LogitSpec: Accelerating Retrieval-based Speculative Decoding via Next Next Token Speculation

Tianyu Liu[1,2*] **Qitan Lv**[1,2*] **Hao Li**[1,2] **Xing Gao**[2] **Xiao Sun**[2†] **Xiaoyan Sun**[1]
[1]University of Science and Technology of China [2]Shanghai AI Laboratory
{tianyu_liu, qitanlv, lihaohn}@mail.ustc.edu.cn
sunxiao@pjlab.org.cn sunxiaoyan@ustc.edu.cn

Abstract

Speculative decoding (SD), where a small draft model is employed to propose *draft* tokens in advance and then the target model validates them in parallel, has emerged as a promising technique for LLM inference acceleration. Many endeavors to improve SD are to eliminate the need for a draft model and generate draft tokens in a retrieval-based manner in order to further alleviate the drafting overhead and significantly reduce the difficulty in deployment and applications. However, retrieval-based SD relies on a matching paradigm to retrieve the most relevant reference as the draft tokens, where these methods often fail to find matched and accurate draft tokens. To address this challenge, we propose *LogitSpec* to effectively expand the retrieval range and find the most relevant reference as drafts. *LogitSpec* is motivated by the observation that the logit of the last token can not only predict **the next token**, but also speculate **the next next token**. Specifically, *LogitSpec* generates draft tokens in two steps: (1) utilizing the last logit to speculate the next next token; (2) retrieving relevant reference for both the next token and the next next token. *LogitSpec* is training-free and plug-and-play, which can be easily integrated into existing LLM inference frameworks. Extensive experiments on a wide range of text generation benchmarks demonstrate that *LogitSpec* can achieve up to 2.61× speedup and 3.28 mean accepted tokens per decoding step. Our code is available at `https://github.com/smart-lty/LogitSpec`.

1 Introduction

Large Language Models (LLMs), such as GPT-4.5 (OpenAI, 2024), DeepSeek R1 (DeepSeek-AI, 2025), Qwen 2.5 (Team, 2025), and LLaMA-3 (Team, 2024), have demonstrated remarkable capabilities across a wide range of natural language processing tasks, including question answering (Caljiorne Soares and Parreiras, 2020), code generation (Jiang et al., 2024), and dialogue systems (Yi et al., 2024). While they achieve success by scaling up the model parameters, the increase in scale comes with significant inference challenges. The most straightforward challenge is *auto-regressive decoding*, where each LLM decoding step can only generate one token. This token-by-token decoding process incurs exacerbated latency with both the length of generated tokens and the model scale.

To address this challenge, speculative decoding (SD) (Leviathan et al., 2023; Chen et al., 2023) has emerged as a promising approach for lossless LLM inference acceleration. The key idea of SD is to employ a small draft model to first generate γ draft tokens, and then the target model validates these tokens in parallel within a single forward, ensuring the output distribution to be unchanged. However, deploying an extra draft model introduces considerable overhead and difficulties, including (a) **complex implementation and deployment**, especially in integration with other techniques,(Liu et al., 2025) (b) **additional memory overhead**, especially in long-context scenarios. Moreover, when there exists no available draft model for SD, it takes **substantial training cost** to construct a compact draft model (Li et al., 2025b). (Please refer to Section 2 for more details.)

Therefore, many existing works are dedicated to developing draft-model-free SD methods (Saxena, 2023; Fu et al., 2024; He et al., 2024), where the draft tokens are generated in a retrieval-based manner. At each decoding step, they retrieve the relevant reference **according to the last few tokens** and extract draft tokens from the reference. In this way, they can explicitly eliminate the need for a draft model and reduce the drafting overhead.

Despite their promising efficacy and implementation simplicity, these methods rely on a matching paradigm to effectively retrieve the most relevant

[*]Equal Contribution.
[†]The Corresponding Author.

Findings of the Association for Computational Linguistics: ACL 2026, pages 33070–33092
July 2-7, 2026 ©2026 Association for Computational Linguistics

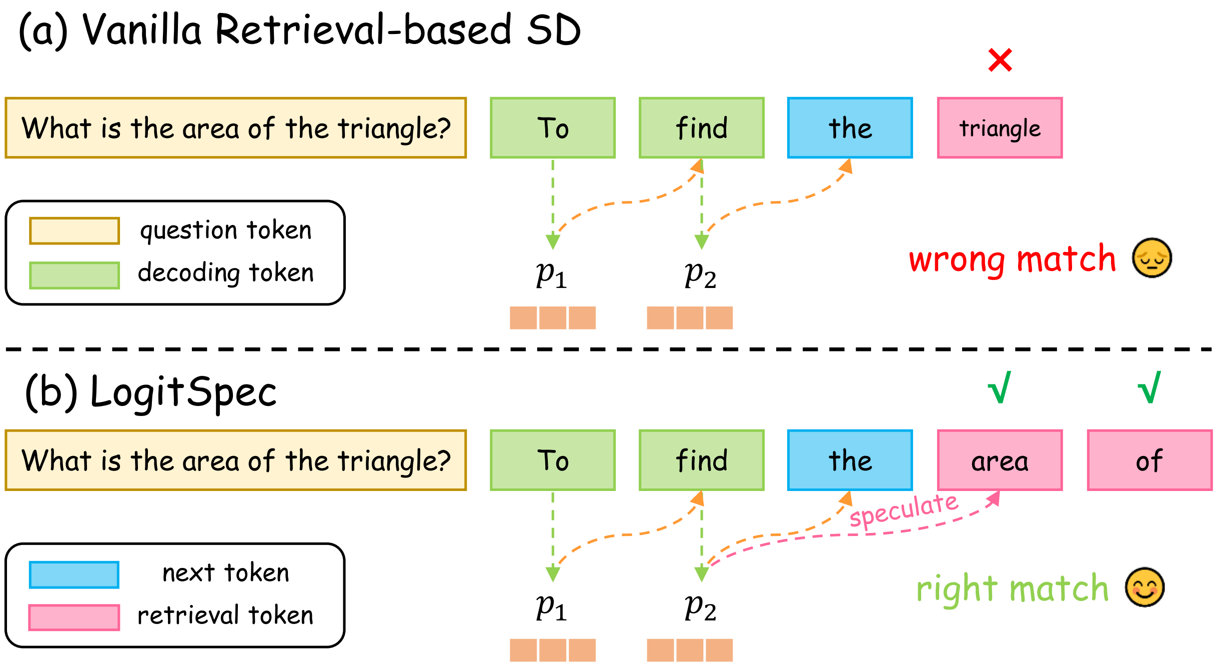

Figure 1: Illustration of vanilla retrieval-based SD method and our *LogitSpec*. (a) Retrieval-based SD retrieves the wrong token "**triangle**" according to the next token "the". (b) *LogitSpec* first speculates the next next token "area", and then retrieves the right relevant reference "**area of**" according to "the area". This simple example illustrates how *LogitSpec* utilizes the last logit to speculate the next next token and improves the retrieval accuracy.

reference as draft tokens and often struggle to find matched and accurate tokens. As shown in Figure 1 (a), with a simple prompt "*What is the area of the triangle?*", vanilla retrieval-based SD methods fail to effectively retrieve the right token "area" according to the next token "the", as the most relevant reference is "the triangle". Moreover, it is often the case that no matched reference can be found, e.g., there are no matched tokens in more than 30% of decoding steps in PLD (Saxena, 2023).

In this paper, we propose a simple yet effective retrieval-based SD framework, namely *LogitSpec*, which leverages the logit of the last token (*last logit*) to predict the next next token and improves the accuracy of the retrieval reference. *LogitSpec* is motivated by the observation that **the last logit can speculate the next next token with a relatively high accuracy.** Based on this observation, we use the speculated next next token as guidance to retrieve reference. Specifically, *LogitSpec* generates draft tokens in two steps: (1) utilizing the last logit to speculate the next next token; (2) retrieving relevant reference for both the next token and the next next token. As shown in Figure 1 (b), with the speculated token "area", *LogitSpec* successfully retrieves the correct draft tokens "area of". This two-step process enables better retrieval accuracy—it can help filter the most relevant reference when the reference is redundant, while extending the searching space when there exists no relevant reference. To summarize, our contributions are:

(a) We empirically observe that the last logit can speculate the next next token with a relatively high accuracy. Unlike other retrieval techniques, which are sensitive to the specific task, this property is **robust** and **effective** across various tasks.

(b) Building on this observation, we propose *LogitSpec*, a **plug-and-play** retrieval-based SD framework which can improve the retrieval accuracy and achieve better speedup.

(c) We conduct extensive experiments on various text generation benchmarks to demonstrate the effectiveness of *LogitSpec*. Notably, *LogitSpec* achieves up to 2.61$\times$ speedup and 3.28 mean accepted tokens per decoding step without the need for an extra draft model.

2 Related Work

In this section, we briefly review the two main directions of speculative decoding, draft-model-based speculative decoding and draft-model-free speculative decoding. More discussions are provided in Appendix A.

Draft-model-based speculative decoding. This line of research focuses on improving the draft quality, which primarily invests heavily in post-training a specialized draft model to generate high-quality drafts. Medusa (Cai et al., 2024) pioneers this direction by adding extra MLP heads at the top of the target model, reusing the hidden states from the target model, and predicting the next few tokens in parallel. However, the generation from these decoding heads is independent, which harms the accuracy of the draft tokens. Therefore, some works (Ankner et al., 2024; Xiao et al., 2024) propose to add sequential dependencies for better performance. Glide (Du et al., 2024) takes a similar idea of Medusa and proposes to reuse the KV cache from the target model. Recently, Eagle (Li et al., 2025c, 2024, 2025b) has dominated this research line by training an auto-regressive head and scaling up the training data. **However, while these methods achieve superior speedup, all of them rely on an extra draft model, which necessitates extra parameters or extensive training.** The existence of the draft model significantly limits its application: (a) deploying a draft model requires complex and careful implementation. (b) Both the draft model and its KV cache require additional GPU memory, which may incur load imbalance. In the long context settings, even the KV cache for the draft model may exceed the capacity of 1 GPU and lead to significant resource competitions; (c) the training cost is substantial, and the draft model requires retraining when the target model is updated.

Draft-model-free speculative decoding. This line of research focuses on maximizing drafting efficiency by eliminating the need for a draft model entirely, aiming for universal, zero-cost acceleration. Some existing works (Zhang et al., 2024a; Elhoushi et al., 2024; Xia et al., 2025) observe that the layer sparsity of LLMs means that not all the layers are necessary to predict the next token. They propose methods to generate draft tokens with a subset of the layers of the target model. However, the number of skipped layers is limited, which affects the overall speedup. Recently, retrieval-based SD (He et al., 2024; Saxena, 2023; Fu et al., 2024) has dominated the draft-model-free speculative decoding by retrieving the relevant reference as the draft tokens. The relevant reference can be derived from the question and generated tokens, or an external database. **However, there exist two common issues during the retrieval process**: (a) retrieval from the context often fails to find matched draft tokens; (b) retrieval from an external database often struggles to find accurate draft tokens. These two issues motivate us to develop a more appropriate retrieval-based speculative decoding framework. To the best of our knowledge, *LogitSpec* is the first to use a speculated next-next token as an additional retrieval anchor; this mechanism is training-free and complementary to multi-token prediction methods that require trained extra heads.

3 Background

Speculative Decoding. Let $\boldsymbol{x}$ denote an input sequence (prefix). A speculative decoding step consists of a drafting phase and a verification phase. During the drafting phase, the draft model $\mathcal{M}_q$ is employed to give γ draft tokens $x_1, x_2, ..., x_\gamma$ by running γ times the model forward and sampling. Here, we denote the output logit $\mathcal{M}_q(\boldsymbol{x} + [x_1, ..., x_{i-1}])$ as q_{i-1}, then each draft token is given by $x_i \sim q_{i-1}, i = 1, ..., \gamma$. During the verification phase, the prefix $\boldsymbol{x}$ together with γ draft tokens are sent to $\mathcal{M}_p$ for verification. The target model $\mathcal{M}_p$ inputs $\boldsymbol{x} + [x_1, ..., x_\gamma]$ and outputs the logits $p_0, p_1, ..., p_\gamma$. Then SD sequentially verifies x_i via speculative sampling (Leviathan et al., 2023; Chen et al., 2023), where the acceptance rate is given by:

$$\alpha_i = \begin{cases} 1 & p_{i-1}[x_i] \geq q_{i-1}[x_i], \\ \dfrac{p_{i-1}[x_i]}{q_{i-1}[x_i]} & p_{i-1}[x_i] < q_{i-1}[x_i], \end{cases} \quad (1)$$

If SD rejects x_i, it will resample a token from $norm(\max(0, p_{i-1} - q_{i-1}))$, otherwise, SD accepts all the draft tokens and samples an additional token from p_γ. In this way, each SD step generates at least 1 token and at most $\gamma + 1$, leading to theoretical lossless quality and efficiency acceleration.

Retrieval-based Speculative Decoding. The retrieval-based SD methods generate draft tokens in a retrieval-based manner. As retrieval-based SD methods do not require an additional draft model, we omit the notation for $\mathcal{M}_q$ and abbreviate $\mathcal{M}_p$ as $\mathcal{M}$. A retrieval model $\mathcal{R}$ is employed to store n-grams (referred as **reference**) from corpus

$$\mathcal{R} = \{(x_i^1, x_i^2, ..., x_i^n)\}_{i=1}^N. \quad (2)$$

Here, N denotes the total number of n-grams in $\mathcal{R}$. Existing retrieval-based SD methods mainly differ in constructing $\mathcal{R}$. At each SD decoding step,

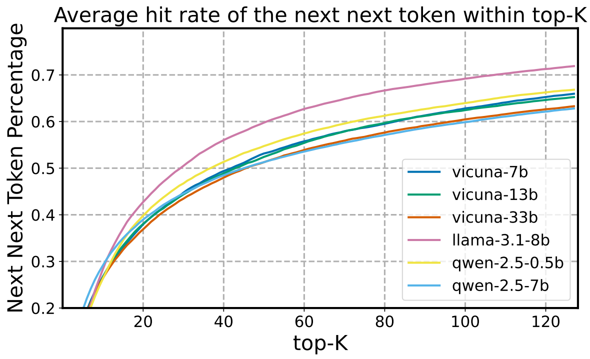

Figure 2: Motivated observation I: the last logit can speculate the next next token with high accuracy. In over 50% decoding steps, the next next token can be found in the top-60 entries within the last logit across model sizes and architectures.

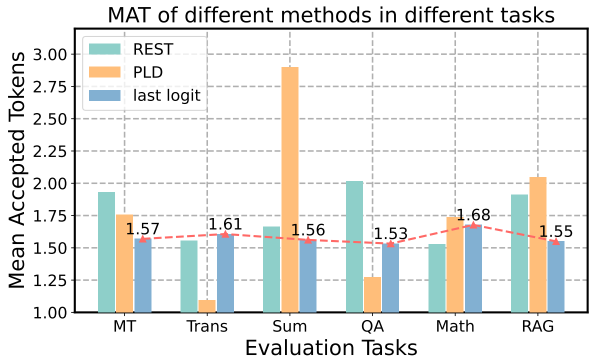

Figure 3: Motivated observation II: compared with other retrieval-based methods, the prediction of the last logit demonstrates robustness to downstream tasks, motivating us to utilize it to guide the retrieval process.

given a query m-gram tokens $x_q^1, x_q^2, ..., x_q^m$, the retrieval model first traverses the reference, finds matched n-grams and returns the subsequent tokens of the matched tokens:

$$\textsc{Match}\big(\mathcal{R}, (x_q^1, x_q^2, ..., x_q^m)\big) \\ = \{(x_i^{m+1}, ..., x_i^n) \mid i \in \mathcal{S}\}. \quad (3)$$

where $\mathcal{S} = \{i | x_i^t = x_q^t, \forall t = 1, ..., m\}$. Intuitively, a larger m leads to more precise matches but lower match probabilities.

4 Method

In this section, we introduce our *LogitSpec*, a novel retrieval-based SD framework that utilizes *last logit* as a guidance for retrieval and effectively improves the retrieval performance. We first conduct a motivated experiment in Section 4.1, and then introduce the whole framework of *LogitSpec* in Section 4.2 and 4.3. An overview of *LogitSpec* is shown in Figure 4.

4.1 Motivated Observation

Drawing insights from multi-token prediction (Gloeckle et al., 2024) that fine-tuned LLMs can predict multiple tokens in a single forward, we are interested in the ability of LLMs to predict the next next token **without fine-tuning**. Unlike MTP, which operates at the hidden-state level and requires training extra heads, we study the logit-level signal of off-the-shelf LLMs (see Appendix B for an intuition). We conduct a simple experiment to investigate the rank of the next next token in the last logit, which is used to predict the next token. Specifically, we use a small fraction of Spec-Bench (Xia et al., 2024) that contains 13 sub-tasks (detailed in Appendix B) and 6 different LLMs as backbones. Results in Figure 2 demonstrate that **the last logit has a strong potential to predict the next next token**. In over 50% decoding steps, the next next token can be found within the top-60 entries of the last logit. For Llama 3.1 8B (Team, 2024) with a large 128K vocabulary, this percentage even increases to 64% within the top-60 entries of the last logit. The results of 6 different LLMs on 13 downstream tasks demonstrate that this observation holds consistently across different models (Vicuna, Llama 3.1, and Qwen) and scales (ranging from 0.5B to 33B).

We further conduct experiments on Spec-Bench with Vicuna 7B to investigate the robustness of the predictive capability of the last logit across different tasks. We propose a minimal speculative decoding algorithm with last logit, *last logit decoding* for this experiment, where we directly use the top-60 entries of the last logit to serve as the draft tokens. Each draft token is a guess at the next next token. If the target model accepts any one of the draft tokens, the others are dropped. Consequently, at each decoding step, the accepted length is either 2 or 1. As in Figure 3, this simple SD method yields mean accepted tokens per decoding step (MAT) of over 1.5. Compared with other retrieval-based SD methods such as PLD (Saxena, 2023) and REST (He et al., 2024), which are highly affected by the task type, **the MAT of *last logit decoding* exhibits superior robustness to the task.** These results demonstrate that the ability of the last logit to predict the next next token widely exists in modern LLMs themselves, and strongly motivate us to improve retrieval-based SD methods with the last logit. We provide a detailed experimental setup for the motivation experiments in Appendix B.

4.2 *LogitSpec* Drafting

While *last logit decoding* has shown great simplicity in implementation and robustness to the downstream tasks, its theoretical upper bound of the speedup ratio is 2, as all the draft tokens are guesses for the next next token. In our experiments, the *last logit decoding* achieves an overall speedup ratio of $1.2\times \sim 1.4\times$, which hinders the real-world applications of *last logit decoding* due to the unsatisfactory speedup ratio. Therefore, to further improve the overall inference speedup, we propose to utilize the prediction ability of the last logit as **a guidance for retrieval**. Specifically, for a given input prefix $\mathbf{x} = (x_1, x_2, ..., x_i)$ and the last logit $p_i = \mathcal{M}(x_1, x_2, ..., x_i)$, we first sample the next token $x_{i+1} \sim p_i$. Then, we utilize the top-k entries of the last logit (except the next token x_{i+1}) as the speculation to the next next token $\widetilde{x}_{i+2}$:

$$\mathcal{L} = \{\widetilde{x}_{i+2} \mid \widetilde{x}_{i+2} \in \textsc{Top}_k(p_i),\ \widetilde{x}_{i+2} \neq x_{i+1}\}. \tag{4}$$

Then, we use the retrieval model $\mathcal{R}$ to retrieve reference for both the next token and next next tokens:

$$\mathcal{D}_{i+1}(x_{i+1}) = \textsc{Match}\big(\mathcal{R}, \boldsymbol{g}_i^{(1)}(x_{i+1})\big), \tag{5}$$

$$\mathcal{D}_{i+2}(\widetilde{x}_{i+2}) = \textsc{Match}\big(\mathcal{R}, \boldsymbol{g}_i^{(2)}(\widetilde{x}_{i+2})\big), \tag{6}$$

$$\mathcal{D} = \mathcal{D}_{i+1}(x_{i+1}) \cup \bigcup_{\widetilde{x}_{i+2} \in \mathcal{L}} \mathcal{D}_{i+2}(\widetilde{x}_{i+2}). \tag{7}$$

Here, $\boldsymbol{g}_i^{(1)}(x_{i+1}) = (x_{i-m+2}, ..., x_i, x_{i+1})$ denotes the m-gram query for the next token, and $\boldsymbol{g}_i^{(2)}(\widetilde{x}_{i+2}) = (x_{i-m+3}, ..., x_i, x_{i+1}, \widetilde{x}_{i+2})$ denotes the query for the next next token. In our experiments, we retrieve draft tokens with $m = 3$ first. If no matched tokens are found, we decrease m to 2 and so on. For simplicity, we only use the user-input prompt and decoded tokens to construct the retrieval model $\mathcal{R}$, that is to say, **our *LogitSpec* is query-independent, and will not be affected by other queries.** We give the detailed implementation of the retrieval process in Appendix D.3.

Taking the example in Figure 4, we first select the top-k entries of the last logit as the speculation for the next next token. Then, we retrieve reference for both the next token "the" and the next next token "area", and return "of the". After that, we organize these draft tokens into a draft tree, where the next token serves as the root. Finally, the token sequence "area of the" is accepted by $\mathcal{M}$.

After the retrieval process, we employ a simple pruning strategy to control the number of draft tokens. Specifically, we do not prune the retrieval tokens for the next token. For each speculated next next token, we prune the retrieval tokens with a simple heuristic strategy: if the rank of the next next token is below 8, we preserve 4 tokens; if the rank of the next next token is below 32, we preserve 3 tokens; otherwise, we only preserve the speculated next next token itself. The retrieval process is terminated until the total number of draft tokens exceeds a specific draft tree capacity K. These thresholds are chosen as simple fractions of K without careful tuning, implementing a pyramid allocation that gives more retrieved tokens to higher-confidence speculations. Furthermore, we explored the impact of various pruning strategies, with results detailed in Table 15 of Appendix E.8.

4.3 *LogitSpec* Verification

After retrieving multiple draft token sequences $\mathcal{D}$, we organize these token sequences into a draft tree and prepare a tree attention for parallel verification.

Then, we prepare an attention mask to make each draft token sequence invisible to other sequences. Let $\Lambda(l)$ denote a causal mask with length l, and m_j denote the length of the j-th token sequence in $\mathcal{D}$; the attention mask is given by:

$$A_{\text{draft}} = \text{diag}\left(\Lambda(m_1), \ldots, \Lambda(m_j)\right). \tag{8}$$

An illustration of this attention mask is shown in Figure 4(c) (note that the next token is visible to all sequences, corresponding to the first column). Both the preparation of the attention mask and the verification of the draft tree are consistent with previous works (Miao et al., 2024; Cai et al., 2024). We provide a pseudo code of the attention mask preparations in Appendix C.

5 Experiments

5.1 Experimental Setup

Tasks and Datasets. We evaluate *LogitSpec* on a broad suite of text generation benchmarks. We first use Spec-Bench (Xia et al., 2024), a widely used comprehensive benchmark that covers diverse

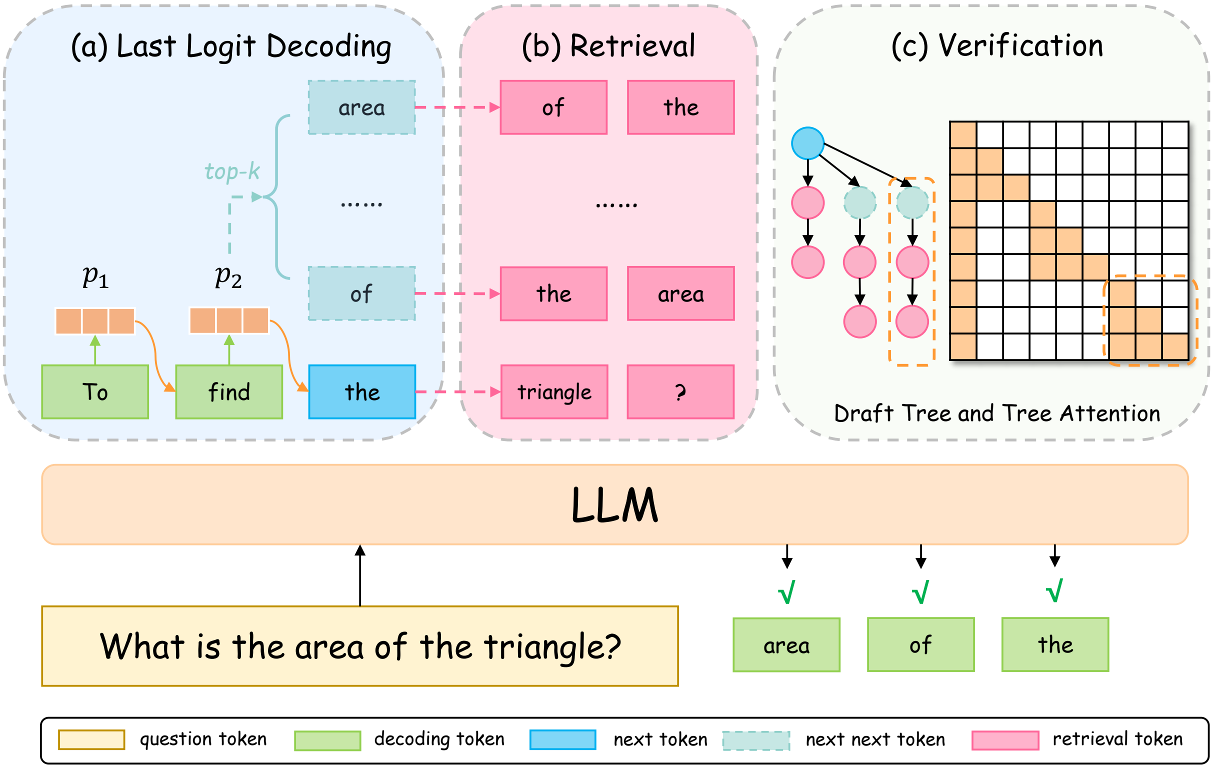

Figure 4: An overview of *LogitSpec*. At each decoding step, *LogitSpec* first utilizes the top-k entries of the last logit as the speculation for the next next token. Then, *LogitSpec* retrieves relevant references for both the next token and the next next token. Finally, *LogitSpec* organizes the draft tokens into a draft tree and prepares a tree attention for parallel verification.

application scenarios, including multi-turn conversation (MT), translation (Trans), summarization (Sum), question answering (QA), mathematical reasoning (Math), and retrieval-augmented generation (RAG). We then further evaluate *LogitSpec* on HumanEval (Chen et al., 2021), GSM8K (Cobbe et al., 2021), CNN/DM (Nallapati et al., 2016), MATH (Hendrycks et al., 2021), AIME 24&25 (Mathematical Association of America, 2025), and LongBench (Bai et al., 2024), which are widely used benchmarks for code generation, math reasoning, summarization, and long-context generation.

Implementation Details. We follow standard inference settings with temperature 0 and batch size 1. All implementation details (baselines, hardware, software versions, model configurations) are provided in Appendix D. We use two widely used metrics for evaluation: mean accepted tokens per decoding step (denoted as MAT) and overall speedup ratio (denoted as Speedup). We further evaluate *LogitSpec* under non-zero temperatures ($T > 0$) in Appendix E.5.

5.2 Main Results

We conduct experiments on various text generation benchmarks to demonstrate the effectiveness of *LogitSpec*. As in Table 1 (with the Spec-Bench breakdown deferred to Appendix E.3), *LogitSpec* outperforms retrieval-based SD baselines by a large margin. We adopt the Vicuna series as the main-table backbone because existing baselines (e.g., REST, Lookahead) only provide stable, reproducible implementations for the Vicuna/Llama-2 family; results on modern Llama-3.1 and Qwen3 backbones are reported in Appendix E.2. We further present more in-depth analysis of *LogitSpec* including (i) more LLM backbones (Llama2 series, Llama-3.1-Instruct-8B and Qwen-3-8B), (ii) more benchmarks (MATH and AIME datasets), and (iii) long context benchmarks (the LongBench dataset) in Appendix E. Comparisons against draft-model-based methods (EAGLE, Medusa) and stronger retrieval-based baselines (Token Recycling, SAM Decoding) are deferred to Appendix E.1 and Appendix E.2, respectively.

We observe the following: (a) *LogitSpec* significantly improves MAT to 2.13 $\sim$ 3.28, achiev-

Table 1: Experimental results of *LogitSpec* on CNN/DM (Nallapati et al., 2016), GSM8K (Cobbe et al., 2021) and HumanEval (Chen et al., 2021) with **Vicuna**. We report the mean accepted tokens per decoding step (MAT) and overall speedup ratio. We **bold** the best results and underline the suboptimal results for each backbone model.

Models	Method	CNN/DM		GSM8K		HumanEval		Spec-Bench		Average	
		MAT	Speedup	MAT	Speedup	MAT	Speedup	MAT	Speedup	MAT	Speedup
Vicuna 7B	Lookahead	1.54	1.28	1.92	1.64	1.79	1.57	1.66	1.27	1.73	1.44
	REST	1.64	1.19	1.55	1.13	1.96	1.51	1.82	1.48	1.74	1.33
	PLD	2.61	2.26	1.80	1.62	1.84	1.65	1.73	1.59	2.00	1.78
	SpS	2.34	1.58	2.08	1.49	2.56	1.80	2.28	1.49	2.32	1.59
	LogitSpec	**3.28**	**2.61**	**2.69**	**2.20**	**2.62**	**2.24**	**2.44**	**2.01**	**2.76**	**2.26**
Vicuna 13B	Lookahead	1.46	1.12	1.88	1.61	1.75	1.57	1.63	1.22	1.68	1.38
	REST	1.65	1.22	1.57	1.18	1.94	1.55	1.82	1.38	1.75	1.33
	PLD	2.34	1.85	1.76	1.61	1.98	1.77	1.68	1.48	1.94	1.68
	SpS	2.18	1.48	2.00	1.55	2.66	1.95	2.18	1.49	2.26	1.62
	LogitSpec	**2.90**	**2.17**	**2.59**	**2.13**	**2.88**	**2.47**	**2.32**	**1.93**	**2.67**	**2.18**
Vicuna 33B	Lookahead	1.50	1.18	1.89	1.44	1.66	1.33	1.61	1.24	1.67	1.30
	REST	1.65	1.41	1.57	1.32	1.99	1.64	1.80	1.48	1.75	1.46
	PLD	2.07	1.72	1.66	1.41	1.60	1.42	1.55	1.42	1.72	1.49
	SpS	2.01	1.57	1.97	1.54	2.27	1.80	2.01	1.61	2.07	1.63
	LogitSpec	**2.66**	**2.01**	**2.46**	**1.93**	**2.41**	**1.95**	**2.13**	**1.76**	**2.42**	**1.92**

ing up to 2.61× speedup on CNN/DM with Vicuna 7B and 2.47× on HumanEval with Vicuna 13B, substantially outperforming REST (<1.5× on CNN/DM). (b) Thanks to the robustness of last logit speculation, *LogitSpec* maintains strong performance even in challenging scenarios with limited context repetition, where other retrieval-based methods struggle (see detailed per-task analysis in Appendix E.3). (c) *LogitSpec* significantly outperforms PLD across all benchmarks and model sizes, demonstrating that next-next token speculation effectively guides retrieval and improves draft quality, especially when output has low overlap with input context.

5.3 Ablation Study

To further provide more insights of *LogitSpec*, we conduct extensive ablation studies on the aforementioned 4 benchmarks in Table 2. Specifically, we mainly focus on two components of *LogitSpec*: *last logit* and *retrieval*. We denote *LogitSpec* without the last logit decoding and only using retrieval as *w/o last logit*, and *LogitSpec* without retrieval and only using last logit decoding as *w/o retrieval*. As shown in Table 2, the absence of any component results in a performance degradation of the entire framework.

Our findings are as follows. First, the absence of *retrieval* exhibits more importance to the final acceleration, which is consistent with the discussion in Section 4.2 that the theoretical upper bound of *last logit decoding* severely hinders its real-world application. Second, the absence of *retrieval* and the absence of *last logit* show different effects in different sub-tasks. For example, the absence of *retrieval* decreases MAT by 0.87 on Spec-Bench with Vicuna 7B, while it decreases MAT by 1.71 on CNN/DM. In contrast, the absence of *last logit* leads to a more consistent MAT degradation across different tasks, which further highlights the robustness of *last logit decoding*. Finally, these results demonstrate that combining *last logit* with *retrieval* improves the retrieval performance and overall speedup. We also conduct ablation studies on pruning strategies in Appendix E.8 and E.9. The experimental results suggest that varying the pruning strategy yields only minor differences.

5.4 Case Study

In-depth Running Time Analysis. We conduct experiments to analyze the running time allocation of the whole decoding. Specifically, there are **five non-negligible components** in *LogitSpec*, including (a) *retrieving draft tokens*: the process of retrieving reference for the next token and the next next tokens; (b) *preparation*: preparing attention mask for the draft tokens; (c) *model forward*: conducting one-pass model forward; (d) *verification*: validating the draft tokens with speculative sampling; (e) *update*: necessary update of KV cache

Table 2: Ablation experiments of *LogitSpec* on Spec-Bench (Xia et al., 2024), CNN/DM (Nallapati et al., 2016), GSM8K (Cobbe et al., 2021) and HumanEval (Chen et al., 2021) with **Vicuna**. We report the ablation results with MAT reduction and Speedup reduction with down arrow ↓.

Models	Method	Spec-Bench		CNN/DM		GSM8K		HumanEval	
		MAT	Speedup	MAT	Speedup	MAT	Speedup	MAT	Speedup
Vicuna 7B	*w/o last logit*	$1.72_{\downarrow.72}$	$1.27_{\downarrow.74}$	$2.66_{\downarrow.62}$	$2.24_{\downarrow.37}$	$1.81_{\downarrow.88}$	$1.63_{\downarrow.57}$	$1.89_{\downarrow.73}$	$1.69_{\downarrow.55}$
	w/o retrieval	$1.57_{\downarrow.87}$	$1.24_{\downarrow.77}$	$1.57_{\downarrow1.71}$	$1.23_{\downarrow1.38}$	$1.71_{\downarrow.98}$	$1.39_{\downarrow.81}$	$1.69_{\downarrow.93}$	$1.39_{\downarrow.85}$
	LogitSpec	**2.44**	**2.01**	**3.28**	**2.61**	**2.69**	**2.20**	**2.62**	**2.24**
Vicuna 13B	*w/o last logit*	$1.66_{\downarrow.66}$	$1.23_{\downarrow.70}$	$2.34_{\downarrow.56}$	$1.81_{\downarrow.36}$	$1.78_{\downarrow.81}$	$1.61_{\downarrow.52}$	$2.02_{\downarrow.86}$	$1.75_{\downarrow.72}$
	w/o retrieval	$1.59_{\downarrow.73}$	$1.19_{\downarrow.74}$	$1.58_{\downarrow1.32}$	$1.22_{\downarrow.95}$	$1.71_{\downarrow.88}$	$1.45_{\downarrow.68}$	$1.63_{\downarrow1.25}$	$1.37_{\downarrow1.10}$
	LogitSpec	**2.32**	**1.93**	**2.90**	**2.17**	**2.59**	**2.13**	**2.88**	**2.47**
Vicuna 33B	*w/o last logit*	$1.55_{\downarrow.58}$	$1.22_{\downarrow.54}$	$2.05_{\downarrow.61}$	$1.72_{\downarrow.29}$	$1.64_{\downarrow.82}$	$1.39_{\downarrow.54}$	$1.59_{\downarrow.82}$	$1.40_{\downarrow.55}$
	w/o retrieval	$1.61_{\downarrow.52}$	$1.25_{\downarrow.51}$	$1.62_{\downarrow1.04}$	$1.26_{\downarrow.75}$	$1.69_{\downarrow.77}$	$1.33_{\downarrow.60}$	$1.60_{\downarrow.81}$	$1.34_{\downarrow.61}$
	LogitSpec	**2.13**	**1.76**	**2.66**	**2.01**	**2.46**	**1.93**	**2.41**	**1.95**

Table 3: Case study experiments of *LogitSpec* on Spec-Bench and HumanEval with **Vicuna**. We report the successful retrieval rate of each method. We report the relative improvements with ↑.

Method	Spec-Bench			HumanEval		
	Vicuna 7B	Vicuna 13B	Vicuna 33B	Vicuna 7B	Vicuna 13B	Vicuna 33B
PLD	63.88	63.08	62.71	69.03	69.51	67.67
LogitSpec	$97.76_{\uparrow53.04\%}$	$97.64_{\uparrow54.79\%}$	$97.93_{\uparrow56.16\%}$	$99.29_{\uparrow43.84\%}$	$99.31_{\uparrow42.87\%}$	$99.37_{\uparrow46.84\%}$

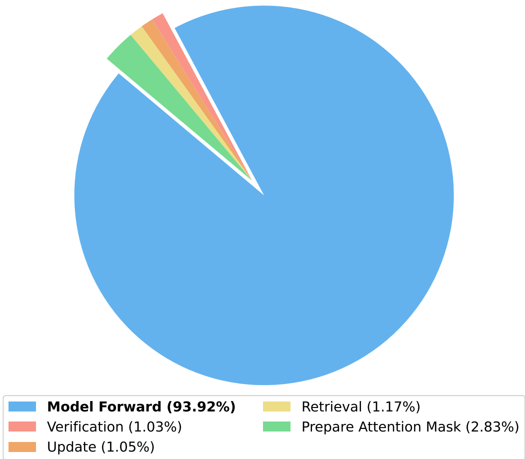

Figure 5: Running time breakdown of the whole decoding process on Spec-Bench with Vicuna 7B.

and retrieval model.

As shown in Figure 5, *model forward* occupies the majority of wall-clock time. Compared with vanilla AR decoding, the overhead introduced by *LogitSpec*, i.e., retrieving overhead, only takes 1.17% of the whole decoding process. It can be further alleviated by parallel techniques, as the retrieval process is independent, which brings negligible overhead as well.

Retrieval Performance. As mentioned in Section 4.2, *LogitSpec* expands the retrieval range and improves the retrieval performance. We conduct a simple experiment to test the retrieval success rate, i.e., whether the model successfully retrieves the reference as draft tokens. As in Table 3, PLD cannot retrieve any matched tokens in more than 30% decoding steps, while *LogitSpec* retrieves matched reference in most decoding steps. These results further demonstrate the effectiveness of *LogitSpec*.

Real-world examples. We also provide a real-world example in Appendix E.10 to illustrate how *next-next token speculation* allows *LogitSpec* to succeed where standard retrieval methods fail.

6 Conclusion

In this paper, we observe that the logit of the last token can predict **the next next token** with a relatively high accuracy without **any fine-tuning**. Based upon this observation, we propose a novel retrieval-based SD framework, namely *LogitSpec*, which utilizes the prediction ability of the last logit to effectively expand the retrieval range and find the most relevant reference as the draft tokens. *LogitSpec* does not require an additional draft model and is a fully **plug-and-play** method, which can be easily implemented and integrated into existing LLM frameworks. Extensive experiments demonstrate that *LogitSpec* can effectively improve the retrieval performance, leading to a $1.8\times \sim 2.6\times$ speedup across all the evaluation benchmarks.

Limitations

We consider a few limitations and future works as follows. **(i)** While our *LogitSpec* is a fully plug-and-play SD framework, its real-world inference acceleration is less competitive. Our future works involve integrating *LogitSpec* into existing draft-model-based SD methods for further acceleration. **(ii)** Currently, *LogitSpec* retrieves relevant reference from the prompt, which may incur lower speedup when the prompt is short. Concretely, *LogitSpec* still achieves $\sim 1.4\times$ speedup on translation tasks where context n-grams are scarce (Table 6) and rises to $2.01\times$ on LongBench with longer contexts (Appendix E.7). We consider integrating an external database to boost the retrieval model as a future work. **(iii)** Our acceleration estimates rely on offline decoding with curated prompts and NVIDIA A100 GPUs, so the reported speedups may vary on other workloads or hardware. *LogitSpec* currently retrieves draft candidates from the on-the-fly prompt context only; short prompts or out-of-domain corpora reduce its benefits.

Ethical Considerations

Inheritance of Model Behaviors. *LogitSpec* is an inference acceleration framework designed to speed up existing Large Language Models without modifying their weights. As a speculative decoding method, it aims to losslessly recover the distribution of the target model. Consequently, *LogitSpec* inherits the ethical properties, biases, and potential safety risks of the underlying target LLM and draft model. It does not introduce new capabilities for generating harmful content, nor does it mitigate existing biases in the base models. Users should continue to apply standard safety guardrails and alignment techniques to the target models deployed with *LogitSpec*.

References

Zachary Ankner, Rishab Parthasarathy, Aniruddha Nrusimha, Christopher Rinard, Jonathan Ragan-Kelley, and William Brandon. 2024. Hydra: Sequentially-dependent draft heads for medusa decoding. In *First Conference on Language Modeling*.

Gregor Bachmann, Sotiris Anagnostidis, Albert Pumarola, Markos Georgopoulos, Artsiom Sanakoyeu, Yuming Du, Edgar Schönfeld, Ali Thabet, and Jonas Kohler. 2025. Judge decoding: Faster speculative sampling requires going beyond model alignment. *Preprint*, arXiv:2501.19309.

Sangmin Bae, Jongwoo Ko, Hwanjun Song, and Se-Young Yun. 2023. Fast and robust early-exiting framework for autoregressive language models with synchronized parallel decoding. *Preprint*, arXiv:2310.05424.

Yushi Bai, Xin Lv, Jiajie Zhang, Hongchang Lyu, Jiankai Tang, Zhidian Huang, Zhengxiao Du, Xiao Liu, Aohan Zeng, Lei Hou, Yuxiao Dong, Jie Tang, and Juanzi Li. 2024. LongBench: A bilingual, multitask benchmark for long context understanding. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 3119–3137, Bangkok, Thailand. Association for Computational Linguistics.

Tianle Cai, Yuhong Li, Zhengyang Geng, Hongwu Peng, Jason D. Lee, Deming Chen, and Tri Dao. 2024. Medusa: Simple llm inference acceleration framework with multiple decoding heads. In *Proceedings of the 41st International Conference on Machine Learning*, ICML'24. JMLR.org.

Marco Antonio Calijorne Soares and Fernando Silva Parreiras. 2020. A literature review on question answering techniques, paradigms and systems. *Journal of King Saud University - Computer and Information Sciences*, 32(6):635–646.

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

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, and 39 others. 2021. Evaluating large language models trained on code. *Preprint*, arXiv:2107.03374.

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

DeepSeek-AI. 2025. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *Preprint*, arXiv:2501.12948.

Cunxiao Du, Jing Jiang, Xu Yuanchen, Jiawei Wu, Sicheng Yu, Yongqi Li, Shenggui Li, Kai Xu, Liqiang Nie, Zhaopeng Tu, and Yang You. 2024. Glide with a cape: a low-hassle method to accelerate speculative decoding. In *Proceedings of the 41st International Conference on Machine Learning*, ICML'24. JMLR.org.

Mostafa Elhoushi, Akshat Shrivastava, Diana Liskovich, Basil Hosmer, Bram Wasti, Liangzhen Lai, Anas Mahmoud, Bilge Acun, Saurabh Agarwal, Ahmed Roman, Ahmed A Aly, Beidi Chen, and Carole-Jean Wu. 2024. Layerskip: Enabling early exit inference and self-speculative decoding.

Yichao Fu, Peter Bailis, Ion Stoica, and Hao Zhang. 2024. Break the sequential dependency of llm inference using lookahead decoding. In *Proceedings of the 41st International Conference on Machine Learning*, ICML'24. JMLR.org.

Fabian Gloeckle, Badr Youbi Idrissi, Baptiste Rozière, David Lopez-Paz, and Gabriel Synnaeve. 2024. Better & faster large language models via multi-token prediction. *Preprint*, arXiv:2404.19737.

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

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. 2021. Measuring mathematical problem solving with the MATH dataset. In *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 2)*.

Coleman Hooper, Sehoon Kim, Hiva Mohammadzadeh, Hasan Genc, Kurt Keutzer, Amir Gholami, and Sophia Shao. 2024. Speed: Speculative pipelined execution for efficient decoding. *Preprint*, arXiv:2310.12072.

Yuxuan Hu, Ke Wang, Xiaokang Zhang, Fanjin Zhang, Cuiping Li, Hong Chen, and Jing Zhang. 2024. Sam decoding: Speculative decoding via suffix automaton. *Preprint*, arXiv:2411.10666.

Kaixuan Huang, Xudong Guo, and Mengdi Wang. 2025. Specdec++: Boosting speculative decoding via adaptive candidate lengths.

Juyong Jiang, Fan Wang, Jiasi Shen, Sungju Kim, and Sunghun Kim. 2024. A survey on large language models for code generation. *Preprint*, arXiv:2406.00515.

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

Jinze Li, Yixing Xu, Haiduo Huang, Xuanwu Yin, Dong Li, Edith C. H. Ngai, and Emad Barsoum. 2025a. Gumiho: A hybrid architecture to prioritize early tokens in speculative decoding. *Preprint*, arXiv:2503.10135.

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

Yuhui Li, Fangyun Wei, Chao Zhang, and Hongyang Zhang. 2025b. Eagle-3: Scaling up inference acceleration of large language models via training-time test. *Preprint*, arXiv:2503.01840.

Yuhui Li, Fangyun Wei, Chao Zhang, and Hongyang Zhang. 2025c. Eagle: Speculative sampling requires rethinking feature uncertainty. *Preprint*, arXiv:2401.15077.

Jiahao Liu, Qifan Wang, Jingang Wang, and Xunliang Cai. 2024. Speculative decoding via early-exiting for faster llm inference with thompson sampling control mechanism. *Preprint*, arXiv:2406.03853.

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*.

Xianzhen Luo, Yixuan Wang, Qingfu Zhu, Zhiming Zhang, Xuanyu Zhang, Qing Yang, Dongliang Xu, and Wanxiang Che. 2024. Turning trash into treasure: Accelerating inference of large language models with token recycling. *Preprint*, arXiv:2408.08696.

Mathematical Association of America. 2025. Maa invitational competitions. `https://maa.org/maa-invitational-competitions/`. Accessed: 2025-09-09.

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

Ramesh Nallapati, Bowen Zhou, Cicero dos Santos, Çağlar Gulçehre, and Bing Xiang. 2016. Abstractive text summarization using sequence-to-sequence RNNs and beyond. In *Proceedings of the 20th SIGNLL Conference on Computational Natural Language Learning*, pages 280–290, Berlin, Germany. Association for Computational Linguistics.

NVIDIA, Péter Vingelmann, and Frank H.P. Fitzek. 2020. Cuda, release: 10.2.89.

OpenAI. 2024. Gpt-4 technical report. *Preprint*, arXiv:2303.08774.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Köpf, Edward Yang, Zach DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, and 2 others. 2019. Pytorch: An imperative style, high-performance deep learning library. *Preprint*, arXiv:1912.01703.

Apoorv Saxena. 2023. Prompt lookup decoding.

Llama Team. 2024. The llama 3 herd of models. *Preprint*, arXiv:2407.21783.

Qwen Team. 2025. Qwen2.5 technical report. *Preprint*, arXiv:2412.15115.

Jikai Wang, Yi Su, Juntao Li, Qingrong Xia, Zi Ye, Xinyu Duan, Zhefeng Wang, and Min Zhang. 2024. Opt-tree: Speculative decoding with adaptive draft tree structure. *Preprint*, arXiv:2406.17276.

Yepeng Weng, Dianwen Mei, Huishi Qiu, Xujie Chen, Li Liu, Jiang Tian, and Zhongchao Shi. 2025. Coral: Learning consistent representations across multi-step training with lighter speculative drafter. *Preprint*, arXiv:2502.16880.

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

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

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

Bin Xiao, Chunan Shi, Xiaonan Nie, Fan Yang, Xiangwei Deng, Lei Su, Weipeng Chen, and Bin Cui. 2024. Clover: Regressive lightweight speculative decoding with sequential knowledge. *Preprint*, arXiv:2405.00263.

Siheng Xiong, Joe Zou, Faramarz Fekri, and Yae Jee Cho. Long-context modeling with dynamic hierarchical sparse attention for on-device llms. In *NeurIPS 2025 Workshop on Efficient Reasoning*.

Penghui Yang, Cunxiao Du, Fengzhuo Zhang, Haonan Wang, Tianyu Pang, Chao Du, and Bo An. 2026. Longspec: Long-context lossless speculative decoding with efficient drafting and verification. *Preprint*, arXiv:2502.17421.

Zihao Yi, Jiarui Ouyang, Yuwen Liu, Tianhao Liao, Zhe Xu, and Ying Shen. 2024. A survey on recent advances in llm-based multi-turn dialogue systems. *Preprint*, arXiv:2402.18013.

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

Lefan Zhang, Xiaodan Wang, Yanhua Huang, and Ruiwen Xu. 2025. Learning harmonized representations for speculative sampling. In *The Thirteenth International Conference on Learning Representations*.

Situo Zhang, Hankun Wang, Da Ma, Zichen Zhu, Lu Chen, Kunyao Lan, and Kai Yu. 2024b. Adaeagle: Optimizing speculative decoding via explicit modeling of adaptive draft structures. *Preprint*, arXiv:2412.18910.

A More Discussions to Related Work

We provide additional discussion of existing works in both draft-model-based speculative decoding and draft-model-free speculative decoding. An intuitive comparison is shown in Table 4.

Draft-model-based speculative decoding. Besides the discussions in Section 2, we discuss additional works on draft-model-based speculative decoding. For example, HASS (Zhang et al., 2025) identifies the inconsistency between EAGLE's training and inference phases, and proposes a multi-step training framework to address this. CORAL (Weng et al., 2025) proposes a cross-step representation alignment to address this problem. Judge Decoding (Bachmann et al., 2025) recognizes the potential of accepting high-quality but refused draft tokens to further improve the acceleration. Gumiho (Li et al., 2025a) demonstrates that the initial draft token is more important and proposes a hybrid model to combine serial and parallel draft heads. LongSpec (Yang et al., 2026) proposes a long-context lossless speculative decoding framework with efficient drafting and verification.

Besides these methods that focus on the training process (Xiong et al.), the process of verification also draws extensive interest, mainly focusing on adaptive draft length. SpecDec++ (Huang et al., 2025) formulates the verification process as a Markov decision process to adaptively determine the draft length. OPT-Tree (Wang et al., 2024) proposes a method to search for the optimal tree structure that maximizes the mathematical expectation of the acceptance length in each decoding step. AdaEAGLE (Zhang et al., 2024b) proposes a novel framework to explicitly model the draft tree structure for EAGLE series models. PEARL (Liu et al., 2025) pioneers this direction by serving the draft model and the target model in parallel to achieve a segmented adaptive draft length.

Draft-model-free speculative decoding. For layer sparsity, SPEED (Hooper et al., 2024) proposes a method to speculatively execute multiple future tokens in parallel with the current token using predicted values based on early-layer hidden states. FREE (Bae et al., 2023) proposes a shallow-deep module and a synchronized parallel decoding to improve the efficiency. EESD (Liu et al., 2024) proposes an early-exiting framework with a self-distillation method and leverages Thompson Sampling to regulate the generation processes. For retrieval-based SD, Token Recycling (Luo et al., 2024) proposes a method to utilize the dropped draft tokens and generate draft tokens via an adjacency matrix. Different from our method, Token Recycling is a **query-dependent** method that utilizes information from other queries, which may result in limitations when applied to real-world applications with complex and dynamic user inputs. SAM Decoding (Hu et al., 2024) utilizes a common text corpus and dynamic text sequence as retrieved sources and proposes a suffix automaton to efficiently obtain more accurate match positions.

B Details of Motivation Experiments

To investigate the prediction ability of the last logit, we conduct two motivation experiments to demonstrate its effectiveness and robustness. For the effectiveness of the last logit, as shown in Figure 2, we conduct autoregressive inference for the 6 models and record the logits for each decoded token. Then, for each decoded token x_i which is sampled from the logit p_{i-1}, we investigate the rank of x_i in p_{i-2}, i.e., the corresponding last logit, and visualize the statistics of the rank in Figure 2. For the robustness of the last logit, as shown in Figure 3, we conduct *last logit decoding* and investigate its mean accepted tokens per decoding step. Both experiments are conducted on a small subset of Spec-Bench, where we randomly sample 2 questions for each sub-category of MT, and 10 questions for other categories (Trans, Sum, QA, Math, RAG).

Why Not Use Next-Next-Next Tokens (NNNT)? A natural question is whether we can extend the approach to predict even further tokens. We investigated this: the ground-truth NNNT appears in the top-60 last logits with ≥40% probability, showing the phenomenon does extend.

However, we did **not** use NNNT due to combinatorial explosion. If we speculate top-60 for both NNT and NNNT, the draft tree would contain at least $60^2 = 3{,}600$ tokens, far exceeding efficient verification capacity. As shown in Table 16 (Appendix E.9), tree capacity beyond 64-128 tokens yields diminishing returns as verification overhead dominates.

Our chosen approach (NNT + retrieval with K=64) represents the optimal efficiency-accuracy trade-off given current hardware constraints.

Table 4: Training and deployment comparison for different methods. Example training costs: EAGLE 8× RTX 3090 for 1–2 days; HYDRA 8× A100 for training; MEDUSA 1× A100 for 5 hours.

Methods	Training Cost	Additional Parameters	Lossless Quality?	Deployment Difficulty
EAGLE	High	AR Heads	✓	High
HYDRA	High	MLP Heads	✗	Moderate
MEDUSA	Moderate	MLP Heads	✗	Moderate
LogitSpec (Ours)	**None**	**None**	✓	**Plug-and-Play**

Intuition: Why Can Last Logits Predict Next-Next Tokens? While our contribution is an empirical discovery, we offer intuition for why this phenomenon exists:

The last logit $p_{i-1} = P(x_i|x_1, \ldots, x_{i-1})$ is computed from hidden states encoding the full context up to position $i-1$. While the model is trained to predict x_i, the rich contextual representation necessarily contains information about likely continuations beyond x_i—otherwise, the model could not generate coherent multi-token sequences.

Our findings suggest that top-ranked tokens in p_{i-1} correlate with likely values of x_{i+1}, particularly in structured text where multi-token patterns (phrases, named entities, common collocations) are prevalent. This predictive signal exists *without fine-tuning*, making it immediately exploitable for inference acceleration.

We leave formal theoretical analysis of this phenomenon to future work.

C Pseudo Code to Prepare Attention Inputs

We provide pseudo code to organize the retrieved multiple draft token sequences into a draft tree and prepare its attention mask. More detailed implementations can be found in our attached code.

```
def prepare_attention_inputs(past_len,
    next_token, candidate_list,
    num_draft_tokens):
    'LogitSpec organizes draft tokens in
        a tree manner. Each sub-
        sequence corresponds to a local
        causal mask.'

    seq_len = num_draft_tokens + 1

    # organize the candidate list into a
        sequence
    draft_ids = [next_token] + [token
        for sub in candidate_list for
        token in sub]

    # prepare original position ids and
        attention mask
    position_ids = torch.zeros((1,
        seq_len), dtype=torch.long)
    causal_mask = torch.full((seq_len,
        past_len + seq_len), fill_value
        =0)
    causal_mask[:, :past_len+1] = 1

    # prepare causal mask
    idx = 1
    for sub_sequence in candidate_list:
        l = len(sub_sequence)
        sub_mask = torch.tril(torch.ones
            ((l, l)))
        causal_mask[idx:idx+l, idx+
            past_len:idx+past_len+l] =
            sub_mask
        position_ids[0, idx:idx+l] =
            torch.arange(l) + 1
        idx += l

    position_ids += past_len
    return draft_ids, causal_mask,
        position_ids
```

D More Implementation Details

D.1 Baselines

We compare *LogitSpec* against strong speculative decoding baselines. For draft-model-free, plug-and-play methods, we include: (a) Lookahead Decoding (Fu et al., 2024), which utilizes dropped draft tokens as a retrieval corpus; (b) REST (He et al., 2024), which employs an external knowledge base as the retrieval model; and (c) PLD (Saxena, 2023), which extracts n-grams from the current user-input prompt and decoded tokens as the retrieval model. In addition, we report results for a draft-model-based baseline, vanilla speculative sampling (SpS) (Leviathan et al., 2023; Chen et al., 2023) with a small Vicuna-68M draft model, to illustrate the trade-off between extra model parameters / training and acceleration. We use Vicuna series models (7B, 13B, 33B), Llama-2 series models (7B, 13B, 70B), Llama-3.1-Instruct-8B and Qwen-3-8B as backbones, and report results with baseline methods on Vicuna and Llama-2 as the main results, where the performance of baseline methods can be reproduced with their official implementations.

D.2 Hardware and Software Configurations

We follow the setting of Spec-Bench with Pytorch (Paszke et al., 2019) 2.6.0 and CUDA (NVIDIA et al., 2020) 12.4. Our implementation is based on Hugging Face transformers. For main experiments with Vicuna and Llama-2 models (Section 5.1 and all baseline comparisons), we use transformers (Wolf et al., 2020) version 4.37.1. For additional experiments with Llama-3.1-8B-Instruct and Qwen3-8B (Section E.2), we use transformers version 4.57.1, which is necessary to support these newer model architectures. Experiments with models between 7B and 33B are conducted on a single NVIDIA SXM A100 80G GPU, while experiments with 70B models are conducted on 2 NVIDIA SXM A100 80G GPUs. The inference precision is `float16`. Following prior work, we conduct experiments with temperature `0` and batch size 1. The maximal generation lengths are 1024.

D.3 Retrieval Process

The retrieval process significantly affects overall acceleration. A naive implementation would be computationally prohibitive for *LogitSpec*.

Naive Approach. For each of k speculated next-next tokens, scan the entire prompt (length L) to find matching m-grams, yielding $O(k \cdot m \cdot L)$ complexity.

Our Hash-Based Optimization. We reduce this to $O(n + k)$ per decoding step:

1. **Preprocessing (amortized):** Slide a window of size m over the prompt once, building a hash table mapping each m-gram (key) to its continuation span (up to n tokens as value). Cost: $O(L)$, done once per prompt.

2. **Per-step retrieval:** For each of k next-next token candidates:

   - Construct the m-gram query: $O(m) \approx O(1)$ (since m is small, typically 3)
   - Hash table lookup: $O(1)$
   - Copy continuation span: $O(n)$

   Total per step: $O(k \cdot (1 + n)) = O(k + kn)$.

Memory Cost. The hash table requires $O(mL)$ memory, which is negligible compared to KV cache (typically several GB for long contexts).

Empirical Validation. Figure 5 shows retrieval accounts for only 1.17% of total wall-clock time, confirming our optimization is effective in practice.

D.4 Evaluation Instructions

In our experiments, we employ different instructions for different evaluation tasks. Specifically, for Spec-Bench, we use its standard instructions:

Prompt Templates for Spec-Bench

```
A chat between a curious user and an artificial intelligence assistant. The assistant gives helpful, detailed, and polite answers to the user's questions.

USER: Question

ASSISTANT:
```

For CNN/DM, we prepend “Summarize: ” to the question:

Prompt Templates for CNN/DM

```
A chat between a curious user
and an artificial intelligence
assistant. The assistant gives
helpful, detailed, and polite
answers to the user's questions.

USER: Summarize: Question

ASSISTANT:
```

For GSM8K, we follow the setting with (Liu et al., 2025) and use an 8-shot CoT for inference:

Prompt Templates for GSM8K

```
A chat between a curious user
and an artificial intelligence
assistant. The assistant gives
helpful, detailed, and polite
answers to the user's questions.

USER: 8-shot CoT Q: Question

ASSISTANT:
```

For HumanEval, we add a simple instruction "Please help me to complete this code, just output your codes directly.":

Prompt Templates for HumanEval

```
A chat between a curious user
and an artificial intelligence
assistant. The assistant gives
helpful, detailed, and polite
answers to the user's questions.

USER: Please help me to complete
this code, just output your codes
directly. Question

ASSISTANT:
```

D.5 Dataset Configurations

In our experiments, we evaluate *LogitSpec* on four categories of text generation tasks, including Spec-Bench, CNN/DM, GSM8K, and HumanEval. For Spec-Bench and HumanEval, we use the full data for evaluation. For CNN/DM and GSM8K, we randomly sample 1000 questions for evaluation. The maximal generation length is set as 1024 across all the experiments.

D.6 Model Configurations

In our experiments, all the models are deployed with precision `float16`. The `<|eos token|>` matches the tokenizer's `<|eos token|>`. To effectively alleviate the overhead of rolling back the KV cache to the accepted draft tokens, we follow Medusa (Cai et al., 2024) to allocate continuous GPU memory for all the KV cache.

Ethics Statement

This work adheres to the ACL Code of Ethics. Our study does not involve human subjects or personally identifiable information, and we only use publicly available datasets under their respective licenses. We transparently report our methods and potential risks and do not recommend deployment in high-stakes settings without further safety assessments.

E More In-depth Analysis of *LogitSpec*

E.1 Comparison with Draft-Model-Based Methods

For completeness, we also compare *LogitSpec* against the draft-model-based methods EAGLE (Li et al., 2025c) and Medusa (Cai et al., 2024) on Spec-Bench with Vicuna-7B. As shown in Table 5, *LogitSpec* attains a speedup comparable to EAGLE and clearly above Medusa, while being entirely training-free and requiring no extra parameters.

Table 5: Comparison with draft-model-based methods on Spec-Bench with **Vicuna-7B**. *LogitSpec* is training-free, while EAGLE and Medusa require training extra heads.

Method	MAT	Speedup	Note
EAGLE	3.58	2.10×	Requires training & extra heads
Medusa	2.31	1.71×	Requires training & extra heads
LogitSpec	**2.44**	2.01×	**Training-free & plug-and-play**

E.2 More Results on Different LLM Backbones

Comparison with State-of-the-Art Retrieval-Based Methods. To position *LogitSpec* against recent advances in retrieval-based speculative decoding, we compare with SAM Decoding (Hu et al., 2024) and Token Recycling (Luo et al., 2024). Both methods are plug-and-play and do not require draft models, making them directly comparable to *LogitSpec*.

Table 7 shows results on Llama-3.1-8B-Instruct across AIME, LongBench, and MATH. *LogitSpec* achieves 7.5% higher average speedup (2.58× vs 2.40× for Token Recycling, 2.34× for SAM). The key advantage comes from *LogitSpec*'s next-next token speculation:

- **vs. Token Recycling**: Token Recycling reuses previously rejected draft tokens via an adjacency matrix, which is query-dependent and requires cross-query information. In contrast, *LogitSpec* is query-independent and uses the target model's own predictive capability to guide retrieval, making it more robust when query patterns are diverse.
- **vs. SAM Decoding**: SAM uses suffix automata for efficient n-gram matching from a static corpus. While this improves matching speed, it still faces the fundamental challenge of low match rates when context repetition is limited. *LogitSpec* addresses this by speculating next-next tokens, effectively expanding the retrieval space beyond observed n-grams.

Notably, on AIME (a challenging reasoning dataset with low repetition), *LogitSpec* achieves 3.09× speedup compared to 2.82× for Token Recycling, demonstrating the value of predictive guidance when retrieval alone struggles.

Here, we provide additional results for different LLM backbones to provide more insights into *LogitSpec*, including Llama 2 chat series models, LLaMA-3.1-8B-Instruct, and Qwen3-8B.

We present the results of Llama 2 chat series models on Spec-Bench, CNN/DM, GSM8K, and HumanEval in Table 8 and Table 9. *LogitSpec* consistently achieves the best acceleration among all baselines across Spec-Bench, CNN/DM, GSM8K, and HumanEval, further validating its effectiveness. For completeness, Table 6 reports the per-subtask Spec-Bench breakdown on Vicuna 7B/13B/33B, complementing the summary statistics in the main text.

E.3 Per-Task Spec-Bench Breakdown on Vicuna

Per-task Analysis. The detailed breakdown reveals important insights into *LogitSpec*'s robustness across diverse tasks. In the Translation task, where almost no reference is available to serve as draft tokens, PLD and Lookahead achieve poor acceleration. Even equipped with a large external database (≈ 12 GB), REST only achieves 1.17× ∼ 1.31× speedup, while *LogitSpec* achieves 1.38× ∼ 1.43× speedup without external databases, demonstrating the prediction ability of the last logit on the next next token. In contrast, on tasks with high context repetition such as Summarization and RAG, *LogitSpec* achieves even stronger performance, highlighting how next-next token speculation complements retrieval-based approaches across varying task characteristics.

Table 7 compares *LogitSpec* against SAM Decoding and Token Recycling on AIME, LongBench, and MATH with the Llama-3.1-8B-Instruct backbone under identical decoding settings ($T = 0$, $K = 64$, greedy sampling). *LogitSpec* consistently attains the best MAT and speedup across all datasets.

We also present more results applying LLaMA-3.1-8B-Instruct and Qwen3-8B on Spec-Bench, CNN/DM, GSM8K, and HumanEval in Tables 10 and 11. We observe that compared to standard autoregressive decoding, *LogitSpec* consistently achieves around 2× inference acceleration on average across these datasets.

E.4 More Results on Different Benchmarks

We provide more results for *LogitSpec* on complex math reasoning tasks including MATH (Hendrycks et al., 2021) and AIME 24 & 25 (Mathematical Association of America, 2025) datasets in Figure 6 and Table 12. For each of these benchmarks, we report the real-world speedup for each subset, the overall mean accepted tokens per decoding step (MAT), and the overall speedup. These expanded experimental results further corroborate *LogitSpec*'s effectiveness and applicability, yielding an overall MAT of 3.32 and an overall speedup of 2.78× on MATH, and up to 3.76 MAT and 3.33× speedup on AIME25. This demonstrates *LogitSpec*'s robust performance even on challenging reasoning tasks.

E.5 Sampling with Non-Zero Temperatures

Why LogitSpec Remains Effective Under Sampling. A key question is whether LogitSpec's reliance on top-k logits makes it sensitive to temperature scaling. We clarify the mechanism and provide empirical validation.

Candidate Selection is Temperature-Invariant. In our implementation, speculative next-next token candidates are selected from the **raw logits** before applying temperature. Since

Table 6: Experimental results of *LogitSpec* on Spec-Bench (Xia et al., 2024) with **Vicuna**. We report the speedup ratio on each sub task, mean accepted tokens per decoding step (MAT) and overall speedup ratio. We **bold** the best results and <u>underline</u> the suboptimal results for each backbone model.

Models	Method	MT	Trans	Sum	QA	Math	RAG	MAT	Speedup
Vicuna 7B	Lookahead	1.40	1.14	1.19	1.24	1.55	1.09	1.66	1.27
	REST	1.63	<u>1.31</u>	1.36	<u>1.66</u>	1.21	<u>1.73</u>	1.82	1.48
	PLD	1.64	1.04	<u>2.43</u>	1.14	<u>1.61</u>	1.71	1.73	<u>1.59</u>
	SpS	<u>1.66</u>	1.13	1.62	1.49	1.47	1.55	<u>2.28</u>	1.49
	LogitSpec	**1.92**	**1.39**	**2.68**	**1.70**	**2.22**	**1.87**	**2.44**	**2.01**
Vicuna 13B	Lookahead	1.30	1.06	1.20	1.12	1.48	1.12	1.63	1.22
	REST	1.52	<u>1.17</u>	1.37	**1.53**	1.19	1.55	1.82	1.38
	PLD	1.47	1.02	<u>2.19</u>	1.03	<u>1.57</u>	<u>1.71</u>	1.68	1.48
	SpS	<u>1.60</u>	1.13	1.68	1.39	1.53	1.67	<u>2.18</u>	<u>1.49</u>
	LogitSpec	**1.89**	**1.43**	**2.33**	<u>1.43</u>	**2.23**	**1.93**	**2.32**	**1.93**
Vicuna 33B	Lookahead	1.32	1.08	1.20	1.06	1.54	1.15	1.61	1.24
	REST	1.63	1.27	1.45	**1.61**	1.30	1.61	1.80	1.48
	PLD	1.44	1.06	<u>2.00</u>	1.07	1.55	1.45	1.55	1.42
	SpS	<u>1.75</u>	<u>1.28</u>	1.76	<u>1.53</u>	<u>1.69</u>	<u>1.68</u>	<u>2.01</u>	<u>1.61</u>
	LogitSpec	**1.77**	**1.38**	**2.15**	1.37	**2.00**	**1.69**	**2.13**	**1.76**

Table 7: Comparison with stronger retrieval-based baselines on **Llama-3.1-8B-Instruct**. We report MAT and end-to-end speedup on AIME, LongBench, and MATH, as well as the macro average. All methods share the same hardware, decoding parameters (temperature $T = 0$), and tree capacity $K = 64$.

Method	**AIME**		**LongBench**		**MATH**		**Average**	
	MAT	Speedup	MAT	Speedup	MAT	Speedup	MAT	Speedup
AR (Greedy)	1.00	1.00×	1.00	1.00×	1.00	1.00×	1.00	1.00×
SAM Decoding	3.11	2.73×	3.01	2.04×	2.66	2.26×	2.92	2.34×
Token Recycling	3.10	2.82×	2.96	1.97×	2.80	2.40×	2.96	2.40×
LogitSpec	**3.57**	**3.09×**	**3.08**	**2.10×**	**3.05**	**2.56×**	**3.23**	**2.58×**

temperature scaling is a monotonic transformation, it preserves ranking:

$$\begin{aligned} z_i > z_j \quad &\iff \quad \exp(z_i/T) > \exp(z_j/T) \\ &\iff \quad p_i(T) > p_j(T), \quad \forall T > 0 \end{aligned} \tag{9}$$

where $p(T) = \text{softmax}(z/T)$. Therefore, the top-$k$ candidate set used for speculation is **identical** across all temperatures and does not "fill with noise tokens" as T increases.

What Changes: Acceptance Probability. Higher temperatures make the actual sampled trajectory deviate more from the most-likely path, reducing acceptance rates. This is an inherent property of all speculative decoding methods under sampling, not specific to LogitSpec.

To study the robustness of *LogitSpec* under stochastic decoding, we evaluate performance across temperatures $T \in \{0.2, 0.4, 0.6, 0.8\}$ on AIME, LongBench, and MATH with Llama-3.1-8B-Instruct. As shown in Table 13, *LogitSpec* maintains strong performance even at higher temperatures, still achieving speedups above $1.8\times$ even at $T = 0.8$. This demonstrates that *LogitSpec* remains effective across diverse generation scenarios.

Table 8: Experimental results of *LogitSpec* on CNN/DM (Nallapati et al., 2016), GSM8K (Cobbe et al., 2021) and HumanEval (Chen et al., 2021) with **Llama-2**. We report the mean accepted tokens per decoding step (MAT) and overall speedup ratio. We **bold** the best results and underline the suboptimal results for each backbone model.

Models	Method	CNN/DM		GSM8K		HumanEval		Overall	
		MAT	Speedup	MAT	Speedup	MAT	Speedup	MAT	Speedup
Llama 2 7B	Lookahead	1.58	1.36	2.02	1.67	1.77	1.53	1.79	1.52
	REST	1.71	1.33	1.51	1.18	1.97	1.52	1.73	1.34
	PLD	1.89	1.73	3.32	2.98	1.57	1.41	2.26	2.04
	SpS	1.99	1.46	2.83	1.87	2.13	1.51	2.32	1.61
	LogitSpec	**2.41**	**2.02**	**4.44**	**3.68**	**2.17**	**1.75**	**3.01**	**2.48**
Llama 2 13B	Lookahead	1.56	1.18	2.08	1.52	1.84	1.66	1.83	1.45
	REST	1.71	1.34	1.53	1.26	1.96	1.63	1.73	1.41
	PLD	1.89	1.52	3.24	2.54	1.73	1.63	2.29	1.90
	SpS	1.95	1.34	2.87	1.85	2.33	1.76	2.38	1.65
	LogitSpec	**2.43**	**2.03**	**4.31**	**3.24**	**2.38**	**2.10**	**3.04**	**2.46**
Llama 2 70B	Lookahead	1.53	1.28	1.90	1.57	1.86	1.57	1.76	1.47
	REST	1.67	1.35	1.63	1.32	1.96	1.66	1.75	1.44
	PLD	1.98	1.74	1.63	1.46	1.62	1.49	1.74	1.56
	SpS	2.01	1.71	1.98	1.69	2.21	1.70	2.07	1.70
	LogitSpec	**2.67**	**2.10**	**2.37**	**1.87**	**2.33**	**1.93**	**2.46**	**1.97**

Table 9: Experimental results of *LogitSpec* on Spec-Bench (Xia et al., 2024) with **Llama-2**. We report the speedup ratio on each sub task, mean accepted tokens per decoding step (MAT) and overall speedup ratio. We **bold** the best results and underline the suboptimal results for each backbone model.

Models	Method	MT	Trans	Sum	QA	Math	RAG	MAT	Speedup
Llama 2 7B	Lookahead	1.55	1.44	1.42	1.48	1.69	1.46	1.69	1.51
	REST	1.58	1.20	1.38	1.61	1.31	1.55	1.83	1.48
	PLD	1.46	1.34	1.89	1.23	1.65	1.58	1.59	1.49
	SpS	1.57	1.38	1.55	1.46	1.55	1.60	2.07	1.53
	LogitSpec	**1.83**	**1.72**	**2.19**	**1.63**	**2.15**	**1.94**	**2.15**	**1.87**
Llama 2 13B	Lookahead	1.39	1.36	1.21	1.39	1.69	1.25	1.68	1.37
	REST	1.53	1.14	1.30	1.51	1.23	1.45	1.85	1.42
	PLD	1.36	1.19	1.58	1.16	1.62	1.37	1.56	1.37
	SpS	1.52	1.26	1.43	1.42	1.54	1.50	2.03	1.48
	LogitSpec	**1.73**	**1.57**	**1.86**	**1.53**	**2.14**	**1.73**	**2.12**	**1.74**
Llama 2 70B	Lookahead	1.45	1.35	1.28	1.38	1.71	1.31	1.66	1.41
	REST	1.63	1.33	1.38	**1.67**	1.35	1.55	1.83	1.53
	PLD	1.34	1.32	1.76	1.18	1.63	1.47	1.51	1.39
	SpS	1.65	1.50	1.62	1.57	1.70	1.68	1.88	1.63
	LogitSpec	**1.66**	**1.58**	**1.95**	1.58	**2.03**	**1.78**	**2.12**	**1.72**

E.6 Cross-Lingual Next-Next Token Coverage

To validate the universality of the last logit's predictive capability across different linguistic structures, we conduct experiments on the MMMLU benchmark covering five languages: English, Chinese, Japanese, German, and French. Table 14 reports the fraction of ground-truth next-next to-

Table 10: Experimental results of *LogitSpec* on Spec-Bench (Xia et al., 2024) with **LLaMA-3.1-8B-Instruct and Qwen3-8B**. We report the speedup ratio on each sub task, mean accepted tokens per decoding step (MAT) and overall speedup ratio. We **bold** the best results for each backbone model.

Model	**Method**	**MT**	**Trans**	**Sum**	**QA**	**MATH**	**RAG**	**MAT**	**Speedup**
Llama-3.1-8B-Instruct	Vanilla	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
Llama-3.1-8B-Instruct	***LogitSpec***	**1.89**	**1.67**	**1.94**	**1.68**	**2.01**	**1.77**	**2.11**	**1.88**
Qwen-3-8B	Vanilla	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
Qwen-3-8B	***LogitSpec***	**1.71**	**1.74**	**1.64**	**1.65**	**1.89**	**1.68**	**1.95**	**1.75**

Table 11: Experimental results of *LogitSpec* on CNN/DM (Nallapati et al., 2016), GSM8K (Cobbe et al., 2021) and HumanEval (Chen et al., 2021) with **LLaMA-3.1-8B-Instruct and Qwen3-8B**. We report the mean accepted tokens per decoding step (MAT) and overall speedup ratio. We **bold** the best results for each backbone model.

Model	Method	**CNN/DM**		**GSM8K**		**Humaneval**		**Overall**	
		MAT	Speedup	MAT	Speedup	MAT	Speedup	MAT	Speedup
Llama-3.1-8B-Instruct	Vanilla	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
Llama-3.1-8B-Instruct	***LogitSpec***	**2.04**	**1.85**	**2.18**	**1.95**	**2.63**	**2.31**	**2.28**	**2.04**
Qwen-3-8B	Vanilla	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
Qwen-3-8B	***LogitSpec***	**1.77**	**1.59**	**2.18**	**1.94**	**2.18**	**1.92**	**2.04**	**1.82**

Table 12: Results of *LogitSpec* on AIME datasets.

Method	AIME24		AIME25	
	MAT	Speedup	MAT	Speedup
Vanilla	1.00	1.00	1.00	1.00
LogitSpec	3.41	3.25	3.76	3.33

kens appearing in the top-k logits for each language. The results show consistent coverage across all languages: at $k = 64$, over 50% of next-next tokens fall within the top-k predictions for every language. This demonstrates that our observation about the last logit's predictive capability is not restricted to English or specific linguistic structures, but rather represents a fundamental property of modern LLMs that generalizes across diverse languages with different word orders and morphological characteristics.

These results show that the last logit's predictive capability is not restricted to English or specific linguistic structures. The consistency across languages with markedly different characteristics—Chinese (logographic, subject-verb-object), Japanese (agglutinative, subject-object-verb), German (inflectional, flexible word order), and French (Romance, subject-verb-object)—suggests this is a fundamental property of modern LLMs' internal representations rather than a language-specific artifact. Combined with our diverse task coverage (reasoning, QA, summarization, translation, code generation), this cross-lingual validation provides strong evidence for the generality of our core observation.

E.7 More Results on Long-Context Scenarios

We conduct a new set of experiments on LongBench using Llama-3.1-8B-Instruct as the backbone model, randomly sampling 100 problems for evaluation. We observe that *LogitSpec* is highly effective in these long-context scenarios. As shown in Figure 7, our method achieves an overall MAT of 3.09 and an overall speedup of 2.01. This strong performance is directly linked to the core mechanics of our method. *LogitSpec*'s retrieval model is built from the user-input prompt and previously decoded tokens. A longer context generally provides a richer database for this retrieval process. However, a large context also increases the chance of finding ambiguous or incorrect n-grams, which is precisely where *LogitSpec*'s "next next token speculation" offers a significant advantage. By using a more specific multi-token query, it effectively disambiguates the retrieval process, which is particularly crucial in a long context with many repetitive

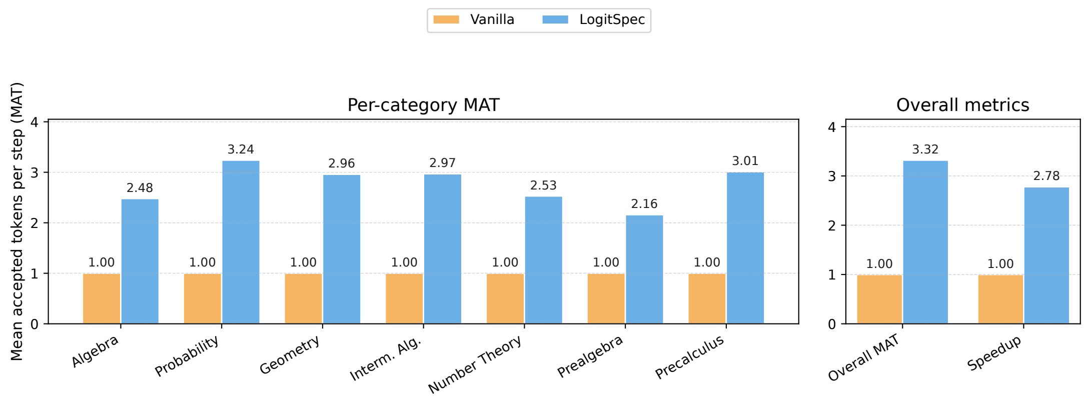

Figure 6: LogitSpec performance on MATH benchmark subsets.

Table 13: LogitSpec under different sampling temperatures on Llama-3.1-8B-Instruct. Draft candidates are drawn from raw logits before temperature scaling, so the speculative candidate set is unchanged across T; higher T only changes the sampled path.

Temperature	AIME MAT	AIME Speedup	LongBench MAT	LongBench Speedup	MATH MAT	MATH Speedup
0.0	3.57	3.09×	3.08	2.10×	3.05	2.56×
0.2	3.37	2.85×	2.94	1.94×	3.20	2.64×
0.4	3.51	2.96×	2.92	1.93×	3.03	2.55×
0.6	3.26	2.76×	2.87	1.89×	2.91	2.46×
0.8	3.15	2.66×	2.84	1.86×	2.81	2.42×

Table 14: Fraction of ground-truth next-next tokens that appear within the top-k logits of the last token on MMMLU subsets (Llama-3.1-8B-Instruct). Each language averages 20 questions.

Language	$k = 4$	$k = 8$	$k = 16$	$k = 32$	$k = 64$
English	15.25%	22.02%	33.60%	42.06%	51.83%
Chinese	17.14%	24.21%	36.29%	45.68%	55.07%
Japanese	15.16%	24.30%	34.72%	46.17%	55.87%
German	14.64%	21.44%	32.87%	42.42%	50.64%
French	14.86%	21.77%	32.21%	42.78%	52.22%

phrases. Furthermore, our retrieval implementation was designed for efficiency, using a hash table to ensure that the overhead remains negligible (around 1.17% of the total decoding time) even as the sequence length grows. Therefore, these new results confirm that *LogitSpec* is a robust and effective solution for accelerating inference in demanding long-context scenarios.

E.8 Ablation Study on Different Pruning Strategies

In Section 4.2, we apply a heuristic pruning strategy to control the number of draft tokens. The role of the pruning algorithm is to effectively control the size of the resulting draft tree, keeping the decoding overhead low without sacrificing too many possibilities, which is a strategy to balance breadth and cost. To provide more insight into the pruning strategy, we conduct ablation studies in Table 15. Specifically, we consider two different pruning strategies:

(a) Strategy 1: A rank-based heuristic: if the rank is <4, we preserve 5 tokens; <8, 4 tokens; <16, 3 tokens; <32, 2 tokens; else 1 token.

(b) Strategy 2: A simple heuristic: preserving 4 tokens for all speculated next next tokens.

The results in Table 15 demonstrate that while different pruning strategies have some effect on the final performance, the overall performance of *LogitSpec* is quite robust. The speedup remains stable at approximately 1.9× across these different approaches.

We also would like to clarify that the core contribution of our work is **next next token speculation** using the last logit to guide and enhance the efficiency and accuracy of retrieval-based speculative decoding. The primary purpose of the pruning algorithm is to serve as an auxiliary module for our core mechanism.

E.9 Ablation Study on Draft Tree Capacity

As mentioned in Section 4.2, we set the capacity of the draft tree to K=64. To provide further in-

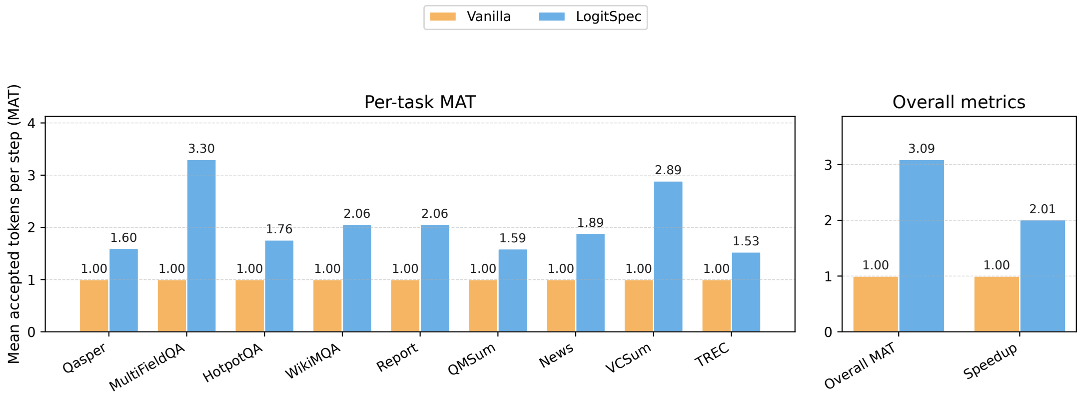

Figure 7: LogitSpec performance on LongBench subsets.

Table 15: Ablation study on pruning strategies. *LogitSpec* $_{with\ s1}$ and *LogitSpec* $_{with\ s2}$ denote *LogitSpec* with Strategy 1 and Strategy 2, respectively.

Method	MT	Trans	Sum	QA	MATH	RAG	MAT	Speedup
LogitSpec $_{with\ s1}$	**1.90**	**1.68**	**1.94**	1.66	**2.04**	**1.78**	**2.12**	**1.88**
LogitSpec $_{with\ s2}$	1.86	1.67	1.87	**1.68**	2.03	1.74	2.02	1.85
***LogitSpec* (ori)**	1.89	1.67	**1.94**	**1.68**	2.01	1.77	2.11	**1.88**

sight, we evaluate a range of K values from 32 to 128 in Table 16. These results reveal a clear trade-off: as K increases, the MAT per step improves (from 2.03 to 2.30), since a larger tree offers more opportunities for token acceptance. However, the overall speedup ratio peaks at K=64 (1.92×) and subsequently declines. This is because verifying an overly large draft tree introduces significant computational overhead that ultimately negates the advantages of a higher acceptance rate.

E.10 Real-World Example

To provide more insights into *LogitSpec*, we present a real-world example to illustrate how "next next token speculation" allows *LogitSpec* to succeed where standard retrieval methods fail.

Taking the prefix

```
Q: A pen costs as much as a pencil and
    eraser combined. A pencil costs $1
    .20 and an eraser costs $0.30. How
    much will 8 pens cost?
A: To find the cost of 8 pens, we first
    need to find the cost of one pen. A
    pencil costs $1.20 and an eraser
    costs $0.30. The
```

At the first step, *LogitSpec* generates the next token "combined" and then speculates the next next token with the following high-probability candidates:

- .
- total
- cost
- combination
- pen
- eraser

Then, *LogitSpec* extends each candidate with retrieved n-grams and verifies these draft sequences:

- [×] . A pencil costs
- [×] total they had
- [✓] cost of one pen
- [×] combination (no matched n-grams retrieved)
- [×] pen.
- [×] eraser costs $

Table 16: Ablation study on pruning hyperparameters K using LLaMA-3.1-8B-Instruct as the backbone model.

Method	MT	Trans	Sum	QA	MATH	RAG	MAT	Speedup
Vanilla	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
LogitSpec $_{32}$	1.79	1.62	1.82	1.56	1.93	1.67	1.95	1.79
LogitSpec $_{48}$	1.86	1.66	1.90	1.67	2.00	1.73	2.04	1.85
LogitSpec $_{64}$	**1.89**	**1.67**	**1.94**	**1.68**	**2.01**	**1.77**	2.11	**1.88**
LogitSpec $_{80}$	1.85	1.64	1.86	1.67	1.84	1.74	2.14	1.83
LogitSpec $_{96}$	1.81	1.65	1.83	1.66	1.84	1.73	2.17	1.80
LogitSpec $_{112}$	1.79	1.61	1.77	1.64	1.83	1.69	2.21	1.77
LogitSpec $_{128}$	1.74	1.62	1.73	1.58	1.81	1.62	**2.22**	1.73

Finally, with the guidance of the last logit, we successfully accept 3 draft tokens and generate "the combined cost of one pen". In the next decoding step, *LogitSpec* again accepts 3 draft tokens and generates "the combined cost of one pencil and one eraser". However, without last-logit speculation, standard retrieval can only retrieve the first n-gram "A pencil costs".

F Discussion: Integration with Production Frameworks

vLLM Integration Feasibility. While our current implementation uses Hugging Face transformers, several reviewers asked about integration with production inference frameworks like vLLM. We discuss technical feasibility:

Tree Attention Support. vLLM has recently added native tree attention support:

- Flash attention kernel for tree structures: https://github.com/vllm-project/flash-attention/pull/81
- Official tree attention backend: https://docs.vllm.ai/en/latest/api/vllm/v1/attention/backends/tree_attn/

Compatibility with vLLM's Architecture. LogitSpec's draft tree can be naturally represented as multiple sequences with shared prefixes, aligning with vLLM's batched inference paradigm. Each branch corresponds to a different draft sequence, similar to the prefill process for multiple sequences. The nano-PEARL repository demonstrates a similar integration approach for tree-based verification in vLLM.

Minimal Overhead. Beyond tree attention, LogitSpec only requires:

- The last logit at each step (already computed by vLLM)
- CPU-based retrieval and tree construction (can run in parallel with GPU inference)

We conclude that there are no fundamental algorithmic obstacles to deploying LogitSpec in vLLM; the remaining work is engineering integration, which we leave to future work.

G LLM Usage

We used a large language model (LLM)–based writing assistant solely for grammar and wording improvements on draft text. The LLM did not generate research ideas, claims, proofs, algorithms, code, figures, or analyses, and it did not have access to any non-public data. All edits suggested by the LLM were manually reviewed and either accepted or rewritten by the authors, who take full responsibility for the final content. The LLM is not an author of this paper.