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 thresholds: one to determine when to stop candidate token generation and another to decide token acceptance, both 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 49% speedup over standard speculative decoding while limiting accuracy degradation to under 2%, 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 utility 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 improve LLM inference efficiency. AdaSD introduces two adaptive thresholds: one to determine when to terminate candidate token generation by the draft model, and the other to decide the acceptance criterion for candidate tokens verified by the target model. Specifically, we employ entropy and Jensen–Shannon distance as principled measures to set these thresholds. Furthermore, both thresholds 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 decoding method that jointly adjusts candidate token generation length and acceptance criteria to balance inference speed and accuracy.
- 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 49% speedup over standard speculative decoding, while limiting accuracy degradation to less than 2%.

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 M_q, referred to as the *draft model*, is introduced to approximate the target model's output distribution.

Figure 1a illustrates the process of speculative decoding. 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.

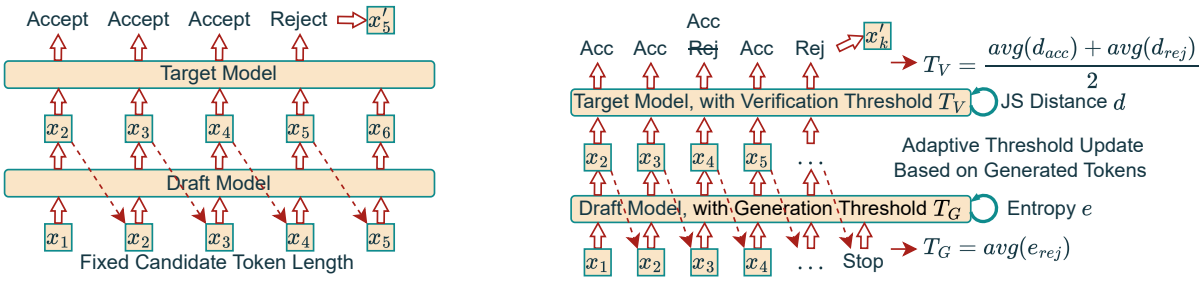
2.1.3 Sampling Strategy

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. Verification proceeds sequentially, and tokens continue to be accepted until the first mismatch or all candidate tokens are successfully verified.

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



(a) Speculative Decoding

(b) AdaSD

Figure 1: Progression of language model inference. (a) Speculative decoding accelerates inference by introducing a draft model and a generation–verification mechanism. (b) AdaSD employs adaptive thresholds based on entropy and Jensen–Shannon distance to dynamically control the decoding process.

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 Information Theory

2.2.1 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:

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

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.

2.2.2 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:

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

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.

2.2.3 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:

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

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.

2.2.4 Jensen–Shannon Divergence

To address the unbounded nature of KL divergence, the Jensen–Shannon (JS) divergence is defined as:

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

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

2.2.5 Jensen–Shannon Distance

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 the draft model's transformer layers to better align with the target distribution. Both approaches, however, entail significant model adaptation and require extra training before yielding performance gains.

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 probability threshold, terminating 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. SpecDec++ (Huang et al., 2025) formulates the problem as a Markov decision process and introduces 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. 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.

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 our 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, and it optimizes 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 operates with two thresholds. The *generation threshold* (Section 4.3) determines when the draft model should stop generating candidate tokens and hand them off to the target model. The *verification threshold* (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 1b.

To further improve efficiency, AdaSD introduces a heuristic feedback mechanism that adaptively adjusts thresholds 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, and uses them to update thresholds on the fly. Further details of the algorithm are provided in Appendix A.

4.2 Empirical Study

We begin by considering the generation threshold of the draft model. As discussed earlier, entropy quantifies the uncertainty associated with a *single* random variable. Therefore, we use entropy 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 (Elhoushi et al., 2024).

For the verification threshold, 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 of both models and effectively distinguishes accepted from rejected tokens. At the same time, the indicator must be bounded to ensure that the adaptive threshold functions reliably and robustly against outliers, preventing extreme values from destabilizing the verification process.

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. For example, this can occur when the draft model generates a rare token that the target model assigns near-zero probability. To address this issue, we propose to use JS distance. While both JS divergence and JS distance are bounded, JS distance offers the additional advantage of defining a proper metric, which allows the threshold to be directly and consistently adjusted within a metric space.

To validate the effectiveness of these measures, we conduct an 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., 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 both models, we examine the distribution of JS distances across the sample data. As shown in Figure 3, a certain samples of rejected tokens have JS distances near 1, whereas such cases would that produce extreme values that destabilize the threshold with 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 provide insights that JS distance effectively captures model agreement and measures uncertainty, allowing robust thresholding for token verification in AdaSD.

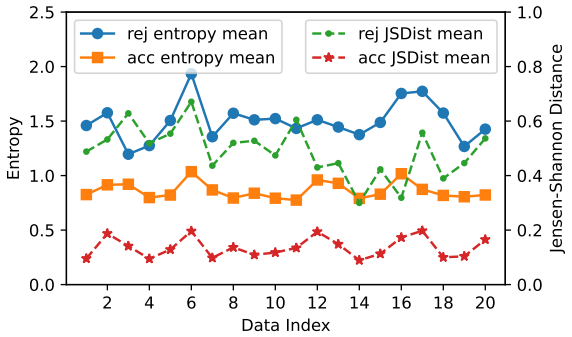


Figure 2: Mean entropy and Jensen–Shannon distance for accepted and rejected tokens, evaluated individually on 20 samples from Alpaca dataset, using Llama 3.1 8B and 70B as the draft and target models, respectively.

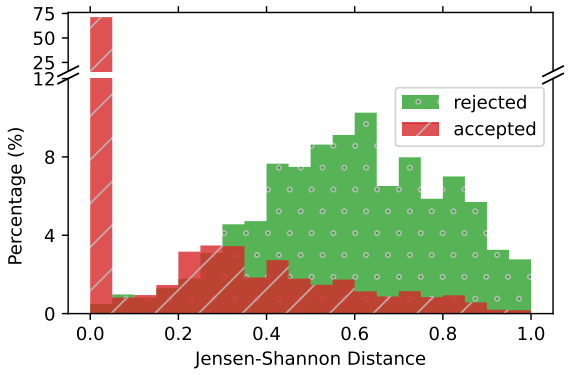


Figure 3: Jensen–Shannon distance distribution of accepted and rejected tokens across 20 samples from the Alpaca dataset.

4.3 Generation Threshold

The generation threshold derived from entropy determines when to stop candidate token generation in the draft model. If the entropy of the newly generated candidate token exceeds the threshold, indicating that the token is likely to be rejected, the generation process is terminated and controls proceed to the verification step.

The generation threshold is updated adaptively during inference. At each iteration, the entropy

of the rejected candidate token is recorded. The threshold is then recalculated as the mean entropy of all rejected tokens collected from the start of the process. Let T_G denote the generation threshold, and $avg(e_r)$ represent the mean entropy of the previously rejected tokens. The generation threshold is then updated as follows:

$$T_G = avg(e_r).$$

This adaptive mechanism enables the threshold to adjust dynamically based on the draft model's own output, ensuring that token generation stops at the most appropriate point.

4.4 Verification Threshold

The verification threshold leverages the JS distance to regulate token acceptance in the target model. At each verification step, we compute the JS distance between the probability distributions of the verification pair (the candidate token proposed by the draft model and the token predicted by the target model). If the computed distance is smaller than the threshold, the candidate token is accepted, regardless of whether the actual generated tokens from both models are identical or not.

Determining an appropriate distance threshold, however, is non-trivial. One possible strategy is to set the threshold based on the mean JS distance of previously accepted tokens. While this ensures conservative verification and high accuracy, it results in few tokens being accepted, thereby reducing the throughput of speculative decoding. Alternatively, we could set the threshold using the mean JS distance of previously rejected tokens. However, this could lead to excessive acceptance of low-confidence tokens, and thus, significantly degrade overall accuracy. Therefore, an effective verification threshold must strike a careful balance between efficiency and reliability. It should be tight enough to preserve semantic alignment with the target model while being loose enough to enable meaningful acceleration during inference.

To this end, we incorporate information from both accepted and rejected tokens. As shown in Figure 2, the JS distances of rejected tokens are consistently higher and clearly separable from those of accepted tokens. Motivated by this observation, we define the verification threshold as the midpoint between the mean JS distances of the previous accepted and rejected tokens. That is,

$$T_V = \frac{avg(d_a) + avg(d_r)}{2},$$

where T_V denotes the verification threshold, and $avg(d_a)$ and $avg(d_r)$ represent the mean JS distances of previously accepted and rejected tokens, respectively. This heuristic strikes a balance that guides the verification process between precision and efficiency.

Similar to the generation threshold, the verification threshold is adjusted adaptively throughout inference. At each iteration, the JS distance of each newly generated token is recorded separately based on whether it was accepted or rejected. The mean JS distance of all accepted and rejected tokens since the start of the speculative decoding process are updated accordingly, and the verification threshold is subsequently adjusted to reflect these updated statistics. For example, assume we have recorded twenty accepted and twenty rejected tokens with JS distances using the numbers shown in Figure 2. In this case, the mean JS distances of accepted and rejected tokens are 0.15 and 0.49 respectively, and the verification threshold will be 0.32.

In addition to this approach, we explore several more sophisticated heuristics to adjust the verification threshold, which are described in Appendix B. However, these methods do not outperform the proposed simple yet effective heuristic.

5 Experiments

5.1 Experimental Setup

Benchmarks. 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) benchmarks programming and code generation. MMLU (Hendrycks et al., 2021) covers multiple-choice questions across a wide range of subjects.

Models. We select two representative model families, Llama 3 (Grattafiori et al., 2024) and Qwen 2.5 (Yang et al., 2025), with different size configurations. We 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.

Baselines. 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), which extends the state-of-the-art DISCO method (Mamou et al., 2024) without the need to train a classifier beforehand. In the following experiments, we use *Vanilla* and *AssistedGen* to represent both methods, respectively.

Implementation. All experiments are conducted using the Hugging Face Transformer framework (Wolf et al., 2020) version 4.55. We run tests on four NVIDIA A6000 GPUs and enable "`device_map=auto`" to evenly split the model on all available GPUs. All evaluated approaches use sampling to select the predicted tokens.

5.2 Performance Results

Table 1 presents the throughput and accuracy results. The performance of draft and target models are also included for reference. For AdaSD, we also evaluate two variants: *Gen-Only*, which applies only the generation threshold, and *Verify-Only*, which applies only the verification threshold. Efficiency is measured as the speedup over the vanilla speculative decoding method. Because our AdaSD adjusts the acceptance criteria, we also report the resulting accuracy. Note that the accuracies of all other methods may differ slightly from the target model due to the use of sampling.

Overall, AdaSD consistently 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 23–49% higher throughput than Vanilla and 10% higher than AssistedGen, with accuracy of only 0.5–2% lower than 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 only about 8% of improvement. This is because MMLU is a multiple-choice benchmark, where the model generates only a smalls 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. Consequently, this can lead to relatively more number of candidate tokens, which increases both generation and verification overhead, and the likelihood of tokens being rejected as well. This explains why AssistedGen outperforms AdaSD in the Qwen 2.5 experiments. These findings suggest that AdaSD's efficiency could be further improved by incorporating more precise generation and acceptance-aware control mechanisms.

5.3 Ablation Studies

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

Gen-Only. Gen-Only demonstrates strong inference speedup over Vanilla, achieving a 17–42% improvement across the experiments. In some cases, it even outperforms or achieves comparable efficiency with AssistedGen. This improvement arises from generating longer candidate sequences while using an effective control mechanism. The draft model continues generating tokens until the token entropy exceeds the generation threshold, 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 7% 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 threshold, AdaSD achieves an additional 13% improvement. These

	GSM8K				HumanEval				MMLU			
Llama 3.1 70B – Llama 3.1 8B												
	tks/sec	JSDist	acc	speedup	tks/sec	JSDist	acc	speedup	tks/sec	JSDist	acc	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	14.431	0.310	0.933	1.039	14.499	0.309	0.793	1.013	8.481	0.418	0.833	1.072
AdaSD	17.505	0.309	0.937	**1.260**	18.839	0.313	0.744	**1.316**	8.623	0.419	0.822	**1.090**
Llama 3.1 70B – Llama 3.2 1B												
	tks/sec	JSDist	acc	speedup	tks/sec	JSDist	acc	speedup	tks/sec	JSDist	acc	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	17.192	0.375	0.941	1.046	17.793	0.359	0.726	1.019	9.385	0.465	0.816	1.073
AdaSD	23.202	0.369	0.931	**1.412**	26.045	0.356	0.738	**1.491**	9.937	0.463	0.813	**1.136**
Qwen 2.5 72B – Qwen 2.5 7B												
	tks/sec	JSDist	acc	speedup	tks/sec	JSDist	acc	speedup	tks/sec	JSDist	acc	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	14.663	0.321	0.918	1.008	13.821	0.354	0.780	1.014	10.952	0.408	0.849	1.042
AdaSD	18.588	0.324	0.914	1.278	16.752	0.344	0.817	1.229	11.395	0.407	0.835	1.084

Table 1: Comparison of speculative decoding methods across different benchmarks for three draft–target model pairs. Metrics include decoding speed (tks/sec), average JS distance at the verification threshold, accuracy, and relative speedup compared to the Vanilla baseline. Gen-Only and Verify-Only represent variants of AdaSD that use only the generation threshold and verification threshold, respectively.

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	4.287	0.857	1.008	5.000	4.026	0.805	1.014	5.000	2.983	0.597	1.042
AdaSD	14.491	10.434	0.720	1.278	13.252	8.879	0.670	1.229	9.205	4.596	0.499	1.084

Table 2: Decoding results on different datasets using Qwen 2.5. The result demonstrates how the number of generated candidates (#cand), accepted tokens (#match), and the resulting acceptance rate (AccRate) collectively influence the decoding speedup.

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 significantly improves speculative decoding speed, with only a minor reduction in accuracy. 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, designing more effective heuristics to adjust the JS distance could improve the acceptance rate and further reduce accuracy degradation. 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

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.

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.

Mostafa Elhoushi, Akshat Shrivastava, Diana Liskovich, Basil Hosmer, Bram Wasti, Liangzhen Lai, Anas Mahmoud, Bilge Acun, Saurabh Agarwal, Ahmed Roman, Ahmed Aly, Beidi Chen, and Carole-Jean Wu. 2024. LayerSkip: Enabling early exit inference and self-speculative decoding. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics*, pages 12622–12642.

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*.

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.

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*.

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.

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.

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.

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.

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.

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 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 prevent 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.

B Extensive Experiments

We explore three alternative heuristics for adjusting the verification threshold to better understand the influence of JS distance. We use T_V to denote the verification threshold, d_a and d_r to denote the JS distances of an accepted and a rejected token, respectively, and n_a and n_r to denote the total numbers of accepted and rejected tokens that have been recorded.

- **Variant A**: Compute the verification threshold by the mean JS distances of accepted tokens:

$$T_V = \frac{\sum_{i=1}^{n_a} d_a^{(i)}}{n_a}.$$

- **Variant B**: Compute the verification threshold by weighting the mean JS distances of accepted and rejected tokens with their respective total counts:

$$T_V = \frac{\sum_{i=1}^{n_a} d_a^{(i)} + \sum_{j=1}^{n_r} d_r^{(j)}}{n_a + n_r}.$$

Algorithm 1: Adaptive Speculative Decoding

Input: the target model M_p, the draft model M_q, the context X, the maximum generated tokens K, the maximum window size W

Output: the final generated context X

Initialization : $L_E^R = L_D^R = L_D^A = [\text{null}]$ /* L: list, E: entropy, D: JS distance, A: accepted, R: rejected */,
$T_G = T_V = 0$ /* T: threshold, G: generation, V: verification */,
$w = 0$ /* window length 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 $T_G < H(q_i)$ **then** // Check generation threshold
      break
  // 2. The target model verifies tokens in parallel
  $p_1, p_2, \ldots, p_{w+1} \leftarrow M_p(X, x_{1:0}), M_p(X, x_{1:1}), \ldots, M_p(X, x_{1:w})$
  $y_1, y_2, \ldots, y_{w+1} \sim p_1, p_2, \ldots, p_{w+1}$ // Sample y_i from p_i
  for $i \leftarrow 1$ **to** w **do**
    if $x_i \neq y_i$ ***and*** $T_V < D_{\text{JS}}(p_i \parallel q_i)$ **then** // Check verification threshold
      L_E^R.append$(H(q_i))$ // Record rejected entropy
      L_D^R.append$(D_{\text{JS}}(p_i \parallel q_i))$ // Record rejected JS distance
      $w \leftarrow i - 1$ // Record accepted window length
      break
    L_D^A.append$(D_{\text{JS}}(p_i \parallel q_i))$ // 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
  $T_V \leftarrow \left(\text{average}(L_D^A) + \text{average}(L_D^R)\right)/2$ // Update verification threshold

- **Variant C**: We define two additional counters, c_a and c_r, to record the numbers of accepted and rejected sequences, respectively. For each draft token sequence, if it contains at least one accepted token, we increase c_a by one; if it contains at least one rejected token, we increase c_r by one. The verification threshold is defined as:

$$T_V = \frac{c_a \times \frac{\sum_{i=1}^{n_a} d_a^{(i)}}{n_a} + c_r \times \frac{\sum_{j=1}^{n_r} d_r^{(j)}}{n_r}}{c_a + c_r}.$$

  When the numbers of accepted and rejected sequences are equal (i.e., $c_a = c_r$), this is the same as AdaSD.

Tables 3, 4, and 5 present the performance results on the GSM8K, HumanEval, and MMLU datasets, respectively. For a fair comparison, we fix the random seed using "`torch.manual_seed(48763)`". Overall, these heuristic variants do not outperform AdaSD. The verification thresholds derived from Variants A and B are close to zero, which fail to provide any speedup and even cause accuracy degradation. In contrast, Variant C achieves a better balance between accuracy and speedup. We observe that c_a is typically greater than c_r, which leads Variant C to a lower verification threshold compared to AdaSD and, thus, a lower token acceptance rate. As a result, Variant C achieves higher accuracy than AdaSD, but its speedup is slightly lower.

We argue that setting AdaSD's threshold at half the JS distances of accepted and rejected tokens is sufficiently conservative. If the verification threshold is increased beyond this point, many tokens that should be rejected would instead be accepted, potentially leading to a drop in accuracy.

openai/gsm8k, main, test, num_rows = 1319							
Llama 3.1 70B – Llama 3.1 8B							
	tks/sec	JSDist	#cand	#match	AccRate	acc	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	14.431	0.310	5.000	4.270	0.854	0.933	1.039
AdaSD	17.505	0.309	11.472	8.737	0.762	0.937	1.260
Variant A	16.213	0.048	11.059	7.824	0.707	0.938	1.167
Variant B	16.399	0.069	11.121	7.935	0.714	0.942	1.181
Variant C	16.733	0.179	11.387	8.261	0.725	0.931	1.205
Llama 3.1 70B – Llama 3.2 1B							
	tks/sec	JSDist	#cand	#match	AccRate	acc	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	17.192	0.375	4.967	3.806	0.766	0.941	1.046
AdaSD	23.202	0.369	10.823	7.007	0.647	0.931	1.412
Variant A	20.899	0.059	10.394	6.218	0.598	0.935	1.272
Variant B	21.214	0.099	10.404	6.243	0.600	0.943	1.291
Variant C	21.811	0.254	10.581	6.562	0.620	0.943	1.327
Qwen 2.5 72B – Qwen 2.5 7B							
	tks/sec	JSDist	#cand	#match	AccRate	acc	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	14.663	0.321	4.971	4.287	0.862	0.918	1.008
AdaSD	18.588	0.324	14.491	10.434	0.720	0.914	1.278
Variant A	18.267	0.025	14.431	10.225	0.709	0.914	1.256
Variant B	18.265	0.044	14.361	10.176	0.709	0.914	1.256
Variant C	18.364	0.174	14.347	10.236	0.713	0.914	1.262

Table 3: 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	JSDist	#cand	#match	AccRate	acc	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	14.499	0.309	4.970	4.361	0.877	0.793	1.013
AdaSD	18.839	0.313	13.491	10.634	0.788	0.744	1.316
Variant A	17.896	0.030	13.168	9.889	0.751	0.762	1.250
Variant B	18.000	0.048	12.874	9.789	0.760	0.701	1.257
Variant C	18.197	0.167	13.069	10.022	0.767	0.774	1.271
Llama 3.1 70B – Llama 3.2 1B							
	tks/sec	JSDist	#cand	#match	AccRate	acc	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	17.793	0.359	5.000	4.041	0.808	0.726	1.019
AdaSD	26.045	0.356	12.240	8.512	0.695	0.738	1.491
Variant A	24.943	0.039	11.950	8.055	0.674	0.750	1.428
Variant B	24.809	0.070	11.826	8.001	0.677	0.738	1.420
Variant C	24.795	0.229	11.936	7.993	0.670	0.780	1.420
Qwen 2.5 72B – Qwen 2.5 7B							
	tks/sec	JSDist	#cand	#match	AccRate	acc	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	13.821	0.354	5.000	4.026	0.805	0.780	1.014
AdaSD	16.752	0.344	13.252	8.879	0.670	0.817	1.229
Variant A	16.711	0.025	13.199	8.832	0.669	0.787	1.226
Variant B	16.712	0.051	13.199	8.832	0.669	0.787	1.226
Variant C	16.416	0.206	13.282	8.705	0.655	0.793	1.204

Table 4: 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	JSDist	#cand	#match	AccRate	acc	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	8.481	0.418	5.000	2.386	0.477	0.833	1.072
AdaSD	8.623	0.419	6.147	3.032	0.493	0.822	1.090
Variant A	8.044	0.123	6.278	2.729	0.435	0.839	1.017
Variant B	8.170	0.230	6.169	2.770	0.449	0.832	1.033
Variant C	8.504	0.351	6.316	2.966	0.470	0.837	1.075
Llama 3.1 70B – Llama 3.2 1B							
	tks/sec	JSDist	#cand	#match	AccRate	acc	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	9.385	0.465	5.000	1.938	0.388	0.816	1.073
AdaSD	9.937	0.463	5.284	2.265	0.429	0.813	1.136
Variant A	9.039	0.150	5.209	1.970	0.378	0.824	1.034
Variant B	9.316	0.311	5.342	2.074	0.388	0.816	1.065
Variant C	9.817	0.432	5.275	2.246	0.426	0.807	1.122
Qwen 2.5 72B – Qwen 2.5 7B							
	tks/sec	JSDist	#cand	#match	AccRate	acc	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	10.952	0.408	4.917	2.983	0.607	0.849	1.042
AdaSD	11.395	0.407	9.205	4.596	0.499	0.835	1.084
Variant A	10.687	0.079	8.956	4.195	0.468	0.837	1.017
Variant B	10.735	0.168	8.960	4.216	0.471	0.830	1.021
Variant C	11.146	0.341	9.084	4.441	0.489	0.843	1.060

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