# Hierarchical Speculative Decoding with Dynamic Windows for Efficient Language Model Inference

**Shen-sian Syu**
National Taiwan University, Taiwan
d07921013@ntu.edu.tw

**Hung-yi Lee**
National Taiwan University, Taiwan
hungyilee@ntu.edu.tw

## Abstract

Speculative Decoding (SD) utilizes an efficient draft model to generate multiple tokens, which are subsequently verified in parallel by a target model. This approach has shown significant potential for accelerating inference in large language models (LLMs), with performance heavily reliant on the hyperparameter $K$—the window size. However, previous methods often depend on simple heuristics to select $K$ or dynamically adjust the window size, which may necessitate additional training or careful resource management to avoid competition. To address these challenges, we propose **H**ierarchical **S**peculative **D**ecoding with **D**ynamic **W**indow (HSDDW), a straightforward framework that eliminates the need for additional training. Specifically, we introduce a *self-verify* mechanism that enables the draft model to autonomously decide when to stop generating tokens. Additionally, by integrating a hierarchical structure that leverages the capabilities of models of different sizes, we significantly enhance the overall speed of the system. HSDDW demonstrates competitive performance across four datasets, achieving notable speedups of $2.91\times$ on MT-Bench and $2.99\times$ on Alpaca, outperforming existing state-of-the-art methods.

## 1 Introduction

In recent years, large language models (LLMs) like GPT-4, LLaMA-2, Vicuna, and Mistral (OpenAI, 2023; Touvron et al., 2023a; Chiang et al., 2023; Jiang et al., 2023) have demonstrated exceptional performance across a wide range of tasks. However, LLMs face significant challenges, particularly due to memory bandwidth constraints (Patterson, 2004; Shazeer, 2019; Xia et al., 2024), and the bottleneck of auto-regressive inference, where the token-by-token decoding process introduces substantial latency during inference. These issues are especially critical in environments with limited computational resources, such as mobile devices.

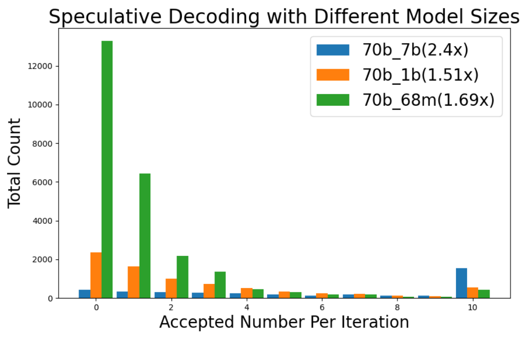


Figure 1: Comparison of the total number of accepted tokens for draft models of different sizes across various acceptance counts in speculative decoding. The target model used is Llama-2-Chat-70B, with draft models including Llama-2-Chat-7B, Llama-2-Chat-1B, and Llama-68M, respectively. The values in parentheses represent the speed improvement compared to autoregressive methods. Note that the fixed window size is set to $K = 10$.

To address these challenges, Speculative Decoding (SD) has been proposed (Stern et al., 2018; Xia et al., 2023; Leviathan et al., 2023; Chen et al., 2023). Speculative decoding accelerates inference by leveraging a smaller and faster draft model to initially predict $K$ tokens, where $K$ is a fixed hyperparameter that defines the window size. These tokens are then verified in parallel by the larger target model, ensuring that the output distribution aligns with that of the target model. This approach significantly reduces latency, providing more than a $2\times$ speedup over using only the target mode.

Enhancing the speed of the draft model has become a central focus in ongoing speculative decoding research. One research direction focuses on dynamically adjusting the window size $K$. As shown in Figure 1, smaller draft models do not consistently lead to faster decoding, and the number of accepted tokens significantly decreases with smaller models. If $K$ is too large, the draft model may generate too many tokens that do not align with the target model, while a window size that is too small limits the draft model's potential for speed improvement. Moreover, the choice of window size is closely related to the draft model's acceptance rate (Leviathan et al., 2023), which is unavailable during inference.

To tackle this issue, recent research has explored enabling the draft model to dynamically determine the number of tokens generated at each inference step, rather than relying on a fixed hyperparameter window size $K$. This approach reduces redundancy when the draft model makes incorrect predictions. For instance, SpecDec++ (Huang et al., 2024) introduces an acceptance prediction head, trained on the draft model, to determine when to stop token generation. Similarly, PEARL (Liu et al., 2024) proposes parallelizing the drafting and verification phases using two strategies—pre-verification and post-verification—to allow adaptive draft lengths based on varying scenarios. However, it is important to note that SpecDec++ (Huang et al., 2024) requires additional training and specific datasets. On the other hand, PEARL (Liu et al., 2024) involves simultaneous inference of the draft and target models, which necessitates efficient GPU resource scheduling to avoid resource competition.

Therefore, in this Chapter, we aim to address the following questions: *Can we dynamically adjust the window size K without requiring any additional training?* To explore this, we propose a straightforward method, *Self-verify*, where the draft model assesses its own confidence in the generated predictions to decide whether to continue or stop token generation, thereby achieving a dynamic window size. Additionally, We know that LLaMA-68M (68m) has a faster inference speed than LLaMA-7B (7b). However, as shown in Figure 1, the speculative decoding model using 70b-68m is slower than the 70b-7b model. This leads us to our second question: *Can we leverage models of different sizes to enhance speed?* Building on this concept, we introduce **H**ierarchical **S**peculative **D**ecoding with a **D**ynamic **W**indow (HSDDW). In HSDDW, we integrate a stronger draft model, closer in capability to the target model, to take over the verification phase, which we term *pre-verify*. By utilizing this hierarchical structure, HSDDW can effectively leverage models of different sizes, further accelerating speculative decoding.

Our key contributions can be summarized as follows:

- We propose HSDDW, a novel inference acceleration framework, which employs a simple yet effective *self-verify* mechanism and a hierarchical structure, requiring no additional training to solve the dynamic window problem.
- Our method is insensitive to the window size $K$, demonstrating greater robustness and stability in performance.
- HSDDW demonstrates competitive performance across four datasets, achieving notable speedups of $2.91\times$ on MT-Bench, $2.92\times$ on HumanEval, and $2.99\times$ on Alpaca, outperforming existing state-of-the-art methods.

## 2 Background

### 2.1 Speculative Decoding

Consider a language model $\mathcal{M}$, which generates an output sequence $y$ based on a given input prefix $x$. To speed up the generation process, speculative decoding employs a **compact draft model** $\mathcal{M}_q$ to propose candidate tokens, which are subsequently validated by a **larger target model** $\mathcal{M}_p$. speculative decoding can be divided into two steps: *draft* and *verify*, as illustrated in Figure 2. In the drafting step, $\mathcal{M}_q$ samples tokens according to its next-token probability distribution $q(y_t \mid x, y_{<t})$ at each time step $t$, while $\mathcal{M}_p$ produces predictions using $p(y_t \mid x, y_{<t})$. When generating the entire sequence $y$, the sequence-level distributions are represented by $p_{\leq K}(y \mid x)$ for $\mathcal{M}_p$ and $q_{\leq K}(y \mid x)$ for $\mathcal{M}_q$. Generation concludes either when an end-of-sequence token is produced or when a predefined window fixed size $K$ is reached. After generating a sequence block, each token in $y_i$ is sequentially verified by $\mathcal{M}_p$, where $i = 1, 2, ..., K$. If any token is rejected during this process, the first rejected token is resampled from an adjusted probability distribution, and all tokens following the rejection are discarded, the rejection sampling can be found in (Chen et al., 2023). The generation stops either when an end-of-sequence token is sampled or the maximum sequence length $T$ is reached.

### 2.2 Related Work

Accelerating speculative decoding has become a critical approach in enhancing the efficiency of autoregressive models. Prior works (Xia et al., 2023; Zhou et al., 2023) have focused on improving the alignment between the draft model's distribution

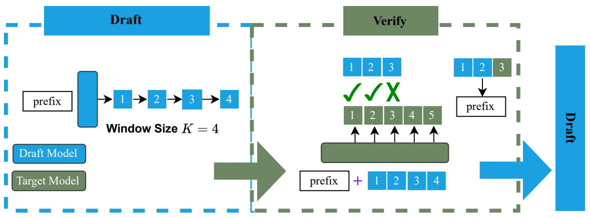


Figure 2: Speculative Decoding first efficiently drafts multiple tokens by given fixed window size $K$ and then verifies them in parallel using the target LLM.

and that of the target model to enhance prediction accuracy. Additionally, some studies explore retrieval-based techniques, utilizing a repository of previously generated knowledge to retrieve and apply relevant tokens based on the current context (He et al., 2024; Zhao et al., 2024). Furthermore, approaches such as (Miao et al., 2024; Cai et al., 2024; Spector and Re, 2023) leverage token trees, enabling the target model to verify multiple outputs from the draft model simultaneously.

Triforce (Sun et al., 2024) combines the original model weights with a dynamic sparse KV cache optimized via retrieval, employing a draft model as an intermediate layer within a hierarchical structure. Additionally, a smaller model is used to speculate on the draft outputs, effectively reducing drafting latency while preserving both accuracy and efficiency.

These methods are compatible with HSDDW or *Self-verify* and can be stacked without conflict. As mentioned earlier in the introduction 1, both SpecDec++ (Huang et al., 2024) and PEARL (Liu et al., 2024) aim to design dynamic window sizes to accelerate speculative decoding. However, the key difference between HSDDW and SpecDec++ is that HSDDW requires no additional training. Unlike PEARL, our approach eliminates the need for special GPU resource scheduling to avoid resource competition. Additionally, our method differs from Triforce (Sun et al., 2024), where the retrieval cache draft model is solely used for verification. In contrast, the additional draft model in HSDDW not only generates tokens but can also perform verification.

## 3 Methodology

The simplest way to enhance the draft model's speed is by using a more efficient, smaller-sized model. However, smaller draft models often have greater discrepancies from the target model's distribution, making it harder for the draft model to generate long sequences that the target model can accept in a single pass. As shown in Figure 1, speculative decoding with the Llama-2-Chat-70B (70b) and Llama-68M (68m) pair 70b-68m often results in the first token being rejected, leading the draft model to waste time generating tokens that don't align with the target model. Consequently, 70b-68m speed is slower than the other two speculative decoding configurations. Our goal, therefore, is to develop a mechanism where the draft model stops generating tokens when it predicts they will not be accepted, effectively implementing a dynamic window size. We propose the *Self-verify* method, which uses entropy measurements to achieve this dynamic window, and we further introduce the HSDDW method that integrates a hierarchical structure to improve overall performance.

### 3.1 Self-verify: verify the draft token by itself.

The purpose of this method is to reduce the number of unaccepted tokens by measuring the entropy $\mathcal{H}$ of the draft model's token distribution to assess the model's confidence in the currently generated tokens, specifically $\mathcal{H}(q(y_t \mid x, y_{<t}))$. When the confidence is high, indicated by $\mathcal{H}(q(y_t \mid x, y_{<t})) \leq \tau$, the draft model continues generating tokens. Conversely, if the confidence is low, such that $\mathcal{H}(q(y_t \mid x, y_{<t})) > \tau$, the generation process is halted. We refer to $\tau$ as the *Confidence Threshold*. This method, where the draft model decides whether to stop based on its own confidence without completing the window size $K$ iterations, is referred to as *self-verify*.

Specifically, the draft model generates tokens autoregressively and simultaneously calculates the entropy $\mathcal{H}(q_t)$ for each position $t$[1]. As illustrated in Figure 3, when $\mathcal{H}(q_3) > \tau$ at position 3, the generation process stops, and the tokens are sent to the target model for verification. Here, confidence threshold $\tau$ is dynamic and is calculated as the average of $\mathcal{H}$ for all tokens rejected by the target model in the current sentence. Further details are provided in Algorithm 1.

[1]We use $q_t$ to represent $q(y_t|x, y_{<t})$ for simplicity.

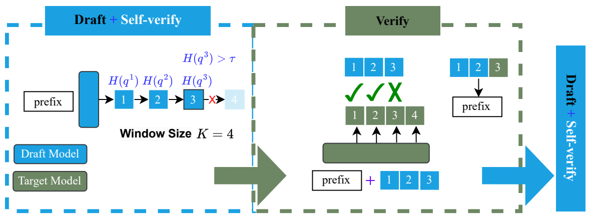


Figure 3: A framework of speculative decoding with dynamic window using *Self-verify*. In this example, when the draft model generates the third token, its entropy $\mathcal{H}(q_3) > \tau$, causing the draft model to stop generating the fourth token (with $K = 4$). After verification by the target model, the token at the third position is rejected, prompting a resampling process for the third token.

## 3.2 HSDDW

Next, we introduce a hierarchical structure, aiming to leverage a stronger draft model to assist the target model in early-stage verification, referred to as *Pre-verify*. This approach reduces the number of target model inferences, thereby accelerating the overall model speed. Furthermore, we incorporate the self-verify method to further enhance performance.

Specifically, HSDDW employs two draft models and one target model: a fast draft model $\mathcal{M}_1$, a stronger draft model $\mathcal{M}_2$, and the target model $\mathcal{M}_p$. The framework has two main objectives: first, to offload some of the verification burden from $\mathcal{M}_p$ by utilizing $\mathcal{M}_2$, thereby reducing the number of inferences performed by $\mathcal{M}_p$; second, to efficiently leverage the speed of $\mathcal{M}_1$. In our approach, $\mathcal{M}_1$ is responsible for the primary drafting phase, while $\mathcal{M}_2$ performs *pre-verification*. Once $\mathcal{M}_2$ loses confidence in its own verification, the generated tokens are passed to $\mathcal{M}_p$ for final verification. The complete architecture of HSDDW is illustrated in Figure 4, consisting of three stages: the first stage is **Draft + Self-verify**, the second stage is **Pre-verify + Self-verify**, and the third stage is **Verify**.

The process of HSDDW is outlined as follows. Initially, the framework enters the first stage, **Draft + Self-verify**, where $\mathcal{M}_1$ performs drafting while conducting self-verification. The tokens generated by $\mathcal{M}_1$, after self-verification, are then passed to the second stage, **Pre-verify + Self-verify**, where $\mathcal{M}_2$ carries out both pre-verification and self-verification. This stage branches into two cases: (i) When $\mathcal{H}(q_2^i) \leq \tau_2$, indicating that $\mathcal{M}_2$ is confident in the current output, the process reverts to the first stage, allowing $\mathcal{M}_1$ to continue drafting. (ii) When $\mathcal{H}(q_2^i) > \tau_2$, indicating lower confidence from $\mathcal{M}_2$, the process proceeds to the third stage. In the third stage, similar to the standard verification phase, all tokens generated by both $\mathcal{M}_1$ and $\mathcal{M}_2$ are verified by the target model. More details can be found in Algorithm 2.

## 4 Experiment

### 4.1 Experimental setup

**Tasks and Datasets.** In this study, we followed the methodology of (Huang et al., 2024) and utilized three datasets: Alpaca[2] (Taori et al., 2023), HumanEval[3] (Chen et al., 2021), and GSM8K[4] (Cobbe et al., 2021). We sampled 150 examples from HumanEval and GSM8K for performance benchmarking. Additionally, we incorporated the MT-Bench dataset[5] (Zheng et al., 2024), which comprises a total of 80 multi-turn conversational tasks. Notably, for comparisons with baseline models, we utilized the multi-turn results to ensure a fair evaluation. However, in other analytical experiments, to reduce resource consumption, we conducted evaluations using single-turn tasks.

**Baseline Method.** We selected related studies on dynamic length prediction as our baselines. **(i) SD**:the vanilla speculative decoding (Leviathan et al., 2023; Chen et al., 2023) **(ii) SpecDec++**: (Huang et al., 2024) trains a decision head to determine whether the draft model should continue predicting tokens at each step. **(iii) PEARL**: (Liu et al., 2024) simultaneously executes the drafting and verification phases by employing two strategies that enable adaptive draft lengths for different scenarios, effectively mitigating the issue of mutual waiting. **(iv) EAGLE** (Li et al., 2024): This method effectively addresses uncertainty by enabling accurate second-to-top-layer feature prediction with minimal computational overhead, making it a robust and competitive benchmark.

**Model Pairs.** For model selection, we employed the Llama-2-chat models (Touvron et al., 2023b), using Llama-68m[6] (Miao et al., 2024) and Llama-2-chat 7B as the draft models, and Llama-2-chat 70B as the target model. For the vanilla SD setup, we used the Llama-2-chat 70B as the target model and Llama-2-chat 7B as the draft model. In the

[2]https://huggingface.co/datasets/tatsu-lab/alpaca
[3]https://huggingface.co/datasets/openai/openai_humaneval
[4]https://huggingface.co/datasets/openai/gsm8k
[5]https://huggingface.co/spaces/lmsys/mt-bench/tree/main
[6]https://huggingface.co/JackFram/llama-68m

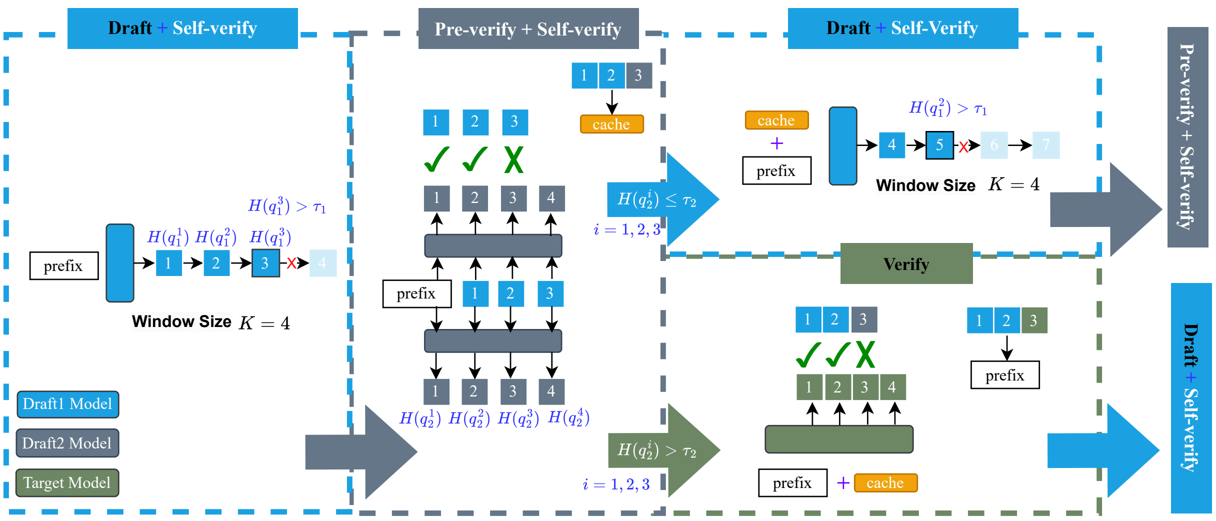


Figure 4: A framework of HSDDW. HSDDW utilizes two draft models, Draft1 Model ($\mathcal{M}_1$) and Draft2 Model ($\mathcal{M}_2$), along with one target model ($\mathcal{M}_p$). $\mathcal{M}_1$ is responsible for the **Draft+Self-verify** stage, $\mathcal{M}_2$ handles the **Pre-verify + Self-verify** stage, and the target model operates in the **Verify** stage. In the **Pre-verify + Self-verify** stage, the entropy of $\mathcal{M}_2$'s output is used to determine whether to continue drafting with $\mathcal{M}_1$ or hand over the tokens to $\mathcal{M}_p$ for final verification.

HSDDW framework, Llama-2-chat 70B, Llama-2-chat 7B, and Llama-68m were used as the target model $\mathcal{M}_p$, the stronger draft model $\mathcal{M}_2$, and the fast draft model $\mathcal{M}_1$, respectively. Additional evaluation details are provided in Appendix A.1 and A.2

## 4.2 Main Results

Our proposed method, HSDDW, demonstrates substantial improvements in decoding efficiency across multiple benchmarks, as illustrated in Table 1. The results are categorized based on two temperature ($temp$) settings: $temp = 1$ and $temp = 0$ (greedy).

For $temp = 1$, HSDDW achieves remarkable speedups, particularly on MT-Bench, HumanEval, and Alpaca, with gains of 2.91×, 2.92×, and 2.99×, respectively, surpassing or matching all baseline methods. Although EAGLE (Li et al., 2024) performs competitively on GSM8K, HSDDW still delivers a strong 2.64× speedup on this benchmark. Importantly, unlike SpecDec++ (Huang et al., 2024) and EAGLE (Li et al., 2024), which require additional training, HSDDW can be directly applied to any off-the-shelf LLM without further modifications.

For $temp = 0$, HSDDW outperforms both PEARL (Liu et al., 2024) and EAGLE (Li et al., 2024) on MT-Bench, GSM8K, and Alpaca benchmarks, demonstrating its robustness and efficiency under greedy decoding conditions. Across the four datasets and two temperature settings, HSDDW exhibits comprehensive and robust speedup performance.

Notably, the speedups achieved under $temp = 0$ are consistently higher than those under $temp = 1$. This observation aligns with findings in prior works (Leviathan et al., 2023; Li et al., 2024; Zhao et al., 2024; Fu et al., 2024; Zhang et al., 2023), which attribute this difference to the increased uncertainty at higher temperatures. Specifically, under higher temperature settings, the draft model faces greater difficulty in predicting tokens that align with the target model's output, as opposed to the lower uncertainty present at $temp = 0$.

Additionally, in Section 4.4, we further discuss "*Why HSDDW's performance on Humaneval and GSM8K is relatively lower?* ", providing insights into its limitations on these specific benchmarks.

## 4.3 Ablation Study

HSDDW integrates both self-verify and pre-verify mechanisms. In the following sections, we will provide a detailed analysis of each stage and component of the architecture, examining their individual contributions to the overall performance. In Table 2, we present the results of our ablation study, which analyzes the impact of the various verification strategies employed in our model across four benchmarks. Specifically, we evaluate the individual effects of Pre-verify and Self-verify, as well as the combined performance of Pre-verify with

| | Methods | MT-Bench | HumanEval | GSM8K | Alpaca |
|---|---|---|---|---|---|
| *T=1* | **AT** | $1.00\times$ | $1.00\times$ | $1.00\times$ | $1.00\times$ |
| | **SpecDec++**(Huang et al., 2024) | - | $2.23\times^{\diamond}$ | $2.26\times^{\diamond}$ | $2.04\times^{\diamond}$ |
| | **EAGLE**(Li et al., 2024) | $2.67\times^{\diamond}$ | $\mathbf{2.92}\times^{\diamond}$ | $\mathbf{2.74}\times^{\diamond}$ | $2.65\times^{\diamond}$ |
| | **HSDDW** | $\mathbf{2.91}\times$ | $\mathbf{2.92}\times$ | $2.64\times$ | $\mathbf{2.99}\times$ |
| *T=0* | **PERAL**(Liu et al., 2024) | $2.48\times^{\diamond}$ | $3.01\times^{\diamond}$ | $2.87\times^{\diamond}$ | - |
| | **EAGLE**(Li et al., 2024) | $3.01\times^{\diamond}$ | $\mathbf{3.52}\times^{\diamond}$ | $3.03\times^{\diamond}$ | $2.97\times^{\diamond}$ |
| | **HSDDW** | $\mathbf{3.13}\times$ | $3.39\times$ | $\mathbf{3.36}\times$ | $\mathbf{3.24}\times$ |

Table 1: Comparison of Different Methods on Various Model Configurations. $T = 0$ represents $temperature = 0$ (greedy decoding), and $T = 1$ represents $temperature = 1$. $\diamond$ denotes results directly cited from the original paper. The symbol ‘ - ’ denotes that the results has not been report in original papers. We bold the best results.

Self-verify. We also compare these findings to the full HSDDW method to highlight the differences in performance. Please note that HSDDW differs from merely combining Pre-verify with Self-verify. Figure 5 illustrates Pre-verify alone and Pre-verify combined with Self-verify.

From the results, several key trends emerge. First, self-verify shows significant improvements across all four benchmarks. Notably, it surpasses PEARL (Liu et al., 2024) on MT-Bench with a speedup of $2.63\times$, and outperforms SpecDec++ (Huang et al., 2024) on HumanEval, reaching a speedup of $2.57\times$. This demonstrates the effectiveness of the Self-verify approach, even as a standalone method.

The inclusion of Pre-verify provides a performance boost. For example, it improves the speedup from $2.40\times$ to $2.53\times$ on MT-Bench and from $2.22\times$ to $2.31\times$ on HumanEval. However, on GSM8K and Alpaca, Pre-verify yields minimal changes compared to the vanilla speculative decoding.

When both Pre-verify and Self-verify are combined, the performance further improves across the board, with a speedup of $2.69\times$ on MT-Bench and $2.66\times$ on HumanEval. Finally, the HSDDW method consistently delivers the best performance, achieving a remarkable speedup of $2.95\times$ on MT-Bench and $2.99\times$ on Alpaca, clearly demonstrating its superiority over both individual and combined verification strategies. For more details, refer to Section 4.4.

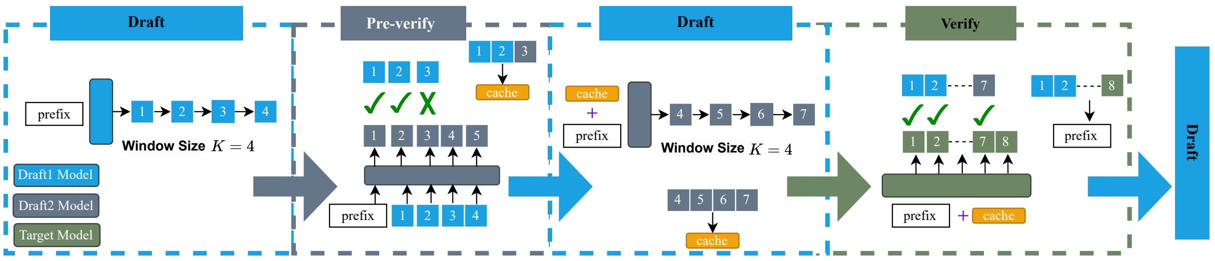
### 4.4 Analysis

**Confidence Threshold** $\tau$ In the self-verify method, entropy is used to evaluate the draft model's confidence in its generated tokens. When $\mathcal{H}(q) > \tau$, the generation process halts. As illustrated in Figure 6, we compared the entropy values of accepted and rejected tokens as determined by the target model in the vanilla speculative decoding setup. Specifically, the "accepted entropy" refers to the average entropy of tokens that were accepted by the target model within their respective sentence contexts. In contrast, the "rejected entropy" represents the average entropy of tokens that were rejected by the target model within the corresponding sentence context. This comparison provides insights into how entropy correlates with token acceptance and rejection during speculative decoding. The results, across both 70b-7b and 7b-68m configurations, reveal a distinct decision boundary in entropy between accepted and rejected tokens. This further confirms entropy as a reliable metric for determining when the draft model should stop generating tokens, aligning well with the target model's decisions during the verification phase in speculative decoding. Based on the above analysis, we design a confidence threshold to enable the draft model to decide when to stop generating tokens.

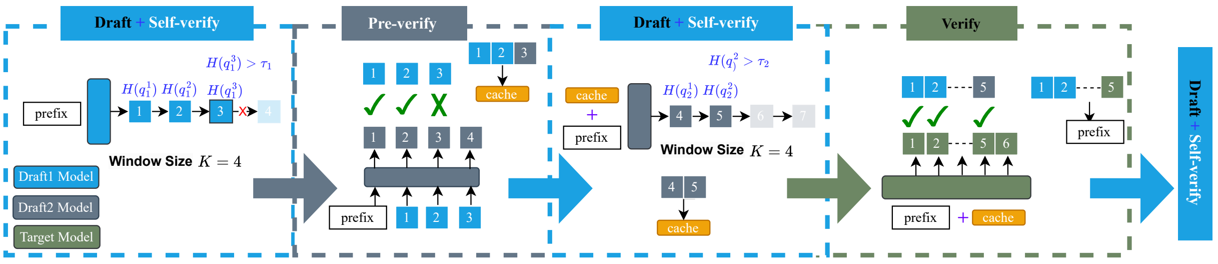
Let $S_{\text{reject}}$ represent the set of positions in a sentence where the draft model's tokens were rejected by the target model. For each rejected token, the corresponding entropy is denoted by ( $\mathcal{H}(q_i)$). When a new token is rejected (at position $i \in S_{\text{reject}}$), the update rule for the threshold $\tau$ is defined as:

$$\tau = \frac{1}{|S_{\text{reject}}|} \sum_{i \in S_{\text{reject}}} \mathcal{H}(q_i)$$

where $|S_{\text{reject}}|$ is the total number of rejected tokens, and $\mathcal{H}(q_i)$) is the entropy of the $i$-th rejected token. This formula calculates $\tau$ as the average en-



(a) SD + Pre-verify



(b) SD + Pre-verify + Self-verify

Figure 5: The upper diagram (a) illustrates speculative decoding with *Pre-verify*, which can be viewed as two speculative decoding models connected sequentially. The bottom diagram (b) shows speculative decoding with both *Pre-verify* and *Self-verify*, which can be considered an extension of (a) with the addition of *Self-verify*.

| **Pre-verify** | **Self-verify** | **HSDDW** | **MT-Bench** | **HumanEval** | **GSM8K** | **Alpaca** |
|---|---|---|---|---|---|---|
| | | | 2.40× | 2.22× | 2.51× | 2.50× |
| ✓ | | | 2.53× | 2.31× | 2.47 × | 2.48× |
| | ✓ | | 2.63× | 2.57× | 2.68× | 2.54× |
| ✓ | ✓ | | 2.69× | 2.66× | 2.65× | 2.63× |
| | | ✓ | 2.95× | 2.92× | 2.64× | 2.99× |

Table 2: Ablation study results showing the impact of Pre-verify and Self-verify strategies on decoding speed across four benchmarks (MT-Bench, HumanEval, GSM8K, and Alpaca).

tropy of all the tokens that have been rejected so far, allowing the threshold to be updated dynamically based on the model's rejection history.

**Speed Comparison Across Different Window Sizes** In speculative decoding, the choice of window size $K$ is crucial for optimizing performance, particularly when paired with the accepted rate of the SD model (Leviathan et al., 2023). As shown in Figure 7, models that leverage both self-verify and pre-verify, as well as the HSDDW model, demonstrate smaller variations in speed across different window sizes compared to vanilla SD. Notably, the speed of HSDDW consistently surpasses that of vanilla SD at all window sizes, indicating greater robustness and stability in performance across different configurations.

**Why HSDDW's performance on Humaneval and GSM8K is relatively lower?** In Table 3, HSDDW demonstrates relatively weaker performance on math-heavy datasets like Humaneval and GSM8K. Specifically, in the MT-Bench dataset, where 10 out of the 80 samples involve mathematical tasks, we observe a notable trend. For purely math-related questions, the performance of vanilla speculative decoding sees a slight decrease, from 2.40× to 2.29×. In contrast, HSDDW experiences a more significant drop, going from 2.95× to 2.08×. However, when math-related problems are removed from MT-Bench, the score for vanilla speculative decoding remains relatively unchanged, while HSDDW shows an increase from 2.08× to 3.04×.

This disparity is likely due to HSDDW utilizing the 68m draft model, which underperforms on mathematical tasks compared to the larger 70b and 7b models. The 68m model's training predominantly involves data from Wikipedia and por-

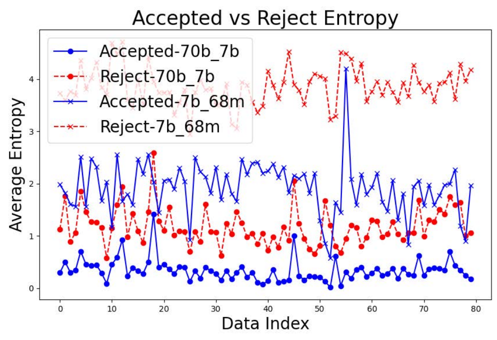


Figure 6: The speculative decoding models for 70b-7b and 70b-68m, showing the entropy values for accepted and rejected tokens on the MT-Bench dataset. A clear decision boundary is observed between acceptance and rejection based on entropy.

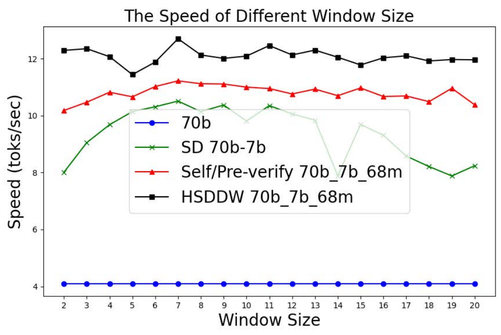


Figure 7: Speed comparison of three speculative decoding methods with varying window sizes on MT-Bench. "Self/Pre-verify" refers to speculative decoding that combines both Self-verify and Pre-verify mechanisms.

tions of the C4-en and C4-realnewslike datasets (Raffel et al., 2020), which are less relevant for math-related tasks. To further enhance HSDDW's effectiveness on math-focused datasets like Humaneval and GSM8K, more targeted training on math-specific domains is essential. This highlights a limitation of HSDDW in handling mathematical problems and suggests areas for potential improvement.

**Further Discussion on the Acceleration of HSDDW** In this section, we aim to provide a detailed explanation for the significant performance improvement of HSDDW (70b-7b-68m) compared to vanilla speculative decoding using the same 68m draft model. In both scenarios, the 68m model generates the draft during the drafting stage. However, HSDDW consistently demonstrates better efficiency. We introduce two key metrics to analyze this improvement: redundancy ($\sigma$) and target model utilization ($\eta$), offering a more comprehensive view of the system's performance.

| | **All** | **only Math** | **rm Math** |
|---|---|---|---|
| **AT** | 1.00× | 1.00× | 1.00× |
| **SD** | 2.40× | 2.29× | 2.41× |
| **HSDDW** | 2.95× | 2.08× | 3.04× |

Table 3: Comparison of Autoregressive (AT), Speculative Decoding (SD), and HSDDW speeds on MT-Bench. The term "All" refers to all data, "only Math" denotes data related only to mathematical content, and "rm Math" indicates the removal of math-related data.

The redundancy metric, $\sigma$, quantifies the proportion of tokens generated by the draft model that are not ultimately retained in the final output:

$$\sigma = 1 - \frac{\text{Number of accepted tokens}}{\text{Total tokens generated by the draft model}}.$$

A lower value of $\sigma$ indicates fewer unused tokens, which is desirable. However, reducing $\sigma$ alone could be misleading, as it could simply result from using a smaller window size for drafting. Therefore, we introduce the second metric, target model utilization ($\eta$), to balance this effect. The metric $\eta$ represents the fraction of the final output length that required inference steps from the target model:

$$\eta = \frac{\text{Number of target model inferences}}{\text{Final output length}}.$$

By evaluating both $\sigma$ and $\eta$, we can better understand why HSDDW surpasses speculative decoding, particularly in terms of computational efficiency and effectiveness.

In our comparison of the four models—70b-7b, 70b-68m, Pre-verify combined with Self-verify (referred to as Pre+Self-verify), and HSDDW—the results are summarized in Table 4. The table shows that models 70b-7b, Pre+Self-verify, and HSDDW exhibit similar values for redundancy ($\sigma$) and target model utilization ($\eta$). However, the observed performance differences can be attributed to variations in their underlying architectures.

In the 70b-7b model, the primary draft model during the drafting phase is the 7b model, whereas in Pre+Self-verify, both the 7b and 68m models are involved. In contrast, HSDDW primarily employs the 68m model for drafting, which contributes to its faster drafting speed. On the other hand, despite also using the 68m model for drafting, the 70b-68m model shows less efficiency, as reflected in its $\sigma$ and $\eta$ values. This highlights how HSDDW more

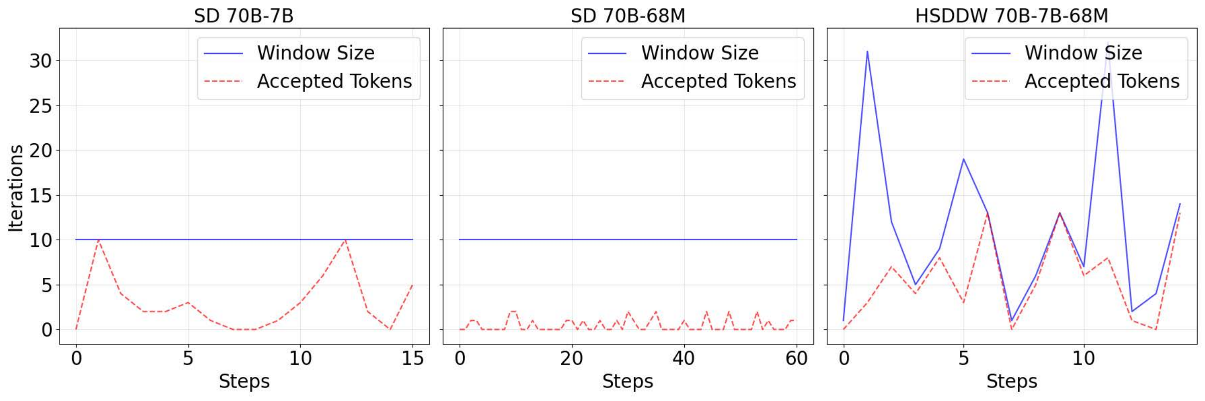


Figure 8: Visualization of window size across three configurations on the 28th question of the MT-Bench dataset.

effectively leverages the 68m model's efficiency, leading to a noticeable speedup in inference time.

| **Model** | $\sigma$ **(%)** | $\eta$ **(%)** | **Speedup** |
|---|---|---|---|
| **70b-7b** | 39.87 | 14.42 | 2.40× |
| **70b-68m** | 89.69 | 49.30 | 1.69× |
| **Pre+Self-verify** | 39.60 | 14.85 | 2.69× |
| **HSDDW** | 44.36 | 14.46 | 2.95× |

Table 4: Comparison of four models in terms of redundancy ($\sigma$) and target model utilization ($\eta$). The values in the table are percentages. "Pre+Self-verify" refers to Pre-verify combined with Self-verify.

**Case Study: Visualizing Dynamic Draft Lengths** We visualize the changes in window sizes of the draft model over time for the 70b-7b, 70b-68m, and 70b-7b-68m configurations. A random sample question from the MT-Bench dataset is used as an example.

The results are shown in Figure 8. We observe that HSDDW exhibits dynamic window sizes at different steps. Notably, the HSDDW 70b-7b-68m configuration demonstrates a significantly higher acceptance rate compared to Vanilla-SD with the 70b-68m setup. This improvement plays a crucial role in the speedup achieved by our method.

## 5 Conclusion

In this paper, we propose HSDDW, a novel inference acceleration framework that significantly enhances LLM inference efficiency without the need for additional training. The framework employs two simple yet effective strategies: Self-verify, which uses entropy to assess the draft model's confidence, and a hierarchical structure that incorporates pre-verification, boosting the overall robustness and performance of the model. A key advantage of HSDDW is its ability to mitigate the significant impact of window size on SD performance, making the method more robust to this parameter. Our experiments demonstrate that HSDDW is fully compatible with state-of-the-art methods and consistently delivers competitive results across various text generation benchmarks.

## 6 Limitations

Our model, HSDDW, leverages a hierarchical structure to accelerate the decoding process by relying on the draft model's inherent capabilities. However, if the task involves content that the draft model has not encountered before, the effectiveness of the acceleration may be reduced. This limitation arises because the draft model's familiarity with the task plays a crucial role in determining how much of the workload can be offloaded from the target model, directly impacting the overall speedup.

## References

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. *CoRR*, abs/2302.01318.

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

Wei-Lin Chiang, Zhuohan Li, Zi Lin, Ying Sheng, Zhanghao Wu, Hao Zhang, Lianmin Zheng, Siyuan Zhuang, Yonghao Zhuang, Joseph E. Gonzalez, Ion Stoica, and Eric P. Xing. 2023. Vicuna: An open-source chatbot impressing gpt-4 with 90%* chatgpt quality.

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

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

Zhenyu He, Zexuan Zhong, Tianle Cai, Jason Lee, and Di He. 2024. REST: Retrieval-based speculative decoding. In *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pages 1582–1595, Mexico City, Mexico. Association for Computational Linguistics.

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

Albert Q. Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de Las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, Lélio Renard Lavaud, Marie-Anne Lachaux, Pierre Stock, Teven Le Scao, Thibaut Lavril, Thomas Wang, Timothée Lacroix, and William El Sayed. 2023. Mistral 7b. *CoRR*, abs/2310.06825.

Yaniv Leviathan, Matan Kalman, and Yossi Matias. 2023. Fast inference from transformers via speculative decoding. In *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pages 19274–19286. PMLR.

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

Tianyu Liu, Yun Li, Qitan Lv, Kai Liu, Jianchen Zhu, and Winston Hu. 2024. Parallel speculative decoding with adaptive draft length. *Preprint*, arXiv:2408.11850.

Xupeng Miao, Gabriele Oliaro, Zhihao Zhang, Xinhao Cheng, Zeyu Wang, Zhengxin Zhang, Rae Ying Yee Wong, Alan Zhu, Lijie Yang, Xiaoxiang Shi, Chunan Shi, Zhuoming Chen, Daiyaan Arfeen, Reyna Abhyankar, and Zhihao Jia. 2024. Specinfer: Accelerating large language model serving with tree-based speculative inference and verification. In *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 3*, ASPLOS '24, page 932–949, New York, NY, USA. Association for Computing Machinery.

OpenAI. 2023. GPT-4 technical report. *CoRR*, abs/2303.08774.

David A. Patterson. 2004. Latency lags bandwith. *Commun. ACM*, 47(10):71–75.

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. *J. Mach. Learn. Res.*, 21(1).

Noam Shazeer. 2019. Fast transformer decoding: One write-head is all you need. *CoRR*, abs/1911.02150.

Benjamin Spector and Chris Re. 2023. Accelerating llm inference with staged speculative decoding. In *ES-FOMO at International Conference on Machine Learning*.

Mitchell Stern, Noam Shazeer, and Jakob Uszkoreit. 2018. Blockwise parallel decoding for deep autoregressive models. In *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc.

Hanshi Sun, Zhuoming Chen, Xinyu Yang, Yuandong Tian, and Beidi Chen. 2024. Triforce: Lossless acceleration of long sequence generation with hierarchical speculative decoding. In *First Conference on Language Modeling*.

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`.

Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, Dan Bikel, Lukas Blecher, Cristian Canton-Ferrer, Moya Chen, Guillem Cucurull, David Esiobu, Jude Fernandes, Jeremy Fu, Wenyin Fu, Brian Fuller, Cynthia Gao, Vedanuj Goswami, Naman Goyal, Anthony Hartshorn, Saghar Hosseini, Rui Hou, Hakan Inan, Marcin Kardas, Viktor Kerkez, Madian Khabsa, Isabel Kloumann, Artem Korenev, Punit Singh Koura, Marie-Anne Lachaux, Thibaut Lavril, Jenya Lee, Diana Liskovich, Yinghai Lu, Yuning Mao, Xavier Martinet, Todor Mihaylov, Pushkar Mishra, Igor Molybog, Yixin Nie, Andrew Poulton, Jeremy Reizenstein, Rashi Rungta, Kalyan Saladi, Alan Schelten, Ruan Silva, Eric Michael Smith, Ranjan Subramanian, Xiaoqing Ellen Tan, Binh Tang, Ross Taylor, Adina Williams, Jian Xiang Kuan, Puxin Xu, Zheng Yan, Iliyan Zarov, Yuchen Zhang, Angela Fan, Melanie Kambadur, Sharan Narang, Aurélien Rodriguez, Robert Stojnic, Sergey Edunov, and Thomas Scialom. 2023a. Llama 2: Open foundation and fine-tuned chat models. *CoRR*, abs/2307.09288.

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

Heming Xia, Tao Ge, Peiyi Wang, Si-Qing Chen, Furu Wei, and Zhifang Sui. 2023. Speculative decoding: Exploiting speculative execution for accelerating seq2seq generation. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, pages 3909–3925, Singapore. Association for Computational Linguistics.

Heming Xia, Zhe Yang, Qingxiu Dong, Peiyi Wang, Yongqi Li, Tao Ge, Tianyu Liu, Wenjie Li, and Zhifang Sui. 2024. Unlocking efficiency in large language model inference: A comprehensive survey of speculative decoding. In *Findings of the Association for Computational Linguistics ACL 2024*, pages 7655–7671, Bangkok, Thailand and virtual meeting. Association for Computational Linguistics.

Jun Zhang, Jue Wang, Huan Li, Lidan Shou, Ke Chen, Gang Chen, and Sharad Mehrotra. 2023. Draft & verify: Lossless large language model acceleration via self-speculative decoding. *CoRR*, abs/2309.08168.

Weilin Zhao, Yuxiang Huang, Xu Han, Wang Xu, Chaojun Xiao, Xinrong Zhang, Yewei Fang, Kaihuo Zhang, Zhiyuan Liu, and Maosong Sun. 2024. Ouroboros: Generating longer drafts phrase by phrase for faster speculative decoding. *Preprint*, arXiv:2402.13720.

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric Xing, et al. 2024. Judging llm-as-a-judge with mt-bench and chatbot arena. *Advances in Neural Information Processing Systems*, 36.

Yongchao Zhou, Kaifeng Lyu, Ankit Singh Rawat, Aditya Krishna Menon, Afshin Rostamizadeh, Sanjiv Kumar, Jean-François Kagy, and Rishabh Agarwal. 2023. Distillspec: Improving speculative decoding via knowledge distillation. *ArXiv*, abs/2310.08461.

## A Evaluation Details

### A.1 Inference Setting

We set the maximum sequence length to 2048 and applied a temperature of $T = 1$. Consistent with (Leviathan et al., 2023), the fixed window size was set to $K = 10$.

### A.2 GPU Compute Resources

The majority of the experiments in this study were conducted on 8 NVIDIA V100-32G GPUs. An exception is in Section 4.4, where we evaluated the 7B-68M pair specifically for entropy analysis. This experiment was performed on a single GPU because entropy differences are independent of the number of GPUs. To ensure transparency, a description of this specific setup will be included in the paper.

For our GPU setup, we utilized PyTorch in combination with the HuggingFace Transformers library [7]. Specifically, the `AutoModelForCausalLM.from_pretrained` function was employed with the `device_map="auto"` parameter, enabling automatic model distribution across available devices. FP16 precision was utilized to optimize memory usage.

The allocation of GPU resources was as follows:

- **70B Model**: Distributed across all 8 GPUs (device0 to device7).
- **7B and 68M Models**: Both models were allocated to a single GPU (device0).

## B License

**Dataset** The licenses for the datasets used in our research are as follows:

1. **Alpaca**[8] (Taori et al., 2023): Creative Commons Attribution Non Commercial 4.0
2. **HumanEval**[9] (Chen et al., 2021): MIT License
3. **GSM8K**[10] (Cobbe et al., 2021): MIT License
4. **MT-bench dataset**[11] (Zheng et al., 2024): Apache License 2.0

**Model** The licenses for the models used in our research are as follows:

1. **Llama-2-chat models** (Touvron et al., 2023b): Meta AI Llama2 Community License Agreement
2. **Llama-68m**[12]: Apache License 2.0

[7] `https://huggingface.co/docs/transformers/model_doc/auto`

[8] `https://huggingface.co/datasets/tatsu-lab/alpaca`

[9] `https://huggingface.co/datasets/openai/openai_humaneval`

[10] `https://huggingface.co/datasets/openai/gsm8k`

[11] `https://huggingface.co/spaces/lmsys/mt-bench/tree/main`

[12] `https://huggingface.co/JackFram/llama-68m`

**Code** The license for the code used in our research is as follows:

1. **SD**[13]: Apache License 2.0

[13] https://github.com/feifeibear/LLMSpeculativeSampling

## C Algorithm: SD + Self-verify

Here, we give the whole algorithm of SD + Self-verify in detail.

**Algorithm 1** SD+Self-Verify.

```
Input: the draft models $\mathcal{M}_q$ , the target model $\mathcal{M}_p$, the input prefix $\mathbf{x}$, the max generate tokens $L$, the window size $K$.
1: Initialization: $M_p = \mathcal{M}_q, M_q = \mathcal{M}_p, H_q = [\,], \tau_q = 0$
2: while $len(\mathbf{x}) < L$ do
3:   ▷ Draft
4:   for $i = 1$ to $K$ do
5:     $q_i \leftarrow M_q(\mathbf{x} + [x_1, ..., x_{i-1}])$
6:     $x_i \sim q_i$
7:     $l = i$
8:     ▷ Self-verify strategy
9:     if $\mathcal{H}(q_i) > \tau_q$ then
10:      break
11:    end if
12:  end for
13:  ▷ Verify
14:  $\mathbf{x}, [x_1, x_2, ..., x_l] \leftarrow \mathbf{x}$     ▷ split the prefix to get the last $n_{tot}$ draft tokens
15:  $p_1, p_2, ..., p_l \leftarrow M_p(\mathbf{x} + [x_1]), M_p(\mathbf{x} + [x_1, x_2]), ..., M_p(\mathbf{x} + [x_1, ..., x_l])$
16:  retrival $q_1, q_2, ..., q_l$ from the cache
17:  $r_1 \sim U(0, 1), ..., r_l \sim U(0, 1)$
18:  $n \leftarrow \min(\{i - 1 | 1 \leq i \leq l, r_i > \frac{p_i[x_i]}{q_i[x_i]}\} \cup \{l\})$
19:  if $n = l$ then
20:    ✓ accept all draft tokens
21:    $\mathbf{x} \leftarrow \mathbf{x} + [x_1, ..., x_l]$
22:  else
23:    × reject someone
24:    $y \sim norm(max(0, p_{n+1} - q_{n+1}))$
25:    $\mathbf{x} \leftarrow \mathbf{x} + [x_1, ..., x_n, y]$
26:    $\tau_q = Average(H_q.append(\mathcal{H}(q_n))$     ▷ update the draft model entropy threshold
27:  end if
28:  mode ← "Draft"
29: end while
```

## D Algorithm : HSDDW

Here, we give the whole algorithm of HSDDW in details.

**Algorithm 2** Hierarchical Speculative Decoding with Dynamic Window.

```
Input: the draft models $\mathcal{M}_{q1}$ and $\mathcal{M}_{q2}$, the target model $\mathcal{M}_p$, the input prefix $\mathbf{x}$, the max generate tokens $L$, the window size $K$.
    ▷ The *Draft* strategy is used first.
Initialization: mode ← "Draft", $M_p = \mathcal{M}_{q2}, M_q = \mathcal{M}_{q1}, H_{q1} = H_{q2} = [\,], \tau_{q1} = \tau_{q2} = 0, n_{tot} = 0$
while $len(\mathbf{x}) < L$ do
    if mode = "Draft" then
        for $i = 1$ to $K$ do
            $q_i \leftarrow M_q(\mathbf{x} + [x_1, ..., x_{i-1}])$
            $x_i \sim q_i$
            $l = i$
            ▷ Self-verify strategy
            if $\mathcal{H}(q_i) > \tau_{q1}$ then
                break
            end if
        end for
        $p_1, p_2, ..., p_l \leftarrow M_p(\mathbf{x} + [x_1]), M_p(\mathbf{x} + [x_1, x_2]), ..., M_p(\mathbf{x} + [x_1, ..., x_l])$
        retrival $q_1, q_2, ..., q_l$ from the cache
        $r_1 \sim U(0,1), ..., r_l \sim U(0,1)$
        ▷ Pre-verify strategy
        $n \leftarrow \min(\{i - 1 | 1 \le i \le l, r_i > \frac{p_i[x_i]}{q_i[x_i]}\} \cup \{l\})$
        if $n = l$ then
            ✓ accept all draft tokens
            $\mathbf{x} \leftarrow \mathbf{x} + [x_1, ..., x_l]$
        else
            × reject someone
            $y \sim norm(max(0, p_{n+1} - q_{n+1}))$
            $\mathbf{x} \leftarrow \mathbf{x} + [x_1, ..., x_n, y]$
            $\tau_{q1} = Average(H_{q1}.append(\mathcal{H}(q_n))$        ▷ update the q1 threshold
        end if
        ▷ Self-verify strategy
        $j \leftarrow \min(\{i | 1 \le i \le l, \mathcal{H}(p_i) > \tau_{q2}\} \cup \{0\})$
        $n_{tot} = n + n_{tot}$
        if $j = 0$ then
            ✓ accept all draft tokens
            mode ← "Draft"
            $M_p = \mathcal{M}_{q2}, M_q = \mathcal{M}_{q1}$
        else
            × reject someone
            mode ← "verify"
            $M_p = \mathcal{M}_p, M_q = \mathcal{M}_{q2}$
        end if
    else
        ▷ Verify strategy
        $\mathbf{x}, [x_1, x_2, ..., x_{n_{tot}}] \leftarrow \mathbf{x}$        ▷ split the prefix to get the last $n_{tot}$ draft tokens
        $p_1, p_2, ..., p_{n_{tot}} \leftarrow M_p(\mathbf{x} + [x_1]), M_p(\mathbf{x} + [x_1, x_2]), ..., M_p(\mathbf{x} + [x_1, ..., x_{n_{tot}}])$
        retrival $q_1, q_2, ..., q_{n_{tot}}$ from the cache
        $r_1 \sim U(0,1), ..., r_{n_{tot}} \sim U(0,1)$
        $n \leftarrow \min(\{i - 1 | 1 \le i \le n_{tot}, r_i > \frac{p_i[x_i]}{q_i[x_i]}\} \cup \{n_{tot}\})$
        if $n = n_{tot}$ then
            ✓ accept all draft tokens
            $\mathbf{x} \leftarrow \mathbf{x} + [x_1, ..., x_{n_{tot}}]$
        else
            × reject someone
            $y \sim norm(max(0, p_{n+1} - q_{n+1}))$
            $\mathbf{x} \leftarrow \mathbf{x} + [x_1, ..., x_n, y]$
            $\tau_{q2} = Average(H_{q2}.append(\mathcal{H}(q_n))$        ▷ update the q2 threshold
        end if
        mode ← "Draft"
    end if
end while
```