

Boosting the Performance of Tree-Based Speculative Decoding of LLMs on FPGAs

Tielong Liu^{1,2}, Gang Li^{1,*}, Zitao Mo¹, Zeyu Zhu^{1,2}, Minnan Pei¹, Jian Cheng^{1,*}

¹C²DL, Institute of Automation, Chinese Academy of Sciences, Beijing, China

²School of Future Technology, University of Chinese Academy of Sciences, Beijing, China

{liutielong2022, mozitao2017, zhuzeyu2021, peiminnan2022}@ia.ac.cn, gang.li@ia.ac.cn, jcheng@nlpr.ia.ac.cn

Abstract—As an efficient alternative to autoregressive decoding, tree-based speculative decoding (SD) has been widely adopted to accelerate LLM inference on GPUs. However, due to the notable disparity in compute power and memory bandwidth, we observe that a specific target-draft model pair with a proper decoding configuration, despite demonstrating significant performance gains on GPUs, often fails to maintain its efficacy on FPGAs, and may even underperform the standard autoregressive decoding approach.

In this paper, we propose an analytical framework to revive the performance of tree-based speculative decoding on FPGAs. We introduce effective performance, a roofline-based metric designed to: 1) assess whether a specific target-draft model pair can benefit from SD for the given FPGA platform, and 2) determine the optimal decoding configuration to achieve peak performance when SD is applicable. We also propose a prior-score-based search strategy to identify the optimal tree structure for a preset number of nodes, further enhancing the performance. We evaluate our method on AMD FPGA platforms using two state-of-the-art SD algorithms: LongSpec and EAGLE-3. Our approach demonstrates a speedup of 2.54-3.89× over autoregressive decoding.

I. INTRODUCTION

Speculative Decoding (SD) [1] has garnered significant attention for accelerating LLM inference on GPUs. Its key idea is to use a small draft model to assist the inference of a large target model. Currently, state-of-the-art SD algorithms [4], [5] employ a tree-based drafting strategy. During inference, the draft model first generates a candidate token tree, where the total number of tokens is known as the verification length. All tokens in this tree are then fed into the target model in parallel for verification.

As LLM inference on GPUs is typically memory-bound, the parallel verification of a tree of tokens does not significantly increase the latency compared to a single-token decoding step. However, accepting multiple tokens in a single step reduces the total number of target model inference steps required, leading to a net reduction in overall generation time.

On FPGAs, however, computational resources are not as abundant as on GPUs. As the verification length increases, the verification process of the target model may transition from memory-bound to compute-bound. In this scenario, the latency of a single inference step increases with the verification length. This increase in latency can offset the performance gains from a higher number of average accepted tokens (AAT) of tree-based SD, potentially degrading the overall inference throughput. Given the significant performance differences of specu-

lative decoding on GPUs and FPGAs, two critical questions arise: *Can speculative decoding accelerate LLM inference on resource-constrained FPGAs? If so, how can its performance be maximized?*

In this paper, we propose an analytical framework for tree-based speculative decoding on FPGAs. The framework not only identifies the circumstances under which SD yields performance gains on FPGAs versus those where it is ineffective but also provides methods to maximize these gains. Specifically, for a given FPGA platform and a target-draft model pair, our framework establishes a roofline-based [3] model that captures the relationship between inference performance, verification length, and AAT. This integrated model introduces the concept of “effective performance” to determine whether an SD algorithm can achieve performance gains on a particular FPGA device. If the performance ceiling with SD is lower than that of autoregressive decoding, it indicates a hardware-algorithm mismatch. Otherwise, we employ our model to find the optimal verification length for maximum speedup.

As tree-based SD algorithms typically process a tree structure of speculative tokens for verification [6]–[9], different tree structures can lead to varying AAT for the same number of nodes, thus affecting the final speedup. To maximize performance, we propose a prior-score-based strategy to search for the optimal tree structure for any given number of nodes. To our best knowledge, we are the first to investigate the performance of speculative decoding on resource-constrained FPGAs. Our main contributions are summarized as follows:

- We present a comprehensive analytical framework that integrates the roofline model with the AAT model to analyze whether a given workload can benefit from SD and, if so, to determine the optimal configuration for achieving maximum speedup.
- We propose a prior-score-based search strategy to identify the optimal tree structure for any given number of tree nodes.
- We conduct extensive performance tests on the state-of-the-art FPGA-based LLM accelerator, FlightLLM [11]. Across various SD algorithms and target models, our method achieves a speedup of 2.54-3.89× over autoregressive decoding. On different hardware configurations, our approach demonstrates speedups of 2.13-2.49×, validating the generality of our method.

*Corresponding authors.

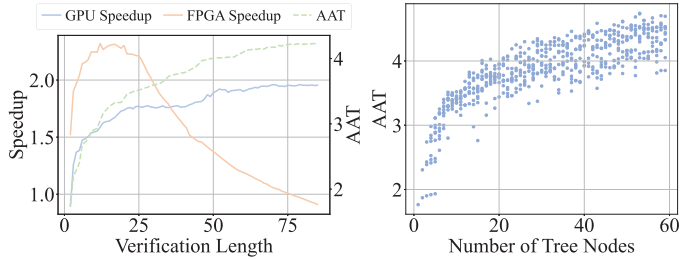


Fig. 1: (a) Speedup of speculative decoding over autoregressive decoding and the AAT curve for LongSpec [4]. (b) AAT vs. verification length (number of tree nodes) for LongSpec.

II. BACKGROUND AND MOTIVATION

A. Tree-Based Speculative Decoding of LLMs

Speculative decoding is a promising method that improves the efficiency of LLM inference without sacrificing generation quality [1], [2]. Among them, tree-based SD [4]–[10] represents the latest advancement in this area, which includes two phases: a prediction phase and a verification phase. During the prediction phase, the draft model uses top-k sampling at each forward step to generate multiple candidate tokens. Each candidate can then generate additional tokens in the next step, collectively forming a tree of draft tokens. In the verification phase, all tokens in this tree are flattened and processed in parallel by the target model. The target model compares its own output probability distribution against that of the draft model for each candidate token. Tokens are accepted where the two distributions are consistent. Finally, the longest continuous accepted sequence from the root of the tree is retained.

B. Motivation

In speculative decoding, the number of average accepted tokens (AAT) and the verification length are two key factors affecting inference performance. When SD is applied to GPUs, current algorithms typically choose a verification length at which the AAT curve approaches saturation [4], [5]. This is because GPUs have abundant computational resources, implying that the latency of a single verification step for the target model remains nearly constant while the average number of accepted tokens is maximized.

However, due to the notable disparity in compute power and memory bandwidth, we observe that the performance of SD on FPGAs differs markedly from that on GPUs. As shown in Fig. 1(a), the speedup of SD over AD on an AMD V80 FPGA can drop below 1 as the verification length increases. This implies that directly applying the AAT and verification length optimized for peak GPU performance gains to FPGAs is unlikely to yield similar benefits, and may even result in performance degradation compared to autoregressive decoding. In addition, for tree-based SD, a single verification length can correspond to multiple tree structures, leading to significantly different AATs, as shown in Fig. 1(b). These findings motivate a deeper understanding of the performance of SD on resource-constrained FPGAs and an investigation into how to enhance its performance on such platforms.

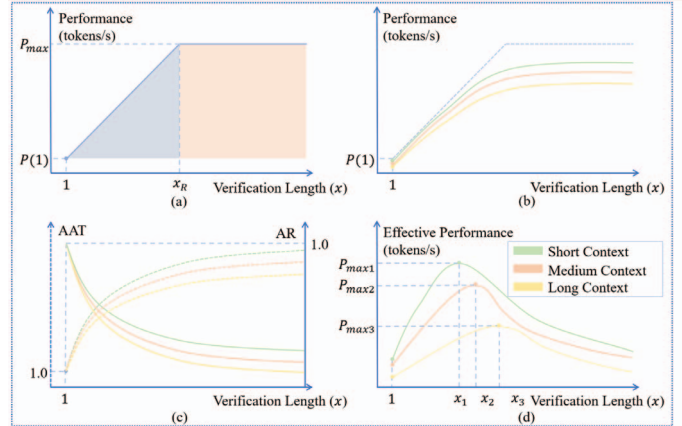


Fig. 2: The analysis model. (a) Primary roofline model; (b) Discounted roofline model; (c) AAT & AR model; (d) Combined model. In (b), (c), and (d), we plot model curves for three different context lengths, indicating that different context lengths correspond to different optimal solutions.

III. METHODOLOGY

A. Establishment of Analysis Model

We begin by defining a workload as a five-tuple: (hardware platform, target model, SD algorithm, context length, task set). We then perform detailed analysis and modeling to evaluate whether a workload is suitable for SD. If it is, our goal is to determine the optimal verification length configuration. The analytical model is illustrated in Fig. 2.

Primary Roofline Model: We adapt the original roofline model by replacing arithmetic intensity with the verification length x . Given that x is typically small relative to the model’s hidden size in our target scenarios, it serves as a reasonable approximation of arithmetic intensity—ignoring units. Furthermore, we substitute the original performance metric—FLOPs/s—with tokens/s, denoting the throughput of input tokens processed per unit time on the target hardware platform. Since the workflow of LLM inference is deterministic, tokens/s and FLOPs/s provide equivalent measures. When $x = 1$, this corresponds to autoregressive decoding, and the performance is denoted as $P(1)$. When $x > 1$, it represents parallel decoding of multiple tokens, which is analogous to the prefill stage of LLM inference. As shown in Fig. 2(a), the relationship between performance and x is approximately linear, similar to the original roofline model. This relationship can be expressed as:

$$P(x) = \begin{cases} ax + b, & \text{for } 1 \leq x < x_R, \\ P_{\max}, & \text{for } x \geq x_R. \end{cases} \quad (1)$$

Where x_R is the ridge point of this model.

Discounted Roofline Model: When performing SD, it’s necessary to consider both the time the target model spends on verification, $T_v(x)$, and the time the draft model spends on generating candidate tokens, $T_d(x)$. For a given SD algorithm, the dataflow for both the draft and verification processes is fixed once the verification length x is determined. This allows us to pre-measure the distributions of $T_v(x)$ and $T_d(x)$ on the

hardware. Therefore, the performance metric $P(x)$ must be multiplied by a discount factor, as expressed below:

$$P_{\text{disc}}(x) = P(x) \cdot \frac{T_v(x)}{T_v(x) + T_d(x)} = \frac{x}{T_v(x) + T_d(x)}. \quad (2)$$

AAT&AR Model: $P_{\text{disc}}(x)$ represents the number of tokens a hardware platform can process per unit of time after applying an SD algorithm. However, during the verification phase, only a subset of the tokens fed to the target model are accepted. AAT, which reflects the alignment between the draft and target models, quantifies the accepted length. The acceptance rate, $AR(x)$, is defined as $AAT(x)/x$. By using $AR(x)$, we can calculate the effective performance of the whole inference process.

Therefore, we need to model both $AAT(x)$ and $AR(x)$, as shown in Fig. 2(c). A complete point-by-point measurement of the $AAT(x)$ model would be extremely time-consuming. Fortunately, we have observed that for various target-draft model combinations, $AAT(x)$ can be approximated by a logarithmic distribution:

$$AAT(x) = A + B \cdot \ln(x - C). \quad (3)$$

Therefore, we can model the complete $AAT(x)$ curve by measuring the AAT for only a few values of x . For typical SD algorithms, when $x \geq 2$:

$$1.5 < AAT(x) < x. \quad (4)$$

It indicates that for the SD algorithms we consider, the first draft token has a greater than 50% chance of being accepted during prediction by the draft model [5]. Based on Eq. 3 and Eq. 4, $AR(x)$ can be modeled as:

$$AR(x) = \frac{AAT(x)}{x} = \frac{A + B \cdot \ln(x - C)}{x} < 1. \quad (5)$$

Combined Model: After obtaining $P_{\text{disc}}(x)$ and $AR(x)$, the number of effective tokens generated per unit of time on the hardware platform can be calculated, and is defined as the effective performance. It is expressed as:

$$P_{\text{eff}}(x) = P_{\text{disc}}(x) \times AR(x). \quad (6)$$

B. Solving for Optimal Performance

1) *Basic Idea:* With the analytical model established, we first consider the question: Under what workloads can SD algorithms provide a performance benefit? By combining Eq. 2, Eq. 5, and Eq. 6, we obtain the following:

$$P_{\text{eff}}(x) = \frac{AAT(x)}{T_v(x) + T_d(x)}. \quad (7)$$

For a given workload, $AAT(x)$, $T_v(x)$, and $T_d(x)$ are all known, allowing us to find the maximum value of $P_{\text{eff}}(x)$. If $\max(P_{\text{eff}}(x)) < P(1)$, we have

$$\text{Speedup} = \frac{\max(P_{\text{eff}}(x))}{P(1)} < 1. \quad (8)$$

In this case, SD is inferior to AD, and the workload should be processed directly using AD. Conversely, if the condition is not met, it indicates that we can find an x for which SD provides a performance benefit. This is our logic for determining whether a specific workload should use SD.

2) *Case Study:* Next, we analyze different workload cases.

- If the compute power is substantially lower than the memory bandwidth, the system becomes compute-bound even during the autoregressive phase. This causes the $P(x)$ curve to degenerate, leaving only the compute roof. In this scenario, $P(x) \equiv P(1)$, $T_v(x) = T_v(1) \cdot x$, so:

$$\text{Speedup} \leq \frac{AAT(x)}{T_v(x) + T_d(x)} \cdot T_v(1) < \frac{x \cdot T_v(1)}{T_v(x)} = 1. \quad (9)$$

In this case, SD will be slower than AD, it is not recommended to apply SD algorithm in this scenario.

- If the compute power is substantially higher than the memory bandwidth, the system remains memory-bound even when x is large enough for the AAT curve to approach saturation during the parallel verification of the target model. In this case, the original roofline model is limited to the memory roof within our scope of analysis. We have:

$$\begin{aligned} T_v(x) &= T_v(1) = T_v, \\ T_d(x) &= a(x) \cdot T_v. \end{aligned} \quad (10)$$

Here, $\alpha(x)$ is approximated as the ratio of the draft model's parameters to those of the target model, scaled by the number of forward passes performed by the draft model in a single draft-verify iteration. This can be expressed as:

$$\alpha(x) = \frac{\text{Param}_d}{\text{Param}_v} \cdot \text{Depth}(x) \quad (11)$$

For SOTA SD algorithms, as the draft model is small enough, $\alpha(x) < \frac{1}{2}$. Therefore,

$$\text{Speedup} \geq \frac{AAT(x)}{(1 + \alpha)T_v} \cdot T_v(1) > \frac{AAT(x)}{(1 + 1/2)} > 1. \quad (12)$$

In this case, SD is guaranteed to accelerate LLM inference. Eq. 12 indicates that a larger x leads to a greater speedup. Thus, a larger x is optimal. The configurations on GPUs generally follow this principle.

- If the ratio of compute power to bandwidth does not fall into the two categories discussed above, the performance is memory-bound for small and compute-bound for large values of x . In this case, the performance should be analyzed in segments to identify the point of maximum performance. Based on Eq. 1, the following analysis can be performed. For $x < x_R$, $T_v(x) = T_v(1)$. Combining this with Eq. 8, we have:

$$\text{Speedup}_1 = \max\left(\frac{1}{1 + \alpha(x)} \cdot AAT(x)\right). \quad (13)$$

Since $\alpha(x)$, $AAT(x)$ are known, the maximum speedup in this segment can be determined and is denoted as Speedup_1 .

When $x \geq x_R$, $T_v(x) = x/x_R \cdot T_v(1)$. We have:

$$\text{Speedup}_2 = \frac{1}{1 + \alpha(x)} \cdot \frac{x_R}{x} \cdot AAT(x). \quad (14)$$

Similarly, we can solve for the maximum speedup in this segment, Speedup_2 . Thus, the final speedup is:

$$\text{Speedup} = \max(\text{Speedup}_1, \text{Speedup}_2). \quad (15)$$

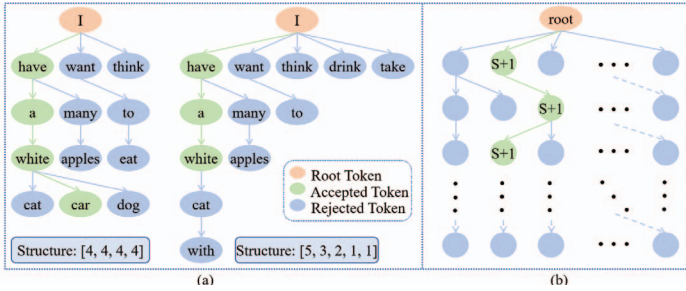


Fig. 3: Exploration of the structure of the draft token tree. (a) Different structure with the same number of tree nodes; (b) Prior-score-based optimal tree structure search method.

In practice, we use a graphical method to solve for the speedup. As shown in Fig. 2(d), once the effective performance curve is obtained, the optimal verification length for different context lengths can be determined, thereby achieving the maximum speedup.

3) *Dynamic Optimal Solution*: As shown in Fig. 2, the hardware-specific roofline model and the algorithm-specific AAT and AR models vary with different context lengths. Therefore, we perform a dynamic analysis across different context length ranges. We establish these combined models offline. During inference, we can dynamically select the optimal verification length configuration based on the changing context length. This ensures that the speculative decoding algorithm consistently achieves the highest possible inference speedup.

C. Prior-Score-Based Optimal Tree Structure Search

For any given verification length, when constructing a token tree using beam search during the drafting process, different tree structures can lead to significant variations in AAT model. As shown in Fig. 3 (a), while the two token trees both contain 12 non-root nodes, the number of tokens accepted by the target model can differ due to the variation in their structure.

To determine the optimal tree structure for any verification length from 2 to x_m , we propose a prior-score-based search strategy. First, we construct a maximal tree with x_m non-root nodes, evenly distributed across levels. We maintain a score list to record the scores of these x_m nodes. Next, we perform SD inference on a validation set. For each draft-verify iteration, if the token corresponding to a node is accepted, its score is incremented by one, as shown in Fig. 3(b). After the inference is complete, we collect the scores for all x_m positions and sort them in descending order. For any case with x verification tokens, we simply extract the top x positions from the score list to construct the token tree. The complete procedure of our method is summarized in Algorithm 1.

IV. EVALUATION

A. Experimental Setting

Speculative Decoding Algorithms. EAGLE-3 [5] and LongSpec [4] are SOTA tree-based SD algorithms designed for short and long contexts, respectively. We deploy these two algorithms on FPGAs to validate the effectiveness of our

Algorithm 1 Overall Procedure

- 1: Establish a primary roofline model based on the hardware characteristics and the target model’s dataflow.
- 2: Build a corresponding discounted roofline model for each SD algorithm.
- 3: Search offline for the optimal tree structure for any number of tree nodes.
- 4: Model AAT and AR across different context lengths.
- 5: Construct the Combined Model to identify the optimal configuration.
- 6: Perform inference on the hardware, switching to the appropriate SD configuration for different context lengths.

method. We select Vicuna-v1.3-13B [12] and Llama-3.1-8B-Instruct [14] as the target models for EAGLE-3. For LongSpec, we adopt Llama-3.1-8B-Instruct [14], Vicuna-v1.5 (7B and 13B) [12], and LongChat (7B and 13B) [13]. All models adhere to their original implementations. Since both algorithms include their respective draft models, no additional training is required.

Test Tasks. The tasks selected for evaluation are consistent with LongSpec, including long-text summarization and code completion. Specifically, we evaluate on GovReport [15], QM-Sum [16], Multi-News [17], LCC [18], and RepoBench-P [19].

Hardware Platform. To evaluate the effectiveness of our proposed method for tree-based speculative decoding, we conduct experiments on FlightLLM [11], a state-of-the-art FPGA-based LLM accelerator. We develop a cycle-accurate simulator, which maintains consistency with FlightLLM in architecture, clock frequency, and off-chip bandwidth utilization. To demonstrate the scalability of our method, we scale the design of FlightLLM to derive four variants on AMD Alveo V80, U280, U250, and Versal VHK158 FPGAs.

B. Main Results

TABLE I presents the speedup achieved by applying our analytical model to the LongSpec algorithm using Llama-3.1-8B-Instruct on the GovReport dataset. We compare against AD, the original set (verification length that gives the highest GPU speedup), and the naive set (verification length where the AAT curve slope $< 2\%$). Our results indicate that the analytical model dynamically selects the optimal configuration for varying context lengths, achieving the highest speedup. For LongSpec, it yields average speedups of $2.44\times$, $3.00\times$, and $1.65\times$ over AD, the original set, and the naive set, respectively, across context lengths from 128 to 16384.

To demonstrate the scalability of our method across different models and SD algorithms, we present the acceleration results using LongSpec and EAGLE-3, as shown in Fig. 4. For LongSpec, our method achieves speedups of $2.54\times$, $2.86\times$, $2.64\times$, and $2.70\times$ over AD for Vicuna-v1.5-7B/13B and LongChat-7B/13B, with average gains of $1.79\times$ and $1.31\times$ over the original and naive sets, respectively. Similar trends are observed for EAGLE-3, with Llama-3.1-8B-Instruct and Vicuna-v1.3-13B achieving average speedups of $2.97\times$ and $3.89\times$. These results confirm that our analytical model is effective across different SD algorithms and LLMs.

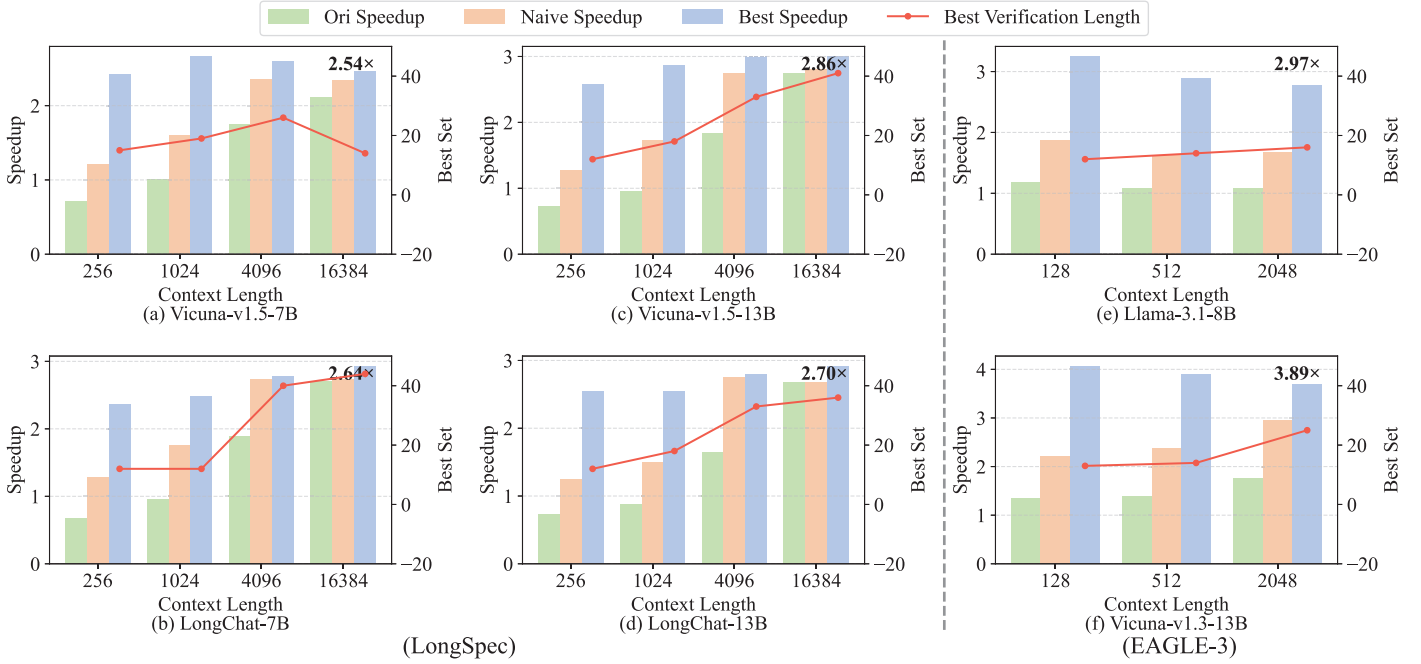


Fig. 4: Speedup and best verification length on LongSpec and EALGE-3. We list the speedup for the original set, naive set, and best set. The lines represent the best verification lengths obtained under different context lengths. The number in the upper right corner of each subgraph represents the average acceleration ratio of our method relative to AD.

TABLE I: Speedup of Llama-3.1-8B-Instruct.

Context Length	Method	Verification Length	Performance (Tokens/s)	Speedup
128	AR	—	40.09	1.00×
	Ori set	69	30.43	0.76×
	Naive set	44	45.47	1.13×
	Best set	13	98.96	2.47×
512	AR	—	38.06	1.00×
	Ori set	69	29.88	0.79×
	Naive set	41	45.10	1.18×
	Best set	13	94.74	2.49×
2048	AR	—	31.66	1.00×
	Ori set	69	26.48	0.84×
	Naive set	37	43.27	1.37×
	Best set	16	77.10	2.44×
4096	AR	—	25.86	1.00×
	Ori set	69	23.76	0.92×
	Naive set	33	42.02	1.62×
	Best set	18	62.14	2.40×
8192	AR	—	18.92	1.00×
	Ori set	69	19.84	1.05×
	Naive set	31	36.44	1.93×
	Best set	17	43.77	2.31×
16384	AR	—	12.32	1.00×
	Ori set	69	16.33	1.33×
	Naive set	31	27.19	2.21×
	Best set	16	28.67	2.33×

C. Consistency of Speedup Across Different Tasks

If the optimal verification length configuration found for certain tasks is not applicable to others, it would undoubtedly reduce the scalability of our method. TABLE II proves the consistency of our method’s results across different tasks.

For different tasks, the AAT curves provided by the original SD algorithm vary, resulting in some variation in the solved

optimal verification length. However, for the same SD algorithm, model, and context length, the AAT curves for different tasks exhibit a certain degree of similarity. Consequently, their optimal verification lengths are very similar, ranging from 12 to 14, as shown in TABLE II. We compute the geometric mean of the AAT curves across tasks to determine a single, unified verification length. Experimental results prove that this unified solution incurs negligible speed loss compared to task-specific solutions.

TABLE II: Performance metrics for different tasks.

Task Name	Gov-Report	QmSum	Multi-News	LCC	RepoBench-P
Best set	13	14	12	13	13
Best Speedup	2.47×	2.67×	1.83×	2.43×	2.33×
Unified Speedup	2.47×	2.66×	1.79×	2.43×	2.33×

D. The Fitting Results of AAT Model

As described in Section III, we apply a logarithmic model to the AAT curve of the speculative decoding algorithm. Fig. 5 illustrates the relationship between the modeled curve plotted from a few test points, and the complete AAT curve. The two curves exhibit a high degree of consistency, with the coefficient of determination R^2 reaching approximately 0.99 across different context scenarios. This demonstrates that the complete AAT curve can be obtained by offline testing of a few points to determine the parameters A, B, and C in Eq. 3.

E. Improvement of Tree Search

To show that our tree search method improves AAT and thus speedup, we compare it with two baseline token tree

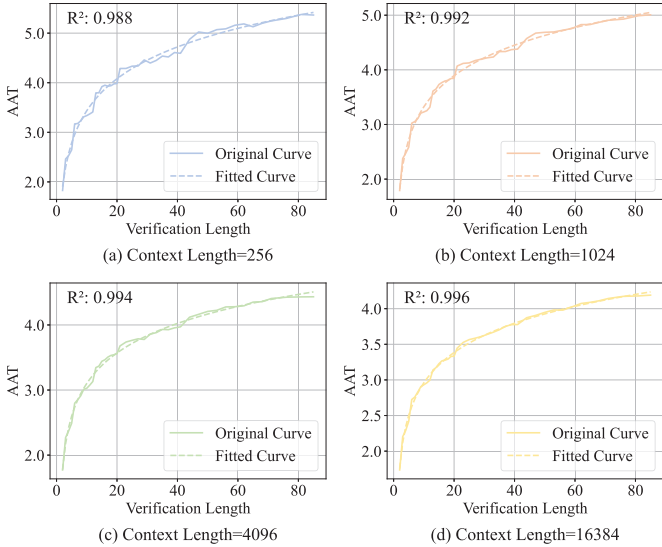


Fig. 5: The fitting results of the AAT curve under different context lengths.

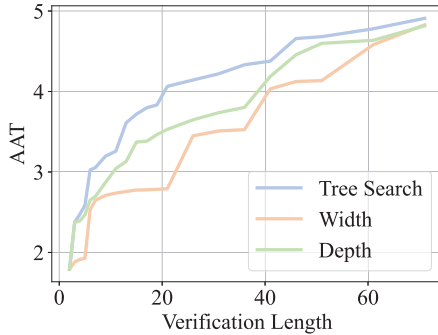


Fig. 6: Comparison of AAT curve of different tree-construction methods.

construction methods. The width-based method fills nodes level by level within a fixed tree shape, while the depth-based method adds consecutive nodes to adjacent levels (if the k^{th} node is at level m , the $(k+1)^{\text{th}}$ goes to level $m+1$).

We evaluate LongSpec on Llama-3.1-8B-Instruct with a context length of 1024 (Fig. 6). Our prior-score-based tree search consistently outperforms the baselines, with average AAT values 0.60 and 0.28 higher than the width- and depth-based methods, respectively. Using the AAT curves to solve for the optimal verification length, our method achieves the highest speedup of $2.46\times$, compared to $2.18\times$ and $1.98\times$ for the width- and depth-based methods.

TABLE III: Improvement of performance under tree search.

Method	Best set	Performance (tokens/s)	Speedup
AR	—	35.66	1.00 \times
Width	9	77.61	2.18 \times
Depth	15	70.44	1.98 \times
Tree Search	13	87.89	2.46 \times

F. Comparison Across Different FPGA Platforms

To evaluate the generalizability of our analytical model across different FPGAs, we conduct experiments on four AMD FPGA development boards: Alveo U280, VHK158, Alveo V80, and Alveo U250. We configure different sizes for the PE arrays based on the varying on-chip resources of these boards. In our experiments, we primarily focus on the PE array size and the HBM bandwidth (GB/s) of each board.

TABLE IV presents the experimental results of LongSpec on various hardware platforms for accelerating the Llama-3.1-8B model. Here, we report only the performance of AD and the maximum performance achieved using our analytical model. As shown, our analytical model achieves average speedups of $2.13\times$, $2.32\times$, $2.44\times$, and $2.49\times$ across the four platforms. The results indicate that platforms with a higher compute-to-bandwidth ratio are more favorable for speculative decoding. This leads to a larger optimal verification length selected by our analytical model, which in turn leads to a higher optimal speedup.

TABLE IV: Best speedup based on LongSpec on different hardware platforms.

Context Length	Board PE Array Bandwidth	U280	VHK158	V80	U250
		64×48 460	128×64 819	128×128 819	128×128 77
256	AR Perf	22.12	39.39	39.39	3.7
	Best set	4	6	13	47
	Best Perf	45.84	93.57	101.25	9.89
	Speedup	2.07 \times	2.38 \times	2.57 \times	2.67\times
1024	AR Perf	20.02	35.66	35.66	3.35
	Best set	4	6	14	19
	Best Perf	40.86	81.58	88.38	8.55
	Speedup	2.04 \times	2.29 \times	2.48 \times	2.55\times
4096	AR Perf	14.52	25.86	25.86	2.43
	Best set	6	8	18	14
	Best Perf	30.83	58.99	62.14	5.84
	Speedup	2.12 \times	2.28 \times	2.40\times	2.40\times
16384	AR Perf	6.92	12.32	12.32	1.16
	Best set	9	12	16	16
	Best Perf	15.93	28.41	28.42	2.69
	Speedup	2.30 \times	2.31 \times	2.31 \times	2.32\times
Average Speedup		2.13 \times	2.32 \times	2.44 \times	2.49\times

V. CONCLUSION

This work proposes a model to evaluate the acceleration potential of SD algorithms on FPGAs and select their optimal configurations. By extending the roofline model with the AAT&AR model, we define effective performance for accurate speedup analysis, and design a prior-score-based tree search to further improve gains. On an Alveo U280 FPGA with LongSpec, our method achieves average speedups of $2.44\times$, $3.00\times$, and $1.65\times$ over AD, the original, and naive sets, and $2.54\text{--}3.89\times$ over AD across diverse algorithms and models, as well as $2.13\text{--}2.49\times$ over AD on various hardware platforms.

VI. ACKNOWLEDGMENT

This work was supported in part by Beijing Natural Science Foundation (No.4254088).

REFERENCES

- [1] Y. Leviathan, M. Kalman, Y. Matias. “Fast Inference from Transformers via Speculative Decoding,” arXiv preprint arXiv:2211.17192, 2022.
- [2] C. Chen, S. Borgeaud, G. Irving, J. Lespiau, L. Sifre, et al. “Accelerating Large Language Model Decoding with Speculative Sampling,” arXiv preprint arXiv:2302.01318, 2023.
- [3] S. Williams, A. Waterman, D. Patterson. “Roofline: an insightful visual performance model for multicore architectures,” in *Commun. ACM*, 52(4):65–76, 2009.
- [4] P. Yang, C. Du, F. Zhang, H. Wang, T. Pang, et al. “LongSpec: Long-Context Speculative Decoding with Efficient Drafting and Verification,” in *International Conference on Machine Learning (ICML)*, 2025.
- [5] Y. Li, F. Wei, C. Zhang, H. Zhang. “EAGLE-3: Scaling up Inference Acceleration of Large Language Models via Training-Time Test,” arXiv preprint arXiv:2503.01840, 2025.
- [6] X. Miao, G. Oliaro, Z. Zhang, X. Cheng, Z. Wang, et al. “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 (ASPLOS)*, Volume 3, pp. 932-949. 2024.
- [7] T. Cai, Y. Li, Z. Geng, H. Peng, J. Lee, et al. “MEDUSA: Simple LLM Inference Acceleration Framework with Multiple Decoding Heads,” in *International Conference on Machine Learning (ICML)*, 2024.
- [8] Y. Li, F. Wei, C. Zhang, H. Zhang. “EAGLE: Speculative Sampling Requires Rethinking Feature Uncertainty,” in *International Conference on Machine Learning (ICML)*, 2024.
- [9] Y. Li, F. Wei, C. Zhang, H. Zhang. “EAGLE-2: Faster Inference of Language Models with Dynamic Draft Trees,” in *Empirical Methods in Natural Language Processing (EMNLP)*, 2025.
- [10] R. Sadhukhan, J. Chen, Z. Chen, V. Tiwari, R. Lai, et al. “MagicDec: Breaking the Latency-Throughput Tradeoff for Long Context Generation with Speculative Decoding,” in *International Conference on Learning Representations (ICLR)*, 2025.
- [11] S. Zeng, J. Liu, G. Dai, X. Yang, T. Fu, et al. “FlightLLM: Efficient Large Language Model Inference with a Complete Mapping Flow on FPGAs,” in *Proceedings of the 2024 ACM/SIGDA International Symposium on Field Programmable Gate Arrays (FPGA)*, pp. 223-234. 2024.
- [12] W. Chiang, Z. Li, Z. Lin, Y. Sheng, Z. Wu, et al. “Vicuna: An Open-Source Chatbot Impressing GPT-4 with 90%* ChatGPT Quality,” url <https://lmsys.org/blog/2023-03-30-vicuna/>, 2023.
- [13] D. Li, R. Shao, A. Xie, Y. Sheng, L. Zheng, et al. “How Long Can Open-Source LLMs Truly Promise on Context Length?” url <https://lmsys.org/blog/2023-06-29-longchat>, 2023.
- [14] A. Grattafiori, A. Dubey, A. Jauhri, A. Pandey, A. Kadian, et al. “The Llama 3 Herd of Models,” arXiv preprint arXiv:2407.21783, 2024.
- [15] L. Huang, S. Cao, N. Parulian, H. Ji, L. Wang. “Efficient attentions for long document summarization,” in *Proceedings of the Conference of the North American Chapter of the Association for Computational Linguistics*, 2021.
- [16] M. Zhong, D. Yin, T. Yu, A. Zaidi, M. Mutuma, et al. “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*, 2021.
- [17] A. Fabbri, I. Li, T. She, S. Li, D. Radev. “MultiNews: A large-scale multi-document summarization dataset and abstractive hierarchical model,” in *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, 2019.
- [18] D. Guo, C. Xu, N. Duan, J. Yin, J. McAuley. “Longcoder: A long-range pre-trained language model for code completion,” in *Proceedings of the International Conference on Machine Learning*, 2023.
- [19] T. Liu, C. Xu, J. McAuley. “RepoBench: Benchmarking repository-level code auto-completion systems,” in *Proceedings of the International Conference on Learning Representations*, 2024.