SPIDE: Serial and Parallel Intertwined Speculative Decoding

Anonymous ACL submission

Abstract

Speculative Decoding (SD) reduces inference latency for Large Language Models (LLMs) by leveraging an efficient draft model to generate candidate tokens, which are subsequently verified by the target model. However, existing SD methods either bring limited accelertation or impose significant computational overhead. In this paper, we propose **S**erial and **P**arallel **I**ntertwined Speculative **DE**coding (SPIDE) — a novel training-free framework that orchestrates dynamic alternation combining serial dynamic drafting with parallel draft verification. We maintain a confidence-acceptance mapping table during the decoding process. In the serial dynamic drafting module, we leverage this table to evaluate the real-time reliability of the sequence and adjust draft lengths adaptively. In the parallel draft verification module, we resolve drafting-termination conflicts that compromise efficiency and update the mapping table synchronously. Experiments across diverse model pairs and text generation tasks demonstrate that SPIDE outperforms auto-regressive decoding and vanilla SD, accelerating token generation by up to **4.56**× and **2.12**×, respectively. Compared to strictly parallel SD frameworks, SPIDE achieves superior acceleration while reducing the additional usage cost of parallelism by **85.3**% on average.

1 Introduction

Transformer-based Large Language Models (LLMs) rely on autoregressive decoding techniques (Chu et al., 2023; Jiao et al., 2023), generating tokens sequentially that incur a substantial latency issue. Speculative decoding (SD) has been investigated to mitigate end-to-end LLM inference latency without compromising generation quality (Chen et al., 2023; Leviathan et al., 2023; Miao et al., 2023). This technique employs a draft model to propose multiple speculative tokens per decoding step, and later verifies all candidates concurrently via the target model. Since the draft model is much more efficient than the target model, SD significantly speeds up inference.

Early vanilla SD methods employ a static sequential strategy. They generate a fixed number (γ) of tokens per block and execute drafting/verification in strict sequence, creating significant idle time as depicted in Figure 1.(a). To address this inefficiency, dynamic drafting methods adaptively determine the draft length using confidence indicators (Mamou et al., 2024; Zhang et al., 2024b).

More recently, parallel SD techniques such as PEARL (Liu et al., 2025) optimize this workflow by introducing pre-verify/post-verify stages (Figure 1.(b)), achieving better acceleration than vanilla SD. However, parallel SD methods face two key challenges. **First**, when drafting is much faster than verification, acceleration is constrained by the need for large values (e.g., γ) to minimize waiting time. For example in Figure 1.(b), a 4th-token error surfaces only after processing two 8-token blocks. As a result, this delayed error detection wastes resources on invalid tokens. **Second**, strictly parallel execution forces continuous LLM engagement. In the Figure 1.(b), the target model remains actively engaged, verifying the first token of the current draft sequence or the tokens of the preceding draft sequence. Given that the target model is a large-scale LLM which is typically more expensive to invoke (Ong et al., 2024), this continuous engagement inevitably leads to high LLM usage costs.

To address these challenges, we propose an efficient, flexible, and training-free framework, named **S**erial and **P**arallel **I**ntertwined Speculative **DE**coding (SPIDE), with a *serial dynamic drafting module* and a *parallel draft verification module*. We divide the confidence scores of draft tokens into intervals, record the acceptance rate of draft tokens in each interval, and maintain a *confidence-acceptance rate mapping table* during the decoding

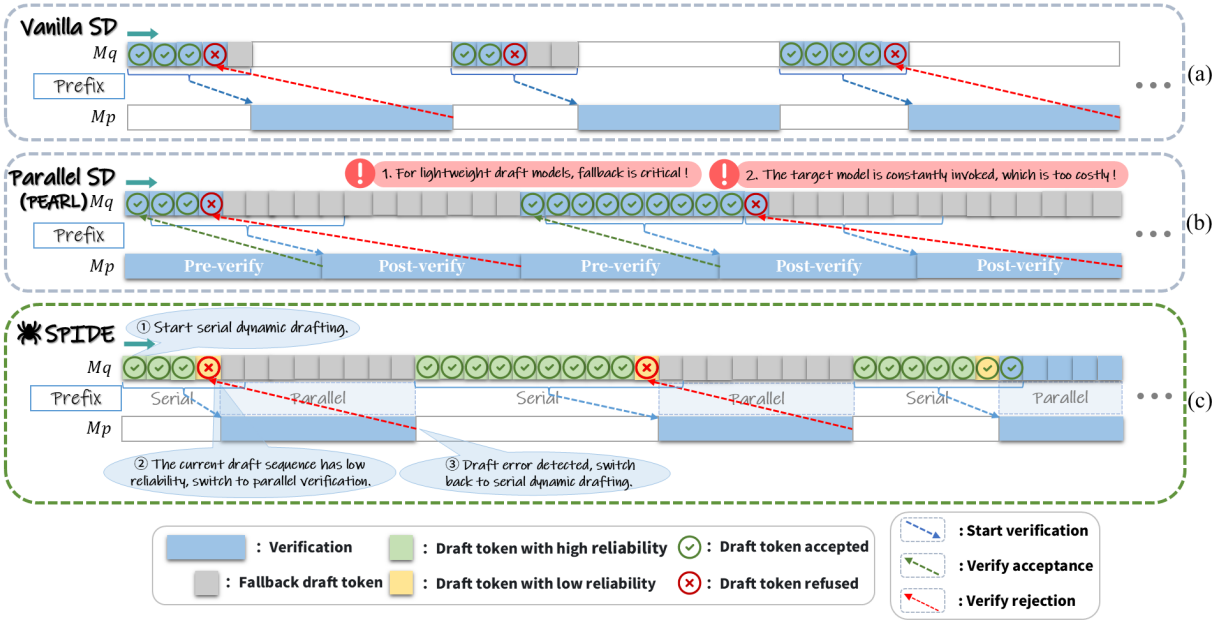

Figure 1: Comparison between vanilla SD, parallel SD, and our SPIDE (M_q and M_p stand for draft model and target model, respectively.): (a) Vanilla SD executes drafting and verification sequentially, often with notable delays. (b) Parallel SD method (i.e., PEARL (Liu et al., 2025)), adopts the "pre-verify" strategy to verify the first token of the current draft sequence, and the " post-verify" strategy to verify the previous draft sequence, such strictly parallel execution approach leads to high LLM usage costs due to frequent invocations of the target models. (c) Our proposed SPIDE dynamically orchestrates serial dynamic drafting tokens and parallel draft verification. This intertwined design achieves superior acceleration while reducing LLM usage costs.

process. In the *serial dynamic drafting module*, we use this mapping table to evaluate the reliability of the current draft sequence after each drafting step. When the reliability is lower than our expectations, we utilize the *parallel draft verification module* to mitigate the inefficiency from premature draft termination. The mapping table is iteratively updated after each verification. In summary, our contributions are as follows:

- We propose a novel training-free decoding framework that dynamically orchestrates serial dynamic drafting tokens and parallel draft verification. Unlike existing solutions with a strict parallelization protocol, SPIDE's intertwined design achieves superior acceleration while reducing LLM usage costs.
- We introduce a dynamic mapping table that aligns token-level confidence metrics with sequence-level acceptance rates. In contrast to dynamic drafting based directly on confidence, our approach provides more fine-grained adaptation and wider applicability to various tasks.
- We conduct comprehensive experiments evaluat SPIDE against baselines across multiple model pairs and various text generation tasks. The results show that SPIDE outperforms baselines in inference speed while reducing computation costs.

2 Related Works

LLM Inference Acceleration. Accelerating LLM inference has been a trending topic. Key solutions include: (1) Efficient structure design (Chitty-Venkata and Somani, 2022), leveraging techniques such as Mixture-of-Experts (MoE) (Shazeer et al., 2017) and low-rank decomposition (Kiruthika et al., 2025) to reduce computational complexity and memory footprint; (2) Model compression (Wang et al., 2024), applying quantization (Lang et al., 2024), pruning (Liu et al., 2018) and knowledge distillation (Hinton et al., 2015) to reduce model size and computational requirements while preserving accuracy; and (3) Inference engine optimization (Park et al., 2025), dedicated to accelerating LLMs' forward pass execution.

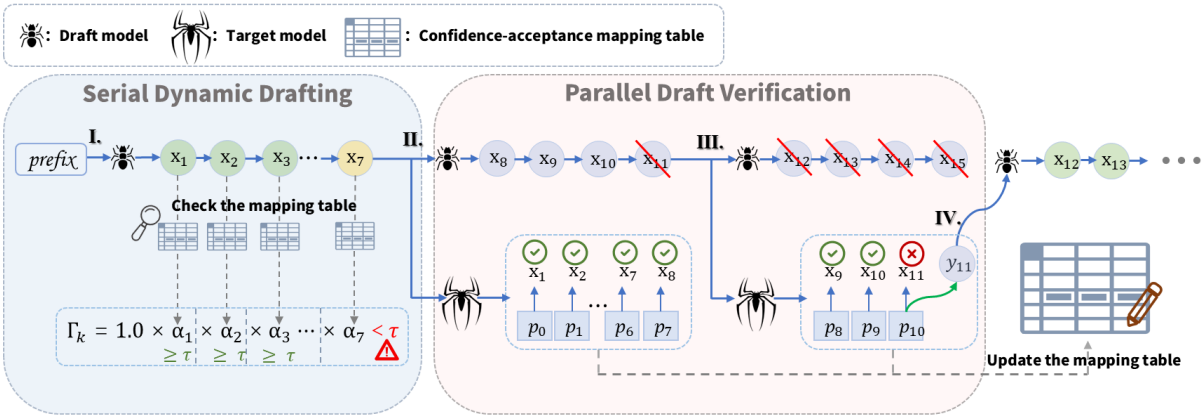

Figure 2: The overall design of SPIDE: I. After inputting the prefix, M_q dynamically drafts $[x_1, x_2, \ldots, x_7]$ sequentially.. Each time x_i is generated, we check the mapping table to obtain the corresponding α_i, and use it to update Γ_k. II. After drafting x_7, the sequence reliability is lower than expectation, we switch to parallel draft verification, M_q continues to draft $[x_8, x_9, x_{10}, x_{11}]$ and M_p verifies $[x_1, x_2, \ldots, x_8]$. III. $[x_1, x_2, \ldots, x_8]$ is fully accepted by M_p, we update the mapping table and continue with the parallel draft verification. M_q then drafts $[x_{12}, x_{13}, x_{14}, x_{15}]$ and M_p verifies $[x_9, x_{10}, x_{11}, x_{12}]$. IV. x_{11} is rejected by M_p, we update the mapping table and correct sampling y_{11}, then switch back to serial dynamic drafting module.

Speculative Decoding. Building on the success of SD in reducing LLM inference latency (Chen et al., 2023; Leviathan et al., 2023; Spector and Re, 2023; Zhao et al., 2024b), substantial efforts have been put into optimizing SD implementation: Proposed optimizations for the draft model include: (1) Multi-candidate batch processing (Yang et al., 2024); (2) KV-cache reuse to enhance draft accuracy (Du et al., 2024); (3) Extending autoregressive decoding to multi-branch drafting (Gao et al., 2025); (4) Employing lightweight variants of the target model itself as the draft model, including quantization (Zhao et al., 2024a), early exit (Elhoushi et al., 2024), and forward filling (Monea et al., 2023); (5) Model augmentation such as adding auxiliary decoding heads (e.g., Medusa (Cai et al., 2024)) or inserting feature-level transformer decoders (e.g., Eagle (Li et al., 2024)).

Further enhancements target the dynamics of the inference process. This line of works is the most relevant to our SPIDE. Methods such as SVIP (Zhang et al., 2024b) adapt the draft length based on token entropy, while PEARL (Liu et al., 2025) achieves adaptive length and reduces pipeline stalls by parallelizing drafting and verification. Draft & Verify (Zhang et al., 2023) proposes an adaptive draft-exiting mechanism based on dynamic confidence threshold. SpecDec++ (Huang et al., 2024) and DISCO (Mamou et al., 2024) respectively train an acceptance prediction head and a basic classifier to achieve dynamic drafting. A comprehensive review over the works on enhancing SD can be found from (Xia et al., 2024). To the best of our knowledge, our SPIDE first proposes the design of decoding framework for serial parallel interleaving structure, which avoids excessive extra computing resource consumption while achieving better efficiency.

3 Method

3.1 Preliminaries

Speculative Decoding. SD is an emerging inference paradigm designed to reduce the high latency of autoregressive decoding in LLMs. At each step, SD efficiently generates multiple draft tokens that are subsequently verified in parallel. Specifically, given a prefix $\mathbf{x}$, SD starts by invoking a draft model M_q to sample a draft sequence of tokens with length γ, denoted as $\tilde{x}_1, \cdots, \tilde{x}_\gamma$, where $\tilde{x}_i \sim q(x|\tilde{x}_1, \cdots, \tilde{x}_{i-1}, \mathbf{x})$. The draft tokens, along with the prefix, are then sent to the target model M_p to obtain their output distribution $p(x|\tilde{x}_1, \cdots, \tilde{x}_i, \mathbf{x})$ in parallel. Finally, the draft tokens are verified sequentially from $\tilde{x}_1$ to $\tilde{x}_\gamma$. To verify token $\tilde{x}_i$, a speculative sampling algorithm is employed to determine whether to accept $\tilde{x}_i$ or not, based on $q(x|\tilde{x}, \cdots, \tilde{x}_{i-1}, \mathbf{x})$ and $p(x|\tilde{x}, \cdots, \tilde{x}_{i-1}, \mathbf{x})$. Once a token is rejected, the next verification terminates and the algorithm returns a new token as the endpoint. If all tokens are

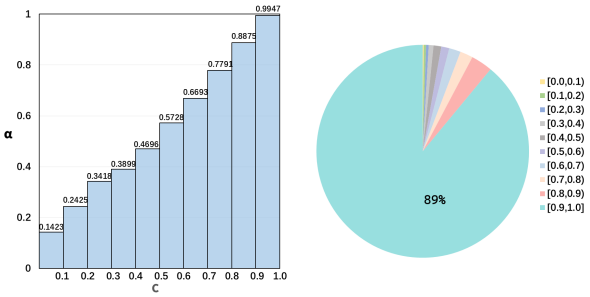

Figure 3: The relationship between the confidence intervals and acceptance rates of draft tokens (left), and the proportion of draft tokens in each confidence interval during the decoding process (right).

Statistics are updated in real time during the decoding process.

Confidence	Total Number (N_T)	Accepted Number (N_A)	Acceptance Rate (N_A/N_T)
[0 , 0.1)	38	15	0.3947
[0.1 , 0.2)	195	66	0.3385
⋮	⋮	⋮	⋮
[0.8 , 0.9)	2365	2099	0.8875
[0.90 , 0.91)	229	213	0.9301
[0.91 , 0.92)	436	413	0.9472
⋮	⋮	⋮	⋮
[0.99 , 1.0)	8270	8217	0.9936
[1.0]	55759	55729	0.9995

Figure 4: A snapshot of our proposed confidence-acceptance mapping table maintained in real time by Deepseek 1.3B & 33B during the decoding process.

accepted, there is an additional token sampled from $p(x|\tilde{x}_1, \cdots, \tilde{x}_\gamma, \mathbf{x})$ as the endpoint.

Speculative Sampling. The importance of speculative sampling lies in its guarantee of distributional equivalence: unconditional acceptance of draft tokens would violate the preservation of the output distribution relative to the target model M_p. Speculative sampling employs an algorithm that accepts $\tilde{x}$ with a probability of $\min(1, \frac{p(\tilde{x})}{q(\tilde{x})})$, resulting in an overall acceptance rate of $\sum_{\tilde{x}} \min(p(\tilde{x}), q(\tilde{x}))$. In the event that $\tilde{x}$ is rejected, a new token is sampled from the distribution defined by:

$$p'(x) = \frac{\max(0, p(x) - q(x))}{\sum_x \max(0, p(x) - q(x))}. \quad (1)$$

It has been shown that speculative sampling preserves the output distribution consistent with the target model (Leviathan et al., 2023; Chen et al., 2023).

3.2 SPIDE

The framework of SPIDE is shown in Figure 2. It consists of two main modules: serial dynamic drafting and parallel draft verification. In addition, we design a confidence acceptance mapping table that can be updated in real time during the decoding process. The serial dynamic drafting module uses the mapping table to dynamically adjust the draft length, and the parallel draft verification module updates the mapping table immediately after each verification. Below, we will first the design of the confidence-acceptance mapping table, and then explain the modules in SPIDE. We include the full algorithm of SPIDE in Appendix A.

Confidence-Acceptance Mapping Table. We maintain a confidence-acceptance mapping table during the decoding process, with the objective of aligning the confidence level with the acceptance rate in real time. This approach allows us to adapt effectively to various generation tasks and pairs of drafting and verification models. In the SPIDE framework, the raw probability is used as the confidence of the draft token generated by M_q, which is calculated based on M_q predicting the maximum probability of the next token x_{t+1} given the previous context $x_{<t+1}$, and is represented as:

$$C = \max\left(p(x_{t+1} \mid x_{<t+1})\right), \quad (2)$$

where C is the confidence, $p(x_{t+1} \mid x_{<t+1})$ is the probability distribution of predicting the next token x_{t+1} given the previous context $x_{<t+1}$.

The acceptance rate α reflects the proportion of tokens generated by M_q that are accepted by M_p. We divide C into ten intervals and record the proportion and α of draft tokens in each interval for different model pairs. The result of Deepseek-1.3B & 33B is shown in Figure 3. We observe that C and α have a strong positive correlation. Moreover, draft tokens with C in the range of $(0.9, 1.0]$ are the vast majority, with an extremely high acceptance rate of almost 100%. Similar situations can be observed in other model pairs, which are included in Appendix B. Taking advantage of this positive correlations, it is intuitive to consider C of the current draft token as the probability that the token will be accepted after verification. We take a step further and propose a more fine-grained strategy based on real-time statistics to align confidence and acceptance probability. Specifically, as shown in Figure 4, we maintain a confidence-acceptance mapping table containing 20 entries, covering the confidence intervals of $[0, 0.1)$, $[0.1, 0.2)$, ... , $[0.8, 0.9)$, $[0.9, 0.91)$, $[0.91, 0.92)$, ... , $[0.99, 1.0)$ and 1.0. During the decoding process, the total number of

draft tokens and the number of accepted tokens within each interval will be counted in real time, and α of draft tokens within each interval will be updated accordingly.

In this simple way, we continuously track and record the acceptance rates of draft tokens in each interval. This mechanism makes the prediction of the probability of the current draft token being accepted no longer rely on the subjective judgment of M_q, via combiing the actual real-time feedback information of M_p. This alleviates the deviation between the confidence of the draft tokens and the actual acceptance probability, so as to align the two in a more explainable and fine-grained manner. Moreover, this strategy comes naturally with adaptive learning capabilities, and can be automatically adjusted according to actual conditions, both in different model pairs and in different tasks or contexts. Compared with SpecDec++ (Huang et al., 2024) for training an acceptance prediction head or DISCO (Mamou et al., 2024) for training a simple classifier , our method does not require an additional training process and only needs to maintain three arrays (total number, accepted number, and acceptance rate). The resource and time consumption brought about are almost negligible.

Serial Dynamic Drafting Module.

This module aims to achieve better dynamic control of draft length by applying the above mapping table. In serial dynamic drafting, we use M_q to generate draft tokens sequentially. After each token is drafted, we determine the corresponding acceptance rate from the current confidence-acceptance mapping table based on the confidence score of that token. This acceptance rate is then utilized as the probability of being accepted upon future verification. At the same time, a reliability value is updated, which represents the overall reliability degree of the current draft sequence:

$$\Gamma_k = \prod_{i=1}^{k} \alpha_{b_i}, \tag{3}$$

where b_i represents the confidence interval to which the i-th token belongs, α represents the acceptance rate of the corresponding interval, k represents the length of the current draft sequence.

In serial dynamic drafting, we decide the next action based on whether the overall reliability Γ_k of the current draft sequence is higher than our expectation (a hyperparameter τ). This logic can be formalized as the following function:

$$\text{DraftAction}(\Gamma_k, \tau) = \begin{cases} 1, & \Gamma_k > \tau, \\ 0, & else, \end{cases} \tag{4}$$

where 1 indicates the continuation of serial dynamic drafting, using M_q to generate the next draft token, and 0 indicates the switch to parallel draft verification, activating M_p to verify the current draft sequence. As shown in Figure 2, after inputting prefix, the serial dynamic drafting begins. M_q autoregressively drafts the sequence $[x_1, x_2, \ldots, x_7]$, and after each generation of x_i $(0 < i \leq 7)$, we will check the acceptance rate α_{b_t} corresponding to x_i in the confidence-acceptance mapping table, and then update Γ_k (the initial value is set to 1.0) with α_{b_t}. After x_7 is generated, Γ_7 is less than τ. Therefore, M_p is activated at this point to switch to parallel draft verification.

We understand the update mechanism of Γ_k as an online Bayesian belief update process:

$$P(H_0|x_{1:k}) \propto \left(\prod_{i=1}^{k} \alpha_{b_i}\right) \cdot \pi_0, \tag{5}$$

assuming H_0 means "the current draft sequence is overall reasonable". We set the prior $P(H_0) = \pi_0$ (in the specific implementation, we set it to 1.0), and continuously update the posterior through the draft token generated at each step. When Γ_k is lower than preset threshold τ, we believe that the overall reliability of the current draft sequence is lower than our expectation, the risk of rejection is high, and it's time to switch to parallel draft verification.

Parallel Draft Verification Module.

The parallel draft verification module contains two lines of work that run in parallel: verification of the previous draft sequence and continued drafting based on the previous draft sequence. It aims to simultaneously mitigate the degradation of efficiency caused by severe rollback and premature stopping of drafting.

As shown in Figure 2, after switching to parallel draft verification, M_q continues to draft $[x_8, x_9, x_{10}, x_{11}]$ based on the assumption that $[x_1, x_2, \ldots, x_7]$ will eventually be fully accepted by M_p. Meanwhile, the previously idle M_p receives $prefix + [x_1, x_2, \ldots, x_7]$ and performs a forward propagation. Using the output logits $[p_0, p_1, \ldots, x_7]$, we verifies $[x_1, x_2, \ldots, x_7, x_8]$

	Vicuna 68M & 13B			Vicuna 160M & 33B			Pythia 160M & 12B			DeepSeek 1.3B & 33B			TinyLlama 1.1B & CodeLlama 34B		
Methods	Speed	Cost	MAT	Speed	Cost	MAT	Speed	Cost	MAT	Speed	Cost	MAT	Speed	Cost	MAT
AD	1.00×	–	–	1.00×	–	–	1.00×	–	–	1.00×	–	–	1.00×	–	–
SD	1.76×	1.00×	1.91	1.97×	1.00×	2.29	1.56×	1.00×	3.30	2.79×	1.00×	8.25	2.33×	1.00×	5.65
SVIP	1.86×	1.02×	1.84	2.10×	0.94×	2.43	1.70×	0.90×	3.82	3.04×	0.92×	9.10	2.77×	**0.64×**	9.57
D&V	1.88×	1.03×	1.86	2.22×	0.97×	2.55	1.88×	0.96×	3.69	3.16×	**0.90×**	9.44	2.89×	0.84×	9.44
PEARL	1.99×	1.27×	1.93	2.42×	1.30×	2.86	2.65×	1.35×	6.44	4.41×	1.77×	23.26	3.65×	1.58×	14.68
SPIDE	**2.38×**	**0.95×**	**2.54**	**2.66×**	**0.91×**	**3.52**	**2.83×**	**0.84×**	**7.89**	**4.52×**	1.26×	**24.83**	**3.86×**	1.20×	**14.94**

Table 1: Results on the HumanEval dataset, a code generation task. We bold **the best results** for each model combination and annotate the verification cost of two parallel methods (i.e., our SPIDE and PEARL) in gray. MAT stands for mean accepted tokens. "–" indicates that this indicator cannot be evaluated since AD does not involve drafting and verification.

	Vicuna 68M & 13B			Vicuna 160M & 33B			Pythia 160M & 12B			DeepSeek 1.3B & 33B			TinyLlama 1.1B & CodeLlama 34B		
Methods	Speed	Cost	MAT	Speed	Cost	MAT	Speed	Cost	MAT	Speed	Cost	MAT	Speed	Cost	MAT
AD	1.00×	–	–	1.00×	–	–	1.00×	–	–	1.00×	–	–	1.00×	–	–
SD	1.64×	**1.00×**	2.08	2.27×	1.00×	4.26	1.35×	1.00×	2.79	2.30×	**1.00×**	6.59	2.52×	1.00×	7.04
SVIP	1.65×	1.05×	1.95	2.57×	**0.99×**	4.43	1.42×	0.87×	3.39	2.56×	1.03×	6.50	2.91×	**0.53×**	14.52
D&V	1.60×	1.09×	1.88	2.44×	1.07×	4.10	1.58×	0.91×	3.78	2.61×	1.09×	6.81	2.76×	0.86×	12.89
PEARL	2.01×	1.33×	2.13	3.22×	1.47×	5.41	2.62×	1.23×	5.90	3.77×	1.68×	14.05	4.45×	1.57×	42.57
SPIDE	**2.41×**	1.02×	**2.98**	**3.76×**	1.09×	**6.89**	**2.86×**	**0.75×**	**7.65**	**3.99×**	1.25×	**15.11**	**4.56×**	1.40×	**44.96**

Table 2: Results on the GSM8K dataset, a arithmetic reasoning task (same result presentation style as in Table 1).

in sequence. Fortunately, our assumption is correct, and $[x_1, x_2, \ldots, x_7]$ is fully accepted by M_p. Therefore, we continue a round of parallel draft verification using the same logic. M_q drafts $[x_{12}, x_{13}, x_{14}, x_{15}]$, and M_p verifies $[x_9, x_{10}, x_{11}, x_{12}]$. Unfortunately, this time x_{11} is rejected, so $[x_{11}, x_{12} \ldots, x_{15}]$ is discarded. After correcting sampling y_{11}, based on the sequence $prefix + [x_1, x_2, \ldots, x_{10}, y_{11}]$, we switch back to serial dynamic drafting. In addition, after these two verifications, we update the confidence-acceptance mapping table in real time based on the actual verification results.

Moreover, in order to minimize the waiting time between M_p and M_q in parallel draft verification (that is, when M_p finishes verifying the previous batch of drafts, M_q also finished drafting the new s draft tokens), we set:

$$s = \left\lfloor \frac{v_d}{v_t} \right\rfloor, \tag{6}$$

where v_d is the running speed of the draft model; v_t is the running speed of the target model.

4 Experiments

4.1 Experimental Setup

Models. We use five pairs of models to evaluate the effectiveness of SPIDE, including Vicuna-68M & 13B, Vicuna-160M & 33B (Chiang et al., 2023), Pythia-160M & 12B (Biderman et al., 2023), TinyLlama-1.1B & CodeLlama-34B (Zhang et al., 2024a; Roziere et al., 2023) and Deepseek-1.3B & 33B (Guo et al., 2024) All models are run on a single-node server with four 3090 Gpus. We provide more model configurations in Appendix C.1.

Datasets. We conduct experiments on four text generation tasks to evaluate the effectiveness of our SPIDE, including HumanEval (code generation task) (Chen et al., 2021), GSM8K (multilingual arithmetic reasoning task) (Cobbe et al., 2021), MGSM (the multilingual translation of GSM8K) (Shi et al., 2022) and MT-bench (multi-round dialogue task) (Zheng et al., 2023). These are representative benchmarks for evaluation. The experiments default to Greedy sampling. We provide more dataset configurations in Appendix C.2. All models and datasets used in this work are publicly

Methods	Bengali	German	English	Spanish	French	Japanese	Russian	Swahili	Tegulu	Thai	Chinese	Avg.
AD	1.00×	1.00×	1.00×	1.00×	1.00×	1.00×	1.00×	1.00×	1.00×	1.00×	1.00×	1.00×
SD	2.40×	1.89×	1.55×	1.65×	1.66×	1.51×	1.50×	2.17×	1.74×	1.77×	1.66×	1.77×
SVIP	2.16×	2.14×	1.69×	1.56×	1.77×	1.57×	1.56×	2.49×	1.64×	1.79×	1.67×	1.82×
D&V	2.26×	2.33×	1.65×	1.76×	1.67×	1.45×	1.87×	2.56×	1.67×	1.67×	1.56×	1.76×
PEARL	3.06×	2.86×	1.75×	1.87×	2.14×	1.58×	2.43×	3.95×	1.80×	1.97×	1.88×	2.30×
SPIDE	**3.94×**	**3.02×**	**2.40×**	**2.12×**	**2.41×**	**1.91×**	**2.92×**	**4.02×**	**2.37×**	**3.28×**	**2.23×**	**2.79×**

Table 3: Results of Vicuna 68M & 13B on MGSM dataset, a multilingual arithmetic reasoning task. We bold **the best results** of each model combination.

Methods	Writing	Roleplay	Reasoning	Math	Coding	Extraction	Stem	Humanities	Avg.
AD	1.00×	1.00×	1.00×	1.00×	1.00×	1.00×	1.00×	1.00×	1.00×
SD	1.55×	1.41×	1.58×	1.70×	1.60×	1.32×	1.47×	1.53×	1.52×
SVIP	1.57×	1.49×	1.71×	1.57×	1.50×	1.31×	1.44×	1.53×	1.51×
D&V	1.68×	1.45×	1.76×	1.51×	1.45×	1.40×	1.48×	1.52×	1.65×
PEARL	1.73×	1.72×	2.38×	2.38×	1.89×	1.55×	1.53×	1.56×	1.84×
SPIDE	**2.06×**	**2.13×**	**2.58×**	**2.54×**	**2.20×**	**1.89×**	**1.98×**	**1.97×**	**2.17×**

Table 4: Results of Vicuna 68M & 13B on MT-bench dataset, a multi-round dialogue task. We bold **the best results** of each model combination.

available under permissive open-source licenses that permit research use and redistribution.

Baseline Methods.

We implement four training-free inference acceleration methods as our baselines. (i) **Autoregressive Decoding (AD) :** the standard paradigm for LLMs text generation. (ii) **Speculate decoding (SD) :** the vanilla SD approach relies on a draft model to draft future tokens and then verify them in parallel (Leviathan et al., 2023; Chen et al., 2023). (iii) **SVIP:** the length of the draft sequence is adaptively determined based on the entropy of each draft token distribution (Zhang et al., 2024b). (iv) **D&V(Draft&Verify):** dynamically adjust the draft length based on the historical acceptance ra0te of draft tokens. (v) **PEARL:** the drafting phase and the verification phase are carried out in parallel, thus alleviating the problem of waiting for each other (Liu et al., 2025).

Evaluation Metrics. We use three metrics to comprehensively evaluate and compare SPIDE and other baseline methods: (i) **Walltime speedup ratio (Speed) :** used to evaluate the speedup of each method relative to autoregressive decoding; (ii) **Verification overhead ratio (Cost) :** the ratio of the number of calls to the target model under the same task, used to evaluate the cost comparison of each method relative to vanilla SD; (iii) **Mean accepted tokens (MAT) :** the average number of tokens generated by the draft model that are successfully accepted by the target model during the decoding process. This metric directly affects the efficiency speedup of decoding. We provide more additional evaluation details in Appendix C.3.

4.2 Main Results

We conducted extensive experiments on the above benchmarks. Taking the speedup ratio as the primary metric, we present the optimal results of each baseline method and our SPIDE. As shown in Table 1 and Table 2, SPIDE outperforms vanill SD, SVIP, and PEARL in terms of speedup and MAT on all model pairs in the HumanEval and GSM8K datasets. SPIDE can achieve up to 4.56× and 2.12× acceleration compared to the autoregressive decoding and vanilla SD, respectively. It is important to highlight that, Compared with vanilla SD, SPIDE only increases the verification cost by 6.7% on average while bringing significantly superior speedup, while PEARL increases the verification cost by 45.5% on average. Compared with PEARL, SPIDE reduces the additional overhead caused by the parallelism of the framework by 85.3% on average. In addition, as shown in Table 3 and Table 4, SPIDE also outperforms baseline methods and achieves

Methods	HumanEval					GSM8K	
	Vicuna 68M & 13B	Vicuna 160M & 33B	Pythia 160M & 12B	DeepSeek 1.3B & 33B	TinyLlama 1.1B & CodeLlama 34B	Pythia 160M & 12B	DeepSeek 1.3B & 33B
SPIDE	2.38×	2.66×	2.83×	4.52×	3.86×	2.86×	3.99×
SPIDE *w/o dyna-draft*	2.15×	2.36×	2.48×	3.83×	3.16×	2.45×	3.29×
SPIDE *w/o para-verify*	2.05×	2.27×	1.96×	3.41×	3.07×	1.63×	2.92×
SD	1.76×	1.97×	1.56×	2.79×	2.33×	1.35×	2.30×
SVIP	1.99×	2.10×	1.70×	3.04×	2.77×	1.42×	2.56×
D&V	1.88×	2.22×	1.88×	3.16×	2.89×	1.42×	2.56×

Table 5: Ablation results of SPIDE on HumanEval and GSM8K datasets. We annotate the comparison of the three serial architectures in gray.

significant inference speedups on 12 multilingual arithmetic tasks and 8 multi-turn dialogue tasks. These results demonstrate the outstanding effectiveness of SPIDE in inference acceleration and cost control. In addition, (1) we provide complete experimental results on MGSM and MT-bench respectively in Appendix D.1 and D.2, respectively. (2) We also investigate the sensitivity of SPIDE to τ, the results and analysis are provided in the Appendix D.3. (3) The actual invocation of the target model on the HumanEval and GSM8K datasets is provided in Appendix D.6.

4.3 Ablation Study

To verify the effectiveness of SPIDE's serial dynamic drafting strategy and parallel draft verification module, we conduct the ablation study. We denote as SPIDE *w/o dyna-draft* that no longer dynamically adjust the length when serial drafting (use the optimal γ of vanilla SD, which is provided in the Appendix D.4), and as SPIDE *w/o para-verify* that no longer use parallel verification. We present the main experimental results for ablation in Table 5. It is evident that the efficiency of both SPIDE *w/o dyna-draft* and SPIDE *w/o para-verify* has decreased compared to the complete SPIDE. Intuitively, SPIDE *w/o dyna-draft* may cause more fallbacks due to static drafting, and SPIDE *w/o para-verify* is prone to wasting time for stopping drafting too early. Moreover, among serial architectures, the acceleration effect of SPIDE *w/o para-verify* is sill superior to that of vanilla SD and SVIP. This further proves the effectiveness and superiority of the method we proposed to dynamically control the draft length based on the mapping mechanism. In addition, we also investigate the sensitivity of SPIDE *w/o para-verify* to τ, the results and analysis are provided in the Appendix D.5.

5 Conclusion

We introduced a speculative decoding framework named SPIDE, which alternates between serial dynamic drafting and parallel draft verification. Experimental results demonstrate that SPIDE outperforms baseline methods in terms of acceleration and reduction of computational costs, which indicates that SPIDE is a more efficient and resource-friendly inference acceleration framework. In the future, we intend to integrate SPIDE with existing accelerated inference techniques to explore more efficient and resource-friendly approaches for LLM acceleration, and we will explore SD acceleration for content generation in other modalities besides text.

6 Limitation

Diverse tasks expansion. Our experiments mainly focus on text generation tasks using several model pairs commonly used in the field of speculative decoding. Although SPIDE can in principle be applied to other models or generation tasks, additional factors such as domain-specific tokenization or specialized text structure may impact overall inference speed. Future work is needed to evaluate the generality of SPIDE across more diverse generation tasks (e.g., images, videos, etc.), as well as to investigate whether the acceleration of multimodal content preserves lossless properties.

Cost of Draft Model. We primarily focus on the invocation of the target model as a critical factor in cost control within the speculative decoding framework. Although the draft model is typically a lightweight language model that incurs minimal invocation costs, it still contributes to certain usage expenses during the decoding process.

References

Stella Biderman, Hailey Schoelkopf, Quentin Gregory Anthony, Herbie Bradley, Kyle O'Brien, Eric Hallahan, Mohammad Aflah Khan, Shivanshu Purohit, USVSN Sai Prashanth, Edward Raff, and 1 others. 2023. Pythia: A suite for analyzing large language models across training and scaling. In *International Conference on Machine Learning*, pages 2397–2430. PMLR.

Tianle Cai, Yuhong Li, Zhengyang Geng, Hongwu Peng, Jason D Lee, Deming Chen, and Tri Dao. 2024. Medusa: Simple llm inference acceleration framework with multiple decoding heads. *arXiv preprint arXiv:2401.10774*.

Charlie Chen, Sebastian Borgeaud, Geoffrey Irving, Jean-Baptiste Lespiau, Laurent Sifre, and John Jumper. 2023. Accelerating large language model decoding with speculative sampling. *arXiv preprint arXiv:2302.01318*.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde De Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, and 1 others. 2021. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*.

Wei-Lin Chiang, Zhuohan Li, Ziqing Lin, Ying Sheng, Zhanghao Wu, Hao Zhang, Lianmin Zheng, Siyuan Zhuang, Yonghao Zhuang, Joseph E Gonzalez, and 1 others. 2023. Vicuna: An open-source chatbot impressing gpt-4 with 90%* chatgpt quality. *See https://vicuna. lmsys. org (accessed 14 April 2023)*, 2(3):6.

Krishna Teja Chitty-Venkata and Arun K Somani. 2022. Neural architecture search survey: A hardware perspective. *ACM Computing Surveys*, 55(4):1–36.

Zhixuan Chu, Hongyan Hao, Xin Ouyang, Simeng Wang, Yan Wang, Yue Shen, Jinjie Gu, Qing Cui, Longfei Li, Siqiao Xue, and 1 others. 2023. Leveraging large language models for pre-trained recommender systems. *arXiv preprint arXiv:2308.10837*.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, and 1 others. 2021. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*.

Cunxiao Du, Jing Jiang, Xu Yuanchen, Jiawei Wu, Sicheng Yu, Yongqi Li, Shenggui Li, Kai Xu, Liqiang Nie, Zhaopeng Tu, and 1 others. 2024. Glide with a cape: A low-hassle method to accelerate speculative decoding. *arXiv preprint arXiv:2402.02082*.

Mostafa Elhoushi, Akshat Shrivastava, Diana Liskovich, Basil Hosmer, Bram Wasti, Liangzhen Lai, Anas Mahmoud, Bilge Acun, Saurabh Agarwal, Ahmed Roman, and 1 others. 2024. Layerskip: Enabling early exit inference and self-speculative decoding. *arXiv preprint arXiv:2404.16710*.

Zipeng Gao, Qingrong Xia, Tong Xu, Xinyu Duan, Zhi Zheng, Zhefeng Wang, and Enhong Chen. 2025. Multi-branch self-drafting for llm inference acceleration. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 39, pages 23942–23950.

Daya Guo, Qihao Zhu, Dejian Yang, Zhenda Xie, Kai Dong, Wentao Zhang, Guanting Chen, Xiao Bi, Yu Wu, YK Li, and 1 others. 2024. Deepseek-coder: When the large language model meets programming–the rise of code intelligence. *arXiv preprint arXiv:2401.14196*.

Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. 2015. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*.

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

Wenxiang Jiao, Wenxuan Wang, Jen-tse Huang, Xing Wang, Shuming Shi, and Zhaopeng Tu. 2023. Is chatgpt a good translator? yes with gpt-4 as the engine. *arXiv preprint arXiv:2301.08745*.

M Kiruthika, Vibhav Krashan Chaurasiya, Balaraman Sundarambal, N Kirubakaran, and 1 others. 2025. A comprehensive review of low-rank decomposition techniques for parameter efficient fine-tuning. In *2025 International Conference on Data Science, Agents & Artificial Intelligence (ICDSAAI)*, pages 1–6. IEEE.

Jiedong Lang, Zhehao Guo, and Shuyu Huang. 2024. A comprehensive study on quantization techniques for large language models. In *2024 4th International Conference on Artificial Intelligence, Robotics, and Communication (ICAIRC)*, pages 224–231. IEEE.

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

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

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

Zhuang Liu, Mingjie Sun, Tinghui Zhou, Gao Huang, and Trevor Darrell. 2018. Rethinking the value of network pruning. *arXiv preprint arXiv:1810.05270*.

Jonathan Mamou, Oren Pereg, Daniel Korat, Moshe Berchansky, Nadav Timor, Moshe Wasserblat, and Roy Schwartz. 2024. Dynamic speculation lookahead accelerates speculative decoding of large language models. *arXiv preprint arXiv:2405.04304*.

Xupeng Miao, Gabriele Oliaro, Zhihao Zhang, Xinhao Cheng, Zeyu Wang, Rae Ying Yee Wong, Zhuoming Chen, Daiyaan Arfeen, Reyna Abhyankar, and Zhihao Jia. 2023. Specinfer: Accelerating generative llm serving with speculative inference and token tree verification. *arXiv preprint arXiv:2305.09781*, 1(2):4.

Giovanni Monea, Armand Joulin, and Edouard Grave. 2023. Pass: Parallel speculative sampling. *arXiv preprint arXiv:2311.13581*.

Isaac Ong, Amjad Almahairi, Vincent Wu, Wei-Lin Chiang, Tianhao Wu, Joseph E Gonzalez, M Waleed Kadous, and Ion Stoica. 2024. Routellm: Learning to route llms with preference data. *arXiv preprint arXiv:2406.18665*.

Sihyeong Park, Sungryeol Jeon, Chaelyn Lee, Seokhun Jeon, Byung-Soo Kim, and Jemin Lee. 2025. A survey on inference engines for large language models: Perspectives on optimization and efficiency. *arXiv preprint arXiv:2505.01658*.

Baptiste Roziere, Jonas Gehring, Fabian Gloeckle, Sten Sootla, Itai Gat, Xiaoqing Ellen Tan, Yossi Adi, Jingyu Liu, Romain Sauvestre, Tal Remez, and 1 others. 2023. Code llama: Open foundation models for code. *arXiv preprint arXiv:2308.12950*.

Noam Shazeer, Azalia Mirhoseini, Krzysztof Maziarz, Andy Davis, Quoc Le, Geoffrey Hinton, and Jeff Dean. 2017. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. *arXiv preprint arXiv:1701.06538*.

Freda Shi, Mirac Suzgun, Markus Freitag, Xuezhi Wang, Suraj Srivats, Soroush Vosoughi, Hyung Won Chung, Yi Tay, Sebastian Ruder, Denny Zhou, and 1 others. 2022. Language models are multilingual chain-of-thought reasoners. *arXiv preprint arXiv:2210.03057*.

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

Wenxiao Wang, Wei Chen, Yicong Luo, Yongliu Long, Zhengkai Lin, Liye Zhang, Binbin Lin, Deng Cai, and Xiaofei He. 2024. Model compression and efficient inference for large language models: A survey. *arXiv preprint arXiv:2402.09748*.

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. *arXiv preprint arXiv:2401.07851*.

Sen Yang, Shujian Huang, Xinyu Dai, and Jiajun Chen. 2024. Multi-candidate speculative decoding. *arXiv preprint arXiv:2401.06706*.

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. *arXiv preprint arXiv:2309.08168*.

Peiyuan Zhang, Guangtao Zeng, Tianduo Wang, and Wei Lu. 2024a. Tinyllama: An open-source small language model. *arXiv preprint arXiv:2401.02385*.

Ziyin Zhang, Jiahao Xu, Tian Liang, Xingyu Chen, Zhiwei He, Rui Wang, and Zhaopeng Tu. 2024b. Draft model knows when to stop: A self-verification length policy for speculative decoding. *arXiv preprint arXiv:2411.18462*.

Juntao Zhao, Wenhao Lu, Sheng Wang, Lingpeng Kong, and Chuan Wu. 2024a. Qspec: Speculative decoding with complementary quantization schemes. *arXiv preprint arXiv:2410.11305*.

Yao Zhao, Zhitian Xie, Chen Liang, Chenyi Zhuang, and Jinjie Gu. 2024b. Lookahead: An inference acceleration framework for large language model with lossless generation accuracy. In *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pages 6344–6355.

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric Xing, and 1 others. 2023. Judging llm-as-a-judge with mt-bench and chatbot arena. *Advances in neural information processing systems*, 36:46595–46623.

A Algorithm

Here, we give the whole algorithm of our SPIDE in detail in Algorithm 1.

B More Observations on Confidence and Acceptance

In the figure 5, we provide observations on the acceptance rate and proportion of draft tokens within each interval when another four pairs of models perform speculative decoding on the Humaneval dataset.

C Evaluation Details

C.1 Model Configurations

We summarize the model configuration in Table 6. In our experiments, all models are loaded with bfloat-16 precision. Our SPIDE does not introduce any additional training, but directly uses these models to evaluate our algorithm. The running speed is measured on the code generation task.

C.2 Dataset Configurations

In our experiments, we evaluate the effectiveness of our SPIDE on 4 categories of text generation

tasks, including code generation, arithmetic reasoning, multilingual inference, and multi-round dialogue. For the code generation task, we employ HumanEval (Chen et al., 2021), a famous code generation benchmark composed of 164 entries. For arithmetic reasoning and multilingual inference, we employ GSM8K and MGSM (Cobbe et al., 2021; Shi et al., 2022) as the evaluation benchmark. For GSM8K, we sample the first 100 entries for evaluation. For the other 10 categories in MGSM, we select 10 entries for each language. For multi-round dialogue, we employ MT-bench (Zheng et al., 2023) as the benchmark. The maximum generation lengths of these tasks are respectively set to 1024, 256, 256, and 256.

C.3 Detailed Experimental Settings

All our experiments, including ablation studies and case studies, were performed on a single-node system equipped with four NVIDIA GeForce RTX 3090 graphics cards. For inference, we use batch size 1, which is common in other speculative decoding works. For the baselines compared, including SD, SVIP, and PEARL, we all reproduce their results on different text generation tasks using the optimal parameters described in their papers or found in our experiments. When evaluating these methods, the model configuration and GPU usage are exactly the same as our SPIDE.

D More Experimental Results

D.1 More Experimental Results on MGSM

We provide more experimental results on the MGSM dataset in Tables 7, 8, 9, 10. SPIDE generally achieves the highest inference speed over all baseline methods on the MGSM dataset.

D.2 More Experimental Results on MT-bench

We provide more experimental results on the MT-bench dataset in Tables 11, 12, 13, 14. SPIDE achieves the highest inference speed over all baseline methods on the MT-bench dataset.

D.3 Sensitive Analysis of the Threshold τ

We conduct some case studies to understand the impact of different threshold hyperparameter τ values on inference speed and usage cost, as shown in the Figure 6. Intuitively speaking, τ represents the degree of conservatism in SPIDE's estimation of the reliability of the current draft sequence during serial dynamic drafting. When the τ is relatively high, the serial dynamic drafting will be more conservative, and the system will enter parallel drafting verification earlier. This effectively avoids the reduction in decoding efficiency caused by rollback. However, entering parallel drafting verification too early will also result in more frequent calls to more expensive target model, causing higher usage costs. In contrast, when τ is relatively low, serial dynamic drafting will be more aggressive in drafting more prediction tokens, and parallel drafting verification will be started later, which saves usage cost, but may increase the rollback phenomenon and cause decrease in the acceleration effect. Based on our case study, we observe that when τ is between 0.9 and 0.5, a better balance can be achieved between speed and cost. By controlling the value of τ, SPIDE can easily trade off the usage cost and the speed of inference to suit different scenarios. Moreover, when the cost of SPIDE is on a par with or even lower than that of vanilla SD, its inference speed is still significantly better than that of vanilla SD. This once again confirms the excellence of SPIDE in both acceleration effect and cost control.

D.4 The Optimal γ for Speculative Decoding

The speedup of speculative decoding is affected by its fixed drafting length γ. Therefore, all the results of speculative decoding in this paper are based on its best γ. We show in Table 15 some search results for γ for speculative decoding.

D.5 Sensitivity Analysis of τ for SPIDE *w/o para-verify*

As shown in Figure 7, the cost of SPIDE *w/o para-verify* is still the same as SPIDE in general. As τ decreases, dynamic drafting becomes relatively more aggressive, verification times decrease, and the overall cost of use becomes lower. For lightweight drafting models, lower τ motivates better speedups and is less costly; The speedup of the non-lightweight drafting model is not very sensitive to τ, but the lower τ can make the use cost much lower than the optimal vanilla SD, while the speedup is still better than the optimal speedup of vanilla SD.

D.6 Calls to the target model

We provide the actual number of calls to the target model by our SPIDE and individual baseline methods on Humaneval and GSMK datasets in Table 16 and 17.

Algorithm 1 SPIDE

Require: the draft model M_q, the target model M_p, the input prefix x, the max generate tokens L, the window size s, the confidence threshold τ, the confidence-acceptance mapping table T

```
# The serial dynamic drafting is used first.
while len(x) < L do
    if mode="serial" then
        # Serial dynamic drafting
        Γ_0 = 1
        for k = 1 to L − len(x) do
            q_k ← M_q(x + [x_1, ..., x_{k−1}])
            x_k ∼ q_k
            b ← the confidence interval of x_k
            α ← T[b]      #look up the table T
            Γ_k = aΓ_{k−1}
            if Γ_k ≤ τ then
                mode←"parallel"
                break      # Stop drafting!
            end if
        end for
    else
        # Parallel drafting verification
        x, [x_1, x_2, ..., x_k] ← x      # split the prefix to get the last k draft tokens
        for i = k + 1 to k + s do
            # running the draft model in parallel to continue drafting.
            q_i ← M_q(x + [x_1, ..., x_{i−1}])
            x_i ∼ q_i
        end for
        # running the target model in parallel to verify the tokens.
        p_1, p_2, ..., p_k ← M_p(x + [x_1]), M_p(x + [x_1, x_2]), ..., M_p(x + [x_1, ..., x_k])
        retrival q_1, q_2, ..., q_k from the cache
        r_1 ∼ U(0, 1), ..., r_k ∼ U(0, 1)
        n ← min({i − 1 | 1 ≤ i ≤ k, r_i > p_i[x_i]/q_i[x_i]} ∪ {k})
        update the confidence-acceptance mapping table T
        if n = k then
            # accept all tokens
            x ← x + [x_1, ..., x_{k+s}]
            k = s
            mode ← "parallel"
        else
            # reject a token
            y ∼ norm(max(0, p_{n+1} − q_{n+1}))
            x ← [x_1, ..., x_n, y]
            mode ← "serial"
        end if
    end if
end while
```

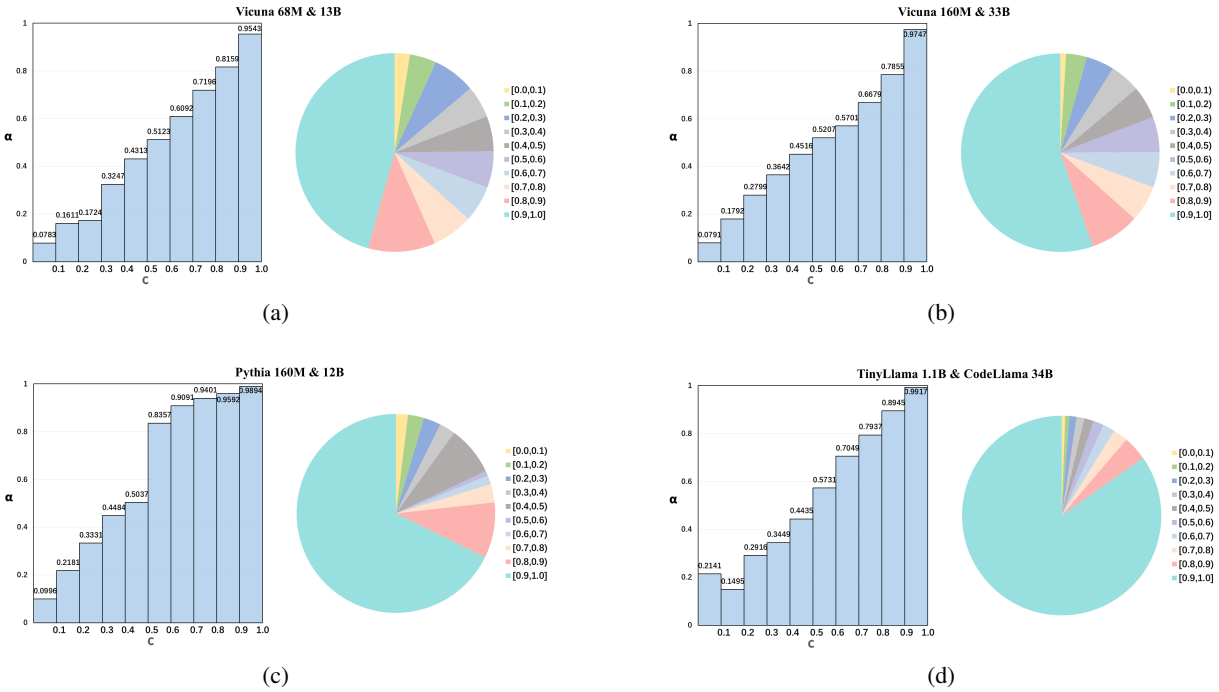

Figure 5: Additional observations about confidence.

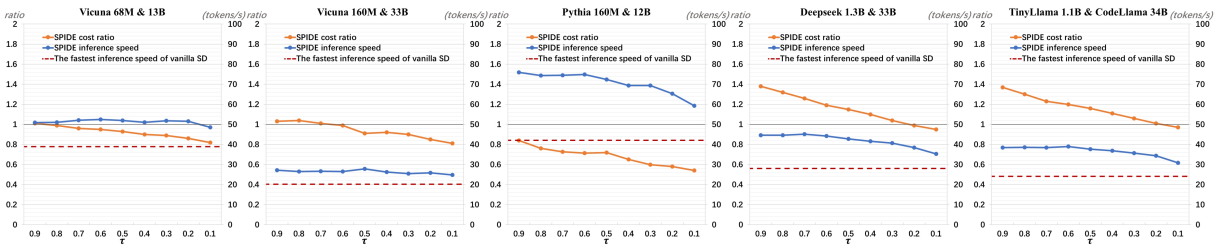

Figure 6: Sensitivity analysis of τ for five pairs of models on the Humaneval dataset.The blue line represents the trend of SPIDE's decoding speed varying with τ. The yellow line represents the trend of SPIDE's cost ratio changing with τ. The dashed red line represents the decoding speed when vanilla SD adopts the best γ, and the solid black line in the middle represents the cost parity with the best vanilla SD.

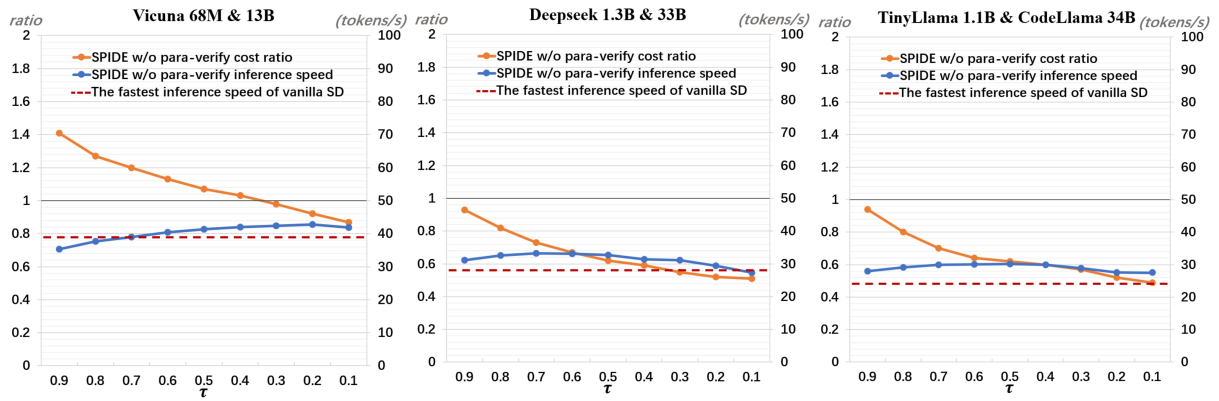

Figure 7: Sensitivity analysis of τ for SPIDE *w/o para-verify* of three pairs of models on the Humaneval dataset.The blue line represents the trend of SPIDE *w/o para-verify*'s decoding speed varying with τ. The yellow line represents the trend of SPIDE *w/o para-verify*'s cost ratio changing with τ. The dashed red line represents the decoding speed when vanilla SD adopts the best γ, and the solid black line in the middle represents the cost parity with the best vanilla SD.

Models	**Layers**	**dim**	**FFN dim**	**Vocab size**
Vicuna-68M	2	768	3072	32000
Vicuna-160M	12	768	3072	32000
Vicuna-13B	40	5120	13824	32000
Vicuna-33B	60	6656	17920	32000
Pythia-160M	12	768	3072	50304
Pythia-12B	36	5120	20480	50688
Deepseek-1.3B	24	2048	5504	32256
Deepseek-33B	62	7168	19200	32256
TinyLlama-1.1B	22	2048	5632	32000
CodeLlama-34B	48	8192	22016	32000

Table 6: Detailed model configurations.

Methods	Bengali	German	English	Spanish	French	Japanese	Russian	Swahili	Tegulu	Thai	Chinese	Avg.
AD	1.00×	1.00×	1.00×	1.00×	1.00×	1.00×	1.00×	1.00×	1.00×	1.00×	1.00×	1.00×
SD	1.68×	1.54×	1.55×	1.54×	1.55×	1.52×	1.51×	1.51×	1.53×	1.53×	1.53×	1.54×
SVIP	1.68×	1.70×	1.68×	1.66×	1.70×	1.69×	1.66×	1.90×	1.63×	1.63×	1.71×	1.69×
D&V	1.71×	1.66×	1.60×	1.65×	1.69×	1.61×	1.69×	1.87×	1.60×	1.57×	1.45×	1.74×
PEARL	2.29×	1.81×	1.52×	1.61×	1.65×	1.70×	1.68×	2.47×	1.86×	1.94×	1.81×	1.84×
SPIDE	**2.54×**	**2.13×**	**1.99×**	**1.86×**	**2.01×**	**1.97×**	**1.84×**	**2.51×**	**2.35×**	**2.28×**	**2.11×**	**2.13×**

Table 7: Results of Vicuna 160M & 33B on MGSM dataset, a multilingual arithmetic reasoning task. We bold **the best results** of each model combination.

Methods	Bengali	German	English	Spanish	French	Japanese	Russian	Swahili	Tegulu	Thai	Chinese	Avg.
AD	1.00×	1.00×	1.00×	1.00×	1.00×	1.00×	1.00×	1.00×	1.00×	1.00×	1.00×	1.00×
SD	1.53×	1.48×	1.42×	1.45×	1.45×	1.47×	1.42×	1.58×	1.46×	1.43×	1.44×	1.46×
SVIP	1.48×	1.56×	1.54×	1.51×	1.57×	1.60×	1.55×	1.69×	1.50×	1.50×	1.51×	1.56×
D&V	1.35×	1.87×	1.66×	1.67×	1.53×	1.68×	1.46×	1.63×	1.59×	1.52×	1.49×	1.59×
PEARL	2.30×	1.98×	1.60×	1.70×	1.79×	1.80×	1.82×	2.34×	2.00×	2.19×	1.95×	1.95×
SPIDE	**2.47×**	**2.09×**	**1.89×**	**1.80×**	**2.01×**	**1.96×**	**1.92×**	**2.35×**	**2.22×**	**2.25×**	**2.06×**	**2.08×**

Table 8: Results of Pythia 160M & 12B on MGSM dataset, a multilingual arithmetic reasoning task. We bold **the best results** of each model combination.

Methods	Bengali	German	English	Spanish	French	Japanese	Russian	Swahili	Tegulu	Thai	Chinese	Avg.
AD	1.00×	1.00×	1.00×	1.00×	1.00×	1.00×	1.00×	1.00×	1.00×	1.00×	1.00×	1.00×
SD	1.67×	1.54×	1.53×	1.52×	1.54×	1.56×	1.51×	1.71×	1.60×	1.53×	1.60×	1.59×
SVIP	1.67×	1.66×	1.64×	1.63×	1.69×	1.67×	1.61×	1.86×	1.65×	1.62×	1.69×	1.66×
D&V	1.75×	1.78×	1.60×	1.61×	1.48×	1.69×	1.56×	1.98×	1.71×	1.69×	1.76×	1.59×
PEARL	2.34×	1.98×	1.71×	1.81×	1.89×	1.93×	1.92×	2.40×	2.20×	2.25×	2.02×	2.04×
SPIDE	**2.60×**	**2.15×**	**2.03×**	**1.94×**	**2.19×**	**2.11×**	**1.95×**	**2.49×**	**2.46×**	**2.40×**	**2.18×**	**2.21×**

Table 9: Results of Deepseek 1.3B & 33B on MGSM dataset, a multilingual arithmetic reasoning task. We bold **the best results** of each model combination.

Methods	Bengali	German	English	Spanish	French	Japanese	Russian	Swahili	Tegulu	Thai	Chinese	Avg.
AD	1.00×	1.00×	1.00×	1.00×	1.00×	1.00×	1.00×	1.00×	1.00×	1.00×	1.00×	1.00×
SD	1.74×	1.63×	1.58×	1.59×	1.65×	1.66×	1.65×	1.81×	1.71×	1.60×	1.68×	1.68×
SVIP	1.72×	1.74×	1.72×	1.70×	1.72×	1.79×	1.73×	1.96×	1.74×	1.71×	1.77×	1.77×
D&V	1.89×	1.87×	1.77×	1.74×	1.87×	1.93×	1.67×	2.09×	1.98×	2.03×	1.89×	1.89×
PEARL	2.37×	2.10×	1.86×	1.93×	2.05×	2.05×	2.07×	2.52×	2.34×	2.33×	2.09×	2.17×
SPIDE	**2.67×**	**2.28×**	**2.19×**	**2.07×**	**2.30×**	**2.25×**	**2.12×**	**2.64×**	**2.61×**	**2.51×**	**2.32×**	**2.37×**

Table 10: Results of Tinyllama 1.1B & Codellama 34B on MGSM dataset, a multilingual arithmetic reasoning task. We bold **the best results** of each model combination.

Methods	Writing	Roleplay	Reasoning	Math	Coding	Extraction	Stem	Humanities	Avg.
AD	1.00×	1.00×	1.00×	1.00×	1.00×	1.00×	1.00×	1.00×	1.00×
SD	1.26×	1.32×	1.42×	1.51×	1.42×	1.31×	1.37×	1.40×	1.38×
SVIP	1.52×	1.53×	1.63×	1.63×	1.51×	1.45×	1.58×	1.58×	1.54×
D&V	1.59×	1.47×	1.69×	1.71×	1.56×	1.49×	1.48×	1.48×	1.51×
PEARL	1.90×	1.47×	2.04×	**2.09×**	1.77×	1.88×	1.57×	1.54×	1.77×
SPIDE	**1.99×**	**1.52×**	**2.11×**	2.06×	**1.95×**	**2.09×**	**1.84×**	**1.65×**	**1.90×**

Table 11: Results of Vicuna 160M & 33B on MT-bench dataset, a multi-round dialogue task. We bold **the best results** of each model combination.

Methods	Writing	Roleplay	Reasoning	Math	Coding	Extraction	Stem	Humanities	Avg.
AD	1.00×	1.00×	1.00×	1.00×	1.00×	1.00×	1.00×	1.00×	1.00×
SD	1.20×	1.25×	1.31×	1.39×	1.32×	1.31×	1.26×	1.32×	1.29×
SVIP	1.44×	1.43×	1.48×	1.50×	1.40×	1.37×	1.44×	1.47×	1.44×
D&V	1.48×	1.56×	1.59×	1.78×	1.67×	1.68×	1.64×	1.57×	1.53×
PEARL	2.23×	1.90×	2.26×	2.18×	2.10×	2.15×	1.86×	1.80×	2.09×
SPIDE	**2.41×**	**2.18×**	**2.31×**	**2.20×**	**2.22×**	**2.30×**	**1.88×**	**1.82×**	**2.20×**

Table 12: Results of Pythia 160M & 12B on MT-bench dataset, a multi-round dialogue task. We bold **the best results** of each model combination.

Methods	Writing	Roleplay	Reasoning	Math	Coding	Extraction	Stem	Humanities	Avg.
AD	1.00×	1.00×	1.00×	1.00×	1.00×	1.00×	1.00×	1.00×	1.00×
SD	1.37×	1.42×	1.52×	1.60×	1.55×	1.40×	1.44×	1.50×	1.47×
SVIP	1.63×	1.62×	1.71×	1.72×	1.62×	1.51×	1.64×	1.63×	1.63×
D&V	1.78×	1.76×	1.69×	1.79×	1.67×	1.47×	1.68×	1.58×	1.68×
PEARL	2.38×	2.09×	2.44×	**2.59×**	**2.36×**	2.33×	1.98×	1.96×	2.25×
SPIDE	**2.68×**	**2.46×**	**2.59×**	2.47×	**2.36×**	**2.37×**	**2.08×**	**2.03×**	**2.39×**

Table 13: Results of Deepseek 1.3B & 33B on MT-bench dataset, a multi-round dialogue task. We bold **the best results** of each model combination.

Methods	Writing	Roleplay	Reasoning	Math	Coding	Extraction	Stem	Humanities	Avg.
AD	1.00×	1.00×	1.00×	1.00×	1.00×	1.00×	1.00×	1.00×	1.00×
SD	1.39×	1.43×	1.52×	1.61×	1.52×	1.38×	1.45×	1.51×	1.47×
SVIP	1.65×	1.62×	1.73×	1.74×	1.61×	1.52×	1.66×	1.67×	1.65×
D&V	1.78×	1.95×	1.84×	1.94×	1.79×	1.60×	1.71×	1.88×	1.90×
PEARL	2.40×	2.15×	2.46×	2.49×	2.27×	2.29×	2.08×	2.06×	2.25×
SPIDE	**2.66×**	**2.46×**	**2.58×**	**2.61×**	**2.31×**	**2.33×**	**2.12×**	**2.11×**	**2.43×**

Table 14: Results of Tinyllama 1.1B & Codellama 34B on MT-bench dataset, a multi-round dialogue task. We bold **the best results** of each model combination.

Vicuna 68M & 13B	Vicuna 160M & 33B	Pythia 160M & 12B	DeepSeek 1.3B & 33B	TinyLlama 1.1B & CodeLlama 34B
39.03 (γ=4)	18.67 (γ=3)	40.09 (γ=3)	27.66 (γ=9)	23.70 (γ=6)
39.18 (γ=5)	19.18 (γ=4)	42.10 (γ=4)	27.84 (γ=10)	23.96 (γ=7)
39.30 (γ=6)	**20.14 (γ=5)**	**42.34 (γ=5)**	**28.14 (γ=11)**	**24.24 (γ=8)**
38.68 (γ=7)	20.00 (γ=6)	41.93 (γ=6)	27.85 (γ=12)	23.95 (γ=9)
37.45 (γ=8)	18.88 (γ=7)	41.66 (γ=7)	27.82 (γ=13)	22.45 (γ=10)

Table 15: Optimal γ values of speculative decoding for each model pair. (Unit: tokens / second)

Methods	Vicuna 68M & 13B	Vicuna 160M & 33B	Pythia 160M & 12B	DeepSeek 1.3B & 33B	TinyLlama 1.1B & CodeLlama 34B
SD	57633	51081	52708	18251	25340
SVIP	60517	50570	45856	18799	13430
D&V	62820	54657	47964	19894	21792
PEARL	76652	75090	64830	30662	39784
SPIDE	58786	55678	39531	22814	35476

Table 16: The number of times each method was called on each model pair on the Humaneval dataset.

Methods	Vicuna 68M & 13B	Vicuna 160M & 33B	Pythia 160M & 12B	DeepSeek 1.3B & 33B	TinyLlama 1.1B & CodeLlama 34B
SD	33286	19515	27062	13560	12736
SVIP	34950	19320	23544	13967	6750
D&V	36282	20881	24626	14780	10953
PEARL	44270	28687	33286	22781	19996
SPIDE	33952	21271	20297	16950	17830

Table 17: The number of times each method was called on each model pair on the GSM8K dataset.