LONGSPEC: Long-Context Lossless Speculative Decoding with Efficient Drafting and Verification

Penghui Yang[2]*, Cunxiao Du[1]*, Fengzhuo Zhang[3], Haonan Wang[3], Tianyu Pang[1], Chao Du[1], Bo An[2]

[1] Sea AI Lab, [2] Nanyang Technological University, [3] National University of Singapore

phyang.cs@gmail.com, cnsdunm@gmail.com, fzzhang@u.nus.edu

Abstract

As Large Language Models (LLMs) can now process extremely long contexts, efficient inference over these extended inputs has become increasingly important, especially for emerging applications like LLM agents that highly depend on this capability. Speculative decoding (SD) offers a promising lossless acceleration technique compared to lossy alternatives such as quantization and model cascades. However, most state-of-the-art SD methods are trained on short texts (typically fewer than 4k tokens), making them unsuitable for long-context scenarios. Specifically, adapting these methods to long contexts presents three key challenges: (1) the excessive memory demands posed by draft models due to large Key-Value (KV) cache; (2) performance degradation resulting from the mismatch between short-context training and long-context inference; and (3) inefficiencies in tree attention mechanisms when managing long token sequences. This work introduces LONGSPEC, a framework that addresses these challenges through three core innovations: a memory-efficient draft model with a constant-sized KV cache; novel position indices that mitigate the training–inference mismatch; and an attention aggregation strategy that combines fast prefix computation with standard tree attention to enable efficient decoding. Experimental results confirm the effectiveness of LONGSPEC, achieving up to a 3.26× speedup over strong `Flash Attention` baselines across five long-context understanding datasets, as well as a 2.34× reduction in wall-clock time on four math reasoning tasks with the QwQ model, demonstrating significant latency improvements for long-context applications.

1 Introduction

Large Language Models (LLMs) have demonstrated remarkable capabilities (Achiam et al., 2023), and their ability to handle extensive contexts is becoming crucial for emerging applications such as LLM agents and long reasoning tasks (Tan et al., 2025; Guo et al., 2025), which now operate over context windows extending to millions of tokens (Team et al., 2024). In these demanding long-context scenarios, the high inference latency of standard autoregressive decoding becomes a pronounced bottleneck. While various acceleration techniques such as quantization (Lin et al., 2024), sparse attention (Li et al., 2024b), and model cascades (Gupta et al., 2024) have been proposed to mitigate this problem, they often compromise the output quality, rendering them lossy solutions. In contrast, speculative decoding (SD) (Leviathan et al., 2023) offers a **lossless** acceleration strategy by using a smaller draft model to propose token sequences, which are then verified in parallel by the larger target model. However, state-of-the-art (SoTA) SD methods (Li et al., 2024c), which often rely on a small and standalone draft model, are mainly designed and evaluated on short-context data, typically with sequences shorter than 4k tokens (see Figure 1). Although some existing SD methods can be extended to longer contexts, they

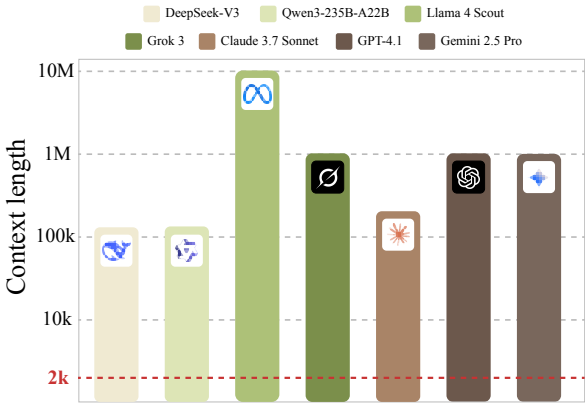


Figure 1: The SoTA SD method, EAGLE, has a training context length of 2048, which is significantly shorter than the context lengths of modern LLMs.

*Equal contribution. Work done during Penghui Yang's associate membership at Sea AI Lab and when Cunxiao Du was at Sea AI Lab.

often use the full target model with a compressed Key-Value (KV) cache as the draft model (Chen et al., 2025b; Tiwari et al., 2025). These approaches avoid the overhead of training a dedicated draft model, but their reliance on full target models, which are not sufficiently lightweight, limits the speed of draft generation. As a result, these methods may underperform compared to SoTA short-context SD techniques. This divergence raises a critical question:

Why cannot SoTA SD methods for short contexts be directly applied to long sequences?

In response to this question, we attribute the difficulty of directly adapting effective short-context SoTA SD techniques to long-context settings to three emergent challenges:

1. **Architecture:** In SoTA SD methods (*e.g.*, EAGLE (Li et al., 2024c)), the draft model's KV cache still grows linearly with context length. This linear growth becomes a prohibitive memory bottleneck as the context length increases.

2. **Training:** Language model training typically relies on plentiful short-sequence data, while long-sequence data remains relatively scarce. The imbalance of training data makes it difficult for the model to generalize to longer contexts. To address this, conventional wisdom in training long-context LLMs employs length extrapolation, in particular by extending the Rotary Position Embedding (RoPE) (Su et al., 2024) base to accommodate longer contexts (Gao et al., 2024; Liu et al., 2024c; Peng et al., 2024). However, this solution is not directly applicable to SoTA SD draft models, because their RoPE base must match that of the target model[1], which is fixed and already scaled for long-context scenarios.

3. **Inference:** The effectiveness of tree attention verification (Miao et al., 2024; Cai et al., 2024) diminishes in long-context scenarios. In particular, common inference optimizations for long-context scenarios are primarily designed to handle regular, structured attention masks and are not optimized for arbitrary or unstructured attention masks. As a result, potential speedups from speculation may be lower than expected.

To address these challenges, we introduce LONGSPEC, a comprehensive framework for efficient long-context lossless speculative decoding. LONGSPEC overcomes the aforementioned obstacles through three key innovations:

1. **Memory-Efficient Architecture.** We propose a draft model architecture with constant memory usage regardless of context length, effectively resolving the scalability limitations of prior SoTA autoregressive draft models.

2. **Effective Training Regimes.** We develop a novel training strategy involving Anchor-Offset Indices, enabling draft models trained on short sequences to robustly generalize to much longer contexts at inference time.

3. **Fast Tree Attention.** We introduce Hybrid Tree Attention, a new computation method that significantly speeds up tree verification by decomposing attention calculations and leveraging optimized Triton kernels.

Experiments on five long-context understanding datasets using five LLMs as target models show that our LONGSPEC can significantly reduce the long-context inference latency, achieving up to a 3.26× speedup over strong baselines with `Flash Attention`[2], and up to a 7× speedup over common baselines using the HuggingFace implementation. Additional experiments on four math reasoning datasets with the long reasoning model QwQ (Qwen, 2024) further validate the effectiveness of LONGSPEC, yielding a 2.34× speedup in wall-clock time. Furthermore, our proposed Anchor-Offset Indices enable models to reach the same loss level 3.93× faster, and our Hybrid Tree Attention reduces attention computation latency by approximately 75% compared to the standard HuggingFace implementation.

2 Related Work

Speculative decoding offers a promising approach to accelerating LLMs without compromising the quality of their outputs[3]. Early efforts (Xia et al., 2023; Leviathan et al., 2023; Liu et al., 2024d; Bae et al., 2023; Liu et al., 2025a) rely on existing

[1]SoTA SD techniques often require the draft model to utilize intermediate features (*e.g.*, hidden states or KV cache) from the target model, which is crucial for providing richer information from the target model, enabling the draft model to better align with and predict the target model's outputs. See more explanations in Appendix B.

[2]In this paper, `Flash Attention` refers to the inference optimization technique `FlashDecoding` (Dao et al., 2023), implemented via the `flash_attn_with_kvcache` function from the Flash Attention library (Dao, 2022).

[3]While this paper focuses on original speculative decoding methods which are lossless, some recent works explore lossy speculative decoding (see Appendix A for a brief overview).

smaller LLMs to generate draft sequences. Some other methods aim to improve upon those early efforts (Sun et al., 2023; Miao et al., 2024; Chen et al., 2024). There are also some works using part of the target model as the draft model (Liu et al., 2024a; Zhang et al., 2024; Elhoushi et al., 2024; Xia et al., 2025). Retrieval-based speculative decoding methods (Fu et al., 2024; He et al., 2024; Zhao et al., 2024; Liu et al., 2025b; Shen et al., 2026) offer an alternative by utilizing N-gram matching rather than relying on smaller models. These approaches bypass the need for additional model training, leveraging pre-existing data patterns to construct draft sequences efficiently.

More recent advancements (Cai et al., 2024; Li et al., 2024c; Du et al., 2024; Huang et al., 2025a) have expanded on these foundations by designing specialized draft models and introducing tree speculation and verification techniques. These methods leverage customized draft models tailored for speculative decoding, achieving higher efficiency and performance. Additionally, the tree-based approaches employed in these methods allow for more adaptive and parallelizable decoding processes, paving the way for broader applications in real-world systems, including vision-language models (Huang et al., 2025b).

Although speculative decoding has progressed significantly for conventional context lengths, only a few existing papers focus on lossless speculative decoding in long-context scenarios. TriForce (Sun et al., 2024) introduces a three-layer speculative decoding system that is scalable for long sequence generation. MagicDec (Chen et al., 2025b) uses speculative decoding to improve both the throughput and latency of LLM inference. QuantSpec (Tiwari et al., 2025) employs a hierarchical 4-bit quantized KV cache and 4-bit quantized weights for draft models. However, these methods mainly utilize the target model with the sparse KV cache as the draft model. The computation-intensive draft models restrict the practical usage of these methods when facing various batch sizes. In contrast, our work focuses on efficiently building a draft model with only one transformer block, achieving more effective performance across different scenarios.

3 Methodology

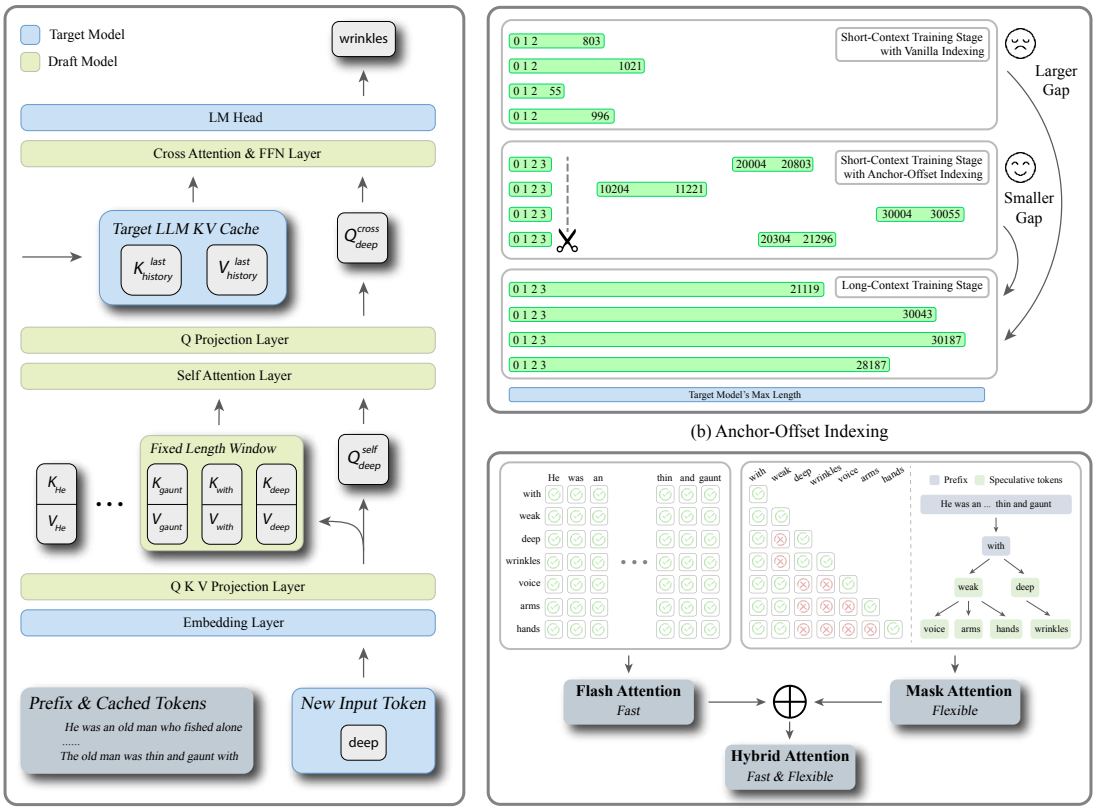
In this section, we present our framework LONGSPEC for Long-Context Speculative Decoding, which addresses three key challenges by (1) designing a lightweight draft model architecture with constant-sized memory overhead, (2) devising the training strategy with anchor-offset indices to handle long contexts effectively, and (3) implementing a fast attention aggregation mechanism that leverages tree-based speculation and verification for practical usage.

3.1 Memory-Efficient Architecture

In previous work, the success of the SoTA model EAGLE depends on two key factors: (1) the hidden states provided by the target model, and (2) its autoregressive structure. However, an autoregressive draft model inevitably requires maintaining its own KV cache, which introduces additional overhead during long-context inference and demands substantial GPU memory, especially for tasks such as LLM agents and long reasoning that involve producing large amounts of output.

To avoid this extra memory overhead, we propose a draft model with constant memory usage independent of context length. As illustrated in Figure 2(a), our model consists of two components: the self-attention module and the following cross-attention module. The self-attention module focuses on modeling local context, while the cross-attention module captures long-range dependencies. To restrict memory usage, we apply a sliding-window attention mechanism to the self-attention module, a technique widely adopted in modern LLMs (Beltagy et al., 2020). Hence, during inference, the self-attention does not exceed the window size, which we set to 512.

For the cross-attention component, inspired by GliDe (Du et al., 2024), we leverage the KV cache of the target model (see Appendix B for a detailed explanation of how this benefits the draft model). This design not only enables better modeling of previous information but also completely removes additional storage overhead for long contexts, since the large model's KV cache must be stored regardless of whether or not speculative decoding is employed. Different from GliDe, we also share the weights of the Embedding Layer and LM Head between the target and draft models, which substantially reduces memory consumption for large-vocabulary LLMs such as LLaMA-3 (vocabulary size: 128,256) (Dubey et al., 2024) and Qwen-2.5 (vocabulary size: 152,064) (Yang et al., 2024).



(a) Memory-Efficient Architecture

(b) Anchor-Offset Indexing

(c) Hybrid Tree Attention

Figure 2: Illustration of the memory-efficient draft model, the Anchor-Offset Indices, and the Hybrid Tree Attention. (a) We use a sliding window self-attention layer to capture the local context information and a cross-attention layer to gather long-context information. (b) The differences between the vanilla indexing and the Anchor-Offset Indices. By introducing a randomly selected offset and some anchor indices, the Anchor-Offset Indices enable the short-context training stage to seamlessly integrate with the long-context training stage. (c) The Hybrid Tree Attention combines the advantages of `Flash Attention` and our Triton-implemented Attention.

3.2 Effective Training Regimes

Anchor-Offset Indices. With vanilla position indices, which consist of successive integers starting from 0, those indices appearing earlier in sequences occur more frequently than larger position indices (An et al., 2025), as shown in Figure 2(b) upper part. Consequently, larger position indices receive insufficient training updates, which leads to a training-inference discrepancy. As we point out in Section 1, the common RoPE-based extrapolation cannot be directly used here because the RoPE base is fixed once the target model is chosen. To leverage the target model's KV cache, our draft model must keep the RoPE base the same as the target model. To tackle this challenge, we can only leverage carefully designed indices. These indices must ensure that (1) the position indices in the draft model can be sufficiently trained using short-context data and (2) the indices would not cause the target model to exhibit out-of-distribution behavior because the target model shares the same indices as the draft model during training.

To satisfy these constraints, we propose the Anchor-Offset Indices strategy. Specifically, we reserve the first four positions $[0, 1, 2, 3]$ as *attention sink* tokens[4], then assign all subsequent tokens to large consecutive indices starting at a random offset (*e.g.*, $[0, 1, 2, 3, 8192, 8193, 8194, \dots]$). By exploiting the *attention sink* phenomenon, we believe that utilizing Anchor-Offset Indices can naturally lead the target model to exhibit in-distribution behavior. The anchor indices and random offset ensure that every position index can be sufficiently trained, addressing the limitation of the vanilla one that repeatedly trains only smaller indices. In our experiments, adopting these indices in the target model only increases the loss by approximately 0.001, indicating that the target model is indeed well-suited to such changes. Pseudo code can be

[4]According to (Xiao et al., 2024), LLM exhibits an *attention sink* phenomenon when dealing with long texts, which means the attention weights primarily concentrate on the first four tokens and the recent tokens.

found in Appendix G.

Flash Noisy Training. During training, our draft model leverages the KV cache from a large model, while this KV cache is not always visible during inference. This is because the large model only updates its KV cache upon verification completion. Concretely, for the t-th cross-attention query Q_t in the draft model, we can only guarantee access to the corresponding key-value states $K_{<t'}$, $V_{<t'}$ satisfying $1 \leq |t' - t| < \gamma$, where γ is the number of speculative steps.

To ensure consistency between training and inference, a straightforward solution would be to add an attention mask (Du et al., 2024). However, this method is incompatible with `Flash Attention`, which would significantly degrade training speed and cause prohibitive memory overhead, particularly in long-context training scenarios. Therefore, we propose a technique called **flash noisy training**. During training, we randomly shift the indices of queries and key-value states with $1 \leq j < \gamma$. Suppose the sequence length is l, then we compute

$$O_{\geq j} = \texttt{attn}\big(Q_{\geq j},\, K_{<l-j},\, V_{<l-j}\big).$$

In this way, we effectively simulate the same visibility constraints as in the inference phase, *i.e.*, $1 \leq |t' - t| < \gamma$, thereby aligning the behavior at training time with the inference behavior. When using Flash Noisy Training, we observe a 14.7% increase in acceptance length compared to training without it, with improvements most concentrated on the final speculated tokens. This highlights its role in mitigating the training-inference gap. Pseudo code can be found in Appendix G.

3.3 Fast Tree Attention

Tree Speculative Decoding (Miao et al., 2024) leverages speculation trees and the causal structure of LLMs so that a draft model can propose multiple candidate sequences, while the target model only needs to verify them once, without altering the final results. In this process, *Tree Attention* plays a key role in ensuring both correctness and efficiency. Early works (Cai et al., 2024; Li et al., 2024c) apply attention masks derived from prefix trees to the $QK^\top$ attention matrix, thus disabling wrong combinations between speculation tokens. However, these methods only run on PyTorch's eager execution mode, precluding more advanced attention kernels such as `Flash Attention`. As a result, the inference speed decreases significantly when the sequence length increases.

To address these performance bottlenecks, we propose a **Hybrid Tree Attention** mechanism, as illustrated in Figure 2(c). Our method is based on two key insights: 1) when performing Tree Attention, as illustrated in the left part of Figure 2(c), the queries and the cached key-value pairs $\{K_{\text{cache}}, V_{\text{cache}}\}$ do not require additional masks; 2) only the queries and the key-value pairs $\{K_{\text{specs}}, V_{\text{specs}}\}$ from the current speculative tokens need masking as illustrated in the right part of Figure 2(c), and the number of such speculative tokens is typically small. Based on these observations, we adopt a divide and aggregate approach that splits the attention computation into two parts and merges them afterward.

Splitting Key-Value Pairs. We partition all key-value pairs into two groups: $\{K_{\text{cache}}, V_{\text{cache}}\}$: the cached part of the main sequence, which requires no attention mask; and $\{K_{\text{specs}}, V_{\text{specs}}\}$: the speculation-stage part, which needs attention masks. For $\{K_{\text{cache}}, V_{\text{cache}}\}$, we invoke the efficient `Flash Attention` kernel. For $\{K_{\text{specs}}, V_{\text{specs}}\}$, we use our custom Triton kernel `fused_mask_attn`, which applies blockwise loading and masking in the KV dimension, enabling fast computation of attention. This step yields two sets of attention outputs $\{O_{\text{cache}}, O_{\text{specs}}\}$ along with their corresponding denominators (*i.e.*, log-sum-exp of all attention scores) $\{\text{LSE}_{\text{cache}}, \text{LSE}_{\text{specs}}\}$.

Aggregation. We then combine these two parts into the final attention output O_{merge} via a log-sum-exp trick. First, we compute

$$\text{LSE}_{\text{merge}} = \log\Big(\exp\big(\text{LSE}_{\text{cache}}\big) + \exp\big(\text{LSE}_{\text{specs}}\big)\Big),$$

and then apply a weighted summation to the two outputs:

$$\begin{aligned} O_{\text{merge}} = {} & O_{\text{cache}} \cdot \exp\big(\text{LSE}_{\text{cache}} - \text{LSE}_{\text{merge}}\big) \\ & + O_{\text{specs}} \cdot \exp\big(\text{LSE}_{\text{specs}} - \text{LSE}_{\text{merge}}\big). \end{aligned}$$

The theoretical guarantee is provided in Appendix C. As outlined above, this hybrid approach employs the highly efficient `Flash Attention` kernel for most of the computations in long-sequence inference and only uses a custom masking attention `fused_mask_attn` for the small number of speculative tokens. The kernel `fused_mask_attn` follows the design philosophy of `Flash Attention 2` (Dao et al., 2023) by splitting Q, K_{specs}, and V_{specs} into small blocks. This strategy reduces global memory I/O and fully leverages GPU streaming multiprocessors. Furthermore, for each

block in the computation of $QK^{\top}_{\text{specs}}$, the mask matrix is loaded and used to apply the masking operation. The Hybrid Tree Attention effectively balances the parallel verification of multiple branches with improved inference speed, all without compromising correctness.

4 Experiments

4.1 Settings

Target and draft models. We select four widely-used long-context LLMs, Vicuna (including 7B and 13B) (Chiang et al., 2023), LongChat (including 7B and 13B) (Li et al., 2023), LLaMA-3.1-8B-Instruct (Dubey et al., 2024), and QwQ-32B (Qwen, 2024), as target models. In order to make the draft model and target model more compatible, our draft model is consistent with the target model in various parameters, such as the number of KV heads.

Training Process. We first train our draft model with Anchor-Offset Indices on the SlimPajama-6B pretraining dataset (Soboleva et al., 2023). The random offset is set as a random integer from 0 to 15k for Vicuna models and LongChat-7B, and 0 to 30k for the other three models because they have longer maximum context length. Then we train our model on a small subset of the Prolong-64k long-context dataset (Gao et al., 2024) in order to gain the ability to handle long texts. Finally, we finetune our model on a self-built long-context supervised-finetuning (SFT) dataset to further improve the model performance. The position index of the last two stages is the vanilla indexing policy because the training data is sufficiently long. We apply flash noisy training during all three stages to mitigate the training and inference inconsistency and the extra overhead of flash noisy training is negligible. More details on model training can be found in Appendix D.

Test Benchmarks. For conventional long-context understanding tasks, we select tasks from the LongBench benchmark (Bai et al., 2024) that involve generating longer outputs, because tasks with shorter outputs, such as document-QA, make it challenging to measure the speedup ratio fairly with speculative decoding. Specifically, we focus on long-document summarization and code completion tasks and conduct tests on five datasets: GovReport (Huang et al., 2021), QMSum (Zhong et al., 2021), Multi-News (Fabbri et al., 2019), LCC (Guo et al., 2023), and RepoBench-P (Liu et al., 2024b). For math reasoning tasks, we test QwQ-32B on four math reasoning datasets: AIME24 (Li et al., 2024a), AMC (Li et al., 2024a), MATH500 (Hendrycks et al., 2021), and Minerva Math (Lewkowycz et al., 2022).

We compare our method with the original target model, PLD (Saxena, 2023), and MagicDec (Chen et al., 2025b). PLD is the most popular retrieval-based method (also known as n-gram SD in vLLM (Kwon et al., 2023)), and MagicDec is a simple prototype of TriForce. To highlight the significance of `Flash Attention` in long-context scenarios, we also present the performance of the original target model using both eager attention implemented by HuggingFace and `Flash Attention` for comparison. To make a fair comparison, we also use `Flash Attention` for baseline MagicDec. The most important metric for speculative decoding is the *walltime speedup ratio*, which is the actual test speedup ratio relative to vanilla autoregressive decoding. We also test the *average acceptance length* τ, *i.e.*, the average number of tokens accepted per forward pass of the target LLM.

4.2 Main Results

Table 1 and Figure 3 show the decoding speeds and average acceptance lengths across the five evaluated datasets at $T = 0$ and $T = 1$, where T denotes the temperature used in LLM sampling. Our proposed method significantly outperforms all other approaches on both summarization tasks and code completion tasks. When $T = 0$, on summarization tasks, our method can achieve an average acceptance length of around 3.5 and a speedup of up to 2.67×; and on code completion tasks, our method can achieve an average acceptance length of around 4 and a speedup of up to 3.26×. This highlights the robustness and generalizability of our speculative decoding approach, particularly in long-text generation tasks. At $T = 1$, our method achieves around 2.5× speedup, maintaining a substantial lead over MagicDec. This indicates that our approach is robust across different temperature settings, further validating its soundness and efficiency.

Although PLD can accelerate generation on many datasets, it still does not match the performance of our proposed LongSpec. In some scenarios (*e.g.*, when retrieval is minimal), PLD can even result in negative acceleration. For another baseline, MagicDec, while it demonstrates competitive acceptance rates compared to LongSpec, its speedup is noticeably lower in our experiments. This is because MagicDec is primarily designed

Table 1: Average acceptance length τ, decoding speed (tokens/s), and speedups across different models and settings. Specifically, "Vanilla HF" refers to HuggingFace's PyTorch-based attention implementation, while "Vanilla FA" employs `Flash Attention`. The speedup statistic calculates the acceleration ratio relative to the Vanilla HF method. For the analysis of the reasons for the low speedup ratio of MagicDec, see Section 4.2 and 4.5. All results are computed at $T = 0$.

	Setting	GovReport			QMSum			Multi-News			LCC			RepoBench-P		
		τ	Tokens/s	Speedup	τ	Tokens/s	Speedup	τ	Tokens/s	Speedup	τ	Tokens/s	Speedup	τ	Tokens/s	Speedup
V-7B	Vanilla HF	1.00	25.25	-	1.00	18.12	-	1.00	27.29	-	1.00	25.25	-	1.00	19.18	-
	Vanilla FA	1.00	45.76	1.00×	1.00	43.68	1.00×	1.00	55.99	1.00×	1.00	54.07	1.00×	1.00	46.61	1.00×
	MagicDec	2.23	41.68	0.91×	2.29	42.91	0.98×	2.31	44.82	0.80×	2.52	46.96	0.87×	2.57	48.75	1.05×
	PLD	2.20	73.91	1.62×	1.22	39.08	0.89×	2.15	72.31	1.29×	2.43	78.41	1.45×	2.23	74.15	1.59×
	LongSpec	**3.57**	**102.23**	**2.23×**	**3.14**	**88.87**	**2.04×**	**3.51**	**100.55**	**1.80×**	**3.73**	**107.30**	**1.99×**	**3.86**	**110.76**	**2.38×**
V-13B	Vanilla HF	1.00	17.25	-	1.00	11.86	-	1.00	18.81	-	1.00	17.25	-	1.00	13.44	-
	Vanilla FA	1.00	28.52	1.00×	1.00	27.43	1.00×	1.00	35.01	1.00×	1.00	33.87	1.00×	1.00	29.14	1.00×
	MagicDec	2.95	38.24	1.34×	2.87	37.15	1.35×	2.97	39.47	1.13×	2.96	38.40	1.13×	2.94	36.66	1.26×
	PLD	1.37	32.10	1.13×	1.28	28.29	1.03×	1.35	34.97	1.00×	1.34	36.24	1.07×	1.32	30.60	1.05×
	LongSpec	**3.31**	**71.08**	**2.49×**	**2.76**	**57.15**	**2.08×**	**3.44**	**78.20**	**2.23×**	**3.57**	**81.00**	**2.39×**	**3.59**	**77.22**	**2.65×**
LC-7B	Vanilla HF	1.00	25.27	-	1.00	14.11	-	1.00	27.66	-	1.00	25.27	-	1.00	17.02	-
	Vanilla FA	1.00	42.14	1.00×	1.00	36.87	1.00×	1.00	50.19	1.00×	1.00	54.17	1.00×	1.00	42.69	1.00×
	MagicDec	2.26	41.90	0.99×	2.20	40.82	1.11×	2.32	43.94	0.88×	2.77	51.73	0.96×	2.57	44.13	1.03×
	PLD	2.10	68.66	1.63×	1.24	36.58	0.99×	2.00	67.66	1.35×	2.48	85.62	1.58×	2.71	89.22	2.09×
	LongSpec	**3.59**	**101.43**	**2.41×**	**3.06**	**85.23**	**2.31×**	**3.41**	**97.93**	**1.95×**	**4.21**	**122.30**	**2.26×**	**4.03**	**115.27**	**2.70×**
LC-13B	Vanilla HF	1.00	17.72	-	1.00	12.08	-	1.00	18.74	-	1.00	17.72	-	1.00	13.85	-
	Vanilla FA	1.00	28.56	1.00×	1.00	27.18	1.00×	1.00	35.37	1.00×	1.00	34.58	1.00×	1.00	29.74	1.00×
	MagicDec	2.40	31.37	1.10×	2.38	30.84	1.13×	2.43	32.58	0.92×	2.68	35.77	1.03×	2.85	35.67	1.20×
	PLD	1.67	35.35	1.24×	1.18	24.10	0.89×	1.85	43.74	1.24×	1.88	49.12	1.42×	1.80	41.07	1.38×
	LongSpec	**3.58**	**76.26**	**2.67×**	**3.15**	**64.41**	**2.37×**	**3.50**	**80.48**	**2.28×**	**4.01**	**90.92**	**2.63×**	**4.46**	**96.96**	**3.26×**
L-8B	Vanilla HF	1.00	21.59	-	1.00	18.67	-	1.00	29.91	-	1.00	29.48	-	1.00	22.77	-
	Vanilla FA	1.00	53.14	1.00×	1.00	51.22	1.00×	1.00	56.94	1.00×	1.00	56.73	1.00×	1.00	54.08	1.00×
	MagicDec	2.04	36.14	0.68×	2.00	35.78	0.70×	2.33	39.57	0.70×	2.65	46.95	0.83×	2.61	44.39	0.82×
	PLD	2.08	77.45	1.46×	1.52	45.76	0.89×	1.94	78.00	1.37×	1.59	54.75	0.97×	1.38	45.70	0.85×
	LongSpec	**3.25**	**84.57**	**1.59×**	**2.99**	**75.68**	**1.48×**	**3.36**	**91.11**	**1.60×**	**3.28**	**89.33**	**1.57×**	**3.39**	**91.28**	**1.69×**

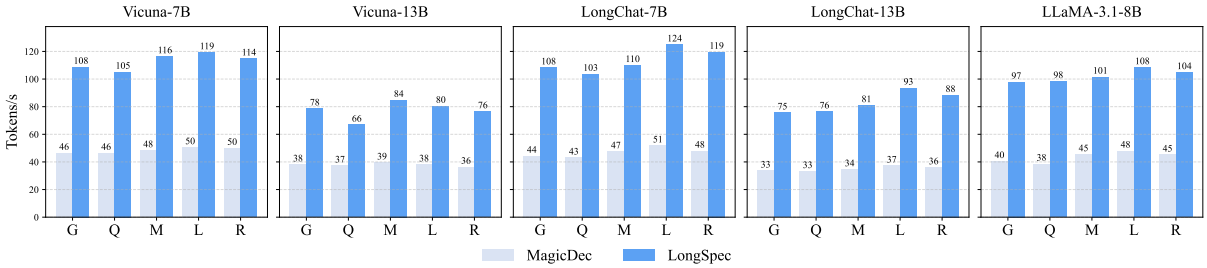


Figure 3: Decoding speed (tokens/s) across different models and settings. All results are computed at $T = 1$. The letters G, Q, M, L, and R on the horizontal axis represent the datasets GovReport, QMSum, Multi-News, LCC, and RepoBench-P respectively.

for scenarios with large batch sizes and tensor parallelism. In low-batch-size settings, its draft model, which leverages all parameters of the target model with a sparse KV cache, becomes excessively heavy. This design choice leads to inefficiencies, as the draft model's computational overhead outweighs its speculative benefits. Our results reveal that MagicDec only achieves acceleration ratios > 1 on partial datasets when using a guess length $\gamma = 2$ and consistently exhibits negative acceleration around $0.7\times$ when $\gamma \geq 3$, further underscoring the limitations of this method in such configurations. The performance of MagicDec in larger batch sizes can be found in Section 4.5.

Lastly, we find that attention implementation plays a critical role in long-context speculative decoding performance. In our experiments, "Vanilla HF" refers to HuggingFace's attention implementation, while "Vanilla FA" employs `Flash Attention`. The latter demonstrates nearly a $2\times$ speedup over the former, even as a standalone component, and our method can achieve up to $6\times$ speedup over HF Attention on code completion datasets. This result underscores the necessity for speculative decoding methods to be compatible with optimized attention mechanisms like `Flash Attention`, especially in long-text settings. Our hybrid tree attention approach achieves this compatibility, allowing us to fully leverage the advantages of `Flash Attention` and further speedup.

Table 2: Performance comparison with and without Anchor-Offset Indices on the Multi-News and RepoBench-P datasets. Models with Anchor-Offset Indices achieve higher output speed and larger acceptance length, highlighting their efficiency and effectiveness.

	Multi-News		**RepoBench-P**	
	τ	Tokens/s	τ	Tokens/s
w/o Anchor-Offset	3.20	85.98	3.26	85.21
w/ Anchor-Offset	3.36	91.11	3.39	91.28

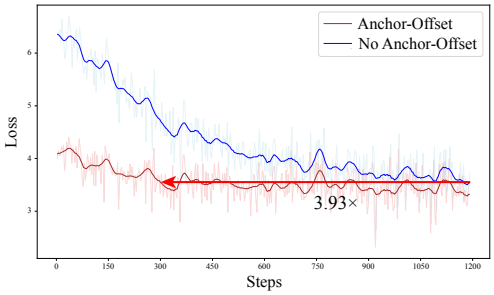


Figure 4: Training loss curves on long-context data. Pretrained models with Anchor-Offset Indices exhibit lower initial and final loss, and reach the same loss level 3.93× faster compared to models without Anchor-Offset Indices.

4.3 Ablation Studies

Anchor-Offset Indices. The experimental results demonstrate the significant benefits of incorporating the Anchor-Offset Indices. Figure 4 shows that the model trained with Anchor-Offset Indices achieves a lower initial loss and final loss compared to the one trained without them when training on the real long-context dataset. Notably, the initialization with Anchor-Offset Indices reaches the same loss level 3.93× faster than its counterpart. Table 2 further highlights the performance improvements across two datasets, a summary dataset Multi-News, and a code completion dataset RepoBench-P. Models with Anchor-Offset Indices exhibit faster output speed and larger average acceptance length τ. These results underscore the effectiveness of Anchor-Offset Indices in enhancing both training efficiency and model performance.

Hybrid Tree Attention. The results presented in Figure 5 highlight the effectiveness of the proposed Hybrid Tree Attention, which combines `Flash Attention` with the Triton kernel `fused_mask_attn`. While the time spent on the draft model forward pass and the target model FFN computations remain comparable across the two methods, the hybrid approach exhibits a significant

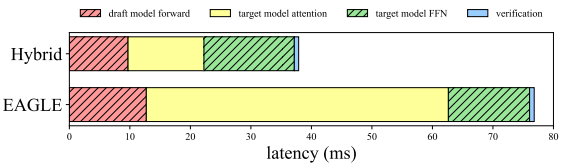


Figure 5: Latency breakdown for a single speculative decoding loop comparing the EAGLE implementation and the proposed Hybrid Tree Attention. Significant latency reduction is observed in the target model's attention layer (the yellow part) using our approach.

Table 3: Performance of our method on the QwQ-32B model on four math reasoning datasets, using a maximum output length of 32k tokens. The table shows the tokens generated per second and the mean number of accepted tokens τ, where our approach achieves about 2.34× higher speed compared to the baseline on average and an average of 3.81 acceptance tokens.

Dataset	Metric	Vanilla	LongSpec	Improvement
AIME24	τ	1.00	3.82	3.82×
	Tokens/s	18.92	42.63	2.25×
AMC	τ	1.00	3.81	3.81×
	Tokens/s	19.41	45.16	2.33×
Minerva	τ	1.00	3.65	3.65×
	Tokens/s	19.46	44.51	2.29×
MATH500	τ	1.00	3.95	3.95×
	Tokens/s	19.59	48.36	2.47×

reduction in latency for the target model's attention layer (the yellow part). Specifically, the attention computation latency decreases from 49.92 ms in the HF implementation to 12.54 ms in the hybrid approach, resulting in an approximately 75% improvement. The verification step time difference is minimal, further solidifying the conclusion that the primary performance gains stem from optimizing the attention mechanism.

4.4 Long Reasoning Acceleration

Long reasoning tasks have gained significant attention recently due to their ability to enable models to perform complex reasoning and problem-solving over extended outputs (Qwen, 2024; OpenAI, 2024). In these tasks, while the prefix input is often relatively short, the generated output can be extremely long, posing unique challenges in terms of efficiency and token acceptance. Our method is particularly well-suited for addressing these challenges, effectively handling scenarios with long outputs. It is worth mentioning that MagicDec is

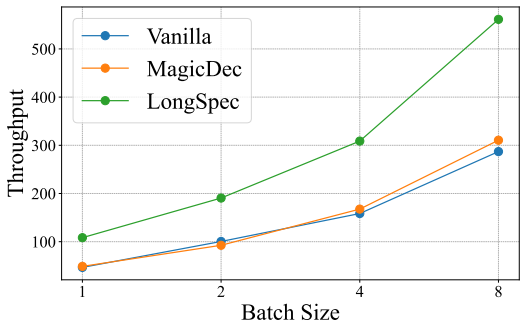


Figure 6: Throughput comparison of Vanilla, MagicDec, and LONGSPEC.

not suitable for such long-output scenarios because the initial inference stage of the long reasoning task is not the same as the traditional long-context task. In long reasoning tasks, where the prefix is relatively short, the draft model in MagicDec will completely degrade into the target model, failing to achieve acceleration.

We evaluate our method on the QwQ-32B model using four widely used benchmarks with a maximum output length set to 32k tokens. The results, illustrated in Table 3, demonstrate a significant improvement in both generation speed and average acceptance tokens. Specifically, our method achieves a generation rate of around 45 tokens/s, 2.34× higher than the strong `Flash Attention` baseline, and an average of 3.81 average acceptance tokens. Notably, QwQ-32B with LONGSPEC achieves even lower latency than the standard 7B model with `Flash Attention`, demonstrating that our method effectively accelerates the long reasoning model. These findings not only highlight the effectiveness of our method in the long reasoning task but also provide new insights into lossless inference acceleration for the o1-like model. We believe speculative decoding will play a crucial role in accelerating this type of model in the future.

4.5 Throughput

As illustrated in Figure 6, the throughput results of Vicuna-7B on the RepoBench-P dataset show that LONGSPEC consistently outperforms both Vanilla and MagicDec across all batch sizes. At a batch size of 8, LONGSPEC achieves a throughput of 561.32 tokens/s, approximately 1.8× higher than MagicDec (310.58 tokens/s) and nearly 2× higher than Vanilla (286.96 tokens/s). MagicDec, designed with throughput optimization in mind, surpasses Vanilla as the batch size increases, reflecting its targeted improvements. However, LONGSPEC still sustains its advantage, maintaining superior throughput across all tested batch sizes.

5 Conclusion

In this paper, we propose LONGSPEC, a novel framework designed to enhance lossless speculative decoding for long-context scenarios. Unlike previous speculative decoding methods that primarily focus on short-context settings, LONGSPEC directly addresses three key challenges: excessive memory overhead, inadequate training for large position indices, and inefficient tree attention computation. To mitigate memory constraints, we introduce an efficient draft model architecture that maintains a constant memory footprint by leveraging a combination of sliding window self-attention and cache-free cross-attention. To resolve the training limitations associated with short-context data, we propose the Anchor-Offset Indices, ensuring that large positional indices are sufficiently trained even within short-sequence datasets. Finally, we introduce Hybrid Tree Attention, which efficiently integrates tree-based speculative decoding with `Flash Attention`. Extensive experiments demonstrate the effectiveness of LONGSPEC in long-context understanding tasks and real-world long reasoning tasks. Our findings highlight the importance of designing speculative decoding methods specifically tailored for long-context settings and point to promising directions for future research in efficient large-scale language model inference.

Limitations

While LONGSPEC demonstrates strong performance in accelerating long-context inference, it is currently designed exclusively for decoder-only transformer-based LLMs. This restricts its applicability to architectures such as RNNs, where speculative decoding and KV cache reuse are not directly compatible with our method. Future work could extend LONGSPEC to a broader range of model architectures and assess performance variability across more diverse settings.

References

Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, and 1 others. 2023. GPT-4 technical report. *arXiv preprint arXiv:2303.08774*.

Chenxin An, Jun Zhang, Ming Zhong, Lei Li, Shansan Gong, Yao Luo, Jingjing Xu, and Lingpeng Kong.

2025. Why does the effective context length of LLMs fall short? In *Proceedings of the International Conference on Learning Representations*.

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. In *Proceedings of the International Conference on Learning Representations*.

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. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*.

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*.

Iz Beltagy, Matthew E Peters, and Arman Cohan. 2020. Longformer: The long-document transformer. *arXiv preprint arXiv:2004.05150*.

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 International Conference on Machine Learning*.

Guanzheng Chen, Qilong Feng, Jinjie Ni, Xin Li, and Michael Qizhe Shieh. 2025a. Long-context inference with retrieval-augmented speculative decoding. In *Proceedings of the International Conference on Machine Learning*.

Jian Chen, Vashisth Tiwari, Ranajoy Sadhukhan, Zhuoming Chen, Jinyuan Shi, Ian En-Hsu Yen, and Beidi Chen. 2025b. MagicDec: Breaking the latency-throughput tradeoff for long context generation with speculative decoding. In *Proceedings of the International Conference on Learning Representations*.

Ziyi Chen, Xiaocong Yang, Jiacheng Lin, Chenkai Sun, Kevin Chang, and Jie Huang. 2024. Cascade speculative drafting for even faster LLM inference. In *Advances in Neural Information Processing Systems*.

Wei-Lin Chiang, Zhuohan Li, Zi 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.

Tri Dao. 2022. The FlashAttention package.

Tri Dao, Daniel Haziza, Francisco Massa, and Grigory Sizov. 2023. Flash-Decoding for long-context inference.

DeepSeek. 2024. DeepSeek's API context caching on disk technology.

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 International Conference on Machine Learning*.

Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, and 1 others. 2024. The LLaMA 3 herd of models. *arXiv preprint arXiv:2407.21783*.

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

Alexander Richard Fabbri, Irene Li, Tianwei She, Suyi Li, and Dragomir Radev. 2019. Multi-News: A large-scale multi-document summarization dataset and abstractive hierarchical model. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*.

Yichao Fu, Peter Bailis, Ion Stoica, and Hao Zhang. 2024. Break the sequential dependency of LLM inference using lookahead decoding. In *Proceedings of the International Conference on Machine Learning*.

Tianyu Gao, Alexander Wettig, Howard Yen, and Danqi Chen. 2024. How to train long-context language models (effectively). *arXiv preprint arXiv:2410.02660*.

Google. 2024. Gemini API context caching feature.

Daya Guo, Canwen Xu, Nan Duan, Jian Yin, and Julian McAuley. 2023. Longcoder: A long-range pre-trained language model for code completion. In *Proceedings of the International Conference on Machine Learning*.

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, and 1 others. 2025. Deepseek-r1: Incentivizing reasoning capability in LLMs via reinforcement learning. *arXiv preprint arXiv:2501.12948*.

Neha Gupta, Harikrishna Narasimhan, Wittawat Jitkrittum, Ankit Singh Rawat, Aditya Krishna Menon, and Sanjiv Kumar. 2024. Language model cascades: Token-level uncertainty and beyond. In *Proceedings of the International Conference on Learning Representations*.

Zhenyu He, Zexuan Zhong, Tianle Cai, Jason Lee, and Di He. 2024. REST: Retrieval-based speculative decoding. In *Proceedings of the Conference of the North American Chapter of the 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. *arXiv preprint arXiv:2103.03874*.

Pin-Lun Hsu, Yun Dai, Vignesh Kothapalli, Qingquan Song, Shao Tang, Siyu Zhu, Steven Shimizu, Shivam Sahni, Haowen Ning, and Yanning Chen. 2024. Liger kernel: Efficient triton kernels for LLM training. *arXiv preprint arXiv:2410.10989*.

Haiduo Huang, Fuwei Yang, Zhenhua Liu, Yixing Xu, Jinze Li, Yang Liu, Xuanwu Yin, Dong Li, Pengju Ren, and Emad Barsoum. 2025a. Jakiro: Boosting speculative decoding with decoupled multi-head via MoE. *arXiv preprint arXiv:2502.06282*.

Haiduo Huang, Fuwei Yang, Zhenhua Liu, Xuanwu Yin, Dong Li, Pengju Ren, and Emad Barsoum. 2025b. SpecVLM: Fast speculative decoding in vision-language models. *arXiv preprint arXiv:2509.11815*.

Luyang Huang, Shuyang Cao, Nikolaus Parulian, Heng Ji, and Lu Wang. 2021. Efficient attentions for long document summarization. In *Proceedings of the Conference of the North American Chapter of the Association for Computational Linguistics*.

Sehoon Kim, Karttikeya Mangalam, Suhong Moon, Jitendra Malik, Michael W Mahoney, Amir Gholami, and Kurt Keutzer. 2023. Speculative decoding with big little decoder. In *Advances in Neural Information Processing Systems*.

Diederik P Kingma and Jimmy Lei Ba. 2015. Adam: A method for stochastic gradient descent. In *Proceedings of the International Conference on Learning Representations*.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. Gonzalez, Hao Zhang, and Ion Stoica. 2023. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the ACM SIGOPS 29th Symposium on Operating Systems Principles*.

Yaniv Leviathan, Matan Kalman, and Yossi Matias. 2023. Fast inference from transformers via speculative decoding. In *Proceedings of the International Conference on Machine Learning*.

Aitor Lewkowycz, Anders Andreassen, David Dohan, Ethan Dyer, Henryk Michalewski, Vinay Ramasesh, Ambrose Slone, Cem Anil, Imanol Schlag, Theo Gutman-Solo, and 1 others. 2022. Solving quantitative reasoning problems with language models. In *Advances in Neural Information Processing Systems*.

Dacheng Li, Rulin Shao, Anze Xie, Ying Sheng, Lianmin Zheng, Joseph E. Gonzalez, Ion Stoica, Xuezhe Ma, and Hao Zhang. 2023. How long can open-source LLMs truly promise on context length?

Jia Li, Edward Beeching, Lewis Tunstall, Ben Lipkin, Roman Soletskyi, Shengyi Huang, Kashif Rasul, Longhui Yu, Albert Q Jiang, Ziju Shen, and 1 others. 2024a. Numinamath: The largest public dataset in AI4Maths with 860k pairs of competition math problems and solutions. *Hugging Face repository*.

Yuhong Li, Yingbing Huang, Bowen Yang, Bharat Venkitesh, Acyr Locatelli, Hanchen Ye, Tianle Cai, Patrick Lewis, and Deming Chen. 2024b. SnapKV: LLM knows what you are looking for before generation. In *Advances in Neural Information Processing Systems*.

Yuhui Li, Fangyun Wei, Chao Zhang, and Hongyang Zhang. 2024c. EAGLE: Speculative sampling requires rethinking feature uncertainty. In *Proceedings of the International Conference on Machine Learning*.

Baohao Liao, Yuhui Xu, Hanze Dong, Junnan Li, Christof Monz, Silvio Savarese, Doyen Sahoo, and Caiming Xiong. 2025. Reward-guided speculative decoding for efficient LLM reasoning. In *Proceedings of the International Conference on Machine Learning*.

Ji Lin, Jiaming Tang, Haotian Tang, Shang Yang, Wei-Ming Chen, Wei-Chen Wang, Guangxuan Xiao, Xingyu Dang, Chuang Gan, and Song Han. 2024. AWQ: Activation-aware weight quantization for on-device LLM compression and acceleration. In *Proceedings of Machine Learning and Systems*.

Fangcheng Liu, Yehui Tang, Zhenhua Liu, Yunsheng Ni, Duyu Tang, Kai Han, and Yunhe Wang. 2024a. Kangaroo: Lossless self-speculative decoding for accelerating LLMs via double early exiting. In *Advances in Neural Information Processing Systems*.

Tianyang Liu, Canwen Xu, and Julian McAuley. 2024b. RepoBench: Benchmarking repository-level code auto-completion systems. In *Proceedings of the International Conference on Learning Representations*.

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

Tianyu Liu, Qitan Lv, Hao Li, Xing Gao, Xiao Sun, and Xiaoyan Sun. 2025b. LogitSpec: Accelerating retrieval-based speculative decoding via next next token speculation. *arXiv preprint arXiv:2507.01449*.

Xiaoran Liu, Hang Yan, Chenxin An, Xipeng Qiu, and Dahua Lin. 2024c. Scaling laws of roPE-based extrapolation. In *Proceedings of the International Conference on Learning Representations*.

Xiaoxuan Liu, Lanxiang Hu, Peter Bailis, Alvin Cheung, Zhijie Deng, Ion Stoica, and Hao Zhang. 2024d. Online speculative decoding. In *Proceedings of the International Conference on Machine Learning*.

Ilya Loshchilov and Frank Hutter. 2017. SGDR: Stochastic gradient descent with warm restarts. In *Proceedings of the International Conference on Learning Representations*.

Xianzhen Luo, Yixuan Wang, Qingfu Zhu, Zhiming Zhang, Xuanyu Zhang, Qing Yang, and Dongliang Xu. 2025. Turning trash into treasure: Accelerating inference of large language models with token recycling. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics*.

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*.

Harikrishna Narasimhan, Wittawat Jitkrittum, Ankit Singh Rawat, Seungyeon Kim, Neha Gupta, Aditya Krishna Menon, and Sanjiv Kumar. 2025. Faster cascades via speculative decoding. In *Proceedings of the International Conference on Learning Representations*.

OpenAI. 2024. Learning to reason with LLMs.

Bowen Peng, Jeffrey Quesnelle, Honglu Fan, and Enrico Shippole. 2024. YaRN: Efficient context window extension of large language models. In *Proceedings of the International Conference on Learning Representations*.

Zongyue Qin, Ziniu Hu, Zifan He, Neha Prakriya, Jason Cong, and Yizhou Sun. 2025. Multi-token joint speculative decoding for accelerating large language model inference. In *Proceedings of the International Conference on Learning Representations*.

Qwen. 2024. QwQ: Reflect deeply on the boundaries of the unknown.

Jeff Rasley, Samyam Rajbhandari, Olatunji Ruwase, and Yuxiong He. 2020. Deepspeed: System optimizations enable training deep learning models with over 100 billion parameters. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*.

Apoorv Saxena. 2023. Prompt lookup decoding.

Yuhao Shen, Tianyu Liu, Junyi Shen, Jinyang Wu, Quan Kong, Li Huan, and Cong Wang. 2026. Double: Breaking the acceleration limit via double retrieval speculative parallelism. *arXiv preprint arXiv:2601.05524*.

Daria Soboleva, Faisal Al-Khateeb, Robert Myers, Jacob R Steeves, Joel Hestness, and Nolan Dey. 2023. SlimPajama: A 627b token cleaned and deduplicated version of redpajama.

Jianlin Su, Murtadha Ahmed, Yu Lu, Shengfeng Pan, Wen Bo, and Yunfeng Liu. 2024. Roformer: Enhanced transformer with rotary position embedding. *Neurocomputing*.

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

Ziteng Sun, Ananda Theertha Suresh, Jae Hun Ro, Ahmad Beirami, Himanshu Jain, and Felix X. Yu. 2023. SpecTr: Fast speculative decoding via optimal transport. In *Advances in Neural Information Processing Systems*.

Weihao Tan, Wentao Zhang, Xinrun Xu, Haochong Xia, Ziluo Ding, Boyu Li, Bohan Zhou, Junpeng Yue, Jiechuan Jiang, Yewen Li, Ruyi An, Molei Qin, Chuqiao Zong, Longtao Zheng, Yujie Wu, Xiaoqiang Chai, Yifei Bi, Tianbao Xie, Pengjie Gu, and 9 others. 2025. Cradle: Empowering foundation agents towards general computer control. In *Proceedings of the International Conference on Machine Learning*.

Gemini Team, Petko Georgiev, Ving Ian Lei, Ryan Burnell, Libin Bai, Anmol Gulati, Garrett Tanzer, Damien Vincent, Zhufeng Pan, Shibo Wang, and 1 others. 2024. Gemini 1.5: Unlocking multimodal understanding across millions of tokens of context. *arXiv preprint arXiv:2403.05530*.

Rishabh Tiwari, Haocheng Xi, Aditya Tomar, Coleman Hooper, Sehoon Kim, Maxwell Horton, Mahyar Najibi, Michael W Mahoney, Kurt Keutzer, and Amir Gholami. 2025. QuantSpec: Self-speculative decoding with hierarchical quantized kv cache. In *Proceedings of the International Conference on Machine Learning*.

Tong Wu, Junzhe Shen, Zixia Jia, Yuxuan Wang, and Zilong Zheng. 2025. From hours to minutes: Achieving lossless acceleration in 100k-token long sequence generation. In *Proceedings of the International Conference on Machine Learning*.

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

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

Guangxuan Xiao, Yuandong Tian, Beidi Chen, Song Han, and Mike Lewis. 2024. Efficient streaming language models with attention sinks. In *Proceedings of the International Conference on Learning Representations*.

An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, and 1 others. 2024. Qwen2.5 technical report. *arXiv preprint arXiv:2412.15115*.

Jun Zhang, Jue Wang, Huan Li, Lidan Shou, Ke Chen, Gang Chen, and Sharad Mehrotra. 2024. Draft & verify: Lossless large language model acceleration via self-speculative decoding. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics*.

Weilin Zhao, Yuxiang Huang, Xu Han, Wang Xu, Chaojun Xiao, Xinrong Zhang, Yewei Fang, Kaihuo Zhang, Zhiyuan Liu, and Maosong Sun. 2024. Ouroboros: Generating longer drafts phrase by phrase for faster speculative decoding. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*.

Ming Zhong, Da Yin, Tao Yu, Ahmad Zaidi, Mutethia Mutuma, Rahul Jha, Ahmed Hassan, Asli Celikyilmaz, Yang Liu, Xipeng Qiu, and 1 others. 2021. QMSum: A new benchmark for query-based multi-domain meeting summarization. In *Proceedings of the Conference of the North American Chapter of the Association for Computational Linguistics*.

A Related Work about Lossy Speculative Decoding

While original speculative decoding methods are mainly lossless, some recent works try to relax the constraints and explore lossy speculative decoding. For instance, BiLD (Kim et al., 2023) employs a small model for autoregressive text generation, with a larger model occasionally invoked non-autoregressively to refine inaccurate predictions, thereby achieving speedups with minimal quality degradation. Narasimhan et al. (Narasimhan et al., 2025) introduce speculative cascading, a method that integrates cascade-style deferral rules with speculative execution to yield better cost-quality trade-offs than either approach alone. Another approach, MTAD (Qin et al., 2025), uses a smaller auxiliary model to approximate the multi-token joint distribution of a larger model, enhancing both inference speed and output effectiveness by accepting a bounded error in this approximation. To address the rejection of high-quality but non-aligned draft tokens, Bachmann et al. (Bachmann et al., 2025) propose adapting the verification step by training a compact "judge" module to recognize valid continuations even without perfect target model alignment, significantly boosting acceptance rates and speed. RSD (Liao et al., 2025) incorporates a process reward model to evaluate intermediate decoding steps, dynamically deciding target model invocation and introducing a controlled bias towards high-reward outputs to optimize the cost-quality trade-off. RAPID (Chen et al., 2025a) employs a RAG-based approach on shortened contexts as its drafter. TokenSwift (Wu et al., 2025) comprehensively uses LLMs with partial KV cache and N-gram tables to accelerate ultra-long sequence generation (up to 100k tokens) while reducing computation time from hours to minutes.

B The Intuition of Why KV Cache Can Help

The KV cache stores the contextual information the model accumulates while processing previous tokens. When predicting the next token, the target model relies on three components: the KV cache (contextual memory), input word embeddings, and model parameters.

In our method, the draft model already shares the input embeddings with the target model, so the primary differences in their predictions stem from the KV cache and internal parameters. By allow-

ing the draft model to use the KV cache generated by the target model, we eliminate another source of variation. As a result, the only remaining difference between their predictions comes from the model parameters. This sharing aligns the draft model's predictions more closely with those of the target model, as it removes discrepancies caused by differing contextual representations.

C Correctness for Attention Aggregation

Because the query matrix Q can be decomposed into several rows, each representing a separate query q, we can only consider the output of each row's q after calculating attention with KV. In this way, we can assume that the KV involved in the calculation has undergone the tree mask, which can simplify our proof. We only need to prove that the output o obtained from each individual q meets the requirements, which can indicate that the overall output O of the entire matrix Q also meets the requirements.

Proposition C.1. *Denote the log-sum-exp of the merged attention as follows:*

$$\mathrm{LSE}_{\mathrm{merge}} = \log\Big(\exp\big(\mathrm{LSE}_{\mathrm{cache}}\big) + \exp\big(\mathrm{LSE}_{\mathrm{specs}}\big)\Big),$$

Then we can write the merged attention output in the following way:

$$\begin{aligned} o_{\mathrm{merge}} =& o_{\mathrm{cache}} \cdot \exp\big(\mathrm{LSE}_{\mathrm{cache}} - \mathrm{LSE}_{\mathrm{merge}}\big) \\ &+ o_{\mathrm{specs}} \cdot \exp\big(\mathrm{LSE}_{\mathrm{specs}} - \mathrm{LSE}_{\mathrm{merge}}\big). \end{aligned}$$

Proof. A standard scaled dot-product attention for q (of size d_{qk}) attending to K_{merge} and V_{merge} (together of size $(M+N)\times d_{qk}$ and $(M+N)\times d_v$ respectively) can be written as:

$$\begin{aligned} o_{\mathrm{merge}} &= \mathsf{mha}\left(q, K_{\mathrm{merge}}, V_{\mathrm{merge}}\right) \\ &= \mathsf{softmax}\left(qK_{\mathrm{merge}}^{\top}/\sqrt{d_{qk}}\right) V_{\mathrm{merge}}. \end{aligned}$$

Because K and V are formed by stacking $(K_{\mathrm{specs}}, K_{\mathrm{cache}})$ and $(V_{\mathrm{specs}}, V_{\mathrm{cache}})$, we split the logit matrix accordingly:

$$qK_{\mathrm{merge}}^{\top}/\sqrt{d_{qk}} = \mathsf{concat}\Big(\underbrace{q\,K_{\mathrm{cache}}^{\top}/\sqrt{d_{qk}}}_{\text{sub-logits for history}}, \underbrace{q\,K_{\mathrm{specs}}^{\top}/\sqrt{d_{qk}}}_{\text{sub-logits for new}}\Big).$$

Denote these sub-logit matrices as:

$$\begin{aligned} Z_{\mathrm{cache}} &= q\,K_{\mathrm{cache}}^{\top}/\sqrt{d_{qk}}, \\ Z_{\mathrm{specs}} &= q\,K_{\mathrm{specs}}^{\top}/\sqrt{d_{qk}}. \end{aligned}$$

Each row i of Z_{specs} corresponds to the dot products between the i-th query in q and all rows in K_{specs}, while rows of Z_{cache} correspond to the same query but with K_{cache}.

In order to combine partial attentions, we keep track of the log of the sum of exponentials of each sub-logit set. Concretely, define:

$$\begin{aligned} \mathrm{LSE}_{\mathrm{cache}} &= \log\left(\sum\nolimits_{j=1}^{N} \exp\left(Z_{\mathrm{cache}}^{(j)}\right)\right), \\ \mathrm{LSE}_{\mathrm{specs}} &= \log\left(\sum\nolimits_{j=1}^{M} \exp\left(Z_{\mathrm{specs}}^{(j)}\right)\right), \end{aligned} \quad (1)$$

where $Z_{\mathrm{specs}}^{(j)}$ denotes the logit for the j-th element, and similarly for $Z_{\mathrm{cache}}^{(j)}$.

Then o_{cache} and o_{specs} can be written as:

$$\begin{aligned} o_{\mathrm{cache}} &= \frac{\sum_{j=1}^{N} \exp\left(Z_{\mathrm{cache}}^{(j)}\right) V_{\mathrm{cache}}^{(j)}}{\exp\left(\mathrm{LSE}_{\mathrm{cache}}\right)}, \\ o_{\mathrm{specs}} &= \frac{\sum_{j=1}^{M} \exp\left(Z_{\mathrm{specs}}^{(j)}\right) V_{\mathrm{specs}}^{(j)}}{\exp\left(\mathrm{LSE}_{\mathrm{specs}}\right)}. \end{aligned} \quad (2)$$

And the whole attention score can be written as:

$$\begin{aligned} N_{\mathrm{num}} &= \sum_{j=1}^{N} \exp\big(Z_{\mathrm{cache}}^{(j)}\big) V_{\mathrm{cache}}^{(j)} \\ &\quad + \sum_{j=1}^{M} \exp\big(Z_{\mathrm{specs}}^{(j)}\big) V_{\mathrm{specs}}^{(j)}, \\ D_{\mathrm{den}} &= \exp\big(\mathrm{LSE}_{\mathrm{cache}}\big) + \exp\big(\mathrm{LSE}_{\mathrm{specs}}\big), \\ o_{\mathrm{merge}} &= \frac{N_{\mathrm{num}}}{D_{\mathrm{den}}}. \end{aligned} \quad (3)$$

By aggregating Equation 2 into Equation 3, we can get the following equation:

$$\begin{aligned} o_{\mathrm{merge}} =& o_{\mathrm{cache}} \cdot \exp\big(\mathrm{LSE}_{\mathrm{cache}} - \mathrm{LSE}_{\mathrm{merge}}\big) \\ &+ o_{\mathrm{specs}} \cdot \exp\big(\mathrm{LSE}_{\mathrm{specs}} - \mathrm{LSE}_{\mathrm{merge}}\big). \end{aligned} \quad (4)$$

□

D Experiments Details

All models are trained using eight A100 80GB GPUs. For the 7B, 8B, and 13B target models trained on short-context data, we employ LONGSPEC with ZeRO-1 (Rasley et al., 2020). For the 7B, 8B, and 13B models trained on long-context data, as well as for all settings of the 33B target models, we utilize ZeRO-3.

Standard cross-entropy is used to optimize the draft model while the parameters of the target model are kept frozen. To mitigate the VRAM peak caused by the computation of the logits, we use a fused-linear-and-cross-entropy loss implemented by the Liger Kernel (Hsu et al., 2024), which computes the LM head and the softmax function together and can greatly alleviate this problem.

For the SlimPajama-6B dataset, we configure the batch size (including accumulation) to 2048, set the maximum learning rate to 5e-4 with a cosine learning rate schedule (Loshchilov and Hutter, 2017), and optimize the draft model using AdamW (Kingma and Ba, 2015). When training on long-context datasets, we adopt a batch size of 256 and a maximum learning rate of 5e-6. The draft model is trained for only one epoch on all datasets.

It is important to note that the primary computational cost arises from forwarding the target model to obtain the KV cache. Recently, some companies have introduced a service known as context caching (DeepSeek, 2024; Google, 2024), which involves storing large volumes of KV cache. Consequently, in real-world deployment, these pre-stored KV caches can be directly utilized as training data, significantly accelerating the training process.

For the tree decoding of LONGSPEC, we employ dynamic beam search to construct the tree. Previous studies have shown that beam search, while achieving high acceptance rates, suffers from slow processing speed in speculative decoding (Du et al., 2024). Our research identifies that this slowdown is primarily caused by KV cache movement. In traditional beam search, nodes that do not fall within the top-k likelihood are discarded, a step that necessitates KV cache movement. However, in speculative decoding, discarding these nodes is unnecessary, as draft sequences are not required to maintain uniform lengths. Instead, we can simply halt the computation of descendant nodes for low-likelihood branches without removing them entirely. By adopting this approach, beam search attains strong performance without excessive computational overhead. In our experiments, the beam width is set to $[4, 16, 16, 16, 16]$ for each speculation step. All inference experiments in this study are conducted using float16 precision on a single A100 80GB GPU.

E Experimental Results of EAGLE and Token Recycling on Long-Context Speculative Decoding

In Table 4, we compare the average acceptance length τ and decoding speed (tokens/s) for two models under four settings: the baseline PyTorch implementation from HuggingFace ("Vanilla HF"), the same model with `Flash Attention` ("Vanilla FA"), Token Recycling (Luo et al., 2025) ("TR", a SoTA retrieval-based method), EAGLE (Li et al., 2024c) (trained with anchor offset indices and inference with HuggingFace), and our LongSpec with hybrid tree attention. Across five datasets (GovReport, QMSum, MultiNews, LCC, and RB-P), Vanilla HF's decoding speeds are limited between 14 and 30 tokens/s, while switching to `Flash Attention` boosts speeds to about 50 tokens/s, a more than 2.5× speedup.

EAGLE extends the acceptance length to around 2 and achieves 26–40 tokens/s, yielding a 30–50% speedup over Vanilla HF. However, because EAGLE cannot leverage `Flash Attention`, its decoding speed remains substantially below that of Vanilla FA in every setting. As for TR, while it extends the acceptance length to around 3 (far larger than EAGLE) and achieves moderate acceleration on many tasks, it consistently underperforms LongSpec across the board.

In contrast, our LongSpec with hybrid tree attention achieves much higher decoding speeds of about 100 tokens/s across all models and datasets. This demonstrates that EAGLE's incompatibility with `Flash Attention` fundamentally limits its decoding performance. Our hybrid tree attention preserves compatibility with `Flash Attention`, thus unlocking substantially higher decoding speed, underscoring the importance of combining tree-structured attention with SoTA long-context inference techniques such as `Flash Attention`.

F Performance Analysis with Varying Prefill Lengths

In Table 5, we show a detailed breakdown of performance as the prefill length increases, with LongChat-7B on GovReport. Across the all token

Table 4: Average acceptance length τ and decoding speed (tokens/s) across different models and settings. Specifically, "Vanilla HF" refers to HuggingFace's PyTorch-based attention implementation, while "Vanilla FA" employs `Flash Attention`. All results are computed at $T = 0$.

	Setting	GovReport		QMSum		MultiNews		LCC		RB-P	
		τ	Tokens/s	τ	Tokens/s	τ	Tokens/s	τ	Tokens/s	τ	Tokens/s
V-7B	Vanilla HF	1.00	25.25	1.00	18.12	1.00	27.29	1.00	25.25	1.00	19.18
	Vanilla FA	1.00	45.76	1.00	43.68	1.00	55.99	1.00	54.07	1.00	46.61
	TR	2.83	94.06	2.13	68.23	2.81	94.51	2.72	87.77	2.83	94.10
	EAGLE	2.02	33.43	1.91	26.78	1.97	36.62	1.92	40.64	1.92	33.84
	LongSpec	3.57	102.23	3.14	88.87	3.51	100.55	3.73	107.30	3.86	110.76
LC-7B	Vanilla HF	1.00	25.27	1.00	14.11	1.00	27.66	1.00	25.27	1.00	17.02
	Vanilla FA	1.00	42.14	1.00	36.87	1.00	50.19	1.00	54.17	1.00	42.69
	TR	2.90	94.82	2.20	64.96	2.75	94.94	2.80	96.67	3.05	100.41
	EAGLE	2.10	32.06	1.94	26.02	2.02	34.38	2.09	38.81	2.10	29.75
	LongSpec	3.59	101.43	3.06	85.23	3.41	97.93	4.21	122.30	4.03	115.27

Table 5: A detailed breakdown of performance as the prefill length increases, with LongChat-7B on GovReport.

Prefill Length	0–5k	5k–10k	10k–15k	15k–20k	20k–25k	25k–32k
Tokens/s	116.65	115.52	114.54	113.47	115.13	103.68
τ	4.01	3.97	3.97	4.12	4.45	3.97
Draft time (ms)	8.91	8.92	8.93	8.98	9.13	9.25
Target time (ms)	25.63	25.66	25.61	27.30	29.08	30.89
Verify time (ms)	6.18	6.22	6.23	6.24	6.27	6.28

ranges, the generation speed remains remarkably stable, only dropping a little in the 25k-32k range. The average acceptance length remains consistent across all ranges, which indicates stable behavior in the number of tokens the system chooses to retain during generation. This stability suggests that the draft quality is unaffected by the length of the prefill, maintaining consistent output dynamics.

In terms of latency, the draft time increases only marginally, from 8.91 ms in the shortest context range to 9.25 ms in the longest, while target time shows a gradual increase from 25.63 ms to 30.89 ms, reflecting the added computational load of managing larger contexts. Verify time remains almost constant across all ranges, increasing only slightly from 6.18 ms to 6.28 ms.

Together, these results demonstrate that the system scales effectively with longer input contexts, maintaining high throughput and consistent drafting quality with only modest increases in latency. This highlights the practicality and robustness of our approach for real-world applications involving extended input sequences.

G Pseudo Code

Here we provide pseudo code for Anchor-Offset Indexing and Flash Noisy Training.

Algorithm 1 Anchor-Offset Indexing

Input: Sequence length N; Max length MAX_LEN; Query states q_s.
Output: Query states with RoPE applied using modified indices.
$P \leftarrow \{0, 1, \ldots, N-1\}$ ▷ Initial position indices
$o \leftarrow \text{RandomInt}(0, \text{MAX\_LEN} - N)$ ▷ Generate random offset
$P[4:] \mathrel{+}= o$ ▷ Apply offset to indices after the first 4 anchors
// e.g., for $N = 128, o = 16257$, P becomes $[0, 1, 2, 3, 16261, \ldots, 16385]$
return $\text{RoPE}(q_s, P)$

Algorithm 2 Flash Noisy Training

Input: Queries Q, Key cache K, Value cache V.
Output: Final attention output.
$j \leftarrow \text{RandomInt}(1, 4)$ ▷ Randomly select number of tokens to drop

//Perform attention on sliced inputs
$Q' \leftarrow Q[j:]$ ▷ Drop first j queries
$K' \leftarrow K[:-j]$ ▷ Drop last j keys from cache
$V' \leftarrow V[:-j]$ ▷ Drop last j values from cache
$attn\_out \leftarrow \text{FlashAttention}(Q', K', V')$

//Pad the output to match original query length
$padded\_out \leftarrow \text{Concat}(\text{Zeros}(j), attn\_out)$
return $\text{OutputProjection}(padded\_out)$

H Case Study

Here we display some illustrative cases from GovReport on the Longchat-7B model, where tokens marked in blue indicate draft tokens accepted by the target model. Due to space limitations, the complete answer is not presented here.

The report discusses the use of opioid medications in healthcare and the potential risks associated with their misuse. Opioid medications are used to treat pain and can also be used to treat other health problems, such as severe coughing. There are three types of opioid medications that are approved for use in the treatment of opioid use disorders: methadone, buprenorphine, and naltrexone. Methadone is a full opioid agonist, meaning it binds to and activates opioid receptors in the body. Buprenorphine is a partial opioid agonist, meaning it also binds to and activates opioid receptors, but to a lesser extent than methadone. Naltrexone is an opioid antagonist, meaning it binds to and blocks the effects of opioid receptors.
The report also discusses the potential risks associated with the use of opioid medications, including the risk of addiction and the risk of overdose. The use of opioid medications can lead to physical dependence and tolerance, which can make it difficult to stop using the medication. Additionally, the misuse of opioid medications can lead to addiction, which can have serious consequences for the individual and their loved ones.
The report also discusses the potential risks associated with the diversion of opioid medications, which is the illegal use of prescription opioids for non-medical purposes. Diversion can lead to increased rates of addiction, overdose, and death.
The report concludes by discussing the importance of proper use and monitoring of opioid medications, as well as the need for continued research and development of new treatments for opioid use disorders.

The Railroad Retirement Board (RRB) is an independent federal agency that administers retirement, survivor, disability, unemployment, and sickness insurance for railroad workers and their families. The RRB covers workers who are employed by railroads engaged in interstate commerce and related subsidiaries, railroad associations, and railroad labor organizations.

The RRB has two main programs: the Railroad Retirement Act (RRA) and the Railroad Unemployment Insurance Act (RUIA). The RRA authorizes retirement, survivor, and disability benefits for railroad workers and their families. The RUIA provides unemployment and sickness benefits for railroad workers.

The number of railroad workers has been declining since the 1950s, although the rate of decline has been irregular. In recent years, railroad employment has increased after reaching an all-time low of 215,000 workers in January 2010. In April 2015, railroad employment peaked at 253,000 workers, the highest level since November 1999, and then declined through FY2017, falling to 221,000 workers.

The RRB's programs are designed to provide comprehensive benefits to railroad workers and their families. The RRA and RUIA are important components of the railroad industry's retirement and benefits system. The RRB's efforts to maintain and improve these programs are crucial for the well-being of railroad workers and their families.

The Congressional Gold Medal is a prestigious award given by the United States Congress to individuals and groups in recognition of their distinguished contributions, achievements, and services to the country. The tradition of awarding gold medals dates back to the late 18th century, and it has been used to honor a wide range of individuals, including military leaders, scientists, artists, and humanitarians.

The first Congressional Gold Medals were issued by the Continental Congress in the late 1700s, and since then, Congress has awarded over 2,000 medals to various individuals and groups. The awarding of the Congressional Gold Medal is not a permanent statutory provision, and it is typically done through special legislation.

The process of awarding the Congressional Gold Medal involves several steps, including the introduction of legislation, the consideration of the legislation by the relevant committees, and the approval of the legislation by both the House of Representatives and the Senate. Once the legislation is approved, the Secretary of the Treasury is responsible for striking the medal, which is then presented

The design of the Congressional Gold Medal is typically determined by the Secretary of the Treasury, in consultation with the Citizens Coinage Advisory Committee and the Commission of Fine Arts. The medal typically features a portrait of the recipient, as well as inscriptions and symbols that reflect the recipient's achievements and contributions.

The Congressional Gold Medal is considered one of the highest civilian honors in the United States, and it is often given to individuals who have made significant contributions to their field or to the country as a whole. The award has been given to a wide range of individuals, including military heroes, civil rights leaders, and artists.

In recent years, the number of Congressional Gold Medals awarded has increased, with over 50 bills introduced in the 113th Congress alone. The award has also been given to a growing number of groups, including military units, organizations, and even entire cities.

The report provides an overview of the annual appropriations for the Department of Homeland Security (DHS) for FY2019. It compares the enacted FY2018 appropriations for DHS, the Trump Administration's FY2019 budget request, and the appropriations measures developed and considered by Congress in response to the request. The report identifies additional informational resources, reports, and policy experts that can provide further information on DHS appropriations.

The report explains several specialized budgetary concepts, including budget authority, obligations, outlays, discretionary and mandatory spending, offsetting collections, allocations, and adjustments to the discretionary spending caps under the Budget Control Act (BCA). It also provides a detailed analysis of the appropriations process for DHS, including the various committees and subcommittees involved, and the role of the Congressional Budget Office (CBO) and the Government Accountability Office (GAO).

The report highlights the key issues and debates surrounding DHS appropriations, including funding for border security, immigration enforcement, cybersecurity, and disaster response. It also discusses the impact of the BCA on DHS appropriations and the potential for future changes to the spending caps.

Overall, the report provides a comprehensive analysis of the annual appropriations for DHS and the factors that influence the allocation of funding. It is a valuable resource for policymakers, analysts, and stakeholders interested in understanding the complexities of DHS appropriations and the challenges facing the department in the coming years.