AdaSD: Adaptive Speculative Decoding for Efficient Language Model Inference

Anonymous ACL submission

Abstract

Large language models (LLMs) have achieved remarkable performance across a wide range of tasks, but their increasing parameter sizes significantly slow down inference. Speculative decoding mitigates this issue by leveraging a smaller draft model to predict candidate tokens, which are then verified by a larger target model. However, existing approaches often require additional training, extensive hyperparameter tuning, or prior analysis of models and tasks before deployment. In this paper, we propose Adaptive Speculative Decoding (AdaSD), a hyperparameter-free decoding scheme that dynamically adjusts generation length and acceptance criteria during inference. AdaSD introduces two adaptive components: one to determine when to stop candidate token generation and the other to decide token acceptance, updated in real time based on token entropy and Jensen-Shannon distance. This approach eliminates the need for pre-analysis or fine-tuning and is compatible with off-the-shelf models. Experiments on benchmark datasets demonstrate that AdaSD achieves up to 1.46× speedup over vanilla speculative decoding while limiting accuracy degradation to under 1.8%, making it a practical solution for efficient and adaptive LLM inference.

1 Introduction

Large language models (LLMs) have rapidly become integral to a wide range of applications, including natural language understanding, content generation, and code synthesis. This progress is driven by the advances of the Transformer architecture (Vaswani et al., 2017). Representative models such as T5 (Raffel et al., 2020), GPT (Brown et al., 2020), Gopher (Rae et al., 2022), and PaLM (Chowdhery et al., 2023) have successively advanced the state of the art, demonstrating that scaling up model parameters can substantially improve performance across many real-world tasks.

Despite these remarkable performance gains, increasing model size also introduces substantial inference overhead. For example, Meta's flagship model Llama 3 (Grattafiori et al., 2024) achieves 87.3% accuracy on the MMLU general knowledge benchmark with its 405 billion parameters. However, at this massive scale, inference becomes memory-bound, as every forward pass requires loading the entire set of model weights from memory, which significantly slows down computation and poses challenges for practical deployment.

Speculative decoding (Leviathan et al., 2023) is a technique to alleviate the memory-bound bottleneck by maximizing the utilization of each weight load during inference. It uses a smaller and faster, but less accurate, *draft model* to generate a sequence of candidate tokens, which are then verified *in parallel* by the original large *target model* to ensure correctness. By offloading token generation to the draft model and verifying multiple tokens simultaneously, speculative decoding significantly improves inference throughput while preserving accuracy, effectively reducing the latency imposed by repeatedly loading the full set of model weights.

Building upon this foundation, several methods have been proposed to improve speculative decoding, such as fine-tuning the draft model to better align with the target model's distribution (Zhou et al., 2024; Li et al., 2024b,a), dynamic mechanisms to determine the optimal length of the candidate token sequences (Mamou et al., 2024; Huang et al., 2025; Liu et al., 2025), and relaxing the acceptance criteria during target model verification (Kim et al., 2023; Holsman et al., 2025).

However, these methods have limitations. First, aligning the draft model often requires additional training, adding computational cost. Second, relaxing acceptance criteria introduces trade-offs between accuracy and efficiency, and requires extra efforts to determine the tolerable levels. Third, many approaches rely heavily on extensive hyper-

parameter tuning, necessitating iterative configuration searches to achieve satisfactory performance.

To address the aforementioned limitations, we propose *Adaptive Speculative Decoding* (AdaSD), a hyperparameter-free approach to enhance LLM inference efficiency. AdaSD consists of two components: one determines the termination of candidate token generation, and the other defines acceptance criteria for verification. Specifically, we employ entropy and Jensen-Shannon distance as principled metrics to set these components separately. Furthermore, both metrics are dynamically adjusted during inference based on information from previously generated tokens, eliminating the need for manual configuration or model-specific tuning.

The contributions of this paper are as follows:

- We propose AdaSD, an adaptive speculative decoding method that jointly adjusts candidate generation length and acceptance criteria to balance inference speed and accuracy.
- We introduce a Bayesian verification mechanism that estimates the acceptance probability of candidate tokens, providing an adaptive alternative to fixed verification rules.
- Our method is compatible with off-the-shelf models, requiring neither model architecture change nor additional training. Moreover, we eliminate the need for hyperparameter tuning.
- Experimental results indicate that AdaSD achieves up to 1.46× speedup over standard speculative decoding, while limiting accuracy degradation to less than 1.8%.

2 Background

2.1 Language Model Acceleration

2.1.1 Auto-Regressive Model

An auto-regressive model employs a decoder-based Transformer architecture to perform text generation. It produces each output token conditioned on the preceding tokens. The newly generated token is then appended to the input sequence, serving as the context for the next step of generation.

Let p denote the probability distribution of the auto-regressive model. At each time step t, the next token x_t is drawn from the conditional probability distribution over the preceding token sequence $x_{<t}$, defined as $x_t \sim p(x \mid x_{<t})$. The token x_t can be selected *deterministically* by choosing the most probable token: $x_t = \arg\max_x p(x \mid x_{<t})$. Alternatively, x_t can be chosen *stochastically* by sampling from the adjusted distribution, using techniques such as temperature scaling, top-k sampling, or top-p sampling. This iterative process continues until the model produces a special end-of-sequence (EOS) token, which terminates generation.

2.1.2 Speculative Decoding

Speculative decoding is designed to address the high inference time of auto-regressive generation in the large language model, denoted as the *target model* M_p. To mitigate this, a smaller and faster model, referred to as the *draft model* M_q, is introduced to approximate the target model's output distribution.

Each iteration of speculative decoding consists of two steps: *generation* and *verification*. In the generation step, the draft model proposes candidate tokens by sequentially producing a number of tokens up to a limit, which may be fixed or adaptively adjusted. In the verification step, the target model evaluates these candidate tokens in parallel by expanding them into a batch and accepting the longest valid prefix to ensure correctness. By combining fast speculative generation with efficient verification, speculative decoding lowers inference latency without compromising output quality.

2.1.3 Sampling Strategy

To determine which tokens are accepted during the verification step of speculative decoding, a straightforward deterministic strategy is *greedy decoding*. In this approach, both the draft model M_q and the target model M_p always select the token with the highest probability:

$$x_t^q = \arg\max_x M_q(x \mid x_{<t}),$$
$$x_t^p = \arg\max_x M_p(x \mid x_{<t}),$$

where x_t^q and x_t^p are the tokens chosen by the draft and target models at step t, respectively. Greedy decoding imposes a strict acceptance rule: the candidate token x_t^q is accepted only if it exactly matches x_t^p. The verification process determines the longest valid prefix by scanning the evaluation results in parallel until the first mismatch occurs.

Although greedy decoding is simple, it is overly restrictive and limits diversity in text generation. To address this limitation, prior work (Leviathan et al., 2023; Chen et al., 2023) introduces *speculative sampling*, which relaxes the strict matching requirement by allowing tokens from the draft model to be probabilistically accepted by the target model.

Speculative sampling preserves the target distribution by jointly considering the distributions of both models. A candidate token $\tilde{x}_t$ is first sampled from the draft model distribution and then accepted by the target model with probability α:

$$\tilde{x}_t \sim M_q(x \mid x_{<t}),$$
$$\alpha = \min\left(1, \frac{M_p(\tilde{x}_t \mid x_{<t})}{M_q(\tilde{x}_t \mid x_{<t})}\right).$$

If the candidate token is rejected, the target model re-samples a new token from an adjusted distribution that excludes $\tilde{x}_t$.

2.2 Jensen-Shannon Divergence and Distance

Entropy and cross-entropy provide fundamental measures for characterizing uncertainty and distribution mismatch. Building on these concepts, Kullback-Leibler (KL) divergence offers a principled way to quantify the difference between two probability distributions. For completeness, the formal definitions of entropy, cross-entropy, and KL divergence are provided in Appendix A.

However, KL divergence is asymmetric and unbounded, making it less desirable for directly comparing two probability distributions. Therefore, Jensen-Shannon (JS) divergence is often adopted as a symmetric and bounded alternative derived from KL divergence. JS divergence is defined as:

$$\begin{aligned} D_{\mathrm{JS}}(p \parallel q) &= \frac{1}{2}D_{\mathrm{KL}}(p \parallel m) + \frac{1}{2}D_{\mathrm{KL}}(q \parallel m) \\ &= H(m) - \frac{1}{2}(H(p) + H(q)), \end{aligned}$$

where $m = \frac{1}{2}(p + q)$ is the mixture distribution of the draft model q and the target model p, $H(\cdot)$ denotes entropy, and $D_{\mathrm{KL}}(\cdot \parallel m)$ denotes the KL divergence with respect to m. Unlike KL divergence, JS divergence is symmetric in p and q, and is bounded in $[0, 1]$ under base-2 logarithm. These properties make it a more stable and interpretable metric for quantifying the dissimilarity between probability distributions.

To provide a more intuitive measure of the difference between distributions, we consider the Jensen-Shannon (JS) distance, defined as the square root of the JS divergence:

$$d_{\mathrm{JS}}(p \parallel q) = \sqrt{D_{\mathrm{JS}}(p \parallel q)}.$$

Since the JS distance satisfies the properties of a metric (Endres and Schindelin, 2003), it provides a well-defined notion of distance between probability distributions within the metric space it induces.

3 Related Work

Several studies enhance speculative decoding by modifying model architectures. Medusa (Cai et al., 2024) retains a single model but equips it with multiple decoding heads, enabling parallel prediction of consecutive future tokens at each step. EAGLE (Li et al., 2024b,a) reconstructs draft transformer layers via additional training and employs dynamic tree search to align with the target distribution. Both approaches, however, entail significant model adaptation and require extra training before yielding performance gains.

To mitigate the memory and computational overhead associated with deploying a separate auxiliary model, some works focus on self-speculative decoding. This paradigm leverages the internal structure of the target model itself to act as the draft model, while the full model is subsequently used for verification. Draft & Verify (Zhang et al., 2024) selectively skips intermediate layers during the draft step based on an offline Bayesian optimization search. EESD (Liu et al., 2024) constructs its draft model by introducing a trainable early-exiting layer immediately following the first few layers of the target model. SWIFT (Xia et al., 2025) dynamically optimizes the set of skipped layers on-the-fly during inference without additional training. However, although self-speculative approaches eliminate the extra memory footprint, their speedup remains inherently bounded. Omitting an excessive number of layers to accelerate drafting inevitably degrades the prediction quality and lowers the acceptance rate.

Another line of research focuses on adjusting the candidate token generation length in draft models. DISCO (Mamou et al., 2024) employs a two-layer feedforward network (FFN) classifier to adaptively set a termination threshold, stopping generation once a candidate token's probability falls below the threshold. Hugging Face's Assisted Generation (Joao Gante, 2023) extends DISCO to an unsupervised variant that preserves the same decision mechanism but replaces the FFN classifier with a receiver operating characteristic (ROC) curve. AdaEDL (Agrawal et al., 2024) introduces a training-free early stopping criterion that halts the drafting process by approximating a lower bound on the expected token acceptance probability using the entropy of the draft model's logits. SpecDec++ (Huang et al., 2025) formulates the problem as a Markov decision process and intro-

duces a trained head that predicts the acceptance probability of candidate tokens to decide when to stop. PEARL (Liu et al., 2025) simulates dynamic draft lengths through two strategies: it first verifies whether a newly generated token is accepted by the target model, and upon success, allows the draft model to continue generating tokens in parallel with verification, creating the effect of dynamic length adjustment. HSDDW (Syu and Lee, 2025) proposes a three-layer hierarchical structure of progressively larger models. The smallest model applies a threshold-based mechanism to control generation or forward tokens to larger models in the hierarchy for verification. C2T (Huo et al., 2026) leverages a trainable lightweight classifier based on joint probability, entropy, and depth to dynamically pre-prune token trees and facilitate early stopping during inference. Although effective, these approaches primarily focus on the generation process of draft models, and leave the target model's role in verification unexplored.

A number of works define acceptance criteria based on the divergence between draft and target distributions. BiLD (Kim et al., 2023) uses a fixed cross-entropy threshold to determine whether to accept draft tokens, while FSD (Holsman et al., 2025) utilizes a fixed JS divergence threshold, accepting tokens when the divergence between draft and target models is sufficiently small. However, both methods require model- or task-specific information in advance for deciding the optimal thresholds, which limits their general applicability.

The summary and comparison of related methods are provided in Appendix B.

4 Adaptive Speculative Decoding

In this section, we first provide an overview of AdaSD. We then present the empirical study, whose insights motivate this work. Finally, we describe the proposed method in detail.

4.1 Overview

We propose Adaptive Speculative Decoding (AdaSD), a hyperparameter-free scheme for efficient LLM inference. AdaSD differs from prior methods in that it requires neither additional training nor any model- or task-specific information in advance. Moreover, it optimizes the decoding decisions by taking both the draft and target models into account. In particular, it eliminates the burden of hyperparameter tuning, which often requires exhaustive search and manual effort.

AdaSD consists of two steps. The *generation* step (Section 4.3) determines when the draft model should stop generating candidate tokens and hand them off to the target model. The *verification* step (Section 4.4) regulates the acceptable level of dissimilarity between the probability distributions of the draft and target models. The workflow of AdaSD is illustrated in Figure 1.

To further improve efficiency, AdaSD introduces a heuristic feedback mechanism that adaptively adjusts decision rules during inference. Instead of relying on pre-analysis of models or tasks, AdaSD dynamically leverages statistics from previously generated tokens to compute entropy and JS distance to update decision rules on the fly. The algorithm of our method is provided in Appendix C.

4.2 Empirical Study

We begin by considering the generation step of the draft model. Since entropy quantifies the uncertainty associated with a *single* random variable, we use it solely within the draft model to estimate the uncertainty of its generated candidate tokens, independent of the target model. This uncertainty measurement is commonly employed in early-exit networks (Agrawal et al., 2024).

For the verification, we propose a joint optimization approach that simultaneously considers both the draft and target models. The main challenge lies in selecting an appropriate indicator that captures the correlation between the two models and determines the acceptance probability of tokens. At the same time, the indicator must be *bounded* to ensure that the resulting acceptance probabilities remain within a well-defined range, facilitating stable analysis and implementation.

Cross-entropy and KL divergence are unsuitable in this context because, when the distributions p and q differ substantially, their values can diverge to infinity, making the threshold unmanageable. To address this issue, we propose using JS distance. While both JS divergence and JS distance are bounded, JS distance offers the additional advantage of defining a proper metric, providing a more stable and consistent signal for adaptive verification within a metric space.

To validate the effectiveness of these measures, we conduct an initial experiment analyzing entropy and JS distance for accepted and rejected tokens. We sample 20 examples from the widely used instruction-tuning dataset Alpaca (Taori et al.,

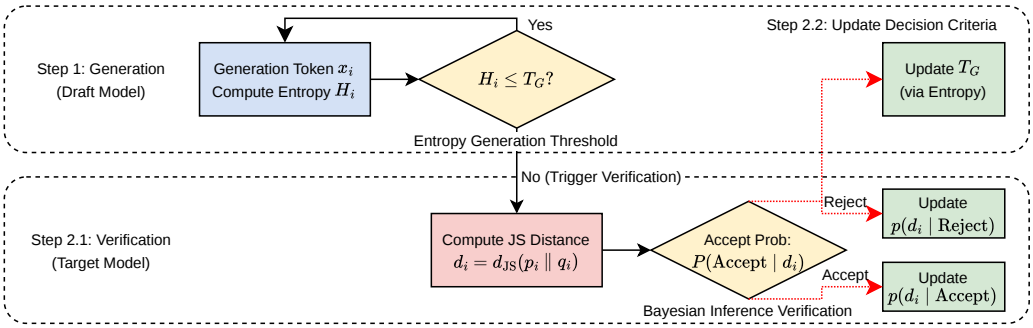

Figure 1: The decoding process of AdaSD.

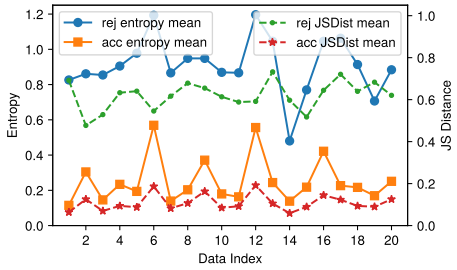

Figure 2: Mean entropy and Jensen-Shannon distance for accepted and rejected tokens, evaluated individually on 20 samples from Alpaca dataset, using Llama 3.1 8B and 70B as the draft and target models, respectively.

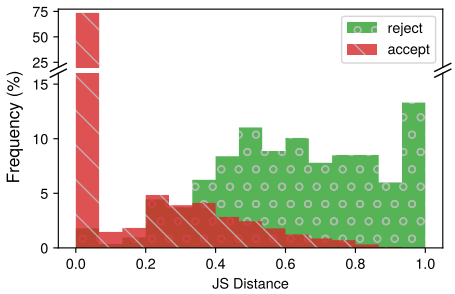

Figure 3: Jensen-Shannon distance distribution of accepted and rejected tokens.

2023), and use Llama 3.1 8B as the draft model and Llama 3.1 70B as the target model. For each example, we compute the mean entropy of the candidate token distributions produced by the draft model, as well as the mean JS distance between the draft and target models for both accepted and rejected tokens. The results, illustrated in Figure 2, show a clear separation. The rejected tokens consistently exhibit higher entropy and JS distance than the accepted tokens, suggesting that both measures can serve as effective signals for predicting token acceptance and guiding the design of adaptive thresholds.

To further understand the correlation between models, we examine the distribution of JS distances across the sample data. As shown in Figure 3, a certain samples of rejected tokens exhibits JS distances close to 1, whereas such cases would produce extreme values that destabilize the threshold when using cross-entropy or KL divergence. Furthermore, the JS distance distribution for accepted tokens shows two distinct clusters. The first cluster (i.e., near zero) occurs when both models assign most of the probability mass to the *same* token, typically in highly predictable contexts. The second cluster appears when both models distribute probability more evenly across multiple options but ultimately select the same token, usually in contexts with weaker contextual correlation. These results offer insights that JS distance effectively captures model agreement and measures uncertainty, allowing to provide robust thresholding for token verification in AdaSD.

4.3 Generation

The generation step in AdaSD employs an entropy-based threshold to control candidate token generation in the draft model. If the entropy of a newly generated candidate token exceeds this threshold, indicating that the token is likely to be rejected, the generation process is terminated and control proceeds to the verification step.

The generation threshold is updated adaptively during inference. Let e_r denote the entropy of a candidate token rejected by the target model during the verification step. We define E_r as the set of all such entropy values collected since the start of the decoding process. The generation threshold T_G is then recalculated as the mean value of the elements in E_r:

$$T_G = \frac{1}{|E_r|} \sum_{e \in E_r} e.$$

This adaptive mechanism ensures that T_G dynamically adjusts based on the draft model's historical

uncertainty for rejected tokens.

4.4 Verification

At each verification step, we measure the discrepancy between the draft and target models by computing the JS distances between their output probability distributions for the candidate tokens. The JS distance provides a symmetric and bounded measure of divergence and serves as the primary signal for regulating token acceptance during verification.

Instead of filtering candidate tokens using a fixed threshold as in previous approaches (Kim et al., 2023; Holsman et al., 2025), we model the *acceptance probability* of a candidate token as a function of the JS distance. Intuitively, smaller distances indicate stronger agreement between the draft and target models and should therefore correspond to higher acceptance probabilities.

To capture this relationship, we employ *Bayesian inference*. Given the JS distance d of a candidate token, the probability of accepting the token is computed from the relative likelihood under two density functions:

$$P(A \mid d) = \frac{p(d \mid A)P(A)}{p(d \mid A)P(A) + p(d \mid R)P(R)},$$

where A and R respectively denote the acceptance and rejection events, $P(A)$ and $P(R)$ represent the prior probabilities, while $p(d \mid A)$ and $p(d \mid R)$ are the conditional probability density functions of JS distances. To estimate these densities, we collect the JS distances of verified tokens from the beginning of decoding and group them according to their verification outcome (accepted or rejected). We then construct the empirical densities $p(d \mid A)$ and $p(d \mid R)$ using kernel density estimation (KDE) (Silverman, 1986).

Note that we exclude tokens whose JS distances are exactly 0 or 1 when computing the KDEs. This design provides three advantages. First, distances of 0 and 1 correspond to cases of absolute agreement and disagreement between the draft and target models. Excluding these cases allows the model to focus on estimating acceptance probabilities for the intermediate distances where the decision is uncertain. Second, computing the KDE incurs a cost proportional to the number of collected samples. As shown in Figure 3, a significant portion of tokens have JS distances equal to 0 or 1. Removing these tokens therefore reduces the number of samples involved in density estimation and significantly lowers the verification overhead without affecting the decision quality. Third, KDE estimation may suffer from boundary leakage, where probability density spills outside the $[0, 1]$ range. Excluding boundary cases therefore improves the accuracy of the density estimation.

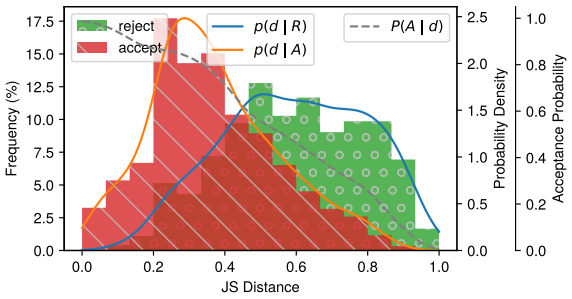

Figure 4: JS distance distribution for accepted and rejected tokens ($d = 0$ and 1 filtered). We estimate probability densities with KDE and compute the acceptance probability as a function of JS distance.

Figure 4 illustrates the acceptance probability as a function of the JS distance for the samples in Figure 3, excluding the distances of 0 and 1. The orange and blue curves correspond to the KDEs for accepted and rejected tokens, respectively. The dashed curve represents the resulting acceptance probability derived from these density estimates, confirming our intuition that smaller distances exhibit higher acceptance probabilities.

After the target model makes the verification decision, JS distances of candidate tokens are added to the corresponding group (accepted or rejected), and the density estimates are updated accordingly. Through this online update process, the acceptance criteria gradually adapts to the empirical distribution of verification outcomes during decoding.

5 Experiments

5.1 Experimental Setup

Benchmark. We select three benchmark datasets, each focusing on a different task. GSM8K (Cobbe et al., 2021) contains grade school-level mathematical word problems. HumanEval (Chen et al., 2021) evaluates programming and code generation. MMLU (Hendrycks et al., 2021) covers multiple-choice questions across a wide range of subjects.

Model. We select two representative model families, Llama 3 (Grattafiori et al., 2024) and Qwen 2.5 (Yang et al., 2025), and consider three target-draft model pairs: (Llama 3.1 70B, Llama 3.1 8B), (Llama 3.1 70B, Llama 3.2 1B), and (Qwen 2.5

72B, Qwen 2.5 7B). These configurations enable a comprehensive evaluation of performance within the same model family across different scales and across model families of comparable sizes.

Baseline. We compare two speculative decoding methods that require no additional training and are agnostic to task-specific information: (a) Vanilla speculative decoding (Leviathan et al., 2023), which uses a fixed candidate token length of five, and (b) Hugging Face's built-in Assisted Generation (Joao Gante, 2023). In the following experiments, we use *Vanilla* and *AssistedGen* to represent both methods, respectively.

5.2 Performance Results

Table 1 presents the throughput and accuracy results for AdaSD and the baseline methods across all evaluated benchmarks, with more detailed data listed in Appendix E. The performance of standalone draft and target models is included for reference, with efficiency measured as the speedup relative to the Vanilla baseline on four NVIDIA A6000 GPUs. All evaluated methods use sampling, which may result in minor accuracy deviations from the original target model. Detailed hardware and software configurations are provided in Appendix D.

Overall, AdaSD achieves the highest speedup among all methods on Llama models and is only slightly slower than AssistedGen on the Qwen model. For GSM8K and HumanEval, AdaSD achieves 23-46% higher throughput than Vanilla. Compared to AssistedGen, AdaSD performs comparably or achieves further speedups depending on the model pairs (up to 14.2% higher on GSM8K with Llama 3.1 70B–1B), while preserving accuracy within 1.8% of the baseline.

A notable observation is the relatively modest speedup achieved by AdaSD on the MMLU dataset. Both AdaSD and AssistedGen exhibit limited acceleration, with the highest improvement being only 12%. This is because MMLU is a multiple-choice benchmark, where the model generates only a small number of tokens corresponding to the selected answer. The short sequence of these outputs inherently limits opportunities for acceleration.

We also observe that AdaSD achieves lower speedup than AssistedGen with Qwen 2.5. To investigate this, we analyze the number of generated and accepted tokens, as well as the corresponding acceptance rates. As Table 2 shows, both the average lengths of generated and accepted tokens in AdaSD are longer than those in AssistedGen. Although AssistedGen produces shorter sequence per speculative decoding step, it can still generate sufficiently long sequences while maintaining a high acceptance rate. This results from its design objective, which jointly maximizes generation efficiency and the acceptance rate. While AssistedGen occasionally adopts a conservative strategy that underestimates the candidate sequence length, it terminates generation at an appropriate point without excessive token generation, thus minimizing the draft model's inference overhead.

In contrast, AdaSD's generation threshold relies on the uncertainty of tokens produced by the draft model. Therefore, this can lead to relatively more candidate tokens and the likelihood of tokens being rejected, which increases both generation and verification overhead. This explains why AssistedGen outperforms AdaSD with Qwen.

We further investigate whether combining the strategies of AdaSD and AssistedGen can improve decoding efficiency for Qwen. As shown in Table 2, the hybrid approach achieves the best performance across all tasks. On GSM8K, the speedup increases from 1.287 (AssistedGen) and 1.257 (AdaSD) to 1.307 with the hybrid approach. Similarly, the speedup improves from 1.282 and 1.250 to 1.286 on HumanEval, and from 1.088 and 1.091 to 1.134 on MMLU. These results indicate that AdaSD's adaptive verification and AssistedGen's generation-length control mechanism are complementary, and their combination enables further improvements in speculative decoding efficiency.

5.3 Ablation Studies

In this section, we analyze the individual impact of AdaSD's components by comparing AdaSD with its two variants, *Gen-Only*, which applies only the generation component, and *Verify-Only*, which applies only the verification component, as reported in Tables 1 and 2. For Verify-Only, we fix the candidate token length to five, consistent with Vanilla.

Gen-Only. Gen-Only demonstrates strong inference speedup over Vanilla, achieving a 16-42% improvement on GSM8K and HumanEval. On MMLU, the improvement is more modest, ranging from 1.4% to 3.9% due to the inherently short generation sequences. This improvement arises from generating longer candidate token sequences. The draft model continues generating tokens until the token entropy exceeds the generation thresh-

	GSM8K			HumanEval			MMLU		
Llama 3.1 70B – Llama 3.1 8B									
	tks/sec	accuracy	speedup	tks/sec	accuracy	speedup	tks/sec	accuracy	speedup
Draft	36.446	0.828	2.624	36.257	0.634	2.532	33.274	0.695	4.206
Target	4.842	0.939	0.349	4.838	0.750	0.338	4.376	0.836	0.553
Vanilla	13.890	0.945	1.000	14.319	0.768	1.000	7.912	0.835	1.000
AssistedGen	16.062	0.944	1.156	18.137	0.756	1.267	7.970	0.835	1.007
Gen-Only	16.243	0.943	1.169	18.017	0.762	1.258	8.023	0.831	1.014
Verify-Only	14.227	0.931	1.024	14.425	0.750	1.007	8.247	0.836	1.042
AdaSD	17.039	0.943	**1.227**	18.348	0.799	**1.281**	8.477	0.834	**1.071**
Llama 3.1 70B – Llama 3.2 1B									
	tks/sec	accuracy	speedup	tks/sec	accuracy	speedup	tks/sec	accuracy	speedup
Draft	99.919	0.375	6.080	100.683	0.348	5.764	95.141	0.453	10.878
Target	4.842	0.939	0.295	4.838	0.750	0.277	4.376	0.836	0.500
Vanilla	16.434	0.939	1.000	17.467	0.756	1.000	8.746	0.838	1.000
AssistedGen	19.892	0.936	1.210	23.265	0.774	1.332	8.920	0.833	1.020
Gen-Only	21.126	0.939	1.286	24.842	0.768	1.422	9.085	0.835	1.039
Verify-Only	17.063	0.926	1.038	17.710	0.726	1.014	9.346	0.817	1.069
AdaSD	22.225	0.937	**1.352**	25.452	0.774	**1.457**	9.795	0.815	**1.120**
Qwen 2.5 72B – Qwen 2.5 7B									
	tks/sec	accuracy	speedup	tks/sec	accuracy	speedup	tks/sec	accuracy	speedup
Draft	38.796	0.867	2.667	38.724	0.720	2.840	38.544	0.725	3.667
Target	4.728	0.913	0.325	4.717	0.787	0.346	4.690	0.845	0.446
Vanilla	14.546	0.912	1.000	13.634	0.799	1.000	10.511	0.837	1.000
AssistedGen	18.721	0.917	**1.287**	17.482	0.799	**1.282**	11.436	0.837	1.088
Gen-Only	18.054	0.920	1.241	16.733	0.787	1.227	10.735	0.841	1.021
Verify-Only	14.705	0.920	1.011	13.838	0.787	1.015	11.042	0.837	1.051
AdaSD	18.289	0.915	1.257	17.041	0.799	1.250	11.470	0.839	**1.091**

Table 1: Performance comparison of speculative decoding methods. Metrics include decoding speed (tks/sec), accuracy, and speedup over the Vanilla baseline. Gen-Only and Verify-Only represent AdaSD variants that use only the generation and verification component, respectively. The bold and underlined values indicate the highest and second-highest speedups.

Qwen 2.5 72B – Qwen 2.5 7B												
	GSM8K				HumanEval				MMLU			
	#cand	#match	AccRate	speedup	#cand	#match	AccRate	speedup	#cand	#match	AccRate	speedup
Vanilla	5.000	4.228	0.846	1.000	5.000	3.942	0.788	1.000	5.000	2.811	0.562	1.000
AssistedGen	10.038	8.335	0.830	1.287	9.745	7.720	0.792	1.282	6.109	3.618	0.592	1.088
Gen-Only	14.254	9.967	0.699	1.241	13.199	8.832	0.669	1.227	8.962	4.210	0.470	1.021
Verify-Only	5.000	4.290	0.863	1.011	5.000	4.024	0.812	1.015	5.000	3.010	0.610	1.051
AdaSD	14.389	10.541	0.731	1.257	13.476	9.141	0.677	1.250	8.904	4.539	0.495	1.091
AdaSD + AssistedGen	10.297	8.639	0.840	**1.307**	9.646	7.727	0.800	**1.286**	5.761	3.743	0.646	**1.134**

Table 2: Decoding results of Qwen on different datasets. The result shows the number of generated candidates (#cand), accepted tokens (#match), and the acceptance rates (AccRate) collectively influence the decoding speedup.

old, which is adaptively updated based on feedback from the target model.

Verify-Only. In contrast, the speedup of Verify-Only is limited when used alone, with at most 6.9% improvement over Vanilla on MMLU. This is because Vanilla achieves a high acceptance rate with its carefully chosen candidate length, effectively covering most general cases. However, when combined with the generation component, AdaSD further improves the efficiency, bringing the total improvement to 12% over Vanilla (5.1% over Verify-Only). These results indicate that our JS distance-based acceptance criteria can further accelerate inference while maintaining accuracy.

6 Conclusion

This paper proposes AdaSD, a novel speculative decoding method without the need for manual hyperparameter tuning. AdaSD improves decoding speed with minimal accuracy loss. While AdaSD enhances decoding efficiency, there is still room for further improvement in its adaptive mechanisms. For generation, incorporating additional indicators beyond entropy could enable more precise decisions. For verification, refining JS distance modeling could further boost acceptance and accuracy. We believe that AdaSD represents a practical step toward more efficient and adaptive decoding for large language models.

Limitations

AdaSD is applicable when the draft and target models share the same vocabulary, meaning that both models must have identical token sizes and token content. This requirement arises because AdaSD relies on computing entropy and JS distance for output tokens. When the vocabulary differs, these metrics cannot be calculated consistently, and determining an appropriate threshold becomes difficult.

Another limitation lies in the challenge of selecting compatible draft and target model pairs. In practice, the optimal size gap between the draft and target models is roughly 50-100 times. However, within this range, few well-known model pairs share an identical vocabulary. This issue is exacerbated by recent model development trends, where smaller models (<1B) often use different vocabularies from larger-sized models (>70B). Moreover, smaller models frequently incorporate architectural modifications beyond simply reducing the number of Transformer layers, such as changes to embedding layers or tokenization schemes, which further limits compatibility and applicability of AdaSD.

References

Sudhanshu Agrawal, Wonseok Jeon, and Mingu Lee. 2024. AdaEDL: Early draft stopping for speculative decoding of large language models via an entropy-based lower bound on token acceptance probability. In *Proceedings of The 4th NeurIPS Efficient Natural Language and Speech Processing Workshop*, volume 262 of *Proceedings of Machine Learning Research*, pages 355–369. PMLR.

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, and 12 others. 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. In *Forty-first International Conference on Machine Learning*.

Charlie Chen, Sebastian Borgeaud, Geoffrey Irving, Jean-Baptiste Lespiau, Laurent Sifre, and John Jumper. 2023. Accelerating large language model decoding with speculative sampling. *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, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, and 39 others. 2021. Evaluating large language models trained on code. *Preprint*, arXiv:2107.03374.

Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, Maarten Bosma, Gaurav Mishra, Adam Roberts, Paul Barham, Hyung Won Chung, Charles Sutton, Sebastian Gehrmann, Parker Schuh, Kensen Shi, Sasha Tsvyashchenko, Joshua Maynez, Abhishek Rao, Parker Barnes, Yi Tay, Noam Shazeer, Vinodkumar Prabhakaran, and 48 others. 2023. Palm: Scaling language modeling with pathways. *Journal of Machine Learning Research*, 24(240):1–113.

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

D.M. Endres and J.E. Schindelin. 2003. A new metric for probability distributions. *IEEE Transactions on Information Theory*, 49(7):1858–1860.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, Amy Yang, Angela Fan, Anirudh Goyal, Anthony Hartshorn, Aobo Yang, Archi Mitra, Archie Sravankumar, Artem Korenev, Arthur Hinsvark, and 542 others. 2024. The llama 3 herd of models. *Preprint*, arXiv:2407.21783.

Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. 2021. Measuring massive multitask language understanding. In *International Conference on Learning Representations*.

Maximilian Holsman, Yukun Huang, and Bhuwan Dhingra. 2025. Fuzzy speculative decoding for a tunable accuracy-runtime tradeoff. In *Findings of the Association for Computational Linguistics: ACL 2025*, pages 26257–26273.

Kaixuan Huang, Xudong Guo, and Mengdi Wang. 2025. Specdec++: Boosting speculative decoding via adaptive candidate lengths. In *Second Conference on Language Modeling*.

Feiye Huo, Jianchao Tan, Zixu Jiang, Kefeng Zhang, Xunliang Cai, and Shengli Sun. 2026. C2t: Classifier-based token tree construction in speculative decoding. In *The Fourteenth International Conference on Learning Representations*.

Joao Gante. 2023. Assisted generation: a new direction toward low-latency text generation.

Sehoon Kim, Karttikeya Mangalam, Suhong Moon, Jitendra Malik, Michael W Mahoney, Amir Gholami, and Kurt Keutzer. 2023. Speculative decoding with big little decoder. In *Advances in Neural Information Processing Systems*, volume 36, pages 39236–39256. Curran Associates, Inc.

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

Yuhui Li, Fangyun Wei, Chao Zhang, and Hongyang Zhang. 2024a. EAGLE-2: Faster inference of language models with dynamic draft trees. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 7421–7432.

Yuhui Li, Fangyun Wei, Chao Zhang, and Hongyang Zhang. 2024b. EAGLE: Speculative sampling requires rethinking feature uncertainty. In *Forty-first International Conference on Machine Learning*.

Jiahao Liu, Qifan Wang, Jingang Wang, and Xunliang Cai. 2024. Speculative decoding via early-exiting for faster LLM inference with Thompson sampling control mechanism. In *Findings of the Association for Computational Linguistics: ACL 2024*, pages 3027–3043, Bangkok, Thailand. Association for Computational Linguistics.

Tianyu Liu, Yun Li, Qitan Lv, Kai Liu, Jianchen Zhu, Winston Hu, and Xiao Sun. 2025. PEARL: Parallel speculative decoding with adaptive draft length. In *The Thirteenth International Conference on Learning Representations*.

Jonathan Mamou, Oren Pereg, Daniel Korat, Moshe Berchansky, Nadav Timor, Moshe Wasserblat, and Roy Schwartz. 2024. Dynamic speculation lookahead accelerates speculative decoding of large language models. In *Proceedings of The 4th NeurIPS Efficient Natural Language and Speech Processing Workshop*, volume 262, pages 456–467.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, and 2 others. 2019. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc.

Jack W. Rae, Sebastian Borgeaud, Trevor Cai, Katie Millican, Jordan Hoffmann, Francis Song, John Aslanides, Sarah Henderson, Roman Ring, Susannah Young, Eliza Rutherford, Tom Hennigan, Jacob Menick, Albin Cassirer, Richard Powell, George van den Driessche, Lisa Anne Hendricks, Maribeth Rauh, Po-Sen Huang, and 61 others. 2022. Scaling language models: Methods, analysis & insights from training gopher. *Preprint*, arXiv:2112.11446.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. 2020. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of Machine Learning Research*, 21(140):1–67.

Bernard. W. Silverman. 1986. *Density Estimation for Statistics and Data Analysis*. Routledge.

Shensian Syu and Hung-yi Lee. 2025. Hierarchical speculative decoding with dynamic window. In *Findings of the Association for Computational Linguistics: NAACL 2025*, pages 8260–8273.

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. https://github.com/tatsu-lab/stanford_alpaca.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Remi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, and 3 others. 2020. Transformers: State-of-the-art natural language processing. In *Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 38–45.

Heming Xia, Yongqi Li, Jun Zhang, Cunxiao Du, and Wenjie Li. 2025. Swift: On-the-fly self-speculative decoding for llm inference acceleration. In *International Conference on Learning Representations*.

An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, Huan Lin, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jingren Zhou, Junyang Lin, Kai Dang, and 23 others. 2025. Qwen2.5 technical report. *Preprint*, arXiv:2412.15115.

Jun Zhang, Jue Wang, Huan Li, Lidan Shou, Ke Chen, Gang Chen, and Sharad Mehrotra. 2024. Draft & verify: Lossless large language model acceleration via self-speculative decoding. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 11263–11282, Bangkok, Thailand. Association for Computational Linguistics.

Yongchao Zhou, Kaifeng Lyu, Ankit Singh Rawat, Aditya Krishna Menon, Afshin Rostamizadeh, Sanjiv Kumar, Jean-François Kagy, and Rishabh Agarwal. 2024. Distillspec: Improving speculative decoding via knowledge distillation. In *The Twelfth International Conference on Learning Representations*.

A Information Theoretic Background

Entropy. In information theory, entropy measures the uncertainty associated with a random variable. For a discrete random variable X taking values $x \in \mathcal{X}$ with a probability distribution $p : \mathcal{X} \to [0, 1]$, the entropy is defined as:

$$\begin{aligned} H(X) &= \mathbb{E}[-\log p(x)] \\ &= -\sum_{x \in \mathcal{X}} p(x) \log p(x). \end{aligned}$$

Entropy can be interpreted as the expected value of the self-information $-\log p(x)$, representing the average number of bits needed to encode the outcome of X. A larger entropy value indicates higher uncertainty or greater variability in the distribution.

Cross-Entropy. Cross-entropy quantifies the difference between two probability distributions p and q defined over the same event space $\mathcal{X}$. Formally, the cross-entropy of an estimated distribution q with respect to a true distribution p is defined as:

$$\begin{aligned} H(p, q) &= \mathbb{E}_p[-\log q(x)] \\ &= -\sum_{x \in \mathcal{X}} p(x) \log q(x). \end{aligned}$$

Cross-entropy represents the expected number of bits needed to encode samples drawn from p using a code optimized for q. Hence, it captures both the inherent uncertainty of p and the additional cost due to the divergence between p and q.

Kullback-Leibler Divergence. The Kullback-Leibler (KL) divergence measures the discrepancy between a true distribution p and an approximating distribution q. It can be expressed as the difference between cross-entropy and entropy:

$$\begin{aligned} D_{\mathrm{KL}}(p \parallel q) &= H(p, q) - H(p) \\ &= \sum_{x \in \mathcal{X}} p(x) \log \frac{p(x)}{q(x)}. \end{aligned}$$

Since KL divergence has no fixed upper bound, if an event occurs frequently under p but rarely under q, the divergence can become arbitrarily large, making it difficult to stop or normalize the measure within a controlled range.

B Method Comparison

This section provides a summary comparison of representative speculative decoding methods. Table 3 highlights several key design choices across prior work and our proposed method. Specifically, we compare whether each method adopts self-speculative decoding, dynamically adjusts the draft sequence length, or introduces relaxed verification mechanisms. The table also indicates whether the method requires architectural modifications or additional training, relies on empirically tuned hyperparameters, or performs task-specific pre-analysis. This comparison helps clarify the design space of speculative decoding methods and highlights the characteristics of AdaSD.

C Algorithm of AdaSD

Algorithm 1 shows the AdaSD procedure, which comprises three main steps: generation, verification, and update. To enable adaptive threshold adjustment, we record the entropy of generated tokens and the JS distances during inference in three lists: rejected entropy (L_E^R), rejected JS distance (L_D^R), and accepted JS distance (L_D^A). These statistics are used to update the generation and verification thresholds. Moreover, two additional constraints are imposed. First, we set the maximum window size W to 20, limiting the number of candidate tokens generated per iteration and preventing unbounded draft expansion. Second, due to the inherent maximum context length of LLMs, we define a limit K on the total number of generated tokens to ensure the context X remains within allowable bounds.

D Implementation Details

Experiments are implemented using the Hugging Face Transformer framework (Wolf et al., 2020) version 4.55. The Transformer backend uses PyTorch (Paszke et al., 2019) version 2.8 with CUDA 12.8 and cuDNN 9.10. We run all tests on four NVIDIA A6000 GPUs and enable "`device_map=auto`" to evenly split the model on all available GPUs. All evaluated methods use speculative sampling for token selection.

E Additional Experimental Results

Tables 4, 5, and 6 present the performance results on the GSM8K, HumanEval, and MMLU datasets respectively.

paper	self-speculative	dynamic draft length	relaxed verification	architecture modification/ extra training	empirically tuned hyperparameter	task pre-analyzing
Medusa (Cai et al., 2024)	V			V		
EAGLE (Li et al., 2024b,a)		V		V		
Draft & Verify (Zhang et al., 2024)	V	V			V	V
EESD (Liu et al., 2024)	V	V		V	V	
SWIFT (Xia et al., 2025)	V	V			V	
DISCO (Mamou et al., 2024)		V		V		V
AdaEDL (Agrawal et al., 2024)		V			V	
SpecDec++ (Huang et al., 2025)		V		V		
PEARL (Liu et al., 2025)		V				V
HSDDW (Syu and Lee, 2025)		V				
C2T (Huo et al., 2026)		V		V	V	
BiLD (Kim et al., 2023)		V	V	V		V
FSD (Holsman et al., 2025)			V		V	V
AssistedGen (Joao Gante, 2023)		V				
AdaSD (Ours)		V	V			

Table 3: Comparison of representative speculative decoding methods and their key design choices.

Algorithm 1: Adaptive Speculative Decoding

Input: target model M_p, draft model M_q, context X, max tokens K, max window size W
Output: final generated context X
Initialization: $L_E^R \leftarrow [\,]$ // List of Entropies (Rejected)
$L_D^A, L_D^R \leftarrow [\,], [\,]$ // Lists of JS Distances (Accepted/Rejected)
$T_G \leftarrow 0$ // Generation Threshold
$w \leftarrow 0$ // Window size counter

while $len(X) < K$ **do**
  // 1. The draft model generates tokens sequentially
  for $i \leftarrow 1$ **to** W **do**
    $q_i \leftarrow M_q(X, x_{1:i-1})$ // Define $x_{1:0}$ is null
    $x_i \sim q_i$ // Sample x_i from q_i
    $w \leftarrow i$ // Record generated window length
    if $H(q_i) > T_G$ **then** // Check generation threshold
      break
  // 2. The target model verifies tokens in parallel
  $p_1, \ldots, p_{w+1} \leftarrow M_p(X, x_{1:0}), \ldots, M_p(X, x_{1:w})$
  $y_1, \ldots, y_{w+1} \sim p_1, \ldots, p_{w+1}$ // sample y_i from p_i
  for $i \leftarrow 1$ **to** w **do**
    $d_i \leftarrow d_{\text{JS}}(p_i \parallel q_i)$
    $p(d_i \mid A) \leftarrow \texttt{KDE}(L_D^A, d_i)$ // density under accepted distances
    $p(d_i \mid R) \leftarrow \texttt{KDE}(L_D^R, d_i)$ // density under rejected distances
    $P(A) \leftarrow \frac{|L_D^A|}{|L_D^A|+|L_D^R|}$ // prior of acceptance
    $P(R) \leftarrow \frac{|L_D^R|}{|L_D^A|+|L_D^R|}$ // prior of rejection
    $P(A \mid d_i) \leftarrow \frac{p(d_i|A)\,P(A)}{p(d_i|A)\,P(A)+p(d_i|R)\,P(R)}$ // acceptance probability
    if $x_i \neq y_i$ ***and*** Reject($P(A \mid d_i)$) **then**
      L_E^R.append($H(q_i)$) // record rejected entropy
      if $0 < d_i < 1$ **then**
        L_D^R.append(d_i) // record rejected JS distance
      $w \leftarrow i - 1$ // record accepted window length
      break
    if $0 < d_i < 1$ **then**
      L_D^A.append(d_i) // record accepted JS distance
  // 3. Update parameters
  X.append($x_{1:w}, y_{w+1}$) // update context
  $T_G \leftarrow$ average(L_E^R) // update generation threshold

openai/gsm8k, main, test, num_rows = 1319						
Llama 3.1 70B – Llama 3.1 8B						
	tks/sec	#cand	#match	AccRate	accuracy	speedup
Draft	36.446	-	-	-	0.828	2.624
Target	4.842	-	-	-	0.939	0.349
Vanilla	13.890	5.000	4.057	0.811	0.945	1.000
AssistedGen	16.062	7.898	6.311	0.799	0.944	1.156
Gen-Only	16.243	11.157	7.850	0.704	0.943	1.169
Verify-Only	14.227	4.969	4.189	0.843	0.931	1.024
AdaSD	17.039	11.168	8.309	0.745	0.943	1.227
Llama 3.1 70B – Llama 3.2 1B						
	tks/sec	#cand	#match	AccRate	accuracy	speedup
Draft	99.919	-	-	-	0.375	6.080
Target	4.842	-	-	-	0.939	0.295
Vanilla	16.434	5.000	3.596	0.719	0.939	1.000
AssistedGen	19.892	6.918	5.047	0.730	0.936	1.210
Gen-Only	21.126	10.438	6.189	0.593	0.939	1.286
Verify-Only	17.063	4.967	3.748	0.755	0.926	1.038
AdaSD	22.225	10.358	6.560	0.636	0.937	1.352
Qwen 2.5 72B – Qwen 2.5 7B						
	tks/sec	#cand	#match	AccRate	accuracy	speedup
Draft	38.796	-	-	-	0.867	2.667
Target	4.728	-	-	-	0.913	0.325
Vanilla	14.546	5.000	4.228	0.846	0.912	1.000
AssistedGen	18.721	10.038	8.335	0.830	0.917	1.287
Gen-Only	18.054	14.254	9.967	0.699	0.920	1.241
Verify-Only	14.705	4.971	4.290	0.863	0.920	1.011
AdaSD	18.289	14.389	10.514	0.731	0.915	1.257

Table 4: Inference results on the GSM8K test set with different schemes across three model pair combinations.

openai/openai_humaneval, test, num_rows = 164						
Llama 3.1 70B – Llama 3.1 8B						
	tks/sec	#cand	#match	AccRate	accuracy	speedup
Draft	36.257	-	-	-	0.634	2.532
Target	4.838	-	-	-	0.750	0.338
Vanilla	14.319	5.000	4.264	0.853	0.768	1.000
AssistedGen	18.137	9.095	7.927	0.872	0.756	1.267
Gen-Only	18.017	13.274	9.972	0.751	0.762	1.258
Verify-Only	14.425	4.968	4.318	0.869	0.750	1.007
AdaSD	18.348	13.015	10.054	0.772	0.799	1.281
Llama 3.1 70B – Llama 3.2 1B						
	tks/sec	#cand	#match	AccRate	accuracy	speedup
Draft	100.683	-	-	-	0.348	5.764
Target	4.838	-	-	-	0.750	0.277
Vanilla	17.467	5.000	3.915	0.783	0.756	1.000
AssistedGen	23.265	7.759	6.362	0.820	0.774	1.332
Gen-Only	24.842	11.962	7.930	0.663	0.768	1.422
Verify-Only	17.710	4.969	3.961	0.797	0.726	1.014
AdaSD	25.452	11.829	8.124	0.686	0.774	1.457
Qwen 2.5 72B – Qwen 2.5 7B						
	tks/sec	#cand	#match	AccRate	accuracy	speedup
Draft	38.724	-	-	-	0.720	2.840
Target	4.717	-	-	-	0.787	0.346
Vanilla	13.634	5.000	3.942	0.788	0.799	1.000
AssistedGen	17.482	9.745	7.720	0.792	0.799	1.282
Gen-Only	16.733	13.199	8.832	0.669	0.787	1.227
Verify-Only	13.838	4.956	4.024	0.812	0.787	1.015
AdaSD	17.041	13.476	9.141	0.677	0.799	1.250

Table 5: Inference results on the HumanEval test set with different schemes across three model pair combinations.

cais/mmlu, all, validation, num_rows = 1531						
Llama 3.1 70B – Llama 3.1 8B						
	tks/sec	#cand	#match	AccRate	accuracy	speedup
Draft	33.274	-	-	-	0.695	4.206
Target	4.376	-	-	-	0.836	0.553
Vanilla	7.912	5.000	2.162	0.432	0.835	1.000
AssistedGen	7.970	5.581	2.482	0.445	0.835	1.007
Gen-Only	8.023	6.297	2.726	0.433	0.831	1.014
Verify-Only	8.247	4.402	2.285	0.511	0.836	1.042
AdaSD	8.477	6.132	2.897	0.490	0.834	1.071
Llama 3.1 70B – Llama 3.2 1B						
	tks/sec	#cand	#match	AccRate	accuracy	speedup
Draft	95.141	-	-	-	0.453	10.878
Target	4.376	-	-	-	0.836	0.500
Vanilla	8.746	5.000	1.714	0.343	0.838	1.000
AssistedGen	8.920	4.759	1.845	0.388	0.833	1.020
Gen-Only	9.085	5.205	1.956	0.376	0.835	1.039
Verify-Only	9.346	4.226	1.903	0.438	0.817	1.069
AdaSD	9.795	5.297	2.193	0.422	0.815	1.120
Qwen 2.5 72B – Qwen 2.5 7B						
	tks/sec	#cand	#match	AccRate	accuracy	speedup
Draft	38.544	-	-	-	0.725	3.667
Target	4.690	-	-	-	0.845	0.446
Vanilla	10.511	5.000	2.811	0.562	0.837	1.000
AssistedGen	11.436	6.109	3.618	0.592	0.837	1.088
Gen-Only	10.735	8.962	4.210	0.470	0.841	1.021
Verify-Only	11.042	4.920	3.010	0.610	0.837	1.051
AdaSD	11.470	8.904	4.539	0.495	0.839	1.091

Table 6: Inference results on the MMLU validation set with different schemes across three model pair combinations.