Faster Speculative Decoding via Effective Draft Decoder with Pruned Candidate Tree

Huanran Zheng, Xiaoling Wang✉
East China Normal University, Shanghai, China
hrzheng@stu.ecnu.edu.cn, xlwang@cs.ecnu.edu.cn

Abstract

Speculative Decoding (SD) is a promising method for reducing the inference latency of large language models (LLMs). A well-designed draft model and an effective draft candidate tree construction method are key to enhancing the acceleration effect of SD. In this paper, we first propose the Effective Draft Decoder (EDD), which treats the LLM as a powerful encoder and generates more accurate draft tokens by leveraging the encoding results as soft prompts. Furthermore, we use KL divergence instead of the standard cross-entropy loss to better align the draft model's output with the LLM. Next, we introduce the Pruned Candidate Tree (PCT) algorithm to construct a more efficient candidate tree. Specifically, we found that the confidence scores predicted by the draft model are well-calibrated with the acceptance probability of draft tokens. Therefore, PCT estimates the expected time gain for each node in the candidate tree based on confidence scores and retains only the nodes that contribute to acceleration, pruning away redundant nodes. We conducted extensive experiments with various LLMs across four datasets. The experimental results verify the effectiveness of our proposed method, which significantly improves the performance of SD and reduces the inference latency of LLMs.

1 Introduction

Large language models (LLMs) have achieved excellent performance in various natural language processing tasks and have garnered much attention (OpenAI, 2023; Touvron et al., 2023; Zheng et al., 2023). However, the autoregressive decoding paradigm adopted by LLMs requires generating tokens one by one, which results in high inference latency. To overcome the speed bottleneck of this serial generation method, speculative decoding (SD) has been proposed as a feasible solution (Leviathan et al., 2022; Chen et al., 2023a). As shown in Figure 1, the primary process of SD involves quickly

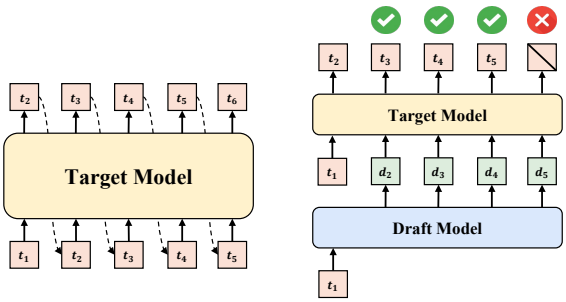


Figure 1: Comparison between autoregressive decoding (left) and speculative decoding (right).

generating multiple draft tokens through a draft model and then verifying whether these tokens are accepted in parallel by a target model (LLM). This process leverages the parallel computing capabilities of devices such as GPUs, significantly reducing inference latency. More importantly, SD is a lossless acceleration method that can strictly ensure the final generated result is completely consistent with autoregressive decoding.

Existing works on SD improvement can be divided into two main directions. The first direction focuses on enhancing the draft model (Cai et al., 2024; Zhou et al., 2023; Elhoushi et al., 2024). For example, MEDUSA (Cai et al., 2024) uses a non-autoregressive approach to generate draft tokens, increasing the draft model's inference speed. DistillSpec (Zhou et al., 2023) adapts knowledge distillation to better align the draft model with the target model, thereby increasing the acceptance rate of draft tokens. However, these methods treat the draft model as an independent model and do not utilize the results encoded by LLMs to assist in generating draft tokens. The second direction is to generate multiple draft candidate sequences to improve the average acceptance length (Du et al., 2024; Cai et al., 2024; Miao et al., 2023). For instance, SpecInfer (Miao et al., 2023) constructs a candidate tree-based speculative inference and verification system to improve the decoding effi-

Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), pages 9856–9868
July 27 - August 1, 2025

ciency of LLMs. However, the candidate trees constructed by previous methods contain many redundant nodes, leading to unnecessary computation and time overhead.

Therefore, in this paper, we propose new solutions from the two aforementioned perspectives to address the shortcomings of previous methods and further improve the efficiency of SD. First, we design an Effective Draft Decoder (EDD), which generates subsequent draft tokens based on the encoding results from LLMs. LLMs have powerful text understanding capabilities and can effectively encode the semantic information of input sentences. Previous studies (Kasai et al., 2020; Gu and Kong, 2020) have shown that, given a powerful encoder, a single-layer autoregressive decoder is sufficient to achieve excellent generation quality. Therefore, we treat the LLM as an encoder to generate soft prompts, which allows the EDD to obtain more contextual information and make more accurate predictions. In addition, unlike previous methods (Du et al., 2024; Cai et al., 2024; Li et al., 2024b) that use cross-entropy loss to fit the output of the draft model to the training set, we use KL divergence to directly align the probability distribution predicted by the draft model with the target model, thereby further improving the acceptance rate.

Second, we found that the prediction confidence of the draft model is well-calibrated with the token acceptance rate through experiments. Based on this, we proposed the Pruned Candidate Tree (PCT) algorithm, which dynamically prunes the candidate tree according to the confidence score of the draft tokens and intelligently determines the depth of the tree. Specifically, we estimate the impact of each node on the expected inference latency and remove nodes that could increase this value, significantly improving the final inference speed.

We chose the LLaMA2-Chat series (Touvron et al., 2023) and the Vicuna1.5 series (Zheng et al., 2023) LLMs as the target models and conducted experiments on four datasets across different tasks. The experimental results verify the effectiveness of our method. First, EDD effectively utilizes the information encoded by LLMs, and its predicted results are closer to the target model. Second, the PCT algorithm fully unleashes the potential of the draft model and significantly increases the average acceptance length. When combining the two, our method achieves speedups of **3.27**× and **3.38**× for LLaMA2-Chat 13B and Vicuna1.5 13B on the MT-Bench (Zheng et al., 2023). The main contributions of this paper are summarized as follows:

- We designed the EDD framework, which incorporates the encoded information from LLMs into the draft model and trains it using the knowledge distillation method, allowing it to better align with the target model.
- We proposed the PCT algorithm, which effectively prunes candidate trees while maintaining a high average acceptance length, significantly reducing inference latency.
- Extensive experiments verify the effectiveness of our proposed method, which can be easily applied to various LLMs and achieves better performance than previous SD methods.

2 Preliminaries

Speculative decoding allows autoregressive LLMs to generate multiple tokens in a single forward pass without compromising generation quality (Leviathan et al., 2022). Each iteration of speculative decoding is divided into two stages: drafting and verification (as shown in Figure 1). In the drafting stage, SD uses a draft model, which is more efficient than the target model, to generate the subsequent m draft tokens $(d_{i+1}, d_{i+2}, ..., d_{i+m})$ based on the previous content $t_{\leq i}$. In the verification phase, SD determines whether to accept these draft tokens based on the prediction results of the target model. There are two acceptance strategies: speculative decoding and speculative sampling, which correspond to the greedy search and standard sampling of the target model, respectively. In this work, we primarily adopt the speculative decoding acceptance strategy. After inputting $t_{\leq i}$ and draft tokens into the target model for a forward pass, the subsequent prediction results $(t_{i+1}, t_{i+2}, \ldots, t_{i+m}, t_{i+m+1})$ are obtained in parallel through greedy search. Among these, due to the causal mask, the generation process of t_{i+1} is identical to autoregressive decoding, so t_{i+1} must be accepted. This ensures that at least one new token is generated in each iteration. Subsequently, SD performs judgments from left to right:

$$\begin{cases} \text{Accept } t_j, \text{ if } d_{j-1} = t_{j-1} \\ \text{Reject } t_j, \text{ if } d_{j-1} \neq t_{j-1} \end{cases} \quad (1)$$

where $i+2 \leq j \leq i+m+1$. In addition, SD only accepts tokens before the first rejection to ensure that the generated results are exactly the same as

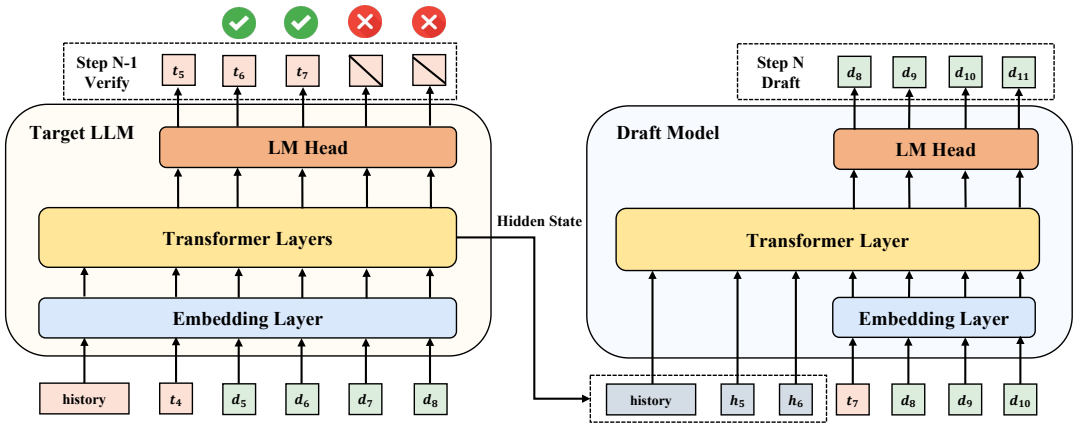


Figure 2: Overview of our proposed Effective Draft Decoder. EDD treats LLM as an encoder and uses the hidden state of LLM as a soft prompt to generate subsequent token drafts autoregressively. To select appropriate hidden states, we conduct hyperparameter experiments in Section 4.3.1.

the autoregressive decoding. Finally, the accepted tokens are concatenated with the previous content as the input for the next iteration. From the SD generation process, it can be seen that ensuring both the efficiency of the draft model and the acceptance rate of the generated draft tokens is the key to reducing inference latency.

3 Method

This section introduces the two components of our approach. In Section 3.1, we present the model architecture and training process of the proposed EDD. In Section 3.2, we analyze the correlation between the predicted confidence and the acceptance rate, and provide the design principle and implementation details of our PCT algorithm.

3.1 Effective Draft Decoder

Model Architecture. In this paper, we use decoder-only LLMs as the target models, which typically consist of an embedding layer, multiple Transformer layers, and an LM head. The main time consumption during the forward pass is caused by the multiple Transformer layers. Our EDD uses the same model structure as the LLMs but contains only one Transformer layer. Therefore, although it still employs autoregressive decoding, its generation speed is significantly faster than that of the LLMs, ensuring the efficiency of the draft model. However, the size gap between the draft model and the target model can lead to a significant performance gap, resulting in a low acceptance rate. As a result, we no longer consider the draft model as a separate model but rather as an extension module of the LLM. Specifically, we treat the LLM as an encoder that can effectively encode previous content. At the same time, the draft model acts as a decoder, generating subsequent draft tokens based on the hidden states provided by the LLM (as shown in Figure 2).

Previous research (Kasai et al., 2020; Gu and Kong, 2020) has shown that, given a powerful encoder, even a shallow decoder can achieve excellent generation quality. Therefore, EDD leverages the powerful encoding capabilities of LLMs, significantly improving model performance. In addition, unlike GLIDE (Du et al., 2024) and EAGLE (Li et al., 2024b), our approach uses the LLM encoding result as a soft prompt for EDD without any modifications to the draft model, ensuring consistency between the draft and target models.

Training Strategies. We copy the parameters of the embedding layer and LM head from the target LLM to EDD for better initialization, and all parameters in EDD are updated. We train the EDD using the standard encoder-decoder model training process. Specifically, we first use the LLM (frozen) to encode the input text $T = (t_1, t_2, \ldots, t_n)$ to obtain the hidden states $H = (h_1, h_2, \ldots, h_n)$. Next, we employ the teacher-forcing strategy to input both T and H into the EDD, generating its predicted probability distribution:

$$P_d(T \mid H) = \prod_{i=1}^{n} P_d\left(t_i \mid t_{<i}, H\right) \qquad (2)$$

However, EDD cannot obtain the encoding information of all texts during inference and can only generate subsequent tokens based on the previous text. To maintain consistency with the inference

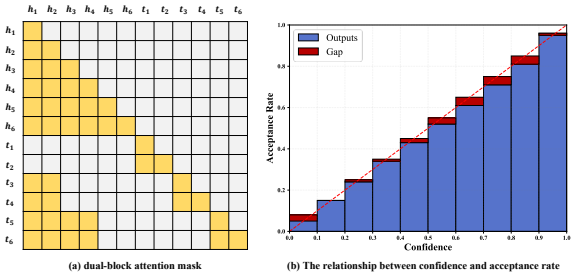


Figure 3: (a) Example of a dual-block attention mask where the block length L is set to 2. (b) Average acceptance rate of draft tokens in different confidence bins.

process, we propose a dual-block mask mechanism for efficient training. Specifically, we divide T into multiple blocks of length L, corresponding to the multiple draft processes. As shown in Figure 3(a), when the draft model predicts t_i in the j-th block, the mask ensures that the draft model can only access the previous tokens in the j-th block and the hidden states corresponding to blocks 0 to $j-1$:

$$P_d(T \mid H) = \prod_{i=1}^{n} P_d\left(t_i \mid t_{j*L:i}, h_{\leq j*L}\right) \quad (3)$$

where $j = \left\lfloor \frac{i}{L} \right\rfloor$. Since the number of accepted draft tokens may vary in each iteration during inference, we randomly select different block lengths for the division at each step during the training phase to improve the model's robustness.

Furthermore, we found that previous methods (Leviathan et al., 2022; Du et al., 2024) use the standard cross-entropy loss to train the draft model, which deviates from the actual training objective. In the SD scenario, we prefer that the probability distribution predicted by the draft model is consistent with the target model rather than the training data. Therefore, we use KL divergence as the training loss to force the probability distribution predicted by the EDD to be close to the target LLM:

$$D_{KL}\left(P_t \| P_d\right) = \sum_x P_t(x) \log \frac{P_t(x)}{P_d(x)} \quad (4)$$

where P_t is the probability distribution predicted by the LLM, which can be obtained simultaneously during the encoding process. Through this soft-label knowledge distillation training method, the EDD can learn more fine-grained information and effectively align with the target model.

3.2 Pruned Candidate Tree

Previous works (Cai et al., 2024; Li et al., 2024b) constructed draft candidate trees to improve the average acceptance length. However, their candidate tree, constructed with a fixed width and depth, contains many redundant nodes, resulting in unnecessary overhead. To address this issue, we propose the PCT algorithm, which intelligently determines the width and depth of the candidate tree.

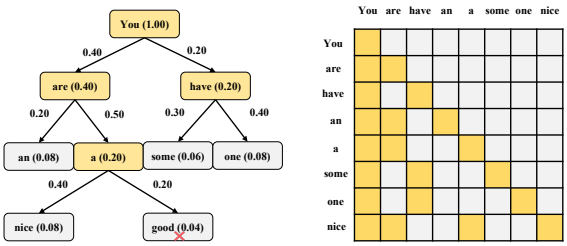
Analysis. Inspired by GLIDE (Du et al., 2024), we first explored the correlation between the draft model's prediction confidence (maximum probability) and the acceptance rate through experiments. Specifically, we trained EDD as the draft model, used LLaMA2-Chat 13B as the target model, and conducted experiments on the MT-Bench (Zheng et al., 2023). We required EDD to generate a draft sequence of length 10 each time and counted the confidence score and acceptance rate of each draft token[1]. We divided the confidence scores into 10 interval bins and plotted the corresponding reliability diagrams. As shown in Figure 3(b), the confidence is well-calibrated (Guo et al., 2017) with the acceptance rate (Expected Calibration Error = 0.019), showing a high positive correlation. Therefore, we can effectively prune the candidate tree based on the confidence score of the draft tokens.

Design Principle. We first focus on a specific draft sequence $(d_1, d_2, \ldots, d_m)$ in the candidate tree. Assume that the time required for a forward pass of the draft model and the target model is s_d and s_t, respectively, and that the probability of accepting the j-th draft token d_j is p_j. So, the expected time gain from performing the j-th drafting step can then be evaluated as follows:

$$(1-p_j)\cdot(s_t+s_d)+p_j\cdot s_d - s_t = s_d - p_j \cdot s_t \quad (5)$$

Therefore, if $p_j < s_d/s_t$, the j-th drafting step is more likely to increase the inference latency and should not be executed. Note that d_j is accepted, meaning that it and all the draft tokens before it are accepted: $p_j = \prod_{i=1}^{j} a_i$, where a_i represents the probability of each token being accepted independently. Furthermore, the results of the above analysis experiment show that the confidence output by the draft model is well-calibrated with the independent acceptance rate, so $p_j \approx \prod_{i=1}^{j} c_i$, where c_i represents the confidence score of each draft token. Therefore, we can estimate the acceptance probability of each node and effectively prune the candidate tree.

[1]Tokens after the first rejection are not included to ensure independence.



(a) Pruned Candidate Tree (b) Tree Attention Mask

Figure 4: An example of PCT implementation, where $s_d/s_t = 0.1$, and leaf nodes with an overall acceptance confidence less than 0.05 are removed.

Implementation Details. Figure 4 shows an example of the PCT construction process. First, the draft model predicts the top-k child nodes based on the expandable nodes from the previous layer to form the current layer. Then, PCT calculates the acceptance confidence of each node in the current layer and marks the nodes with an overall acceptance confidence less than s_d/s_t to prevent them from being used in the next layer's node generation. Finally, the construction process terminates if there are no expandable leaf nodes or the tree depth reaches the maximum depth. Additionally, we remove leaf nodes whose overall acceptance confidence is below a given threshold, further reducing the number of nodes. Finally, we flatten the candidate tree into a one-dimensional sequence and construct a specific tree attention mask (Cai et al., 2024) for the LLM to verify the entire candidate tree in parallel. The recently proposed EAGLE-2 (Li et al., 2024a) is similar to our approach, but the depth used in its candidate tree construction process is fixed. In contrast, our PCT algorithm can adaptively determine the depth of the candidate tree based on the expected time gain at each step in the drafting process, effectively reducing unnecessary computational overhead.

4 Experiments

In this section, we verify the effectiveness of our proposed method on different datasets and LLMs. We first introduce our experimental setup in Section 4.1, then report the main results in Section 4.2. Analysis experiments are presented in Section 4.3.

4.1 Experimental Setup

Models and Datasets. We selected two widely used LLMs, LLaMA2-Chat (7B, 13B) (Touvron et al., 2023) and Vicuna1.5 (7B, 13B) (Zheng et al., 2023), as the target models for our experiments. Our EDD uses the exact same architecture as the target models but contains only one Transformer layer. Similar to previous methods (Du et al., 2024; Cai et al., 2024; Li et al., 2024b), we train the draft models on the ShareGPT (ShareGPT, 2023) dataset. To comprehensively evaluate our method, we selected four datasets from different tasks: MT-Bench (Zheng et al., 2023) for multi-turn dialogue, GSM8k (Cobbe et al., 2021) for mathematical reasoning, CNN/DM (See et al., 2017) for abstractive summarization, and HumanEval (Chen et al., 2021) for code generation. For CNN/DM, we randomly selected 1,000 samples from its test set for our experiments.

Baselines To verify the effectiveness of our method, we selected various baselines for comparison. The baselines for our EDD model include:

1. Vanilla Draft Model (VDM) (Leviathan et al., 2022): Trains an autoregressive draft model independently without utilizing the encoding information of LLMs.

2. MEDUSA-1 (Cai et al., 2024): Adds additional decoding heads to LLM for non-autoregressive decoding. During training, the parameters of the LLMs are frozen, and only the decoding heads are fine-tuned.

3. EAGLE (Li et al., 2024b): The draft model performs autoregressive decoding at the feature level of LLMs.

Because previous works (Leviathan et al., 2022; Cai et al., 2024; Li et al., 2024b) have found that setting the draft sequence length to 5 can achieve good performance in their methods. Therefore, for a fair comparison, we set the sequence length generated by each iteration of all draft models to 5. Additionally, we selected three different candidate tree construction methods as baselines for comparison with PCT:

1. Vanilla Candidate Tree (VCT): Uses fixed width and depth to construct candidate trees. We set the node expansion width to 3 and the tree depth to 5.

2. CAPE (Du et al., 2024): Selects different preset expansion sizes based on the draft model's confidence scores to construct candidate trees. Following the original paper, we fix the candidate tree depth to 5, with the expansion width

of each layer set to 7, 5, 3, and 1 for confidence score levels in the ranges of (0, 0.3], (0.3, 0.6], (0.6, 0.8], and (0.8, 1], respectively.

3. EAGLE-2 (Li et al., 2024a): Re-ranks the nodes of the candidate tree by confidence and retains the top-k nodes. Following the original paper, we set the total number of draft tokens to 50, with a draft tree depth of 6, and select 10 nodes during the expansion phase.

For a fair comparison, all baselines were re-implemented in our experimental environment.

Implementation Details We conducted experiments on NVIDIA RTX A6000 48GB GPUs. During training, we randomly sampled block lengths ranging from 5 to 10 at each step to divide the input text. The hidden states of the fourth-to-last layer of LLMs were used as the encoding result for EDD. We set the batch size to 8, the learning rate to 1e-4, and used AdamW (Kingma and Ba, 2014) to optimize the model parameters. The draft model was trained for only one epoch, and we kept the final checkpoint. We set the temperature to 0 for greedy decoding in the inference phase. The expansion width of each node in the PCT was set to 5, and the maximum depth was set to 10. Additionally, we calculated the average time required for one forward pass of both target models and draft models for PCT construction (7B: 1.1ms/29.8ms, 13B: 1.4ms/50.1ms) and removed leaf nodes whose overall acceptance confidence was less than 0.01. Our evaluation focused on the scenario with a batch size of 1, representing the use case where LLMs are locally hosted for personal use.

Metrics. Since SD can achieve lossless acceleration, we do not need to evaluate the quality of the generated results. We use three metrics to assess the acceleration effect of different methods: (1) **Walltime speedup**: The actual speedup ratio relative to autoregressive decoding, which may be affected by different operating environments. (2) **Acceptance rate** α: The average acceptance ratio of draft tokens. When using draft candidate trees, the acceptance rate is calculated by dividing the final accepted sequence length by the maximum depth of the candidate tree. A higher acceptance rate indicates that the output of the draft model is more consistent with the target model. (3) **Average acceptance length** τ: The average number of new tokens generated in each iteration. If the time consumed by the drafting process is the same, a larger τ indicates a better acceleration effect. Both α and τ are not affected by hardware configuration and provide a more objective evaluation.

4.2 Main Results

The main experimental results are shown in Table 1 and Table 2. More experimental results when temperature=1 are shown in the Table 4 and Table 5. Our proposed EDD and PCT methods effectively improve the acceleration effect of SD and achieve better performance than the baselines.

First, compared to other draft models, EDD generates results closer to the target model. Specifically, when compared with VDM, EDD significantly improves the average acceptance length, indicating that EDD can make full use of the encoding information from the LLM to enhance model performance. MEDUSA-1 improves the generation speed of draft tokens through non-autoregressive decoding, but its average acceptance rate is low, so its speedup ratio is similar to that of VDM. Furthermore, EDD even outperforms the strong baseline, EAGLE. We believe this may be because EDD is more consistent with the target model, and using KL divergence during training helps align its output better with the target model.

Second, PCT can efficiently construct candidate trees and achieve faster inference speeds than baseline methods. In particular, VDM can achieve a high acceptance rate, but its large number of nodes increases inference latency. Although CAPE can automatically determine the number of nodes in each layer of the candidate tree, it only expands based on the top-1 node each time, resulting in decreased candidate diversity. Moreover, while EAGLE-2 re-ranks and filters nodes based on confidence scores, the number of nodes and the tree depth still remain fixed. In contrast, our method adaptively determines the width and depth based on the expected time gain at each node, effectively reducing redundant calculations.

Finally, combining EDD with PCT can achieve excellent acceleration effects. As shown in Table 2, our method can achieve a walltime speedup of **2.04x-3.62x** across different datasets, with the best performance on HumanEval and the worst performance on CNN/DM. We believe this is because the code generation task follows a fixed template, which reduces the difficulty of draft token generation. In contrast, the texts in CNN/DM are more diverse, making the generated drafts more different from the target tokens. In addition, our method

		MT-bench			GSM8K			CNN/DM			HumanEval		
Model	Method	Speedup	α	τ	Speedup	α	τ	Speedup	α	τ	Speedup	α	τ
LLaMA2-Chat 7B	VDM	1.49x	0.39	1.96	1.37x	0.36	1.80	1.11x	0.30	1.51	1.69x	0.40	2.15
	MEDUSA-1	1.48x	0.34	1.71	1.40x	0.33	1.63	1.20x	0.28	1.41	1.78x	0.39	1.97
	EAGLE	1.82x	0.50	2.50	1.75x	0.51	2.54	1.50x	0.38	1.92	2.01x	0.55	2.73
	EDD	**1.97x**	**0.55**	**2.73**	**1.83x**	**0.51**	**2.57**	**1.58x**	**0.41**	**2.06**	**2.09x**	**0.57**	**2.87**
LLaMA2-Chat 13B	VDM	1.53x	0.39	1.95	1.44x	0.37	1.84	1.14x	0.30	1.52	1.68x	0.40	2.03
	MEDUSA-1	1.59x	0.36	1.71	1.46x	0.32	1.58	1.26x	0.28	1.40	1.72x	0.38	1.91
	EAGLE	1.95x	0.53	2.66	1.99x	0.51	2.54	1.48x	0.41	2.04	2.09x	0.55	2.75
	EDD	**2.03x**	**0.54**	**2.68**	**2.08x**	**0.54**	**2.68**	**1.61x**	**0.42**	**2.10**	**2.16x**	**0.57**	**2.86**
Vicuna1.5 7B	VDM	1.53x	0.40	2.02	1.37x	0.37	1.83	1.11x	0.30	1.51	1.69x	0.41	2.06
	MEDUSA-1	1.58x	0.36	1.81	1.37x	0.33	1.63	1.17x	0.28	1.41	1.81x	0.41	2.06
	EAGLE	1.91x	0.54	2.70	1.73x	0.49	2.44	1.45x	0.36	1.81	1.89x	0.52	2.60
	EDD	**2.00x**	**0.55**	**2.77**	**1.81x**	**0.51**	**2.54**	**1.58x**	**0.41**	**2.05**	**2.00x**	**0.55**	**2.77**
Vicuna1.5 13B	VDM	1.58x	0.40	2.01	1.43x	0.37	1.83	1.14x	0.30	1.49	1.72x	0.43	2.13
	MEDUSA-1	1.66x	0.38	1.91	1.47x	0.33	1.63	1.24x	0.28	1.39	1.80x	0.39	1.95
	EAGLE	2.01x	0.54	2.68	1.68x	0.47	2.34	1.36x	0.38	1.90	1.83x	0.51	2.55
	EDD	**2.08x**	**0.56**	**2.80**	**1.80x**	**0.49**	**2.46**	**1.60x**	**0.42**	**2.09**	**1.95x**	**0.53**	**2.63**

Table 1: Performance comparison of our proposed EDD and the baseline draft model. In each draft stage, we allow the draft model to generate a draft sequence of length 5 using greedy search. VDM denotes the vanilla draft model, which has the same model structure as EDD but does not utilize encoding information from LLMs. For a fair comparison, all baselines were re-implemented in our experimental environment.

achieves better speedup on larger LLMs because the speed difference between the draft model and the target model is greater. In the future, we will verify our method on even larger models.

4.3 Analysis

4.3.1 Impact of Hyperparameters

We explore the impact of different hyperparameter settings on our method by conducting experiments on 1,000 samples randomly selected from the GSM8k training set.

Impact of Hidden States. The hidden states at different layers of LLMs contain varying types of information (Men et al., 2024; Jin et al., 2024). We use LLaMA2-Chat 7B as the target model and explore the effects of its hidden states at different layers on EDD. The experimental results are shown in Figure 5. When using hidden states from the last layer or shallow intermediate layers, the average acceptance length is relatively low, as these layers tend to overemphasize either global or local information. Moreover, the best performance is achieved when using the hidden states from the 28th (fourth-to-last) layer, indicating that the information encoded in this layer provides effective assistance for the draft decoding process. Therefore, we utilize the hidden states from the fourth-to-last layer as the encoding results of the LLMs.

Impact of Node Expansion Width. The expansion width of each node significantly affects the

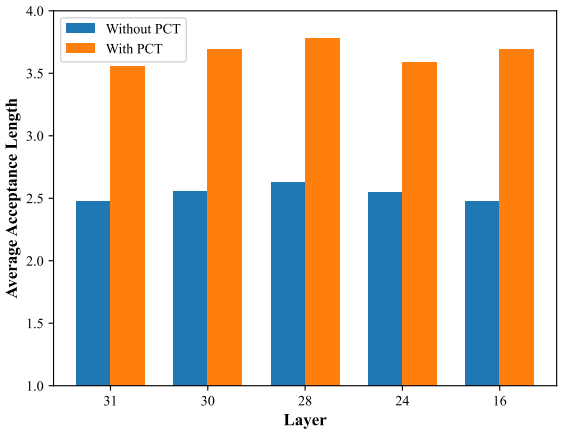


Figure 5: The impact of hidden states at different layers of LLMs on EDD performance.

construction of the candidate tree, so we conducted experiments to explore the impact of different node expansion widths on our PCT method. As shown in Figure 6, when the node expansion width is less than 5, both the average acceptance length and the average number of nodes increase as the width grows. However, when the node expansion width exceeds 5, its effect on PCT becomes negligible. These experimental results demonstrate that PCT can effectively prune candidate trees and remove redundant nodes. More importantly, compared to EAGLE-2, PCT contains only about half the number of nodes while achieving a competitive average acceptance length. This shows that PCT accurately retains the truly valuable nodes through

		MT-bench			GSM8K			CNN/DM			HumanEval		
Model	Method	Speedup	α	τ	Speedup	α	τ	Speedup	α	τ	Speedup	α	τ
LLaMA2-Chat 7B	VCT	2.49x	0.71	3.56	2.36x	0.69	3.44	1.76x	0.50	2.53	2.83x	**0.78**	3.91
	CAPE	2.56x	0.67	3.35	2.38x	0.63	3.13	1.83x	0.49	2.47	2.93x	0.74	3.71
	EAGLE-2	2.87x	0.67	**3.99**	2.68x	0.64	**3.83**	1.91x	0.46	**2.77**	3.13x	0.72	4.31
	PCT	**3.04x**	**0.74**	**3.99**	**2.87x**	**0.72**	3.80	**2.04x**	**0.69**	2.74	**3.35x**	0.75	**4.49**
LLaMA2-Chat 13B	VCT	2.63x	0.74	3.71	2.97x	0.72	3.61	1.86x	0.51	2.53	3.13x	**0.78**	3.88
	CAPE	2.68x	0.71	3.53	3.07x	0.70	3.52	1.81x	0.47	2.35	3.15x	0.73	3.66
	EAGLE-2	3.08x	0.69	**4.15**	3.27x	0.71	**4.24**	2.27x	0.53	**3.15**	3.35x	0.71	4.26
	PCT	**3.27x**	**0.75**	4.13	**3.49x**	**0.75**	**4.24**	**2.39x**	**0.70**	3.03	**3.62x**	0.74	**4.39**
Vicuna1.5 7B	VCT	2.51x	0.72	3.59	2.43x	0.71	3.54	1.78x	0.51	2.55	2.80x	**0.78**	3.90
	CAPE	2.56x	0.67	3.37	2.36x	0.62	3.09	1.83x	0.50	2.49	2.90x	0.72	3.62
	EAGLE-2	3.01x	0.70	**4.20**	2.70x	0.65	3.88	2.03x	0.47	**2.80**	3.05x	0.70	4.19
	PCT	**3.08x**	**0.74**	4.09	**2.95x**	**0.74**	**3.89**	**2.04x**	**0.70**	2.70	**3.30x**	0.72	**4.22**
Vicuna1.5 13B	VCT	2.76x	**0.75**	3.77	2.51x	0.64	3.19	1.92x	0.51	2.53	2.93x	**0.72**	3.58
	CAPE	2.78x	0.73	3.65	2.57x	0.58	2.92	1.97x	0.49	2.45	2.99x	0.71	3.56
	EAGLE-2	3.17x	0.73	4.35	2.77x	0.64	**3.82**	2.17x	0.50	**3.01**	3.15x	0.69	4.12
	PCT	**3.38x**	**0.75**	**4.36**	**2.89x**	**0.70**	**3.72**	**2.33x**	**0.70**	**2.84**	**3.32x**	**0.72**	**4.07**

Table 2: The performance of our proposed PCT and baseline candidate tree construction methods on the EDD. VCT represents the vanilla candidate tree, where the node expansion width is fixed at 3, and the tree depth is fixed at 5. For CAPE (Du et al., 2024) and EAGLE-2 (Li et al., 2024a), we conduct experiments following the settings described in their original papers.

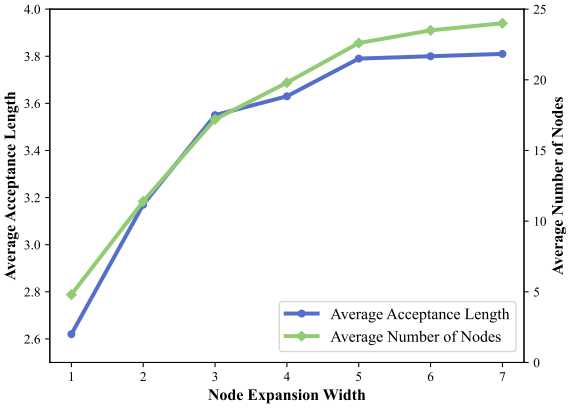


Figure 6: The influence of node expansion width on the average acceptance length and the average number of nodes in the PCT method.

the expected time gain at each step.

4.3.2 Ablation Study

We conduct ablation experiments based on LLaMA2-Chat 7B to verify the effectiveness of each proposed module. Specifically, we compare several variants of our method: (1) **w/o encoding result**: the draft model does not use the encoding results of the LLM; (2) **w/o KL divergence**: the standard cross-entropy loss is used for training; (3) **w/o random division length**: the block length is fixed to 5 during training; (4) **w/o expected time gain**: the expected time gain is not estimated, and confidence score is directly used to prune the candidate tree; (5) **w/o removing leaf nodes**: leaf nodes whose overall acceptance confidence is lower than the threshold are not removed.

The experimental results are shown in Table 3. As we can see, removing any module causes a decline in the performance of our method. First, not using encoding results leads to significant performance degradation. This indicates that the encoding results from the LLM improve the performance of EDD, enabling it to generate more accurate draft tokens. Additionally, KL divergence and random block division length further enhance the EDD training process. Second, directly using confidence to prune candidate trees significantly reduces both the average acceptance rate and the average acceptance length. This demonstrates that expected time gain is a better node evaluation criterion, effectively retaining valuable nodes. Third, failing to delete redundant leaf nodes leads to a decreased speedup ratio. Although this variant does not affect the average acceptance length, it introduces more nodes during the verification phase and increases latency. Due to space constraints, more analysis is presented in Appendix A.

5 Related Work

Significant efforts have been made to improve the efficiency of LLMs, including techniques such as knowledge distillation (Yang et al., 2023; Chen et al., 2023b), pruning (Sun et al., 2023; Yin et al.), quantization (Lin et al., 2024; Yao et al., 2024), and

Method	Speedup	α	τ
EDD+PCT	**3.04x**	**0.74**	**3.99**
w/o encoding result	2.58x	0.55	3.04
w/o KL divergence	2.86x	0.67	3.60
w/o random division length	2.95x	0.69	3.72
w/o expected time gain	2.73x	0.58	3.45
w/o remove leaf nodes	2.95x	0.74	3.99

Table 3: Ablation study of our method on MT-bench.

early exit (Chen et al., 2023c; Zeng et al., 2024). However, these methods can only improve the forward pass speed of LLMs and do not address the speed bottleneck of autoregressive decoding.

To enable LLMs to generate multiple tokens in a single forward pass, SD was proposed (Leviathan et al., 2022). This approach uses a draft model to quickly generate multiple tokens, which are then verified in parallel using LLMs. The performance of SD is primarily determined by the efficiency of the draft model and the acceptance rate of the draft tokens. Therefore, improving the acceptance rate while maintaining the generation speed of the draft model is key to optimizing the SD method.

The vanilla SD method trains a separate autoregressive model that is much smaller than LLMs to generate draft tokens quickly. However, there is a significant gap between the performance of vanilla draft models and LLMs, resulting in a low acceptance rate. Medusa (Cai et al., 2024) adds additional decoding heads to LLMs to generate multiple future tokens in parallel. However, this non-autoregressive generation method suffers from the multimodality problem (Gu et al., 2017), which affects the acceptance rate. Another line of work uses the substructure of LLMs as the draft model to achieve self-speculative decoding (Elhoushi et al., 2024; Liu et al., 2024; Xia et al., 2024). Although these methods can achieve a high acceptance rate, the inference speed of the draft model is slow, leading to an insignificant acceleration effect. Unlike the above methods, EDD fully leverages the encoding ability of LLMs and uses KL divergence to align with LLMs, enabling it to generate draft tokens accurately with a tiny model size.

On the other hand, previous works have shown that predicting multiple candidate sequences to construct a draft candidate tree can significantly improve the acceptance rate (Du et al., 2024; Cai et al., 2024; Li et al., 2024a; Miao et al., 2023; Guan et al., 2024). However, these methods typically generate candidate trees based on fixed widths and depths, resulting in many redundant nodes that increase latency during the draft and verification phases. To address this issue, we propose the PCT algorithm, which effectively prunes candidate trees based on expected time gain, reducing redundant time overhead while maintaining a high acceptance rate.

6 Conclusion

This paper proposes two methods to improve the acceleration effect of speculative decoding, focusing on draft model design and candidate tree construction. First, we design the EDD, which treats the LLM as an encoder and uses its encoding results as soft prompts to help generate more accurate draft tokens. Second, we propose the PCT algorithm, which estimates the expected time gain of each node based on confidence and effectively prunes candidate trees. Experimental results verify the effectiveness of our method, which significantly improves the performance of speculative decoding.

7 Limitations

Although our method can significantly improve the acceleration effect of speculative decoding, it also has some limitations.

First, the proposed EDD needs to be trained. Additionally, since EDD relies on the encoding results of LLMs, this further increases the training time. However, the required computational overhead for training remains acceptable because EDD is very small.

Second, similar to previous speculative decoding methods, our approach is more suitable for scenarios with a batch size of 1. We believe that LLMs will be more widely deployed on personal devices in the future, so our method still holds great potential for application.

Third, we have not yet verified the effectiveness of our method on larger LLMs. In the future, we will conduct experiments on a broader range of LLMs to explore whether our method can achieve better acceleration effects with larger models.

Acknowledgements

This work was supported by NSFC grant (No. 62136002 and 62477014), Ministry of Education Research Joint Fund Project (8091B042239), and Fundamental Research Funds for the Central Universities.

References

Tianle Cai, Yuhong Li, Zhengyang Geng, Hongwu Peng, Jason D. Lee, De huai Chen, and Tri Dao. 2024. Medusa: Simple llm inference acceleration framework with multiple decoding heads. *ArXiv*, abs/2401.10774.

Charlie Chen, Sebastian Borgeaud, Geoffrey Irving, Jean-Baptiste Lespiau, L. Sifre, and John M. Jumper. 2023a. Accelerating large language model decoding with speculative sampling. *ArXiv*, abs/2302.01318.

Hailin Chen, Amrita Saha, Steven Hoi, and Shafiq Joty. 2023b. Personalized distillation: Empowering open-sourced LLMs with adaptive learning for code generation. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 6737–6749, Singapore. Association for Computational Linguistics.

Mark Chen, Jerry Tworek, Heewoo Jun, and et al. 2021. Evaluating large language models trained on code. *ArXiv*, abs/2107.03374.

Yanxi Chen, Xuchen Pan, Yaliang Li, Bolin Ding, and Jingren Zhou. 2023c. Ee-llm: Large-scale training and inference of early-exit large language models with 3d parallelism. *arXiv preprint arXiv:2312.04916*.

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

Cunxiao Du, Jing Jiang, Yuanchen Xu, 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. *ArXiv*, abs/2402.02082.

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. *ArXiv*, abs/2404.16710.

Jiatao Gu, James Bradbury, Caiming Xiong, Victor OK Li, and Richard Socher. 2017. Non-autoregressive neural machine translation. *arXiv preprint arXiv:1711.02281*.

Jiatao Gu and X. Kong. 2020. Fully non-autoregressive neural machine translation: Tricks of the trade. In *Findings*.

Xinyan Guan, Yanjiang Liu, Xinyu Lu, Boxi Cao, Ben He, Xianpei Han, Le Sun, Jie Lou, Bowen Yu, Yaojie Lu, and Hongyu Lin. 2024. Search, verify and feedback: Towards next generation post-training paradigm of foundation models via verifier engineering.

Chuan Guo, Geoff Pleiss, Yu Sun, and Kilian Q. Weinberger. 2017. On calibration of modern neural networks. *ArXiv*, abs/1706.04599.

Mingyu Jin, Qinkai Yu, Jingyuan Huang, Qingcheng Zeng, Zhenting Wang, Wenyue Hua, Haiyan Zhao, Kai Mei, Yanda Meng, Kaize Ding, Fan Yang, Mengnan Du, and Yongfeng Zhang. 2024. Exploring concept depth: How large language models acquire knowledge at different layers? *ArXiv*, abs/2404.07066.

Jungo Kasai, Nikolaos Pappas, Hao Peng, James Cross, and Noah A. Smith. 2020. Deep encoder, shallow decoder: Reevaluating non-autoregressive machine translation. In *International Conference on Learning Representations*.

Diederik P. Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980.

Yaniv Leviathan, Matan Kalman, and Yossi Matias. 2022. Fast inference from transformers via speculative decoding. In *International Conference on Machine Learning*.

Yuhui Li, Fangyun Wei, Chao Zhang, and Hongyang Zhang. 2024a. Eagle-2: Faster inference of language models with dynamic draft trees. In *Conference on Empirical Methods in Natural Language Processing*.

Yuhui Li, Fangyun Wei, Chao Zhang, and Hongyang Zhang. 2024b. Eagle: Speculative sampling requires rethinking feature uncertainty. *ArXiv*, abs/2401.15077.

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. *Proceedings of Machine Learning and Systems*, 6:87–100.

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

Xin Men, Mingyu Xu, Qingyu Zhang, Bingning Wang, Hongyu Lin, Yaojie Lu, Xianpei Han, and Weipeng Chen. 2024. Shortgpt: Layers in large language models are more redundant than you expect. *ArXiv*, abs/2403.03853.

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. 2023. Specinfer: Accelerating large language model serving with tree-based speculative inference and verification. *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 3*.

OpenAI. 2023. Gpt-4 technical report.

Abigail See, Peter J. Liu, and Christopher D. Manning. 2017. Get to the point: Summarization with pointer-generator networks. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1073–1083, Vancouver, Canada. Association for Computational Linguistics.

ShareGPT. 2023. Sharegpt.

Mingjie Sun, Zhuang Liu, Anna Bair, and J. Zico Kolter. 2023. A simple and effective pruning approach for large language models. *ArXiv*, abs/2306.11695.

Hugo Touvron, Louis Martin, Kevin R. Stone, and et al. 2023. Llama 2: Open foundation and fine-tuned chat models. *ArXiv*, abs/2307.09288.

Heming Xia, Yongqi Li, Jun Zhang, Cunxiao Du, and Wenjie Li. 2024. Swift: On-the-fly self-speculative decoding for llm inference acceleration. *ArXiv*, abs/2410.06916.

Bohao Yang, Chen Tang, Kangning Zhao, Chenghao Xiao, and Chenghua Lin. 2023. Effective distillation of table-based reasoning ability from llms. In *International Conference on Language Resources and Evaluation*.

Zhewei Yao, Xiaoxia Wu, Cheng Li, Stephen Youn, and Yuxiong He. 2024. Exploring post-training quantization in llms from comprehensive study to low rank compensation. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pages 19377–19385.

Lu Yin, You Wu, Zhenyu Zhang, Cheng-Yu Hsieh, Yaqing Wang, Yiling Jia, Gen Li, AJAY KUMAR JAISWAL, Mykola Pechenizkiy, Yi Liang, et al. Outlier weighed layerwise sparsity (owl): A missing secret sauce for pruning llms to high sparsity. In *Forty-first International Conference on Machine Learning*.

Ziqian Zeng, Yihuai Hong, Hongliang Dai, Huiping Zhuang, and Cen Chen. 2024. Consistentee: A consistent and hardness-guided early exiting method for accelerating language models inference. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pages 19506–19514.

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, and et al. 2023. Judging llm-as-a-judge with mt-bench and chatbot arena. *ArXiv*, abs/2306.05685.

Yongchao Zhou, Kaifeng Lyu, Ankit Singh Rawat, Aditya Krishna Menon, Afshin Rostamizadeh, Sanjiv Kumar, Jean-François Kagy, and Rishabh Agarwal. 2023. Distillspec: Improving speculative decoding via knowledge distillation. *ArXiv*, abs/2310.08461.

A More Analysis

A.1 Number of Nodes on Different Tasks

Our PCT algorithm can adaptively prune candidate trees. Therefore, we conducted experiments on LLaMA2-Chat 7B to explore whether the number of nodes in the candidate trees constructed by PCT varies across different task categories in the MT-bench. As shown in Figure 7, PCT constructs candidate trees of appropriate sizes based on diverse inputs. Specifically, PCT can distinguish the difficulty of tasks and reserve more nodes for tasks where draft tokens are more likely to be accepted, thereby improving the average acceptance length. Furthermore, for different inputs within the same task category, PCT still generates candidate trees with significant variations in the number of nodes. These experimental results demonstrate that our PCT method can effectively screen each node by estimating its expected time gain, overcoming the limitation of previous methods that generate a fixed number of nodes for any input.

A.2 Case Study

We display generation examples of Vanilla SD and our approach in Table 6. It can be seen that Vanilla SD generates only a few tokens per iteration, resulting in a poor acceleration effect. This result shows the significant gap between the performance of the vanilla draft model and the LLM, making it challenging to align its output with that of the LLM. In contrast, EDD generates multiple tokens per iteration, demonstrating its ability to leverage the encoding information from the LLM to significantly improve the quality of draft tokens. Furthermore, combining PCT with EDD can further enhance performance, enabling the LLM to generate nearly 100 tokens in just a dozen iterations.

		MT-bench			GSM8K			CNN/DM			HumanEval		
Model	Method	Speedup	α	τ	Speedup	α	τ	Speedup	α	τ	Speedup	α	τ
LLaMA2-Chat 7B	VDM	1.30x	0.34	1.70	1.31x	0.32	1.60	1.05x	0.26	1.30	1.58x	0.34	1.70
	MEDUSA-1	1.39x	0.32	1.60	1.35x	0.31	1.55	1.17x	0.27	1.35	1.71x	0.36	1.81
	EAGLE	1.77x	0.46	2.30	1.66x	0.50	2.50	1.45x	0.35	1.75	1.91x	0.50	2.50
	EDD	**1.87x**	**0.51**	**2.55**	**1.75x**	**0.51**	**2.51**	**1.53x**	**0.39**	**1.95**	**2.02x**	**0.55**	**2.75**
LLaMA2-Chat 13B	VDM	1.39x	0.36	1.78	1.40x	0.36	1.96	1.14x	0.31	1.56	1.63x	0.39	1.95
	MEDUSA-1	1.46x	0.31	1.53	1.39x	0.30	1.50	1.24x	0.27	1.36	1.67x	0.36	1.78
	EAGLE	1.85x	0.50	2.51	1.86x	0.48	2.41	1.45x	0.40	2.00	2.03x	0.53	2.65
	EDD	**1.93x**	**0.51**	**2.56**	**2.01x**	**0.52**	**2.60**	**1.53x**	**0.42**	**2.10**	**2.10x**	**0.55**	**2.75**
Vicuna1.5 7B	VDM	1.50x	0.40	2.00	1.33x	0.36	1.80	1.12x	0.31	1.55	1.55x	0.42	2.10
	MEDUSA-1	1.56x	0.35	1.75	1.37x	0.34	1.70	1.17x	0.28	1.48	1.78x	0.40	2.01
	EAGLE	1.85x	0.52	2.60	1.69x	0.49	2.44	1.44x	0.35	1.75	1.80x	0.51	2.55
	EDD	**2.01x**	**0.55**	**2.75**	**1.78x**	**0.51**	**2.55**	**1.55x**	**0.40**	**2.01**	**1.99x**	**0.55**	**2.76**
Vicuna1.5 13B	VDM	1.51x	0.38	1.91	1.40x	0.36	1.80	1.10x	0.29	1.45	1.64x	0.41	2.05
	MEDUSA-1	1.64x	0.37	1.85	1.46x	0.33	1.65	1.22x	0.28	1.40	1.77x	0.39	1.95
	EAGLE	1.95x	0.53	2.65	1.62x	0.46	2.30	1.33x	0.38	1.89	1.80x	0.50	2.49
	EDD	**2.03x**	**0.55**	**2.75**	**1.77x**	**0.50**	**2.48**	**1.53x**	**0.51**	**2.55**	**1.94x**	**0.52**	**2.62**

Table 4: Performance comparison of our proposed EDD and the baseline draft model when temperature=1.

		MT-bench			GSM8K			CNN/DM			HumanEval		
Model	Method	Speedup	α	τ	Speedup	α	τ	Speedup	α	τ	Speedup	α	τ
LLaMA2-Chat 7B	VCT	2.30x	0.68	3.40	2.23x	0.65	3.25	1.66x	0.48	2.42	2.75x	**0.77**	3.86
	CAPE	2.39x	0.65	3.24	2.30x	0.61	3.03	1.70x	0.46	2.32	2.88x	0.70	3.51
	EAGLE-2	2.67x	0.65	**3.90**	2.53x	0.61	**3.65**	1.83x	0.43	**2.58**	3.00x	0.70	4.20
	PCT	**2.91x**	**0.72**	3.87	**2.76x**	**0.70**	3.60	**1.96x**	**0.67**	**2.58**	**3.15x**	0.71	**4.31**
LLaMA2-Chat 13B	VCT	2.40x	0.68	3.39	2.89x	0.69	3.45	1.70x	0.49	2.46	3.01x	**0.75**	3.75
	CAPE	2.51x	0.66	3.31	2.93x	0.68	3.40	1.71x	0.45	2.25	3.08x	0.70	3.51
	EAGLE-2	2.89x	0.66	**3.95**	3.08x	0.68	**4.08**	2.11x	0.50	**3.02**	3.19x	0.69	4.16
	PCT	**3.10x**	**0.72**	3.94	**3.25x**	**0.73**	4.04	**2.24x**	**0.67**	3.01	**3.33x**	0.71	**4.18**
Vicuna1.5 7B	VCT	2.34x	0.69	3.46	2.30x	0.68	3.40	1.68x	0.50	2.49	2.66x	**0.75**	3.75
	CAPE	2.41x	0.64	3.22	2.30x	0.60	2.99	1.77x	0.49	2.45	2.73x	0.70	3.52
	EAGLE-2	2.95x	0.67	**4.02**	2.61x	0.63	**3.78**	1.95x	0.45	**2.70**	3.01x	0.69	4.15
	PCT	**3.02x**	**0.72**	**4.02**	**2.83x**	**0.72**	3.70	**2.01x**	**0.70**	2.63	**3.11x**	0.69	**4.20**
Vicuna1.5 13B	VCT	2.66x	0.73	3.66	2.45x	0.63	3.15	1.81x	0.49	2.45	2.85x	**0.71**	3.49
	CAPE	2.73x	0.71	3.55	2.47x	0.57	2.83	1.89x	0.49	2.44	2.88x	0.69	3.47
	EAGLE-2	3.13x	0.70	4.22	2.61x	0.61	3.68	2.11x	0.49	**2.95**	3.03x	0.66	3.97
	PCT	**3.30x**	**0.74**	**4.23**	**2.75x**	**0.67**	**3.72**	**2.27x**	**0.70**	2.75	**3.29x**	0.70	**4.06**

Table 5: The performance of PCT and other candidate tree construction methods on the EDD when temperature=1.

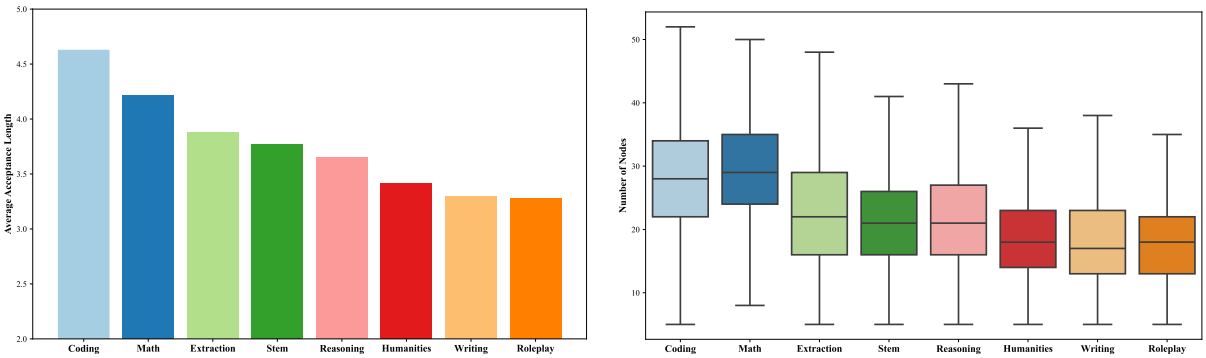


Figure 7: Average acceptance length (left) and distribution of PCT node numbers (right) on different task categories of MT-bench. For simplicity, we do not plot the outliers in the box plots.

Input	Please describe the concept of machine learning. Could you elaborate on the differences between supervised, unsupervised, and reinforcement learning? Provide real-world examples of each.
Vanilla SD	Machine learning / is a / subfield / of artificial / intelligence (/ AI) / that involves / the use of algorithms / and statistical / models to enable / machines / to learn from data / , make decisions / , and improve / their / performance on / a / specific / task over time / . Machine learning algorithms / are designed to recognize / patterns / and relationships in data /, and use / this information / to make predictions / or / decisions without / being explicitly / programmed / to do so. / There are three main types / of machine learning: / super/ vised learning, un / supervised learning, / and rein / forcement learning. · · ·
EDD	Machine learning is a / subfield of artificial / intelligence (/ AI) that involves the / use / of algorithms and / statistical models to enable / machines to learn from / data, make / decisions, and improve / their performance on a / specific task over / time. Machine / learning algorithms / are designed to / recognize / patterns and relationships / in data, / and use this / information to / make predictions / or decisions / without being explicitly / programmed to do so. / There are three / main types of machine learn / ing: super / vised learning, un / supervised learning, / and rein / forcement learning. · · ·
EDD + PCT	Machine learning is a subfield of artificial intelligence / (AI) that involves the use / of algorithms and statistical / models to enable / machines to learn from data / , make decisions / , and improve their performance on a / specific task over time. / Machine learning algorithms are designed to / recognize patterns and relationships / in data, and use / this information to make predictions or / decisions without / being explicitly programmed to do so. / There are three main types / of machine learning: super / vised learning, unsupervised learning, / and reinforcement learning. · · ·
Input	The city of Vega intends to build a bridge that will span the Vegona River, covering a distance of 1.8 kilometers. The proposed location falls within a seismically active area that has experienced several high-magnitude earthquakes. Given these circumstances, what would be the best approach to constructing the bridge?
Vanilla SD	Building / a struct / ural engineer / , I would / recommend / the following / approach to / construct / the bridge / in / a seismically / active area: / 1. Site / -specific / seismic / ha / zard assessment: / Conduct a / thorough analysis of / the / seismic hazard / at the / proposed bridge / location, / including / an assessment of / the maximum / expected / earthquake magnitude / , peak / ground acceleration, / and response spectra / . This will / help determine the / appropriate / design and / construction / methods / to ensure the / bridge can / withstand / seismic / forces. · · ·
EDD	Building / a struc / tural engineer, / I would / recommend the following / approach / to construct the bridge in a / seismically / active / area: / 1. Site / -specific seismic ha / zard assessment: / Conduct a thorough analysis / of the / seismic hazard / at the proposed / bridge location, / including an assessment / of the maximum / expected earthquake / magnitude, / peak ground / acceleration, / and response spectra. / This will help determine / the appropriate design and / construction / methods to ensure / the bridge can withstand / seismic forces. · · ·
EDD + PCT	Building a struct / ural engineer, I would recommend / the following approach to construct / the bridge in / a seismically active area: / 1. Site-specific seismic ha / zard assessment: Conduct a thorough / analysis of the seismic hazard at / the proposed bridge location, / including an assessment of the maximum / expected / earthquake magnitude / , peak ground / acceleration, and response spect/ ra. / This will help determine / the appropriate design and / construction methods to ensure / the bridge can / withstand seismic forces. · · ·
Input	Imagine you are participating in a race with a group of people. If you have just overtaken the second person, what's your current position? Where is the person you just overtook?
Vanilla SD	Great, / an exciting race / ! / Let / 'm see / ... / just / as I overto / ok the / second / person, / the race, / my / position has / now become / 2nd / place. / The / person / I / just / overto / ok is / now in / 3rd place.
EDD	Great, / an exciting race / ! / Let'm see / ... / just / as I overto / ok the second person / , the race / , my position has / now become / 2nd place. The / person / I just overto / ok is now in 3rd / place.
EDD + PCT	Great, an exciting / race! Let'm see... / just as I over / took the second person, the race / , my / position has now become / 2nd place. The person / I just overtook is / now in 3rd place.

Table 6: Examples from MT-bench for our method and Vanilla SD. The target model is LLaMA2-Chat 7B. We employ / to divide the content generated by each iteration.