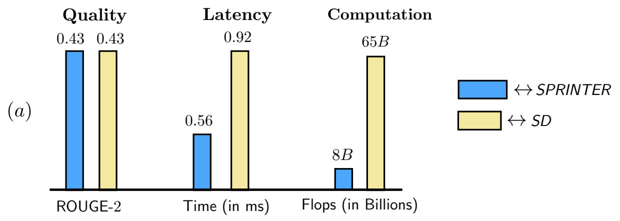
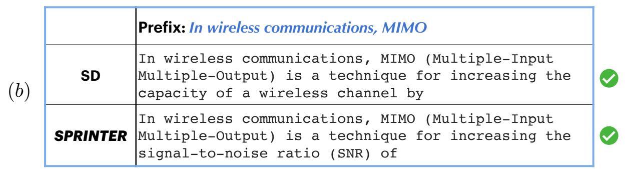
Speeding up Speculative Decoding via Sequential Approximate Verification

Meiyu Zhong [*1] **Noel Teku** [*1] **Ravi Tandon** [1]

Abstract

Speculative Decoding (SD) is a recently proposed technique for faster inference using Large Language Models (LLMs). SD operates by using a smaller draft LLM for autoregressively generating a sequence of tokens and a larger target LLM for parallel verification to ensure statistical consistency. However, periodic parallel calls to the target LLM for verification prevent SD from achieving even lower latencies. We propose *SPRINTER*, which utilizes a low-complexity verifier trained to predict if tokens generated from a draft LLM would be accepted by the target LLM. By performing *sequential approximate verification*, *SPRINTER* does not require verification by the target LLM and is only invoked when a token is deemed unacceptable. This reduces the number of calls to the larger LLM, achieving further speedups and lower computation cost. We present a theoretical analysis of *SPRINTER*, examining the statistical properties of the generated tokens, as well as the expected reduction in latency as a function of the verifier. We evaluate *SPRINTER* on several datasets and model pairs, demonstrating that approximate verification can still maintain high quality generation while further reducing latency.





(b)

	Prefix: *In wireless communications, MIMO*	
SD	In wireless communications, MIMO (Multiple-Input Multiple-Output) is a technique for increasing the capacity of a wireless channel by	✓
SPRINTER	In wireless communications, MIMO (Multiple-Input Multiple-Output) is a technique for increasing the signal-to-noise ratio (SNR) of	✓

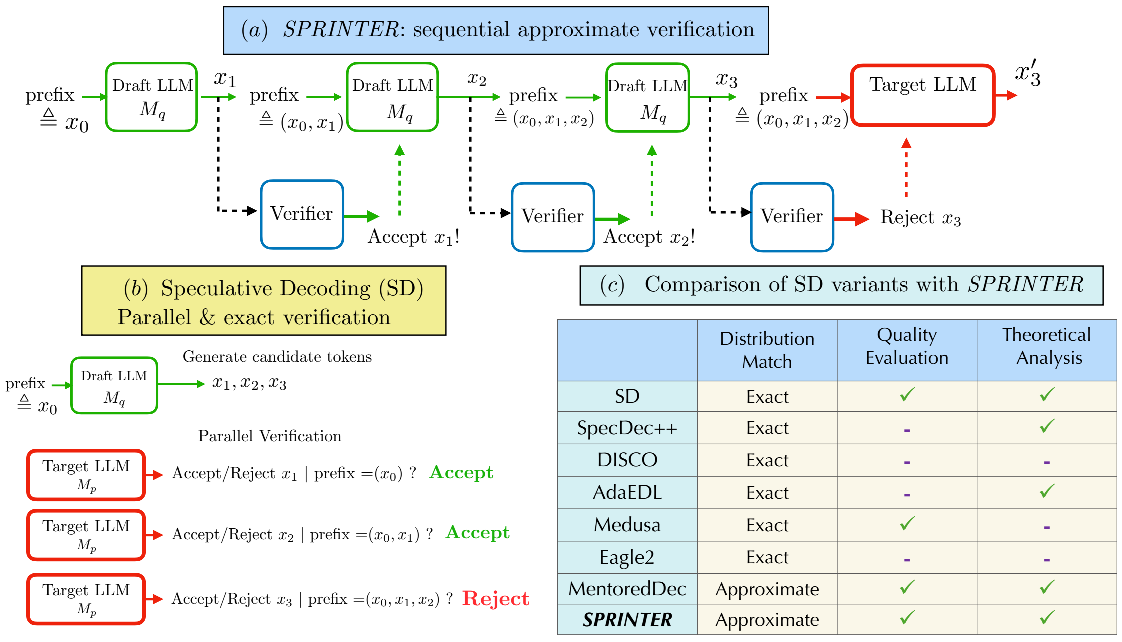
Figure 1. (a) Comparison between *SPRINTER* and SD with respect to *Quality* (ROUGE score), *Latency* (time in ms required to generate a token) and *Computation* (number of flops required to generate 20 consecutive acceptable tokens from the draft model). *SPRINTER* can attain comparable quality, $\mathbf{1.64X}$ speedups, and $\mathbf{8X}$ smaller computation costs compared to SD. (b) Example responses generated via *SPRINTER* vs SD given the prefix: "In wireless communications, MIMO". The response from *SPRINTER* is comparable to SD (for more examples, see Section A.7).

1. Introduction

Large Langauge Models (LLMs) have shown to be very effective in different applications including text generation (Zingale & Kalita, 2024), image analysis (Niu et al., 2024), and video understanding (Tang et al., 2024). However, despite this success, LLM-based solutions for various problem domains are still constrained by *high computational costs incurred during inference* due to large model sizes. To enable faster inference, Speculative Decoding (SD) (Leviathan et al., 2023) has been proposed as a solution for reducing the latencies incurred during the inference of significantly larger LLMs. Under this paradigm, a smaller draft LLM is used to autoregressively generate a certain number of tokens. These tokens are then passed to a larger target LLM, which processes the tokens in parallel to determine how many of them are acceptable (i.e. if the distribution of the generated tokens matches the distribution of what the target model would have generated). If they are not acceptable, then the target model is called to generate replacement tokens. By reducing the amount of times the target model is invoked for autoregressive generation, less latency is incurred. Thus, the objective of SD is to incur smaller inference times while guaranteeing that sampling tokens from the draft model is equivalent to sampling from the target model. (Mamou et al., 2024) proposed using a two layer feedforward network to stop the draft model from generating tokens and initiate the target model's verification process, once the output of the network is less than a threshold. (Huang et al., 2024) models the SD procedure as a Markov Decision Process and uses the draft model with a smaller, multi-layer network to predict

[*]Equal contribution [1]Department of Electrical and Computer Engineering, University of Arizona, Tucson, US. Correspondence to: Meiyu Zhong <meiyuzhong@arizona.edu>, Noel Teku <nteku1@arizona.edu>, Ravi Tandon <tandonr@arizona.edu>.

Proceedings of the 3rd Efficient Systems for Foundation Models Workshop at the 41st International Conference on Machine Learning (ICML), Vancouver, Canada. PMLR 267, 2025. Copyright 2025 by the author(s).



(a) *SPRINTER*: sequential approximate verification
prefix $\triangleq x_0$
Draft LLM M_q
x_1
prefix $\triangleq (x_0, x_1)$
Draft LLM M_q
x_2
prefix $\triangleq (x_0, x_1, x_2)$
Draft LLM M_q
x_3
prefix $\triangleq (x_0, x_1, x_2)$
Target LLM
x'_3
Verifier
Accept x_1!
Verifier
Accept x_2!
Verifier
Reject x_3
(b) Speculative Decoding (SD)
Parallel & exact verification
prefix $\triangleq x_0$
Draft LLM M_q
Generate candidate tokens
x_1, x_2, x_3
Parallel Verification
Target LLM M_p
Accept/Reject x_1 | prefix $=(x_0)$? Accept
Target LLM M_p
Accept/Reject x_2 | prefix $=(x_0, x_1)$? Accept
Target LLM M_p
Accept/Reject x_3 | prefix $=(x_0, x_1, x_2)$? Reject
(c) Comparison of SD variants with *SPRINTER*

	Distribution Match	Quality Evaluation	Theoretical Analysis
SD	Exact	✓	✓
SpecDec++	Exact	-	✓
DISCO	Exact	-	-
AdaEDL	Exact	-	✓
Medusa	Exact	✓	-
Eagle2	Exact	-	-
MentoredDec	Approximate	✓	✓
SPRINTER	Approximate	✓	✓

Figure 2. (a) *SPRINTER* works by generating tokens from a smaller (draft) LLM, which are sequentially accepted/rejected by a *verifier*, a low-complexity small classifier. In *SPRINTER*, the larger (target) LLM is only called if a token is rejected and used only to replace the rejected token. (b) Speculative decoding (SD) works by generating multiple tokens by the draft model, all of them are verified in parallel by the target LLM. (c) Comparison of different speculative decoding based mechanisms with respect to three aspects: i) Approximate or Exact match with the larger LLM (target) probability distribution, ii) Quality Evaluation of Completions and iii) Theoretical Analysis. *SPRINTER* is the first framework to provide in-depth analysis of sequential approximate verification; its impact on the quality-vs-latency tradeoff and provide insights on how to navigate this tradeoff. Additional discussion on related works is presented in Section A.6.

the probability that the current token generated by the draft model should be accepted. The probability that there is at least one draft token that should be rejected is subsequently derived and compared with a threshold to determine if the target model should be invoked for verification.

Overview of SPRINTER: Running the target model for parallel verification, even periodically (i.e. after a certain number of tokens) can still result in significant latencies. Higher speedups can be attained if the constraint that the tokens generated by the draft model must match the distribution of the target model is relaxed. However, we do not want to deviate too far from the target distribution as this would increase the likelihood of the draft model generating inaccurate tokens. To balance this tradeoff, we propose *SPRINTER*, a sampling technique that uses a low-complexity verifier that predicts when it is necessary to invoke the target model to generate a token that replaces the current draft token. Subsequently, under *SPRINTER*, verification is performed *sequentially* as draft tokens are generated, in contrast to SD based approaches which perform parallel verification of multiple draft tokens through the larger LLM, which requires more computational resources to execute.

In many computational settings, including machine learning and optimization, the cost of generating a valid solution can be significantly higher than that of verifying one. This asymmetry, often referred to as the *generation-verification gap*, also provides inspiration for *SPRINTER*: verifying the acceptability of a token could be easier than generating a high quality token from a larger LLM. The verifier used in this work has a significantly lower complexity compared to the draft and target LLMs; subsequently, the additional overhead introduced by training and using it is negligible (as further evidenced in the results). b) Our approach is also aligned with observations made in recent works such as (Bachmann et al., 2025; Melcer et al., 2024), which have shown that *high-quality* tokens can still be generated without necessarily matching the distribution of the larger target model. Figure 1(a) shows that *SPRINTER* achieves lower latency and computational cost while maintaining quality comparable to that of SD. Figure 1(b) presents example responses generated by *SPRINTER* and SD, illustrating that *SPRINTER* produces responses of similar quality as SD. We next summarize the main contributions of this paper.

- ***SPRINTER* framework**. We propose *SPRINTER*, a framework for achieving faster inference from LLMs using a pair of (draft (small), target (large)) LLMs together with the aid of a verifier. The role of the verifier is to perform approximate verification, i.e., if tokens generated by a draft model would be acceptable by the larger target LLM. The key motivating factors and the detailed framework is described in Section 3.

- **Theoretical Analysis**. We present a comprehensive theoretical analysis in Section 3.1 to show the trade-offs between quality of generated tokens versus latency speedups and computational savings offered by *SPRINTER*. Specifically, we demonstrate how the ROC curve characteristics (e.g., false-positive and true-positive rates) of the verifier can be used to balance the tradeoff between latency and quality. Furthermore, we discuss strategies to train the verifier and show how the theoretical results also provide useful design insights.

- **Experiments and Validation**. We present a comprehensive evaluation of *SPRINTER* on several datasets and model pairs in Section 4, demonstrating its ability to reduce latency while requiring significantly less computation and maintaining high quality. Specifically, Win-tie rate and ROUGE metrics are used to evaluate the quality of responses generated by *SPRINTER* against those generated with SD, indicating that only minimal quality degradation is experienced. *SPRINTER* is also shown to outperform target distribution-preserving SD variants (e.g. Eagle2 (Li et al.), Medusa (Cai et al., 2024)) in speed and quality. Furthermore, higher performance improvements are attained using *SPRINTER* compared to Mentored Decoding (Tran-Thien, 2024), which similarly relaxes the requirement that generated tokens match the target distribution.

2. Preliminaries on Speculative Decoding (SD)

SD was originally proposed in (Leviathan et al., 2023) as a novel algorithm for speeding up LLM inference from a larger target LLM M_p through the help of a smaller (faster) draft LLM M_q. $p(x)$ and $q(x)$ represent the probability distributions we get from M_p and M_q respectively for the next token given a specific prefix (i.e. set of already generated tokens $x_{<t}$); specifically, we let $p(x)$ and $q(x)$ denote $p(x|\text{prefix})$ and $q(x|\text{prefix})$ respectively. First, given a prefix, the draft model M_q autoregressively generates γ tokens, where γ is a hyperparameter chosen by the user. These sequence of tokens are then passed to the target model which performs verification of all γ candidate completion sequences in parallel. For the last token x in each sequence, the target model M_p is invoked and it verifies the relationship between $q(x)$ and $p(x)$ (see Figure 2 for an example). If $q(x) < p(x)$, meaning that the probability of the draft token is within the distribution of the target token, it is acceptable. However, if a draft token in any one of the candidate sequences is not accepted, it can still be accepted with probability $\frac{p(x)}{q(x)}$; otherwise, the target model resamples

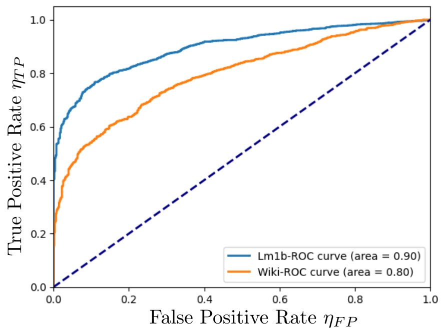


Figure 3. ROC Curve Performance of a trained Verifier (less than $1k$ parameters) on the Wiki-Summary and LM1B Datasets with GPT-Neo-125M as the draft model M_q and GPT-Neo-1.3B as the target model M_p. Despite being orders of magnitude smaller in size compared to the draft and target models, the verifier was able to achieve AU-ROC of 0.8 (respectively 0.9) on the two datasets.

from an alternative distribution given as follows (Leviathan et al., 2023):

$$p'(x) = \text{norm}(\max(0, p(x) - q(x))). \tag{1}$$

SD ensures that the entire sampling process matches the distribution of the target LLM. Speculative decoding has turned out to be powerful in achieving speedups and has led to a large number of recent papers that have proposed variations of SD, including works shown in Figure 2(c). (Mamou et al., 2024) and (Huang et al., 2024), for example, use a low-complexity classifier to determine when the draft model should stop generating tokens while ensuring that the sampled tokens match the distribution of the target model. (Kim et al., 2024) and (Agrawal et al., 2024) propose heuristics that incur less complexity compared to using a verifier; however, they are also ensure that the sampled tokens match the distribution of the target model. We provide a more detailed discussion on these related works in Section A.6.

3. SPRINTER Framework & Analysis

Before presenting our framework, we first discuss some of the key motivating factors behind *SPRINTER*.

Cost of Parallelism. As SD and several variants discussed in related work attempt to match the target distribution, they end up invoking the target model which performs parallel verification. While parallelism ensures that the latency (time) for verifying γ tokens is equivalent to running the target model once, one still has to pay the cost of parallelism as the target model runs γ times. As γ increases, the latency reduces but the cost of parallelism grows proportionally. This is the first idea that motivates us to study sequential verification; instead of verifying by the target LLM in parallel, we instead propose *approximate verification* by a significantly smaller model (named the verifier) in a sequential manner as shown in Fig. 2(a). Depending on the quality of the verifier (i.e., false-positive and true-positive rates), we only call the target LLM if a token is rejected by verifier. Thus, *SPRINTER* can achieve the dual benefit of reducing the number of calls to the target LLM and completely eliminates the cost of running it in parallel.

As an illustration, Fig. 3 shows the ROC curve of a low-complexity verifier (with a single layer and less than $1k$ training parameters) which was trained to accept/reject tokens generated by GPT-Neo-125M (draft model M_q) and GPT-Neo-1.3B as the target model M_p. The fact that we were able to achieve AU-ROC (area under ROC curve) of 0.8 and 0.9 was achievable on LM1B and Wiki-summaries datasets first highlights the feasibility of low-complexity verification (more results are presented in Section 4).

Quality of Smaller Models. Inevitably, if we resort to approximate verification, we have to give up statistical consistency, i.e., one cannot guarantee a match with the target LLM distribution. However, statistical consistency alone may not be a necessary indicator for high quality generation. For instance, recent works such as Mentored Decoding (Tran-Thien, 2024) and Judge Decoding (Bachmann et al., 2025) have shown that even smaller LLMs have generation capabilities that can be comparable with larger ones. Our idea behind *SPRINTER* is to use a smaller trained model (verifier) which is pipelined with the smaller model for sequential verification. It is this interplay between latency, total computational costs and quality that motivate *SPRINTER*. We next describe the framework in detail followed by the theoretical analysis of *SPRINTER*.

Algorithm. We now provide an overview of the *SPRINTER* sampling process. First, a prefix is fed to the draft model and a token is sampled with probability $q(x)$. The draft token is then passed to a verifier V to predict whether or not the ratio between $q(x)$ and $p(x)$ is greater or less than 1. The verifier can take as input various latent features derived from the draft model (e.g. embedding of the draft token x, probability distribution of the LLM's vocabulary). As there is flexibility with which features from the draft model can be extracted, we denote the input to the verifier as $s(x, \text{prefix})$. An ideal verifier would make the following decision:

$$V(s(x, \text{prefix})) = \begin{cases} 1 & \frac{q(x)}{p(x)} \leq 1 \\ 0 & \frac{q(x)}{p(x)} > 1. \end{cases}$$

If $V(s(x, \text{prefix})) = 1$, the verifier predicts that the ratio is less than 1, suggesting that the current token x is acceptable and that M_q should generate the next token. If $V(s(x, \text{prefix})) = 0$, the verifier predicts that the ratio is larger than 1, leading to the rejection of the current token and indicating that M_p should be called. In this scenario, similar to SD, the draft token can be accepted with probability $\frac{p(x)}{q(x)}$ or rejected with probability $1 - \frac{p(x)}{q(x)}$ and replaced with a token sampled from the revised distribution (1). In Section 3.2, we provide details on how to train a verifier. We illustrate the sampling process of *SPRINTER* in Algorithm 1 (full algorithm is presented in Section A.5) and show the *SPRINTER* sampling process in Fig 2(a).

Algorithm 1 *SPRINTER*

```
Input: $M_p, M_q, V$, Prefix, Prediction Threshold $\tau$
Initiate the values
while True do
    Update the Prefix
    Generate the current token $x$ from $M_q$
    Obtain the Verifier's prediction of the current token $V(s(x, \text{Prefix}))$ .
    if $V(s(x, \text{Prefix})) \leq \tau$ then
        Break
    end if
end while
Invoke $M_p$ to verify the last token and re-sample if necessary.
```

3.1. Theoretical Analysis of SPRINTER

In this Section, we present our theoretical results on *SPRINTER*. Through these results, we aim to study the impact of verifier's performance on a) the probability distribution of tokens sampled by *SPRINTER*; b) expected number of consecutive tokens sampled by *SPRINTER* before invoking the target model; c) average latency incurred in the process as well as the amount of computational savings. All theoretical results in our paper are derived under the assumption that the verifier operates in an i.i.d. manner. As we discuss later in this Section, these results also provide practical design insights for navigating the quality-vs-latency tradeoffs.

(a) Statistical Analysis of generated tokens. Let us denote η_{FP} and η_{TP} as the false-positive and true-positive rates of the verifier, respectively. Specifically, a false positive refers to the setting if an unacceptable token (i.e., $q(x)/p(x) > 1$) is deemed acceptable by the verifier. Conversely, a true positive refers to the scenario if an acceptable token (i.e., $q(x)/p(x) \leq 1$) is accepted by the verifier. In our first result, we characterize the distribution of the tokens generated by *SPRINTER*. Proof of Theorem 3.1 can be found in the Appendix A.1.

Theorem 3.1. *The probability of a token x being chosen when running SPRINTER is given as*

$$p_{SPRINTER}(x) = (1 - \eta_{FP})p(x) + \eta_{FP}q(x), \tag{2}$$

where η_{FP} is the false positive rate of the verifier. Furthermore, the total-variation distance between the target and SPRINTER distributions is $d_{TV}(p, p_{SPRINTER}) = \eta_{FP} d_{TV}(p, q)$.

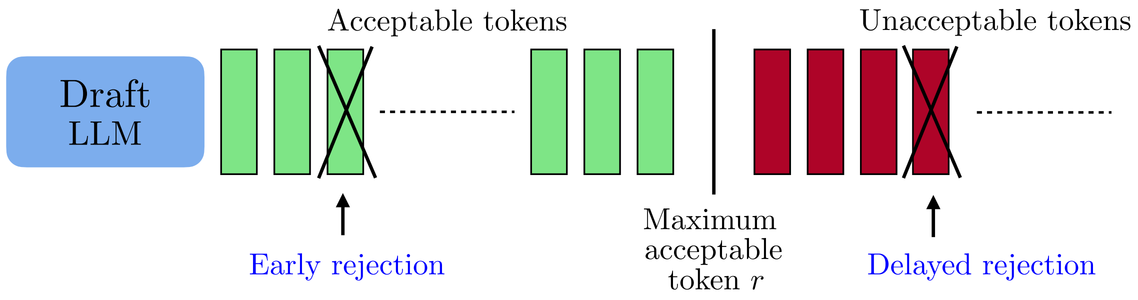


Figure 4. A draft LLM generates r tokens that are acceptable and subsequent tokens that are unacceptable. If the verifier rejects one of the first r consecutive tokens, the speedup attained from continuing to accept tokens until the r^{th} token is lost (i.e. early rejection). If the verifier continues to accept tokens after the r^{th} token, it experiences smaller latencies but at the cost of accepting low-quality tokens (i.e. delayed rejection). Theorem 3.3 characterizes the expected number of generated tokens as a function of η_{TP} and η_{FP}.

Remark 3.2. The false positive rate η_{FP} directly influences the distribution of *SPRINTER* and represents how much the distribution of *SPRINTER* deviates from the distribution of the target model. This observation aligns with the intuition that a perfect verifier with $\eta_{\text{FP}} = 0$ (i.e. *SPRINTER* never samples a token that should be rejected), for example, this results in $p_{\text{SPRINTER}} = p(x)$, effectively matching the distribution of the target model.

(b) Expected number of generated tokens. We now analyze the expected number of tokens accepted by the verifier when using *SPRINTER* as a function of η_{FP} and η_{TP}. For this analysis, we consider the scenario illustrated in Figure 4. Suppose that the ground truth is that given a prefix, the draft model M_q is capable of producing r acceptable tokens sequentially (in other words, the first r generated tokens by M_q are acceptable, whereas subsequent ones are unacceptable). Under this ground truth, let us define the random variable N_{SPRINTER} as the number of consecutive tokens accepted by the verifier. Assuming that the verifier makes decisions in an i.i.d. manner, it can exhibit two types of behavior also shown in Fig. 4:

- **Early Rejection**: This occurs if the verifier accepts the first $(i-1)$ tokens but mistakenly predicts that the i^{th} token (for $i \leq r$) should be rejected, then the verifier will revert to calling the target model rather than continuing to enable the draft model to generate the remaining $r - i$ acceptable tokens. In doing so, the verifier misses out on experiencing an even greater latency reduction while still generating high-quality tokens. This indicates that η_{TP} directly influences the early rejection caused by *SPRINTER*.
- **Delayed Rejection**: On the other hand, it can happen that the verifier continues to accept more than r tokens. Specifically, if the verifier first stops at the i^{th} token (for $i > r$), then it does not invoke the target model until token i, resulting in a higher computational savings and latency speedups but at the cost of accepting $(i-r)$ lower quality tokens. This indicates that η_{FP} directly influences the deviation of the statistical distribution from the target model.

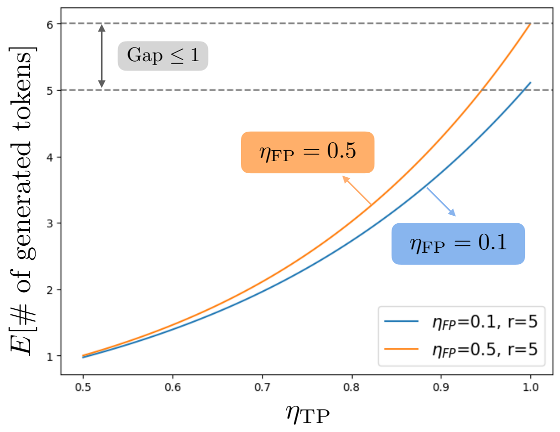


Figure 5. Illustration of expected number of tokens generated by *SPRINTER* as a function of true-positive rate (η_{TP}) for two different values of η_{FP} when the number of consecutively acceptable tokens is $r = 5$. We can observe that as long as $\eta_{FP} \leq 0.5$, the average number of unacceptable tokens (shown as the *"Gap"*) generated by *SPRINTER* never exceeds 1.

Our next Theorem characterizes the properties of N_{SPRINTER} assuming that r consecutive draft tokens are acceptable.

Theorem 3.3. *The probability distribution of the number of generated tokens is given as:*

$$\mathbb{P}(N_{SPRINTER} = i) = \begin{cases} \eta_{TP}^{i}(1-\eta_{TP}) & i < r, \\ \eta_{TP}^{r}(\eta_{FP})^{i-r}(1-\eta_{FP}) & i \geq r. \end{cases}$$

The expected number of generated tokens is given as:

$$\mathbb{E}(N_{SPRINTER}) = \frac{\eta_{TP} - \eta_{TP}^{r}}{1-\eta_{TP}} + \frac{\eta_{TP}^{r}}{1-\eta_{FP}}. \tag{3}$$

The proof of Theorem 3.3 can be found in the Appendix A.2. To gain more insights from this result, Fig. 5 illustrates the

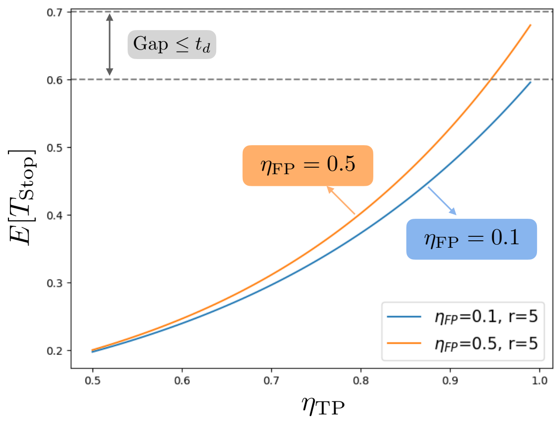


Figure 6. Illustration of expected stopping time of *SPRINTER* as a function of true-positive rate (η_{TP}) for two different values of η_{FP} when $r = 5$ and $t_d = 0.1$. We can again observe that as long as $\eta_{FP} \leq 0.5$, the average stopping time (shown as the *"Gap"*) generated by *SPRINTER* never exceeds t_d.

trade-off between η_{TP} and $\mathbb{E}(N_{\text{SPRINTER}})$ for two different values of η_{FP} assuming that $r = 5$. The figure indicates that as η_{TP} approaches 1, even with a verifier that has a substantial η_{FP} (i.e. 0.5), the additional number of tokens that are accepted past r is marginal. This implies that if an estimate of r could be attained from the training data, then fixing η_{FP} and varying η_{TP}, for example, could bring insights into how well the verifier must perform to generate close to the ideal r tokens. Essentially, η_{TP} and η_{FP} can be varied to determine the optimal false positive and negative rates such that the chances of early and delayed rejection occurring potentially caused by the verifier are minimized.

(c) Latency Analysis and Computational Cost. Given the result in 3.3, we now derive the latency incurred by *SPRINTER* under the scenario in Figure 4. Let t_d, t_t and t_v represent the time required to inference the draft, target and verifier models respectively. We also assume $t_v \leq t_d$, i.e., the time it takes to run the verifier is smaller than running the draft model. Under the above assumption, the next result characterizes the expected stopping time (i.e., the average time before the first rejection by the verifier).

Theorem 3.4. *The expected stopping time is given as:*

$$\mathbb{E}[T_{Stop}] = \frac{(1-\eta_{TP}^r)t_d}{1-\eta_{TP}} + \frac{\eta_{TP}^r t_d}{1-\eta_{FP}}. \tag{4}$$

Proof of the above theorem can be found in Section A.3. From Theorem 3.4, we can observe that the expected stopping time increases as η_{TP} increases. We also observe that the dependence on the false-positive rate η_{FP} is marginal compared to η_{TP}. Furthermore, Fig. 6 shows that as η_{TP} approaches 1, even with a verifier having a relatively high η_{FP} (e.g., 0.5), the additional expected stopping time remains minimal (Gap $\leq t_d$) when $r = 5, t_d = 0.1$.

Savings in Computation. We now compare the total computational cost of *one run of SD* versus *one run of SPRINTER*. For simplicity, assume that one has a perfect verifier (i.e., $\eta_{FP} = 0$ and $\eta_{TP} = 1$). In SD, γ tokens are first generated by the draft model, followed by parallel verification done by the target model. If we denote F_d, F_t and F_v as the number of flops to run the draft, target and verifier models once, then we have

$$\text{SD-Flops}(\gamma) = \gamma F_d + \gamma F_t \tag{5}$$

$$\textit{SPRINTER}\text{-Flops}(\gamma) = \gamma F_d + \gamma F_v + F_t \tag{6}$$

If $F_v \ll F_d \ll F_t$, we can observe that computational savings from *SPRINTER* can be significant and grow proportional to $(\gamma - 1)F_t$ due to sequential verification through a low-complexity model (additional calculations showing the computational savings for model pairs in Section A.8).

3.2. Verifier Training and Architecture

Verifier Training Methodology. The verifier V is a binary classifier trained to predict whether a token should be accepted or rejected. Specifically, $V()$ can take as input various latent features derived from the draft model (e.g. embedding of the draft token x, probability distribution of the LLM's vocabulary). We denote the input to the verifier as $s(x, \text{prefix})$. The data used for training the verifier can be prepared as follows: for a given prefix, a token x is sampled from the draft model M_q. We also run the target model M_p and compute $p(x)$. Subsequently, binary labels are determined for each prefix and token pair by assigning 1 if $\frac{q(x)}{p(x)} \leq 1$ and 0 otherwise. However, rather than comparing the ratio $\frac{q(x)}{p(x)}$ against a threshold of 1, we can increase (decrease) the threshold to λ to bias the verifier to accept (reject) more draft tokens, which would increase (decrease) η_{FP} or decrease (increase) η_{TP}. Additionally, adjusting the inference threshold τ (see Algorithm 1) would achieve a similar effect. Thus, varying λ and τ serve as the two hyperparameters which allow us to influence $(\eta_{\text{FP}}, \eta_{\text{TP}})$.

Our second observation is that during inference, *SPRINTER* would face input as prefixes consisting of interleaved tokens generated in the past by draft and target models. Hence, during training, we expose the verifier to the following possible inputs: (a) an original prefix, (b) a prefix supplied with completions only from the draft model, (c) a prefix supplied with completions only from the target model (d) a prefix with tokens from both draft and target models. To optimize the verifier's performance, we ensure an equal proportional of prefixes from each category.

Verifier architecture used for evaluation. For our experiments, the verifier was implemented as a fully connected linear layer followed by a sigmoid activation, containing significantly fewer parameters than M_p and M_q. V takes as input the last embedding of the previous token and is trained with an Adam optimizer assuming binary cross entropy loss. Our results indicate that a single layer is sufficient to achieve strong performance while maintaining high efficiency, as shown in Fig. 3 on the Wiki-summary and LM1B datasets. We observed that training thresholds of $\lambda = 1.2$ enabled *SPRINTER* to attain an effective performance on Wiki-Summary and LM1B datasets.

4. Experiments and Evaluation

Evaluation goals. To quantify the effectiveness of *SPRINTER* we present the following set of results: (a) We measure the quality of responses generated by *SPRINTER* using two performance metrics: the win-tie rates and ROUGE scores. (b) We compare the latencies incurred by *SPRINTER*, SD (Leviathan et al., 2023), SpecDec++ (Huang et al., 2024), AdaEDL (Agrawal et al., 2024), and Mentored Decoding (Tran-Thien, 2024) in generating text completions given a prefix. (c) We compare the effectiveness of *SPRINTER* in completing tasks from Spec-Bench (Xia et al., 2024) with Medusa (Cai et al., 2024) and Eagle2 (Li et al.). (d) To investigate the impact of verifier training and inference hyperparameters on *SPRINTER*, we perform an ablation study to observe how η_{TP} and η_{FP} affect the ROC. (e) As part of our qualitative comparison, Figure 1(b) and Section A.7 present example responses from *SPRINTER* and SD, illustrating that *SPRINTER* is capable of producing coherent and contextually appropriate tokens without strictly imitating the target model's output distribution.

Code for SPRINTER is available at (Sprinter, 2025).

Dataset and Model Architecture. We use the WiKi-summary (Scheepers et al., 2018), LM1B (Chelba et al., 2013), and Spec-Bench (Xia et al., 2024) datasets for evaluation. Wiki-Summary is a collection of Wikipedia article summaries designed for text summarization tasks. The LM1B (One Billion Word Benchmark) Dataset is a large-scale corpus for language modeling and text generation tasks, extracted from news articles. Spec-Bench compiles questions from various LLM evaluation datasets (e.g. MT-Bench, GSM8K) covering different task categories including summarization and translation. We adopted a similar experimental setup as the prior works: (Fang, 2024; Chakraborty et al., 2024). We present results for three (draft, target) model pairs: GPT-Neo-125M (EleutherAI, 2024b)/GPT-Neo-1.3B (EleutherAI, 2024a), GPT2-Small (124M param.) (OpenAI, 2024a)/GPT2-XL (1.5B param.) (OpenAI, 2024b), and Vicuna 68M (Laboratory, 2024)/7B (LMSYS, 2023).

Quality Analysis. We investigate the quality of responses generated by *SPRINTER* compared with standard SD. To quantify the completion quality of *SPRINTER*, we report results using a) win-tie rates, and b) ROUGE scores.

M_q/M_p	WiKi-Summary	LM1B
GPT-Neo-125M/1.3B	45.2 ± 3.12	35.4 ± 5.08
GPT2-Small/XL	41.0 ± 6.82	41.6 ± 3.98

Table 1. Average win-tie rates of *SPRINTER* against SD for GPT-Neo-125M/GPT-Neo-1.3B and GPT2-Small/GPT2-XL pairs.

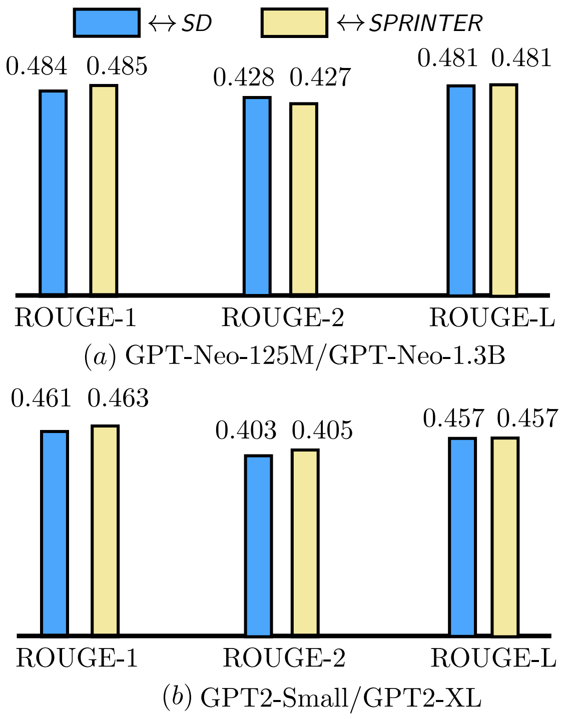


Figure 7. The ROUGE metrics (ROUGE-1, ROUGE-2, ROUGE-2) of *SPRINTER* vs SD for (a) GPT-Neo-125M/GPT-Neo-1.3B and (b) GPT2-Small/GPT2-XL model pairs. This demonstrates that even with faster inference speeds, *SPRINTER* experiences only a minimal drop in quality compared to SD.

Win-tie rate metric has been used extensively in LLM research (Rafailov et al., 2024; Shen et al., 2024). Win-tie rate measurements are taken by presenting GPT-4 with responses from two methods (*SPRINTER*-generated vs SD-generated) and prompting it to decide which response is better based on criteria provided by the user (additional details on win-tie rates are provided in Section A.9). Table 1 reports win-tie rate comparisons of *SPRINTER* with SD, where GPT-4 is provided with an initial prefix and completions from *SPRINTER* and SD. Approximately 30% of each prompt was given as input to each technique, which were constrained to generate 20 tokens per prompt. The table indicates that *SPRINTER* using the GPT-Neo model pair can generate responses of comparable quality to SD. This is especially observed on Wiki-Summary, which wins on-average 45.2% of the time against SD, indicating that *SPRINTER* suffers minimal quality degradation.

ROUGE score comparison. To further illustrate the com-

Model Pair		WiKi-Summary			LM1B		
	Methods	Avg Tokens	Time (ms)	Speedup	Avg Tokens	Time (ms)	Speedup
GPT-Neo-125M / GPT-Neo-1.3B	*SD*	2.17	0.92 ± 0.32	1x	1.95	0.84 ± 0.21	1x
	Specdec^{++}	1.16	0.75 ± 0.25	1.23x	1.08	0.61 ± 0.24	1.38x
	AdaEDL	2.66	0.72 ± 0.30	1.28x	1.91	0.56 ± 0.12	1.50x
	MentoredDec	3.39	0.76 ± 0.18	1.21x	3.11	0.62 ± 0.13	1.35x
	SPRINTER	**11.10**	**0.56 ± 0.22**	**1.64x**	**8.32**	**0.46 ± 0.09**	**1.83x**
GPT2-Small/ GPT2-XL	*SD*	2.01	0.84 ± 0.24	1x	2.05	0.73 ± 0.15	1x
	Specdec^{++}	1.82	0.72 ± 0.23	1.18x	1.04	0.61 ± 0.28	1.20x
	AdaEDL	1.96	0.73 ± 0.25	1.16x	1.52	0.64 ± 0.19	1.14x
	MentoredDec	3.20	0.60 ± 0.19	1.40x	3.12	0.55 ± 0.07	1.33x
	SPRINTER	**10.83**	**0.49 ± 0.20**	**1.69x**	**6.56**	**0.44 ± 0.12**	**1.66x**

Table 2. Latency speedups for *SPRINTER* relative to SD (Leviathan et al., 2023), AdaEDL (Agrawal et al., 2024), Specdec++ (Huang et al., 2024) and MentoredDec (Tran-Thien, 2024) using GPT families as the draft/target models on Wiki-Summary and LM1B datasets.

SPRINTER vs	Win-Tie rate	Speedup	Flops
SD	41	1.64x	$F_d(\gamma) + F_t(\gamma)$
Medusa	38	1.51x	$KF_d(\gamma) + F_t(\tau_n)$
Eagle2	32	1.17x	$F_t(\tau_n) + \tau_d F_d(\frac{\tau_n}{\tau_d})$
Mentored Dec	52	1.35x	$F_d(\gamma) + F_t(\gamma)$

Table 3. Comparison of *SPRINTER* with state-of-the-art methods (Medusa (Cai et al., 2024), Eagle2 (Li et al.) and Mentored Decoding (Tran-Thien, 2024)) in terms of win-tie rate and speedup. Medusa and Eagle2 FLOPS to process γ tokens from (Christopher et al., 2024), where τ_n, τ_d, K are the number of nodes/depth of the draft tree and Medusa heads respectively.

Eval\Train	Latency		Quality	
	WiKi	**Lm1b**	**WiKi**	**Lm1b**
WiKi	0.56 ± 0.22	0.57 ± 0.26	45.2 ± 3.12	47.6 ± 5.78
Lm1b	0.51 ± 0.09	0.46 ± 0.09	38.2 ± 1.94	39.6 ± 1.94

Table 4. Comparison of Latency and Quality between WiKi and Lm1b datasets. The verifier maintains comparable latency and quality when trained on one dataset and evaluated on another. Highlighted entries show the average win-tie rates using verifiers trained/evaluated on the same dataset.

parable quality between *SPRINTER* and SD, Figures 7 show ROUGE scores (Lin, 2004) of the responses made by *SPRINTER* and SD respectively with the reference summaries in Wiki-Summary for both GPT-Neo and GPT2 model pairs. The ROUGE scores measure differing levels of similarity between a provided "candidate" summary and reference summary (Lin, 2004). For both model pairs, the figures indicate that *SPRINTER* is able to generate responses that attain very similar ROUGE scores to SD.

Latency Speedups. Table 2 reports results comparing the latency speedups achieved with *SPRINTER* relative to the methods shown in Figure 2(c) for the Wiki-Summary and LM1B datasets. Given a prefix, 20 additional tokens were generated per prompt by each method. We report the "Avg Token" as the accepted token generated by M_q per single run of the sampling process. As the table indicates, *SPRINTER* achieves higher speedup improvements relative to SD compared to the other baselines, without being restricted to generate tokens that match the target model distribution.

Quality-Latency-Flops Tradeoffs. We further examine the trade-offs between quality and latency using Spec-Bench (Xia et al., 2024) with the Vicuna 68M/7B pair, comparing *SPRINTER* against SD, Medusa (Cai et al., 2024), and Eagle2 (Li et al.). For *SPRINTER*, we employ a verifier trained on LM1B responses from the GPT2-Small/GPT2-XL pair. Each method was constrained to generate 20 tokens per task. We also compare *SPRINTER* and the lossy method Mentored Decoding (Tran-Thien, 2024) on Wiki-Summaries using the GPT-Neo model pair. As shown in Table 3, *SPRINTER* requires significantly fewer FLOPs than SD and other variants, and achieves lower latency while preserving high output quality.

Verifier Ablation Study. Rather than strictly forcing the verifier to only accept draft tokens if the underlying ratio $\left(\frac{q_i(x|\text{prefix})}{p_i(x|\text{prefix})}\right) \leq 1$, the threshold of 1 could be changed to a parameter $\lambda > 1$; allowing for more tokens to be deemed acceptable, providing another method for *SPRINTER* to generate tokens that deviate from the target distribution and accelerate inference. We show ROC curves for the verifier on the LM1B and Wiki dataset when trained at thresholds $\lambda = 1$, 1.2, and 1.5 respectively in Section A.10.

Verifier Transferability. We also explore the generalization capability of the verifier and observe that it can be effectively transferred across tasks and datasets. As shown in Table 4, the verifier achieves comparable latency and quality (measured by win-tie rate) when a model trained on one dataset is used for inference on another. For example, evaluating a verifier trained on Lm1b attains an average win rate of 47.6 and 39.6 on Wiki and Lm1b respectively.

5. Conclusion

We introduced *SPRINTER*, a sampling framework designed to accelerate LLM inference by leveraging a draft-target model pair along with a lightweight verifier. Our theoretical analysis highlights the trade-offs between inference speed, computational efficiency, and output quality, demonstrating how verifier characteristics, such as false-positive and true-positive rates, influence performance. Through extensive experiments on multiple datasets and model pairs, we showed that *SPRINTER* significantly reduces latency while maintaining high-quality outputs. Our result show that sequential approximate verification can be effective in balancing efficiency and quality, making it a promising approach for scalable and efficient LLM deployment.

Acknowledgments

This work was supported by NSF grants CAREER 1651492, CCF-2100013, CNS-2209951, CNS-1822071, CNS-2317192, and by the U.S. Department of Energy, Office of Science, Office of Advanced Scientific Computing under Award Number DE-SC-ERKJ422, and NIH Award R01-CA261457-01A1.

Impact Statement

With the growing use of LLMs in different applications, and the scaling in their size and complexity, inference speed remains a critical bottleneck in real-world applications, affecting latency-sensitive tasks such as conversational AI, real-time translation, and autonomous systems. By introducing the idea of low-complexity sequential verification within the context of speculative decoding, *SPRINTER* can reduce response times and lowers computational costs without significantly compromising on the quality. *SPRINTER* ensures that the target model is not called frequently, resulting in a reduction in energy consumption compared to standard speculative decoding, while ensuring minimal degradation in the quality of its generated responses. Additionally, *SPRINTER* is able to generate these responses at significantly higher speeds compared to SD, making it more feasible for LLMs to be used in time-sensitive applications.

References

Agrawal, S., Jeon, W., and Lee, M. Adaedl: Early draft stopping for speculative decoding of large language models via an entropy-based lower bound on token acceptance probability. In *NeurIPS Efficient Natural Language and Speech Processing Workshop*, pp. 355–369. PMLR, 2024.

Bachmann, G., Anagnostidis, S., Pumarola, A., Georgopoulos, M., Sanakoyeu, A., Du, Y., Schönfeld, E., Thabet, A., and Kohler, J. Judge decoding: Faster speculative sampling requires going beyond model alignment. *arXiv preprint arXiv:2501.19309*, 2025.

Cai, T., Li, Y., Geng, Z., Peng, H., Lee, J. D., Chen, D., and Dao, T. Medusa: Simple llm inference acceleration framework with multiple decoding heads. *arXiv preprint arXiv:2401.10774*, 2024.

Casson, A. Transformer flops. 2023. URL https://adamcasson.com/posts/transformer-flops.

Chakraborty, S., Ghosal, S. S., Yin, M., Manocha, D., Wang, M., Bedi, A. S., and Huang, F. Transfer q star: Principled decoding for llm alignment, 2024. URL https://arxiv.org/abs/2405.20495.

Chelba, C., Mikolov, T., Schuster, M., Ge, Q., Brants, T., Koehn, P., and Robinson, T. One billion word benchmark for measuring progress in statistical language modeling. *arXiv preprint arXiv:1312.3005*, 2013.

Christopher, J. K., Bartoldson, B. R., Ben-Nun, T., Cardei, M., Kailkhura, B., and Fioretto, F. Speculative diffusion decoding: Accelerating language generation through diffusion. *arXiv preprint arXiv:2408.05636*, 2024.

EleutherAI. Eleutherai/gpt-neo-1.3b, 2024a. URL https://huggingface.co/EleutherAI/gpt-neo-1.3B. Accessed: 011-2024.

EleutherAI. Eleutherai/gpt-neo-125m, 2024b. URL https://huggingface.co/EleutherAI/gpt-neo-125m. Accessed: 011-2024.

Fang, J. Llmspeculativesampling. https://github.com/feifeibear/LLMSpeculativeSampling, 2024. Accessed: 012-2024.

Hoffmann, J., Borgeaud, S., Mensch, A., Buchatskaya, E., Cai, T., Rutherford, E., de Las Casas, D., Hendricks, L. A., Welbl, J., Clark, A., Hennigan, T., Noland, E., Millican, K., van den Driessche, G., Damoc, B., Guy, A., Osindero, S., Simonyan, K., Elsen, E., Rae, J. W., Vinyals, O., and Sifre, L. Training compute-optimal large language models, 2022. URL https://arxiv.org/abs/2203.15556.

Huang, K., Guo, X., and Wang, M. Specdec++: Boosting speculative decoding via adaptive candidate lengths. *arXiv preprint arXiv:2405.19715*, 2024.

Jang, D., Park, S., Yang, J. Y., Jung, Y., Yun, J., Kundu, S., Kim, S.-Y., and Yang, E. Lantern: Accelerating visual autoregressive models with relaxed speculative decoding, 2024. URL https://arxiv.org/abs/2410.03355.

Kim, S., Mangalam, K., Moon, S., Malik, J., Mahoney, M. W., Gholami, A., and Keutzer, K. Speculative decoding with big little decoder. *Advances in Neural Information Processing Systems*, 36, 2024.

Laboratory, N. K. double7/vicuna-68m, 2024. URL https://huggingface.co/double7/vicuna-68m. Accessed: 04-2025.

Langley, P. Crafting papers on machine learning. In Langley, P. (ed.), *Proceedings of the 17th International Conference on Machine Learning (ICML 2000)*, pp. 1207–1216, Stanford, CA, 2000. Morgan Kaufmann.

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

Li, Y., Wei, F., Zhang, C., and Zhang, H. Eagle-2: Faster inference of language models with dynamic draft trees, 2024b. *URL https://arxiv. org/abs/2406.16858*.

Li, Y., Wei, F., Zhang, C., and Zhang, H. Eagle: Speculative sampling requires rethinking feature uncertainty. *arXiv preprint arXiv:2401.15077*, 2024.

Lin, C.-Y. ROUGE: A package for automatic evaluation of summaries. In *Text Summarization Branches Out*, pp. 74–81, Barcelona, Spain, July 2004. Association for Computational Linguistics. URL https://aclanthology.org/W04-1013/.

LMSYS. lmsys/vicuna-7b-v1.3, 2023. URL https://huggingface.co/lmsys/vicuna-7b-v1.3. Accessed: 04-2025.

Lu, X., Zeng, Y., Ma, F., Yu, Z., and Levorato, M. Improving multi-candidate speculative decoding. *arXiv preprint arXiv:2409.10644*, 2024.

Mamou, J., Pereg, O., Korat, D., Berchansky, M., Timor, N., Wasserblat, M., and Schwartz, R. Dynamic speculation lookahead accelerates speculative decoding of large language models, 2024. URL https://arxiv.org/abs/2405.04304.

Melcer, D., Gonugondla, S., Perera, P., Qian, H., Chiang, W.-H., Wang, Y., Jain, N., Garg, P., Ma, X., and Deoras, A. Approximately aligned decoding. *arXiv preprint arXiv:2410.01103*, 2024.

Niu, Q., Chen, K., Li, M., Feng, P., Bi, Z., Liu, J., and Peng, B. From text to multimodality: Exploring the evolution and impact of large language models in medical practice. *arXiv preprint arXiv:2410.01812*, 2024.

OpenAI. openai-community/gpt2, 2024a. URL https://huggingface.co/openai-community/gpt2. Accessed: 09-2024.

OpenAI. openai-community/gpt2-xl, 2024b. URL https://huggingface.co/openai-community/gpt2-xl. Accessed: 09-2024.

Rafailov, R., Sharma, A., Mitchell, E., Manning, C. D., Ermon, S., and Finn, C. Direct preference optimization: Your language model is secretly a reward model. *Advances in Neural Information Processing Systems*, 36, 2024.

Scheepers, T., Kanoulas, E., and Gavves, E. Improving word embedding compositionality using lexicographic definitions. In *Proceedings of the 2018 World Wide Web Conference*, pp. 1083–1093, 2018.

Shen, S. Z., Lang, H., Wang, B., Kim, Y., and Sontag, D. Learning to decode collaboratively with multiple language models. *arXiv preprint arXiv:2403.03870*, 2024.

Sprinter. Code for SPRINTER, 2025. URL https://github.com/MeiyuZhong/SPRINTER.git.

Sun, Z., Suresh, A. T., Ro, J. H., Beirami, A., Jain, H., and Yu, F. Spectr: Fast speculative decoding via optimal transport. *Advances in Neural Information Processing Systems*, 36, 2024.

Tang, Y., Bi, J., Xu, S., Song, L., Liang, S., Wang, T., Zhang, D., An, J., Lin, J., Zhu, R., Vosoughi, A., Huang, C., Zhang, Z., Liu, P., Feng, M., Zheng, F., Zhang, J., Luo, P., Luo, J., and Xu, C. Video understanding with large language models: A survey, 2024. URL https://arxiv.org/abs/2312.17432.

Tran-Thien, V. An optimal lossy variant of speculative decoding, June 2024. URL https://huggingface.co/blog/vivien/optimal-lossy-variant-of-speculative-decoding. Accessed: 2025-05-09.

Xia, H., Yang, Z., Dong, Q., Wang, P., Li, Y., Ge, T., Liu, T., Li, W., and Sui, Z. Unlocking efficiency in large language model inference: A comprehensive survey of speculative decoding. In Ku, L.-W., Martins, A., and Srikumar, V. (eds.), *Findings of the Association for Computational Linguistics ACL 2024*, pp. 7655–7671, Bangkok, Thailand and virtual meeting, August 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.findings-acl.456. URL https://aclanthology.org/2024.findings-acl.456.

Yin, M., Chen, M., Huang, K., and Wang, M. A theoretical perspective for speculative decoding algorithm. *arXiv preprint arXiv:2411.00841*, 2024.

Zingale, J. and Kalita, J. Language model sentence completion with a parser-driven rhetorical control method. *arXiv preprint arXiv:2402.06125*, 2024.

A. Appendix

The Appendix is organized as follows:

A.1. Proof of Theorem 3.1

Theorem 3.1. *The probability of a token x being chosen when running SPRINTER is given as*

$$p_{SPRINTER}(x) = (1-\eta_{FP})p(x) + \eta_{FP}q(x), \tag{7}$$

where η_{FP} is the false positive rate of the verifier. Furthermore, the total-variation distance between the target and SPRINTER distributions is $d_{TV}(p, p_{SPRINTER}) = \eta_{FP} d_{TV}(p,q)$.

Proof. We denote η_{FP} and η_{FN} as the false positive and false negative rate of the verifier respectively. We define $x_{\text{acc}} = \{x | \frac{q(x)}{p(x)} \leq 1\}$ and $x_{\text{rej}} = \{x | \frac{q(x)}{p(x)} > 1\}$ to represent the sets of tokens generated by the draft model that should be accepted and rejected by the verifier respectively.

In general, we consider two cases: (a) the token should be rejected while it is accepted by verifier V, (b) the token should be accepted but it is rejected by V. Let us first consider the case when the token should be rejected (i.e. $x \in x_{\text{rej}}$). If $V(s(x, \text{prefix})) = 0$, meaning that the verifier predicts that $\frac{q(x)}{p(x)} > 1$, then we reject the token x sampled from $q(x)$ with probability $1 - \frac{p(x)}{q(x)}$ and re-sample x from an adjusted distribution norm$(\max(0, p(x) - q(x)))$. Therefore, the probability that token x is accepted is:

$$\frac{p(x)}{q(x)} \times (1-\eta_{\text{FP}}) \times q(x). \tag{8}$$

If $V(s(x, \text{prefix})) = 1$, meaning that the verifier predicts that $\frac{q(x)}{p(x)} \leq 1$, we accept the token, which occurs with probability:

$$\eta_{\text{FP}} \times q(x). \tag{9}$$

Therefore, combining (8) and (9), the probability of a token $x \in x_{\text{rej}}$ being accepted under SPRINTER is:

$$(1-\eta_{\text{FP}})p(x) + \eta_{\text{FP}}q(x). \tag{10}$$

Now we consider the case when the token should be accepted (i.e. $x \in x_{\text{acc}}$) . If $V(s(x, \text{prefix})) = 0$, with probability $1 - \eta_{\text{TP}}$, we reject the token x and call the larger model M_p, which will verify that indeed $\frac{q(x)}{p(x)} \leq 1$. Therefore, though the verifier made a mistake, it can be corrected by M_p. The probability that token x is accepted is:

$$q(x) \times (1-\eta_{\text{TP}}). \tag{11}$$

If $V(s(x, \text{prefix})) = 1$, we accept token x with probability:

$$q(x) \times \eta_{\text{TP}}. \tag{12}$$

While the above scenarios relied directly on the decision of the verifier given an acceptable token, there is an additional scenario where a token $x \in x_{\text{acc}}$ is accepted despite not being initially sampled from M_q. Assume that a token sampled from the draft model should be rejected (i.e. $x \in x_{\text{rej}}$) and the verifier accurately predicts to invoke the target model. Under this event, it would be possible for a token that is acceptable to be re-sampled from the adjusted distribution. The probability of token $x \in x_{\text{acc}}$ being accepted under this scenario is:

$$\begin{aligned}
&(1-\eta_{\text{FP}}) \times \left(\sum_{x_{\text{rej}}} q(x_{\text{rej}})(1 - \frac{p(x_{\text{rej}})}{q(x_{\text{rej}})}) \cdot \frac{p(x) - q(x)}{\sum_{x_{\text{acc}}} p(x_{\text{acc}}) - q(x_{\text{acc}})} \right) \\
&= (1-\eta_{\text{FP}}) \times (p(x) - q(x)) \times \frac{\sum_{x_{\text{rej}}} p(x_{\text{rej}}) - q(x_{\text{rej}})}{\sum_{x_{\text{acc}}} p(x_{\text{acc}}) - q(x_{\text{acc}})} \\
&= (1-\eta_{\text{FP}}) \times (p(x) - q(x)).
\end{aligned} \tag{13}$$

Overall, combining (11), (12) and (13), the probability of a token $x \in x_{\text{acc}}$ being accepted under *SPRINTER* is:

$$\begin{aligned}
&= q(x) \times (1 - \eta_{\text{FN}} + \eta_{\text{FN}}) + (1-\eta_{\text{FP}}) \times (p(x) - q(x)) \\
&= q(x) + (1-\eta_{\text{FP}}) \times (p(x) - q(x)) \\
&= (1-\eta_{\text{FP}}) p(x) + \eta_{\text{FP}} q(x).
\end{aligned} \tag{14}$$

Together with (10) and (14), we complete the proof of Theorem 3.1.

Furthermore, the total variation distance between p_{SPRINTER} and p can be calculated as follows:

$$\begin{aligned}
d_{\text{TV}}(p, p_{\textit{SPRINTER}}) &= \frac{1}{2} \sum_x |p(x) - p_{\textit{SPRINTER}}(x)| \\
&= \frac{1}{2} \sum_x |p(x) - (1-\eta_{\text{FP}})p(x) - \eta_{\text{FP}} q(x)| \\
&= \frac{1}{2} \sum_x |\eta_{\text{FP}} p(x) - \eta_{\text{FP}} q(x)| \\
&= \eta_{\text{FP}} \frac{1}{2} \sum_x |p(x) - q(x)| \\
&= \eta_{\text{FP}} d_{\text{TV}}(p, q).
\end{aligned} \tag{15}$$

□

A.2. Proof of Theorem 3.3

Theorem 3.3. *The probability distribution of the number of generated tokens is given as:*

$$\mathbb{P}(N_{\textit{SPRINTER}} = i) = \begin{cases} \eta_{TP}^i (1 - \eta_{TP}) & i < r, \\ \eta_{TP}^r (\eta_{FP})^{i-r} (1 - \eta_{FP}) & i \geq r. \end{cases}$$

The expected number of generated tokens is given as:

$$\mathbb{E}(N_{\textit{SPRINTER}}) = \frac{\eta_{TP} - \eta_{TP}^r}{1 - \eta_{TP}} + \frac{\eta_{TP}^r}{1 - \eta_{FP}}. \tag{16}$$

Proof. We assume that the first r tokens are acceptable, while subsequent tokens are unacceptable. This can be modeled as a finite geometric series for the accepted tokens and an infinite geometric series for the rejected ones. The expected number of generated tokens is derived as follows:

$$\begin{aligned}\mathbb{E}[N_{\mathit{SPRINTER}}] &= \sum_{k=0}^{\infty} kP(N_{\mathit{SPRINTER}} = k) \\ &= \underbrace{\sum_{k=0}^{r-1} kP(N_{\mathit{SPRINTER}} = k)}_{\text{Term 1}} + \underbrace{\sum_{k=r}^{\infty} kP(N_{\text{SPRINTER}} = k)}_{\text{Term 2}} \end{aligned} \tag{17}$$

We first expand Term 1 as follows:

$$\begin{aligned}\sum_{k=0}^{r-1} kP(N_{\mathit{SPRINTER}} = k) &= 0 \times (1-\eta_{\text{TP}}) + 1 \times (1-\eta_{\text{TP}})\eta_{\text{TP}} + 2 \times (1-\eta_{\text{TP}})\eta_{\text{TP}}^2 + \ldots + (r-1) \times (1-\eta_{\text{TP}})\eta_{\text{TP}}^{r-1} \\ &\overset{(a)}{=} (1-\eta_{\text{TP}}) \sum_{k=0}^{r-1} k\eta_{\text{TP}}^k \end{aligned}$$

Note that (a) represents a finite geometric series, allowing us to apply the sum formula for such a series, resulting in:

$$\begin{aligned}(1-\eta_{\text{TP}}) \sum_{k=0}^{r-1} k\eta_{\text{TP}}^k &= (1-\eta_{\text{TP}})\eta_{\text{TP}} \frac{1 - r\eta_{\text{TP}}^{r-1} + (r-1)\eta_{\text{TP}}^r}{(1-\eta_{\text{TP}})^2} \\ &= \frac{\eta_{\text{TP}} - \eta_{\text{TP}}^r}{1-\eta_{\text{TP}}} - (r-1)\eta_{\text{TP}}^r \end{aligned} \tag{18}$$

Similarly, we simplify Term 2 as follows:

$$\begin{aligned}\sum_{k=r}^{\infty} kP(N_{\mathit{SPRINTER}} = k) &\overset{(a)}{=} r \times \eta_{\text{TP}}^r(1-\eta_{\text{FP}}) + (r+1) \times \eta_{\text{TP}}^r\eta_{\text{FP}}(1-\eta_{\text{FP}}) + (r+2) \times \eta_{\text{TP}}^r\eta_{\text{FP}}^2(1-\eta_{\text{FP}}) + \ldots \\ &\overset{(b)}{=} ar + a(r+1)\eta_{\text{FP}} + a(r+2)\eta_{\text{FP}}^2 + \ldots \\ &= a \sum_{k=r}^{\infty} k\eta_{\text{FP}}^{k-r} \\ &= \frac{a}{\eta_{\text{FP}}^r} \sum_{k=r}^{\infty} k\eta_{\text{FP}}^k \\ &\overset{(c)}{=} \frac{a}{\eta_{\text{FP}}^r} \sum_{k=0}^{\infty} k\eta_{\text{FP}}^k - \frac{a}{\eta_{\text{FP}}^r} \sum_{k=0}^{r-1} k\eta_{\text{FP}}^k \end{aligned} \tag{19}$$

where (a) the sum starts from the rth token (i.e. the first unacceptable token), (b) follows from setting $a = \eta_{\text{TP}}^r(1-\eta_{\text{FP}})$. We also observe that (c) is a combination of two geometric series. Therefore, the first term in (19) can be simplified as:

$$\begin{aligned}\frac{a}{\eta_{\text{FP}}^r} \sum_{k=0}^{\infty} k\eta_{\text{FP}}^k &= \frac{a}{\eta_{\text{FP}}^r} \times \frac{\eta_{\text{FP}}}{(1-\eta_{\text{FP}})^2} \\ &= \frac{\eta_{\text{TP}}^r}{\eta_{\text{FP}}^r} \times \frac{\eta_{\text{FP}}}{1-\eta_{\text{FP}}} \end{aligned} \tag{20}$$

Similarly, we simplify the second term in (19) as:

$$\frac{a}{\eta_{\text{FP}}^r} \sum_{k=0}^{r-1} k\eta_{\text{FP}}^k = \frac{\eta_{\text{TP}}^r}{\eta_{\text{FP}}^r} \times \left(\frac{\eta_{\text{FP}} - \eta_{\text{FP}}^r}{1-\eta_{\text{FP}}} - (r-1)\eta_{\text{FP}}^r \right) \tag{21}$$

Plugging (20) and (21) into (19), we have:

$$\begin{aligned}\frac{a}{\eta_{\text{FP}}^r}\sum_{k=0}^{\infty} k\eta_{\text{FP}}^k - \frac{a}{\eta_{\text{FP}}^r}\sum_{k=0}^{r-1} k\eta_{\text{FP}}^k &= \frac{\eta_{\text{TP}}^r}{\eta_{\text{FP}}^r}\left(\frac{\eta_{\text{FP}}}{1-\eta_{\text{FP}}} - \frac{\eta_{\text{FP}}-\eta_{\text{FP}}^r}{1-\eta_{\text{FP}}} + (r-1)\eta_{\text{FP}}^r\right)\\ &= \frac{\eta_{\text{TP}}^r}{\eta_{\text{FP}}^r}\left(\frac{\eta_{\text{FP}}^r}{1-\eta_{\text{FP}}} + (r-1)\eta_{\text{FP}}^r\right)\\ &= \eta_{\text{TP}}^r\left(\frac{1}{1-\eta_{\text{FP}}} + (r-1)\right) \end{aligned} \tag{22}$$

Combining (18) and (22), we obtain:

$$\begin{aligned}\mathbb{E}(N_{\text{SPRINTER}}) &= \frac{\eta_{\text{TP}}-\eta_{\text{TP}}^r}{1-\eta_{\text{TP}}} - (r-1)\eta_{\text{TP}}^r + \eta_{\text{TP}}^r\left(\frac{1}{1-\eta_{\text{FP}}} + (r-1)\right)\\ &= \frac{\eta_{\text{TP}}-\eta_{\text{TP}}^r}{1-\eta_{\text{TP}}} + \frac{\eta_{\text{TP}}^r}{1-\eta_{\text{FP}}}.\end{aligned} \tag{23}$$

The above expression gives Theorem 3.3. □

A.3. Proof of Theorem 3.4

Theorem 3.4. *The expected stopping time is given as:*

$$\mathbb{E}[T_{Stop}] = \frac{(1-\eta_{TP}^r)t_d}{1-\eta_{TP}} + \frac{\eta_{TP}^r t_d}{1-\eta_{FP}}. \tag{24}$$

Proof. Similar as above Theorem, we assume that the first r tokens are acceptable, while all subsequent tokens are not. We also note that the verifier's runtime is negligible compared to that of the draft LLM M_q. Therefore, we consider only t_d for the expected stopping time. We first reduce the expected stopping time as follows:

$$\begin{aligned}\mathbb{E}[T_{\text{Stop}}] &= \sum_{k=0}^{\infty} t_d(k+1)P(T_{\text{Stop}}=k)\\ &= \underbrace{\sum_{k=0}^{r-1} t_d(k+1)P(T_{\text{Stop}}=k)}_{\text{Term 1'}} + \underbrace{\sum_{k=r}^{\infty} t_d(k+1)P(T_{\text{Stop}}=k)}_{\text{Term 2'}}\end{aligned} \tag{25}$$

We now write out Term 1', representing the case that we stop before the rth acceptable token, as follows:

$$\begin{aligned}\sum_{k=0}^{r-1} t_d(k+1)P(T_{\text{Stop}}=k) &= t_d\times(1-\eta_{\text{TP}}) + 2t_d\times(1-\eta_{\text{TP}})\eta_{\text{TP}} + \ldots + rt_d\times(1-\eta_{\text{TP}})\eta_{\text{TP}}^{r-1}\\ &= \sum_{k=0}^{r-1}(1-\eta_{\text{TP}})\eta_{\text{TP}}^k\times(k+1)t_d\\ &= (1-\eta_{\text{TP}})t_d\sum_{k=0}^{r-1}k\eta_{\text{TP}}^k + (1-\eta_{\text{TP}})t_d\sum_{k=0}^{r-1}\eta_{\text{TP}}^k,\end{aligned} \tag{26}$$

where (26) is the combination of two geometric series. Therefore, (26) can be simplified as follows:

$$\begin{aligned}&(1-\eta_{\text{TP}})t_d\sum_{k=0}^{r-1}k\eta_{\text{TP}}^k + (1-\eta_{\text{TP}})t_d\sum_{k=0}^{r-1}\eta_{\text{TP}}^k\\ &= t_d\left(\frac{\eta_{\text{TP}}-\eta_{\text{TP}}^r}{1-\eta_{\text{TP}}} - (r-1)\eta_{\text{TP}}^r\right) + t_d(1-\eta_{\text{TP}}^r)\\ &= t_d\times\left(\frac{\eta_{\text{TP}}-\eta_{\text{TP}}^r}{1-\eta_{\text{TP}}} - r\eta_{\text{TP}}^r + 1\right)\end{aligned} \tag{27}$$

We next expand Term 2', which represents the case that we stop at or after the rth acceptable token, as follows:

$$\begin{aligned}\sum_{k=r}^{\infty} t_d(k+1)P(T_{\text{Stop}}=k) &= (r+1)t_d \times \eta_{\text{TP}}^r(1-\eta_{\text{FP}}) + (r+2)t_d \times \eta_{\text{TP}}^r\eta_{\text{FP}}(1-\eta_{\text{FP}}) + \dots \\ &\overset{(a)}{=} b(r+1) + b(r+2)\eta_{\text{FP}} + \dots \\ &= b\sum_{k=r}^{\infty} k\eta_{\text{FP}}^{k-r} + b\sum_{k=r}^{\infty}\eta_{\text{FP}}^{k-r} \\ &= \frac{b}{\eta_{\text{FP}}^r}\sum_{k=r}^{\infty} k\eta_{\text{FP}}^k + \frac{b}{\eta_{\text{FP}}^r}\sum_{k=r}^{\infty}\eta_{\text{FP}}^k \end{aligned} \tag{28}$$

where (a) follows from setting $b = \eta_{\text{TP}}^r(1-\eta_{\text{FP}})t_d$. Note that the first term in (28) can be written as:

$$\begin{aligned}\frac{b}{\eta_{\text{FP}}^r}\sum_{k=r}^{\infty} k\eta_{\text{FP}}^k &= \frac{b}{\eta_{\text{FP}}^r}\left(\sum_{k=0}^{\infty} k\eta_{\text{FP}}^k - \sum_{k=0}^{r-1} k\eta_{\text{FP}}^k\right) \\ &= \frac{\eta_{\text{TP}}^r}{\eta_{\text{FP}}^r}t_d \times \left(\frac{\eta_{\text{FP}}}{1-\eta_{\text{FP}}} - \frac{\eta_{\text{FP}}-\eta_{\text{FP}}^r}{1-\eta_{\text{FP}}} + (r-1)\eta_{\text{FP}}^r\right)\end{aligned} \tag{29}$$

Similarly, the second term in (28) can be expressed as:

$$\begin{aligned}\frac{b}{\eta_{\text{FP}}^r}\sum_{k=r}^{\infty}\eta_{\text{FP}}^k &= \frac{b}{\eta_{\text{FP}}^r}\left(\sum_{k=0}^{\infty}\eta_{\text{FP}}^k - \sum_{k=0}^{r-1}\eta_{\text{FP}}^k\right) \\ &= \frac{\eta_{\text{TP}}^r}{\eta_{\text{FP}}^r}t_d\left(1-(1-\eta_{\text{FP}}^r)\right)\end{aligned} \tag{30}$$

Plugging (29) and (30) into (28), we obtain the expected stopping time after or at rth acceptable token is:

$$\frac{\eta_{\text{TP}}^r}{1-\eta_{\text{FP}}}t_d + rt_d\eta_{\text{TP}}^r \tag{31}$$

Combining (27) and (31), the final expression of the expected stopping time is given as:

$$\mathbb{E}[T_{\text{Stop}}] = \frac{(1-\eta_{\text{TP}}^r)t_d}{1-\eta_{\text{TP}}} + \frac{\eta_{\text{TP}}^r t_d}{1-\eta_{\text{FP}}}. \tag{32}$$

This completes the proof. □

A.4. Additional theoretical results

In this section, building on previous work by (Leviathan et al., 2023), we examine the acceptance rate of a single run of SPRINTER. This analysis provides insight into how the verifiers' statistical properties influence the acceptance rate.
Calculation of $\alpha_{SPRINTER}$ and $\beta_{SPRINTER}$. It is well known (Leviathan et al., 2023) that the acceptance rate of standard speculative decoding β is given as:

$$\beta = 1 - d_{TV}(p,q), \tag{33}$$

where $d_{TV}(p,q)$ is the total variation distance between the distributions p and q. In the next Theorem, we derive an analogous result showing the acceptance rate of tokens generated by M_q when using *SPRINTER*.

Theorem A.4. *The acceptance rate of SPRINTER $\beta_{SPRINTER}$ is*

$$1-(1-\eta_{FP})\cdot d_{TV}(p,q). \tag{34}$$

Proof. From the *SPRINTER* sampling procedure, we know that $\beta_{SPRINTER}$ satisfies the following:

$$\beta_{SPRINTER} = \mathbb{E}_{x\sim q(x)}\begin{cases}1 & \text{if } q(x)\le p(x) \\ \eta_{\text{FP}} + \frac{p(x)}{q(x)}(1-\eta_{\text{FP}}) & \text{if } q(x) > p(x).\end{cases} \tag{35}$$

The above expectation can be simplified and computed explicitly as follows:

$$\begin{aligned}
\beta_{\textit{SPRINTER}} &= \sum_x \min(q(x), \eta_{\text{FP}} q(x) + (1-\eta_{\text{FP}})p(x)) \\
&= \sum_x \min(\eta_{\text{FP}} q(x) + (1-\eta_{\text{FP}})q(x), \eta_{\text{FP}} q(x) + (1-\eta_{\text{FP}})p(x)) \\
&= \sum_x (1-\eta_{\text{FP}}) \min(q(x), p(x)) + \eta_{\text{FP}} q(x) \\
&= (1-\eta_{\text{FP}}) \cdot (1 - d_{TV}(p,q)) + \eta_{\text{FP}} \\
&= 1 - (1-\eta_{\text{FP}}) \cdot d_{TV}(p,q). \qquad (36)
\end{aligned}$$

□

Remark A.5. Note that when $\eta_{\text{FP}} = 0$, the acceptance rate is the same as that of speculative decoding.

Remark A.6. When $\eta_{\text{FP}} \neq 0$, $\beta_{\textit{SPRINTER}} \geq \beta$ meaning that *SPRINTER* has a higher acceptance rate compared to that of standard speculative decoding. The inherent reason is that the classifier is susceptible to predicting that the token is acceptable when it actually should be rejected, causing a higher acceptance rate and subsequently enabling the draft model to generate more tokens. However, η_{FP} also reflects the distance of the probability distribution of *SPRINTER* from the probability distribution of the target model p. This means that while more tokens are being accepted, the distribution they are sampled from may deviate from the distribution of M_p, causing poorer quality tokens to be generated. Therefore, η_{FP} measures the tradeoff between the accuracy of the tokens generated and the latency incurred from using *SPRINTER*.

A.5. Full SPRINTER Algorithm

Algorithm 2 provides the detailed overview of Algorithm stated in the main paper, detailing the procedure of a full step of SPRINTER.

Algorithm 2 SPRINTER (detailed)

```
Input: $M_p, M_q, V$, Prefix, Prediction Threshold $\tau$
$x = \emptyset$
Sample tokens from $M_q$ while verifier predicts $\frac{q(x)}{p(x)} \leq 1$
$r = 1$
while True do
  Prefix = Prefix + $x$
  $q_r(x) \sim M_q(\text{Prefix})$
  $x \sim q_r(x)$
  if $V(s(x, \text{Prefix})) \geq \tau$ then
    $r += 1$
  else
    Break
  end if
end while
Run $M_p$ Once.
$p_{r+1}(x) \sim M_p(\text{Prefix} + x_r)$
$p'(x) \leftarrow p_{r+1}(x)$
$c \sim U(0,1)$
$n = \min(1, \frac{p_{r+1}(x)}{q_{r+1}(x)})$
if $c > n$ then
  $p'(x) \sim \text{norm}(\max(0, p_{r+1}(x) - q_{r+1}(x)))$
end if
$t \sim p'(x)$
Return Prefix + $t$
```

More Training Details Regarding the Verifier and Hyperparameter Tuning During the training process of the verifier, we vary the value of λ among 1, 1.2, and 1.5 to generate the ground truth regarding whether a token should be accepted or rejected. By increasing λ, we bias the verifier to accept more draft tokens, potentially increasing η_{FP}. Experimental results show that SPRINTER with $\lambda = 1.2$ performs best on the LM1B dataset for both GPT-Neo and GPT2 model pairs. On the Wiki-Summary dataset, SPRINTER with $\lambda = 1$ and $\lambda = 1.2$ achieves superior results using the GPT2 and GPT-Neo model pairs respectively. For the verifier's prediction threshold τ, we observe that $\tau = 0.5$ is sufficient to achieve strong performance.

We split the dataset into train/test sets, using early stopping to mitigate overfitting. The input dimension of the verifier depends on the dimension of the assumed draft model. As GPT-Neo-125M has an output dimension of 768, for example, the verifier used would also have a dimension of 768 neurons.

A.6. Detailed Discussion of Related Works

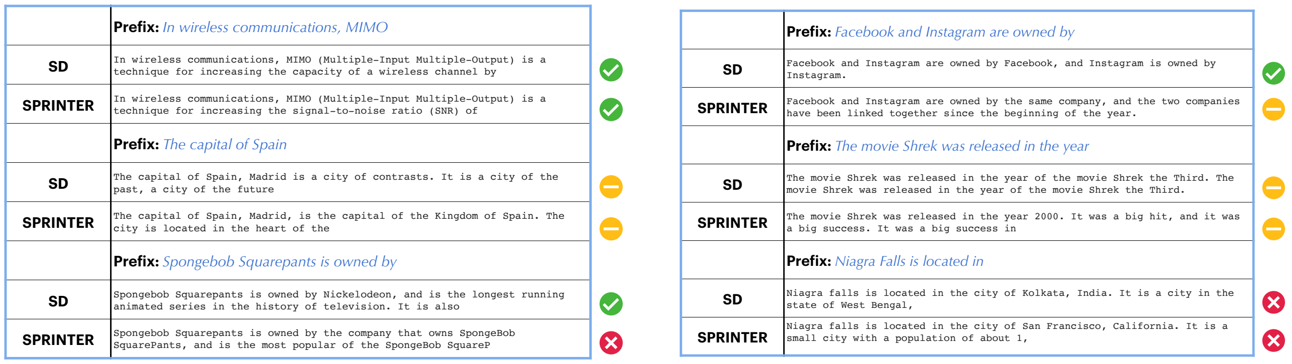
Multiple approaches for optimizing SD have been presented in the literature. SpecTr (Sun et al., 2024) uses optimal transport to effectively manage the draft selection process, ensuring efficient and accurate sampling from large models. Yin et al. (Yin et al., 2024) frame the decoding problem through a Markov chain abstraction and analyzing its key properties—output quality and inference acceleration—from a theoretical perspective. Recently, Judge Decoding (Bachmann et al., 2025) uses a trained linear head to judge token validity/quality beyond mere alignment, significantly speeding up inference while maintaining output quality. Under this method, tokens that may not necessarily match the target distribution may still be accepted as they are still of high-quality. Similarly, (Jang et al., 2024) presents a variant of speculative decoding for visual autoregressive models by relaxing the constraint that tokens must match the target distribution while proposing a mechanism based on total variation distance to ensure that the distribution drift between the generated tokens and the target model does not exceed a certain threshold.

There are also works that have similarly used an additional classifier for determining how many tokens the draft model should generate. (Mamou et al., 2024) proposed using a two layer feedforward network to predict when the draft model should stop generating tokens and initiate the target model's verification procedure. (Huang et al., 2024) models the SD procedure as a Markov Decision Process and uses the draft model with an added head to predict if the draft model should stop generating tokens. However, both methods ensure that sampling the draft model with their method is equivalent to sampling the target model, which hinders improved latency reductions that can be attained if this requirement is relaxed. (Agrawal et al., 2024) proposes prove that a function of the entropy of the distribution of the draft model can be used as a means of indicating when to end a round of drafting tokens. Specifically, the output of this function is compared against a threshold, which is also adjusted dynamically based on the current acceptance rate, to determine if the current drafting round should end. (Kim et al., 2024) provides two heuristics for determining when the target model should take control in generating tokens. The first involves observing the draft model's distribution to see if it has a certain lack of confidence in its current token, which would indicate that the target model should generate a replacement token. The second takes place during verification, when the distributions of the target model and draft model are compared to observe an instant in which the draft model is too confident in its decision. This indicates that the sequence should revert back to this point, with the target model generating a replacement token. A method for fine-tuning the draft model to generate tokens that better align with the target model is also presented.

(Lu et al., 2024) investigated using different classifier architectures for halting the drafting process for multi-candidate speculative decoding. MEDUSA (Cai et al., 2024) propose a framework to accelerate inference in large language models (LLMs) by employing multiple decoding heads. By introducing parallelism in the decoding process, MEDUSA aims to improve the efficiency of LLM operations significantly. EAGLE (Li et al., 2024) presents a speculative sampling framework that improves large language model (LLM) inference by predicting second-to-top-layer features based on a token sequence advanced by one time step.

A.7. Additional Examples of Responses generated by Sprinter and SD

In this section, we feed additional prefixes to *SPRINTER* and SD and show their respective generated responses in Figure 8. We observe that *SPRINTER* can achieve a relatively good performance compared with SD, but can sometimes generate incorrect information.



	Prefix: *In wireless communications, MIMO*	
SD	`In wireless communications, MIMO (Multiple-Input Multiple-Output) is a technique for increasing the capacity of a wireless channel by`	✓
SPRINTER	`In wireless communications, MIMO (Multiple-Input Multiple-Output) is a technique for increasing the signal-to-noise ratio (SNR) of`	✓
	Prefix: *The capital of Spain*	
SD	`The capital of Spain, Madrid is a city of contrasts. It is a city of the past, a city of the future`	⊖
SPRINTER	`The capital of Spain, Madrid, is the capital of the Kingdom of Spain. The city is located in the heart of the`	⊖
	Prefix: *Spongebob Squarepants is owned by*	
SD	`Spongebob Squarepants is owned by Nickelodeon, and is the longest running animated series in the history of television. It is also`	✓
SPRINTER	`Spongebob Squarepants is owned by the company that owns SpongeBob SquarePants, and is the most popular of the SpongeBob SquareP`	✗

	Prefix: *Facebook and Instagram are owned by*	
SD	`Facebook and Instagram are owned by Facebook, and Instagram is owned by Instagram.`	✓
SPRINTER	`Facebook and Instagram are owned by the same company, and the two companies have been linked together since the beginning of the year.`	⊖
	Prefix: *The movie Shrek was released in the year*	
SD	`The movie Shrek was released in the year of the movie Shrek the Third. The movie Shrek was released in the year of the movie Shrek the Third.`	⊖
SPRINTER	`The movie Shrek was released in the year 2000. It was a big hit, and it was a big success. It was a big success in`	⊖
	Prefix: *Niagra Falls is located in*	
SD	`Niagra falls is located in the city of Kolkata, India. It is a city in the state of West Bengal,`	✗
SPRINTER	`Niagra falls is located in the city of San Francisco, California. It is a small city with a population of about 1,`	✗

Figure 8. Comparison of responses generated by *SPRINTER* and SD under the same prefixes.

A.8. Flops Calculation

We adopt the methods used in (Hoffmann et al., 2022; Casson, 2023) to determine the number of floating point operations (FLOPS) performed in a forward pass of the draft and target models used in this work. The main sources of FLOPS considered in (Hoffmann et al., 2022; Casson, 2023) are due to the embedding matrices, self-attention operations in the transformer blocks of the LLM, the feedforward networks in each transfomer block, and the operations required to generate the final logits. Table 10 presents the number of FLOPS needed for each draft and target model to generate 20 tokens. The table shows that GPT-Neo-1.3B and GPT2-XL require roughly 8 and 10 times the number of FLOPS compared to GPT-Neo-125M and GPT2-Small respectively.

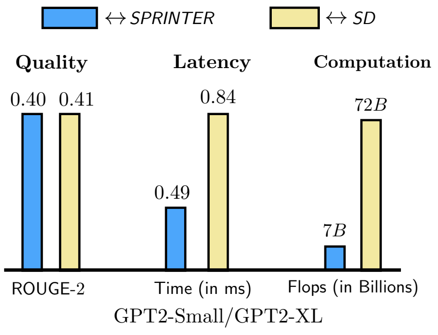


Figure 9. Comparison between *SPRINTER* and SD in terms of *Quality* (ROUGE score), *Latency* (ms to generate a sentence), and *Computation* (FLOPs to generate 20 acceptable tokens) for GPT2-Small and GPT2-XL.

Model	FLOPs for 20 tokens
GPT-Neo-125M	8.01B
GPT-Neo-1.3B	64.66B
GPT2-Small	7.25B
GPT2-XL	71.78B

Figure 10. Estimated FLOPs required for each model to generate 20 tokens.

Recall from Section 3.1, that if γ consecutive tokens are generated by the draft model, our verifier is of a lower complexity to the draft model, and, as evidenced by Figure 10, that the number of FLOPS used by the target model are significantly greater than the FLOPS used by the target model, then the computational savings experienced under *SPRINTER* is $(\gamma - 1)F_t$ where F_t denotes the number of FLOPS of the target model. This further shows that *SPRINTER* is significantly less computationally expensive compared to SD, which relies on the target model for parallel verification every γ tokens.

Figure 9 shows the quality-latency-computational profile experienced by *SPRINTER* and SD assuming the GPT2-Small/GPT2-XL model pair. Similar to Figure 2, which shows the same profile for the GPT-Neo model pair, we observe that *SPRINTER* is able to incur a lower latency than SD while suffering a minimal dip in quality and incurring more computational savings by running inference on the draft model more frequently than the target model.

A.9. Prompt Design for Win-tie Rate Evaluation

In this work, GPT-4 is used to determine the win-tie rates. Specifically, GPT-4 is given the original prefix, a completion generated by *SPRINTER*, and a completion generated by a baseline method. Similar to (Chakraborty et al., 2024), GPT-4 is

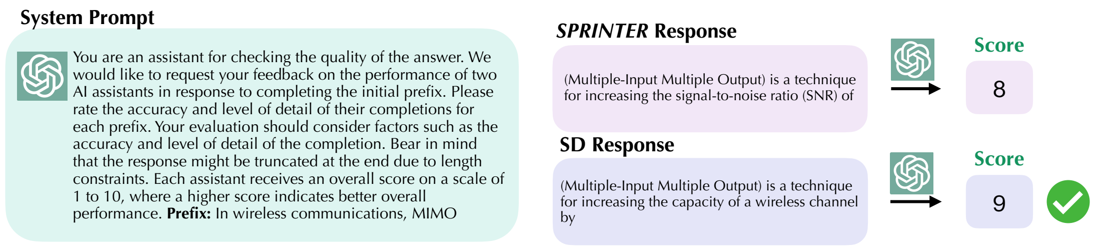
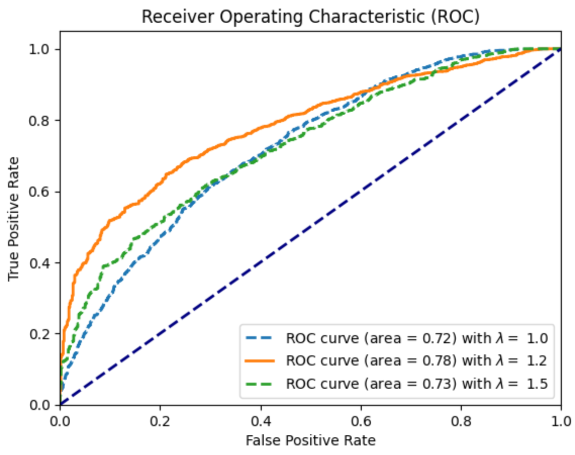

System Prompt
You are an assistant for checking the quality of the answer. We would like to request your feedback on the performance of two AI assistants in response to completing the initial prefix. Please rate the accuracy and level of detail of their completions for each prefix. Your evaluation should consider factors such as the accuracy and level of detail of the completion. Bear in mind that the response might be truncated at the end due to length constraints. Each assistant receives an overall score on a scale of 1 to 10, where a higher score indicates better overall performance. Prefix: In wireless communications, MIMO
SPRINTER Response
(Multiple-Input Multiple Output) is a technique for increasing the signal-to-noise ratio (SNR) of
Score
8
SD Response
(Multiple-Input Multiple Output) is a technique for increasing the capacity of a wireless channel by
Score
9

Figure 11. Illustrating win-tie rates evaluation process.

then prompted to evaluate the quality of completions based on accuracy and level of detail of the responses. We provide an example of using GPT-4 to evaluate two prompt responses pairs in Fig 11. Assume we have the prefix ”In wireless communications, MIMO ’ and feed this to both *SPRINTER* and SD. The completion from SD is “(Multiple-Input Multiple Output) is a technique for increasing the capacity of a wireless channel by ”. The completion from *SPRINTER* is “(Multiple-Input Multiple Output) is a technique for increasing the signal-to-noise ratio (SNR) of”. GPT-4 is given the prefix and the two completions and gives a score of 8 and 9 to *SPRINTER* and SD respectively, which means that SD is assigned the win for this prefix.

A.10. Additional Experimental Results

In this section, we present additional experimental results on the verifier’s hyperparameter tuning. Specifically, we show the ROC curve for the Wiki-Summary dataset using the GPT-Neo-125M / GPT-Neo-1.3B model pair as shown in Figure 12 and Table 5. Figure 13 shows the ROC Curve for the lm1b dataset. An interesting phenomenon emerges: as we increase the verifier decision threshold λ, the area under the curve improves, reaching its optimal performance at $\lambda = 1.2$. The optimal balance at $\lambda = 1.2$ suggests that this threshold best separates acceptable from unacceptable tokens.



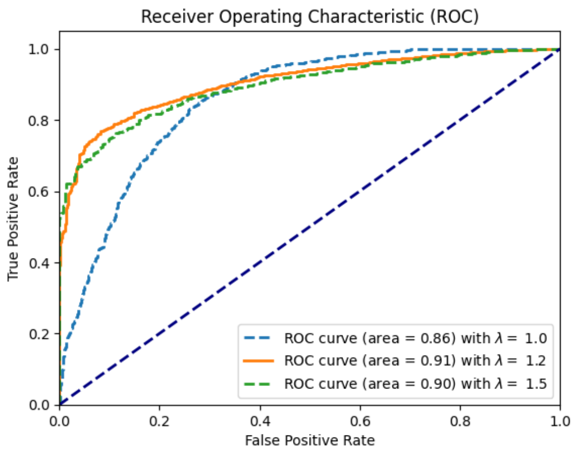


Figure 12. ROC curves on the Wiki-Summary dataset with varying acceptance threshold λ for the $q(.)/p(.)$ ratio during training. We used $\lambda = 1.2$ for generating latency and quality results with GPT2-S/XL and GPT-Neo draft/target pairs.

Figure 13. ROC curves on the LM1B dataset with varying acceptance threshold λ for the $q(.)/p(.)$ ratio during training. We used $\lambda = 1.2$ for generating latency and quality results with GPT2-S/XL and GPT-Neo draft/target pairs.

	Wiki-Summary	LM1B
GPT-Neo Model pair	(λ, τ) = (1.2,0.5)	(λ, τ) = (1.2,0.5)
GPT2 Model pair	(λ, τ) = (1.2,0.5)	(λ, τ) = (1.2,0.5)

Table 5. Comparison of hyperparameters across datasets and model pairs