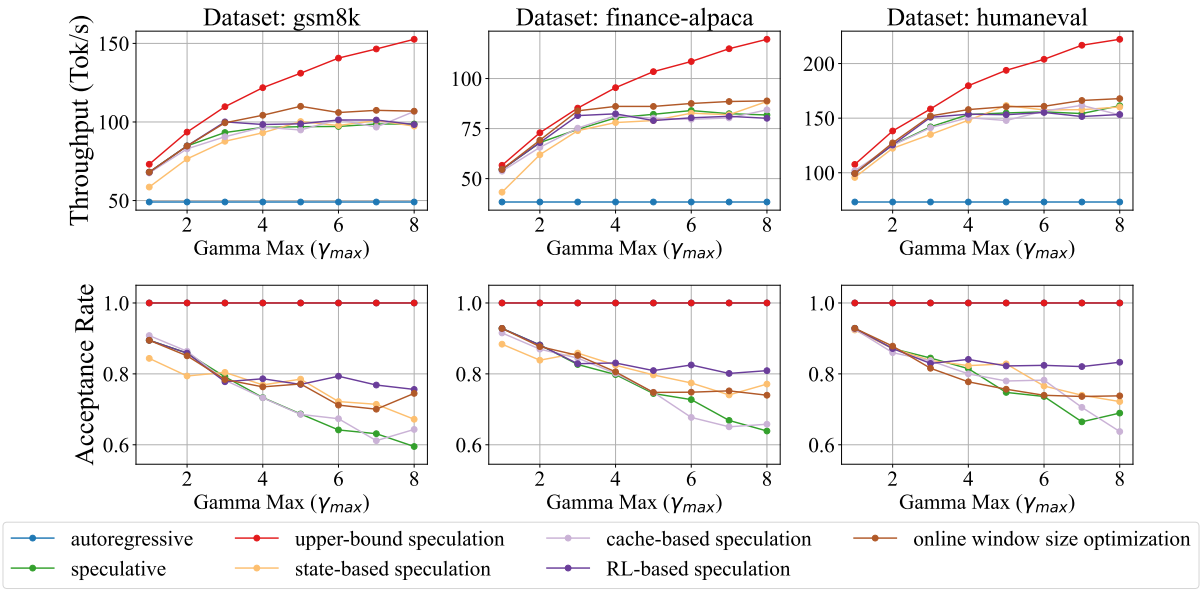

Figure 3: Detailed experimental results of different adaptive methods.

γ denotes the speculation window size for the original speculative decoding method and upper-bound speculation (simply by skipping the validation process); we set a maximum $\gamma_{\max}$ value for adaptive speculation methods, ensuring that γ will not exceed this value. All experiments are conducted on the A100 machine with OPT 13B-125M model pair. From the figure, we find that (i) the analytical model-guided online window size optimization method gives the best overall performance. (ii) Even though RL-based speculation gives better acceptance rates than the other methods, it shows lower throughput. This is because a higher acceptance rate is not directly linked to a higher throughput as in Equation 3. In our case, RL-based speculation remains at a low γ value to keep the acceptance rate high while also losing the potential for more speedups. (iii) cache-based and state-based speculation perform better when prompts are longer (e.g., the humaneval dataset). This can be attributed to a more stable γ prediction as more information is involved in the long prompt.

6.4 Results for Scalability

Comprehensive Chat Dataset. We include evaluations for a comprehensive chat dataset ShareGPT (Community, 2023) in Appendix C.3. Results show that our method achieves an average of $1.71\times$ speedups compared to original autoregressive decoding, and an additional 4.9% improvement over speculative decoding baselines.

Adaptive Speculation for Tree-based Decoding Method. Current speculative decoding uses tree-based methods (Cai et al., 2024; Li et al.). *On-the-fly adaptation of speculative decoding* is complementary to the tree-based decoding. By adaptively changing the draft tree depth, our drop-in method optimizes the draft token sequence length in real time, enhancing decoding performance. We apply our method to the state-of-the-art EAGLE-2 (Li et al., 2024) and report the results in Appendix C.4. On the MT-Bench (Zheng et al., 2023), we achieve up to $3.56\times$ speedups compared to original autoregressive decoding, and an additional 4.2% improvement over SOTA.

7 Conclusion

In this paper, we propose *on-the-fly adaptation for speculative decoding* to accelerate LLM inferences. As a pure software approach, it introduces a two-level adaptation for draft model adaptation and online window size adaptation with no ahead-of-time profiling or training, providing a drop-in optimization for existing LLMs. We experimentally demonstrate the effectiveness of this method and show 3.55% to 16.48% speedups compared to the speculative decoding, and $1.2\times$ to $3.4\times$ over the default LLMs without speculative decoding. Among the several online adaptive methods, we found that the token accuracy-based online window size optimization method works the best, consistently outperforming other methods in terms of the overall LLM throughput.

8 Limitation

This section discusses the limitations of the current work. The drop-in nature of our solution assumes compatibility with existing inference pipelines, but integration challenges may arise in specialized LLM deployments, such as those using custom hardware accelerators or distributed inference systems. Future work should explore more adaptive and model-agnostic strategies to further enhance the robustness and applicability of our approach.

References

Google AI. 2023. Llm inference api - google ai mediapipe solutions. Accessed: 2024-09-15.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel Ziegler, Jeffrey Wu, Clemens Winter, Chris Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. 2020. Language models are few-shot learners. In *Advances in Neural Information Processing Systems*, volume 33, pages 1877–1901. Curran Associates, Inc.

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. 2023. Accelerating large language model decoding with speculative sampling. *arXiv preprint arXiv:2302.01318*.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde De Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, et al. 2021a. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*.

Yulong Chen, Yang Liu, Liang Chen, and Yue Zhang. 2021b. Dialogsum: A real-life scenario dialogue summarization dataset. *arXiv preprint arXiv:2105.06762*.

Google Cloud. 2023. Accelerating ai inference with google cloud tpus and gpus. Accessed: 2024-09-15.

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

ShareGPT Community. 2023. Sharegpt: A platform for sharing gpt conversations. https://sharegpt.com/. Accessed: 2024-11-16.

Mike Conover, Matt Hayes, Ankit Mathur, Jianwei Xie, Jun Wan, Sam Shah, Ali Ghodsi, Patrick Wendell, Matei Zaharia, and Reynold Xin. 2023. Free dolly: Introducing the world's first truly open instruction-tuned llm. *Company Blog of Databricks*.

Yichao Fu, Peter Bailis, Ion Stoica, and Hao Zhang. 2024. Break the sequential dependency of llm inference using lookahead decoding. *arXiv preprint arXiv:2402.02057*.

John L. Hennessy and David A. Patterson. 2017. *Computer Architecture, Sixth Edition: A Quantitative Approach*, 6th edition. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

Geoffrey Hinton. 2015. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*.

Coleman Hooper, Sehoon Kim, Hiva Mohammadzadeh, Hasan Genc, Kurt Keutzer, Amir Gholami, and Sophia Shao. 2023. Speed: Speculative pipelined execution for efficient decoding. *arXiv preprint arXiv:2310.12072*.

Kaixuan Huang, Xudong Guo, and Mengdi Wang. 2024. Specdec++: Boosting speculative decoding via adaptive candidate lengths. *arXiv preprint arXiv:2405.19715*.

Daniel A Jiménez and Calvin Lin. 2001. Dynamic branch prediction with perceptrons. In *Proceedings HPCA Seventh International Symposium on High-Performance Computer Architecture*, pages 197–206. IEEE.

Chih-Chieh Lee, I-CK Chen, and Trevor N Mudge. 1997. The bi-mode branch predictor. In *Proceedings of 30th Annual International Symposium on Microarchitecture*, pages 4–13. IEEE.

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

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

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

Xiaoxuan Liu, Lanxiang Hu, Peter Bailis, Ion Stoica, Zhijie Deng, Alvin Cheung, and Hao Zhang. 2023. Online speculative decoding. *arXiv preprint arXiv:2310.07177*.

Xupeng Miao, G Oliaro, Z Zhang, X Cheng, Z Wang, RYY Wong, A Zhu, L Yang, X Shi, C Shi, et al. 2023. Specinfer: Accelerating generative large language model serving with speculative inference and token tree verification. *arXiv preprint arXiv:2305.09781*.

Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, John Schulman, Jacob Hilton, Fraser Kelton, Luke Miller, Maddie Simens, Amanda Askell, Peter Welinder, Paul F Christiano, Jan Leike, and Ryan Lowe. 2022. Training language models to follow instructions with human feedback. In *Advances in Neural Information Processing Systems*, volume 35, pages 27730–27744. Curran Associates, Inc.

Mike Schuster and Kaisuke Nakajima. 2012. Japanese and korean voice search. In *2012 IEEE international conference on acoustics, speech and signal processing (ICASSP)*, pages 5149–5152. IEEE.

Rico Sennrich. 2015. Neural machine translation of rare words with subword units. *arXiv preprint arXiv:1508.07909*.

James E Smith. 1998. A study of branch prediction strategies. In *25 years of the international symposia on Computer architecture (selected papers)*, pages 202–215.

Benjamin Spector and Chris Re. 2023. Accelerating llm inference with staged speculative decoding. *arXiv preprint arXiv:2308.04623*.

Rohan Taori, Ishaan Gulrajani, Tianyi Zhang, Yann Dubois, Xuechen Li, Carlos Guestrin, Percy Liang, and Tatsunori B Hashimoto. 2023. Stanford alpaca: An instruction-following llama model.

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

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

Minghao Yan, Saurabh Agarwal, and Shivaram Venkataraman. 2024. Decoding speculative decoding. *arXiv preprint arXiv:2402.01528*.

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric Xing, et al. 2023. Judging llm-as-a-judge with mt-bench and chatbot arena. *Advances in Neural Information Processing Systems*, 36:46595–46623.

A Proof

A.1 Proof of Theorem 1.

Proof. Since speculative decoding terminates upon the first failure in a verification window, the number of accepted tokens $V(\gamma, X_{<t})$ follows a truncated geometric distribution:

$$P(V = k) = (1-p)^k p, \quad k \in \{0, 1, \ldots, \gamma - 1\}.$$

Thus, for a fixed window size γ, the failure probability at each step is:

$$q = 1 - (1-p)^\gamma.$$

The number of failures F in N verification steps follows a binomial distribution:

$$F \sim \text{Binomial}\left(N, q\right).$$

For small p, we approximate:

$$q \approx \gamma p.$$

By the Poisson limit theorem, for large N, the failure count can be approximated by:

$$F \sim \text{Poisson}(N\gamma p).$$

Now, both adaptive and fixed methods estimate p using the maximum likelihood estimator:

$$\widehat{p} = \frac{F}{S+F}.$$

Applying the delta method, we approximate the variance of $\widehat{p}$:

$$\text{Var}(\widehat{p}) \approx \frac{p(1-p)}{(S+F)^2}.$$

In the fixed case, the total number of observed tokens is:

$$S_{\text{fixed}} + F_{\text{fixed}} = N\gamma.$$

For the adaptive method, where $\gamma(j)$ is adjusted dynamically, we have:

$$S_{\text{adaptive}} + F_{\text{adaptive}} \geq N\gamma.$$

Thus,

$$\text{Var}(\widehat{p}_{\text{adaptive}}) \leq \text{Var}(\widehat{p}_{\text{fixed}}).$$

Using Hoeffding's inequality,

$$P\left(|\widehat{p} - p| \geq \epsilon\right) \leq 2\exp\left(-2\epsilon^2(S+F)\right),$$

we conclude that the adaptive method has a tighter error bound:

$$P\left(|\widehat{p}_{\text{adaptive}} - p| \geq \epsilon\right) \leq P\left(|\widehat{p}_{\text{fixed}} - p| \geq \epsilon\right).$$

Thus, the expected absolute error is also smaller:

$$\mathbb{E}\left[|\widehat{p}_{\text{adaptive}} - p|\right] \leq \mathbb{E}\left[|\widehat{p}_{\text{fixed}} - p|\right].$$

This completes the proof. □

A.2 Proof of Theorem 2.

Proof. Let $\{\gamma_q^i\}_{i=1}^n$ denote the history of the window sizes during the adaptive speculation and $d_q = \mathbb{E}_{i=1,\cdots,n}(\gamma_q^i)$ be the average window size during speculation. In the following formulations, we omit p and q as the formulations are about a given p and q pair. The throughput R is computed by dividing the length of the answer by the latency t:

$$R = \frac{L}{t}. \tag{6}$$

The total latency of generating outputs for one prompt is computed as

$$\begin{aligned} t &= \sum_{i=1}^{n} a\gamma^i + b(\gamma) = b(\gamma)n + a\sum_{i=1}^{n}\gamma^i \\ &= n(b(\gamma) + a \cdot \mathbb{E}(\gamma^i)). \end{aligned} \tag{7}$$

Inspecting the relations among d, n, ρ and L gives us

$$L = d \cdot n \cdot \rho. \tag{8}$$

Solving for Equations 6, 7 and 8 gives us the expression for throughput. □

B Method Details

B.1 Formulation of Objective 1

This section discusses details on formulating objective 1.

Expected Accepted Token Length. Given the single token accuracy $\beta = Acc(x_t|X_{<t}) \in [0, 1]$, the expected accepted number of tokens is computed as:

$$\begin{aligned} &\mathbb{E}(\#\text{ of accepted tokens}|X_{<t}) \\ &= 1 + \sum_{i=1}^{\gamma-1} i\beta^i(1-\beta) + \gamma\beta^\gamma \\ &= 1 + \sum_{i=1}^{\gamma-1} i\beta^i - \sum_{i=2}^{\gamma}(i-1)\beta^i + \gamma\beta^\gamma \\ &= \sum_{i=0}^{\gamma}\beta^i \\ &= \frac{1-\beta^{\gamma+1}}{1-\beta}. \end{aligned} \tag{9}$$

Formulation of objective. The expected number of verified tokens as correct in a γ-long speculation window is $\frac{1-Acc(x_t|X_{<t})^{\gamma+1}}{1-Acc(x_t|X_{<t})}$. The total latency of one speculation step and verification step is calculated as $\gamma a_q + b_p$. Therefore, the expected number of tokens verified as correct per unit time given a window size γ is

$$\frac{1 - Acc(x_t|X_{<t})^{\gamma+1}}{(1 - Acc(x_t|X_{<t}))(\gamma a_q + b_p)}.$$

B.2 Estimation of $Acc(x_t|X_{<t})$

Let $\beta = Acc(x_t|X_{<t})$. Let Y be a random variable of the number of accepted tokens truncated at $\gamma+1$. The probability function of Y is

$$f(y) = \begin{cases} \dfrac{(1-\beta)\beta^{y-1}}{1-\beta^{\gamma+1}}, & y = 1, 2, 3, \cdots, \gamma+1 \\ 0, & \text{otherwise.} \end{cases} \tag{10}$$

Maximum Likelihood Estimation. For a random sample of size n, the likelihood function is

$$L = (1-\beta^{\gamma+1})^{-n}(1-\beta)^n\beta^{\sum_{i=1}^n y - n}.$$

The following equation 11, a $(\gamma+2)$th-degree polynomial in $\hat{\beta}$, provides the maximum likelihood estimator for β.

$$\begin{aligned} &\left(\sum_{i=1}^{n} y_i - n(\gamma+2) + n\right)\hat{\beta}^{\gamma+2} + \left(n(\gamma+2) - \sum_{i=1}^{n} y_i\right)\hat{\beta}^{\gamma+1} \\ &- \left(\sum_{i=1}^{n} y_i\right)\hat{\beta} + \sum_{i=1}^{n} y_i - n = 0 \end{aligned} \tag{11}$$

Given values of γ, n, and $\sum_{i=1}^n y_i = n\overline{y}$, one can compute the value of $\hat{\beta}$ using an iterative technique such as the Newton-Rhapson method to solve equation 11. It can be shown that there is only one root in the range $0 < \hat{\beta} < 1$.

To eliminate the need for an iterative solution to equation 11, we maintain a table to provide approximate solutions. From equation 11,

$$\overline{y} = \{\gamma\hat{\beta}^{\gamma+2} - (\gamma+2)\hat{\beta}^{\gamma+1} + 1\}/(\hat{\beta}^{\gamma+2} - \hat{\beta}^{\gamma+1} - \hat{\beta} + 1). \tag{12}$$

Further observation gives us that the rate of change of $\overline{y}$ concerning $\hat{\beta}$ appears to be sufficiently constant, making linear interpolation feasible and enabling our approximation in equation 2.

B.3 Optimal Gamma

Given the single token accuracy and inference latency ratio of the draft model to the target model c, the optimal γ value to optimize objective 1 can be determined as in Figure 4.

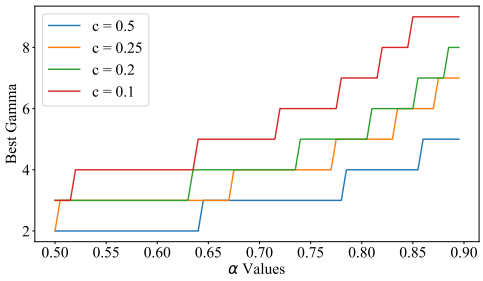

Figure 4: The optimal γ for different α and c values.

Algorithm 2 Reinforcement Learning-Based Speculative

function reinforcementLearningSpeculation(M_p, M_q, $prefix$, Agent)
  ▷ Sample y guesses $x_1, \cdots, x_y$ from M_q autoregressively.
  for $i = 1$ **to** y **do**
    $q_i(x) \sim M_q(prefix + [x_1, \cdots, x_{i-1}])$
    $x_i \sim q_i(x)$
  ▷ Run M_p in parallel.
  $(p_1(x), \cdots, p_{y+1}(x)) \leftarrow$
    $M_p(prefix), \cdots, M_p(prefix + [x_1, \cdots, x_y])$
  ▷ Determine the number of accepted guesses n.
  $r_1 \sim U(0,1), \cdots, r_y \sim U(0,1)$
  $n \leftarrow \min(\{i-1 | 1 \leq i \leq y, r_i > \frac{p_i(x)}{q_i(x)}\} \cup \{y\})$
  ▷ Adjust the distribution from M_p if needed.
  $p'(x) \leftarrow p_{n+1}(x)$
  if $n < y$ **then**
    $p'(x) \leftarrow \mathcal{N}(\max(0, p_{n+1}(x) - q_{n+1}(x)))$
  action ← GetAction(Agent, y)
  y = action
  Reward = the percentage of the accepted speculated tokens
  ▷ Return one token from M_p and n tokens from M_q.
  $t \sim p'(x)$
  return $prefix + [x_1, \cdots, x_n, t]$

B.4 Psudo-code for Reinforcement learning-based speculation

Algorithm 2 detailed the reinforcement learning-based speculation.

C Additional Experiments

This section includes additional experimental results.

C.1 Additional Experimental setups

Software. We primarily use the HuggingFace Transformers library with PyTorch implementations. The flexibility of Python and the availability of pre-trained weights on HuggingFace allow us to experiment with various methods and conduct detailed analyses. The GPU implementations utilize NVIDIA's cuDNN library, which is optimized for large language models (LLMs) and transformers. To ensure the best performance and compatibility with the latest models, we use the most recent versions of Transformers (v4.38.2) and PyTorch (v2.2.1).

Hardware. LLMs demand significant GPU computing power and memory, particularly during inference, where memory bandwidth is critical in achieving high throughput on GPUs. Table 4 lists the GPUs we used, their memory bandwidth, capacity, and the datatypes employed.

Table 4: GPU Hardware

GPU	HBM (GB)	Mem Bandwidth (GB/s)	Datatype
NVIDIA V100	32	900	FP16
NVIDIA A100	80	1555	BF16
NVIDIA RTX4090	24	1008	BF16

We use two NVIDIA A100 GPUs with 80G memory for the LLaMA 70B-7B pair and 70B-13B pair. We distribute the 70B model across two GPUs, which leads to communication overhead during inference with LLaMA 70B. However, for speculative decoding, the 7B (13B) draft model is only loaded on a single GPU, reducing this overhead. For other model pairs, we limit our study to one GPU, loading both the target and draft models on a single device. This approach serves two purposes. First, it allows us to explore the effects of resource constraints on a single GPU, which is relevant for future work on speculative decoding for personal devices. Second, it maximizes efficiency, as splitting a small LLM onto one GPU and a large LLM onto another would underutilize the resources; it is more effective to run both models on a single GPU.

Prompt Dataset. Table 5 consolidates information on the datasets, tasks, and additional details we used to benchmark and compare performance.

Table 5: Prompt Dataset

Dataset	Task	System Prompt
OpenAI HumanEval	Code completion	You are an expert programmer that helps to complete Python code. Given the code from the user, please complete the rest of the code to the best of your ability.
XSum	Summarization	You write two sentence summaries of new articles. Do not write any more. Keep it brief and to the point.
GSM8K	Math Word Problem	You are given a math question, and your task is to answer it. Then provide a step-by-step walkthrough on how you got the answer to the question.
Finance-Alpaca	Finance QA	You are a finance expert. Answer the following questions to the best of your knowledge, and explain as much as possible.

Models. When implementing speculative decoding, selecting appropriate model pairs presents challenges. The parameter ratio is crucial, as a low ratio can negate speed gains if the draft model isn't significantly faster than the target model. Additionally, both models must share the same tokenizer to avoid conversion overhead from differing tokenization

schemes (Schuster and Nakajima, 2012; Sennrich, 2015). Speculative decoding is more effective with models trained on similar datasets, as seen with Meta's LLaMA models (Touvron et al., 2023b,a) or DeepMind's Chinchilla. Mixed precision (FP16 or BF16) is preferred, avoiding quantization due to slowdowns, and using deterministic decoding with a temperature of 0 for consistency (Hinton, 2015). Dolly is an open-source model from Databricks aimed at democratizing LLMs by offering open-source weights and the datasets needed for instruction fine-tuning (Conover et al., 2023). The following table 6 details the model pairs.

Table 6: Model Card

Target Model	Draft Model	Same Vendor?	Ratio
Meta LLaMA 70B	Meta LLaMA 13B	Yes	5.4x
Meta LLaMA 70B	Meta LLaMA 7B	Yes	10x
BigScience BLOOM 7B	BigScience BLOOM 560M	Yes	12.5x
BigScience BLOOM 7B	BigScience BLOOM 1.1B	Yes	7x
Meta OPT 13B	Meta OPT 125M	Yes	96.3x
DataBricks Dolly 12B	DataBricks Dolly 3B	Yes	4.0x

Implementation Details. The FSM-based method and cache-enabled FSM-based method are inspired by branch prediction in computer architecture (Lee et al., 1997; Smith, 1998; Jiménez and Lin, 2001). The reinforcement learning-based speculation involves online learning, so we conducted 25 warmup trials before recording benchmarks. To minimize overhead, the RL algorithm runs on the CPU rather than the GPU, ensuring both inference and training are completed in under 1 millisecond. This makes the overhead negligible when considering the end-to-end latencies compared to standard speculative decoding. AI assistants are used for refining the writing.

C.2 Additional Experiment Results

We include more experiment results. Figure 5 and Figure 6 compare the throughput and acceptance rate for different adaptive speculation methods on the A100 machine with the BLOOM BigScience 7B-560M model pair and LLaMA 70B-7B.

C.3 Comprehensive chat dataset

Table 7 shows the throughput results of adaptive window size selection for different model pairs on different hardware on the shareGPT dataset. The results of the online window optimization methods are reported. The experimental setups are the same as in Section 6.1.

Table 7: Evaluation for the comprehensive chat dataset. SPS denotes the throughput improvement our method achieves over the original speculative decoding. ARS denotes improvements over the default LLMs without speculative decoding.

Hardware	Model Pairing	Dataset	Throughput	
			SPS	ARS
A100	LLaMA 70B/7B	shareGPT	7.89%	2.20×
	LLaMA 70B/13B	shareGPT	3.69%	1.92×
	OPT 13B/125M	shareGPT	4.81%	2.10×
4090	BLOOM 7B/560M	shareGPT	4.58%	1.18×
	BLOOM 7B/1B1	shareGPT	3.50%	1.18×

C.4 Adaptive Speculation for Tree-based Decoding

We implemented our *on-the-fly adaption of speculative decoding* on top of EAGLE-2, dynamically adjusting the draft tree depth (γ) during decoding. For different γ, sequence lengths for different branches of the draft tree are determined using the same expansion and rerank decision process as in the original EAGLE-2. Specifically, the tree depth dynamically changes for each speculation step; For a certain γ in one speculation step, the algorithm first enters the expansion phase: At each layer of the tree, we select the top k nodes with the highest probabilities and expand draft sequences based on these nodes. The longest draft sequence in the tree corresponds to the dynamically determined depth γ. Once the expansion is complete up to the dynamically determined the γ-th layer, we apply a rerank step to select the same number of tokens from the draft tree as in EAGLE-2 and validate the corresponding draft sequences.

Table 8 shows the results of adaptive tree depth selection on EAGLE-2 for different model pairs on different hardware for MT-Bench. The experimental setups are the same as in Section 6.1. We achieve up to 3.56× speedups compared to original autoregressive decoding, and an additional 4.2% improvement over EAGLE-2. We also achieve speedups of up to 4.25×, 3.75×, and 3.85× compared to original autoregressive decoding on the A100 machine for HumanEval, GSM8K, and Alpaca, respectively, with improvements of 4.27%, 5.65%, and 3.83% over EAGLE-2. On the 4090 machine, for HumanEval, GSM8K, and Alpaca, we achieve speedups of up to 2.72×, 3.27×, and 2.52× compared to original autoregressive decoding, respectively, with improvements of 6.23%, 2.55%, and 2.92% over EAGLE-2.

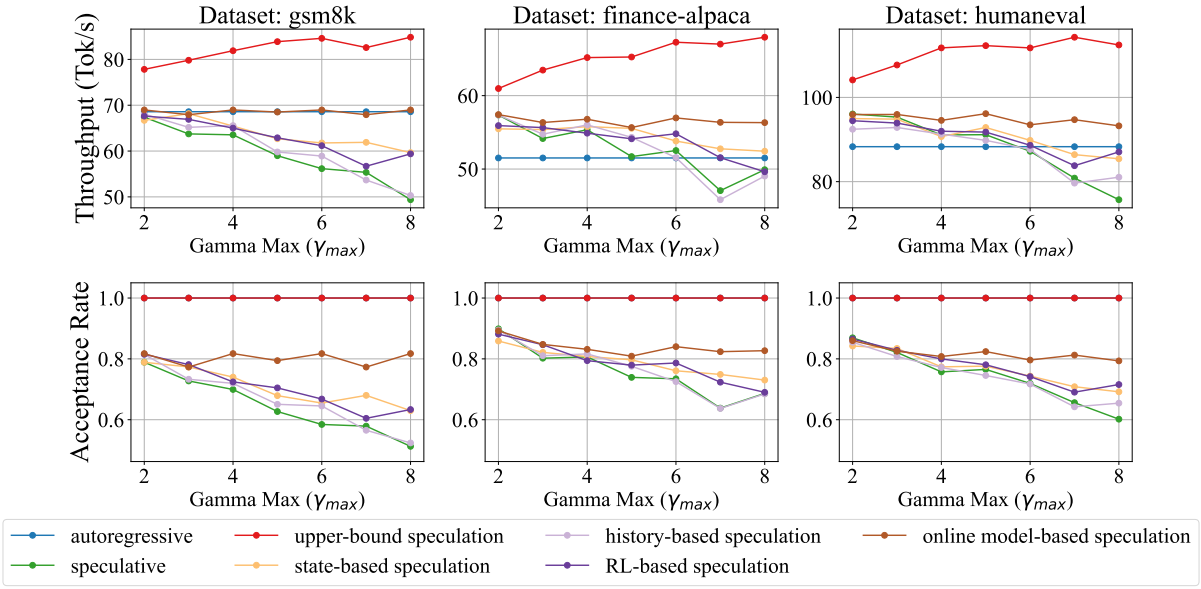

Figure 5: Detailed experimental results for BLOOM 7B-560M.

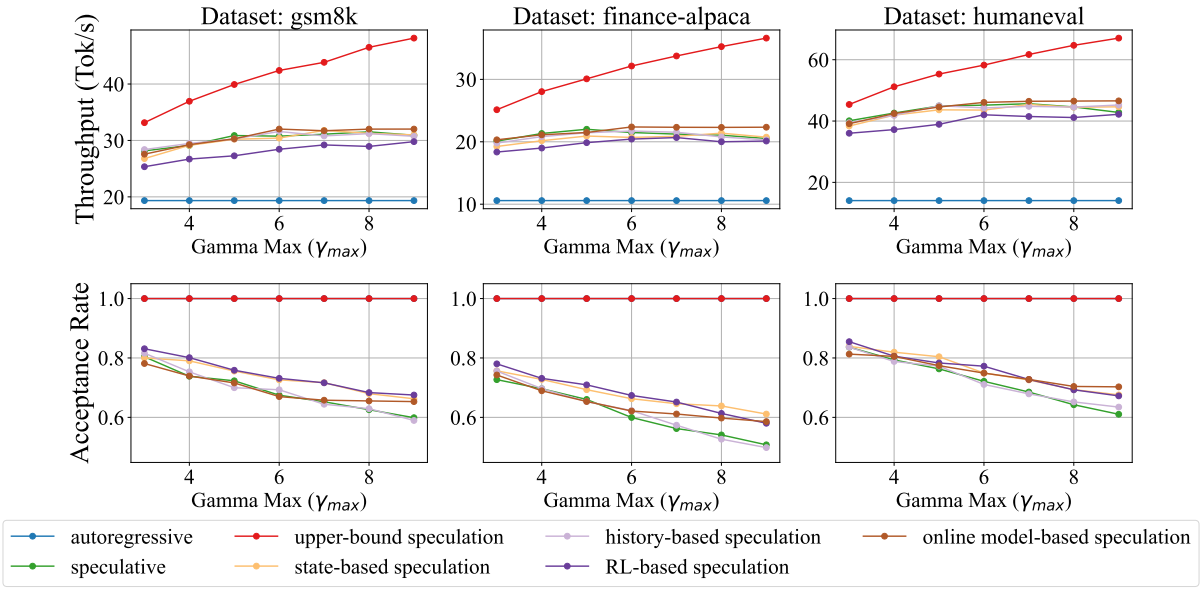

Figure 6: Detailed experimental results for LLaMA 70B-7B.

In addition, Table 9 provides a detailed analysis of serving latency, speculation latency, verification latency, and speculation accuracy. Speculation latency is measured as the number of tokens selected from the draft tree per second. Our method shows lower speculation latency compared to EAGLE-2. This is because, while we dynamically adapt the tree depth, we keep the number of tokens selected from the draft tree the same as in EAGLE-2. However, with a larger tree depth, more tokens might sampled due to the increased number of layers. Verification latency is similar for both EAGLE-2 and our method, as they utilize the same target model. Notably, our method improves the acceptance rate by dynamically adjusting the tree depth, which effectively changes the speculation window size.

C.5 Sensitivity Study

Effects of different history length. Table 10 shows a sensitivity study for the effects of different history lengths when adjusting the window size. The results are collected on the A100 machine for the BLOOM 7B-560M pair.

Effects of vector length. Table shows the sensitivity study for the effects of different vector dimensions for model selection. The results are collected

Table 8: Evaluation for adaptive speculation in improving EAGLE-2, a method for tree-based speculative decoding. SPS denotes the throughput improvement our method achieves over EAGLE-2. ARS denotes improvements over the default LLMs without speculative decoding. ("-": model is out of memory)

Target Model	Dataset	A100		4090	
		SPS	ARS	SPS	ARS
Vicuna-7B-v1.3	MTBench	7.07%	3.21×	6.22%	2.28×
LLaMA2-Chat 7B	MTBench	3.37%	3.29×	6.23%	2.72×
LLaMA2-Chat 13B	MTBench	2.55%	4.01×	-	-
LLaMA2-Chat 70B	MTBench	1.46%	3.56×	-	-
LLaMA3-Inst 70B	MTBench	1.14%	2.68×	-	-

Table 9: Detailed analysis for adaptive speculation in improving EAGLE-2. Data are collected on the MT-Bench. "Speculation" and "Verification" denote speculation throughput and verification throughput, respectively. (Unit for throughput: Toks/sec)

Hardware	Target Model	Method	Serving	Speculation	Verification	Acceptance Rate
A100	Vicuna-7B-v1.3	EAGLE-2	82.44	472.58	708.31	0.67
		Ours	85.27	446.71	666.01	0.72
	LLaMA2-Chat 7B	EAGLE-2	97.81	569.05	4491.42	0.62
		Ours	100.41	299.85	4591.73	0.66
	LLaMA2-Chat 13B	EAGLE-2	79.74	558.02	4535.35	0.61
		Ours	81.51	491.37	4530.73	0.62
	LLaMA2-Chat 70B	EAGLE-2	27.50	389.38	4532.27	0.51
		Ours	27.90	192.02	4491.19	0.65
	LLaMA3-Inst 70B	EAGLE-2	24.33	266.33	3392.65	0.53
		Ours	24.61	132.08	3300.60	0.65
4090	Vicuna-7B-v1.3	EAGLE-2	117.95	665.97	1041.83	0.56
		Ours	125.28	579.34	1164.03	0.56
	LLaMA2-Chat 7B	EAGLE-2	142.15	712.72	8278.28	0.67
		Ours	151.00	643.91	8137.47	0.72

Table 10: Sensitivity study for different history length values when adjusting window size. The best throughput is highlighted for each $\gamma_{\max}$.

Dataset	History Length	$\gamma_{\max}$			
		5	6	7	8
Alpaca	5	52.28	52.49	52.05	52.37
	6	**54.18**	**53.46**	52.71	53.00
	7	53.03	53.01	**54.32**	**53.36**
Humaneval	5	93.98	94.48	**94.21**	**94.21**
	6	**94.84**	**95.18**	93.39	93.39
	7	94.54	94.30	93.41	93.41
gsm8k	5	62.69	**62.15**	63.42	64.03
	6	61.48	61.78	**63.72**	61.84
	7	**64.77**	61.38	62.74	**64.27**

on the 4090 machine for the BLOOM 7B-560M pair.

Table 11: Sensitivity study for different dimensions for model selection.

Dimension	4	8	10	12	16
Throughput	74.29	75.23	74.00	75.46	**75.55**