CORAL: Learning Consistent Representations across Multi-step Training with Lighter Speculative Drafter

Anonymous ACL submission

Abstract

Speculative decoding is a powerful technique that accelerates Large Language Model (LLM) inference by leveraging a lightweight speculative draft model. However, existing designs suffers in performance due to misalignment between training and inference. Recent methods have tried to solve this issue by adopting a multi-step training strategy, but the complex inputs of different training steps make it harder for the draft model to converge. To address this, we propose CORAL, a novel framework that improves both accuracy and efficiency in speculative drafting. CORAL introduces Cross-Step Representation Alignment, a method that enhances consistency across multiple training steps, significantly improving speculative drafting performance. Additionally, we identify the LM head as a major bottleneck in the inference speed of the draft model. We introduce a weight-grouping mechanism that selectively activates a subset of LM head parameters during inference, substantially reducing the latency of the draft model. We evaluate CORAL on three LLM families and three benchmark datasets, achieving speedup ratios of 2.50×-4.07×, outperforming state-of-the-art methods such as EAGLE-2 and HASS. Our results demonstrate that CORAL effectively mitigates training-inference misalignment and delivers significant speedup for modern LLMs with large vocabularies.

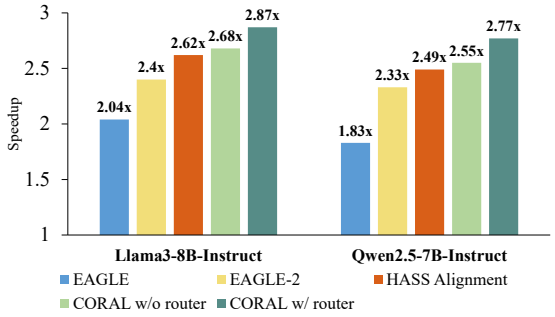

Figure 1: Speedup ratios of different methods on Llama3-8B and Qwen2.5-7B at temperature=0, averaging on MT-bench, HumanEval, and GSM8K datasets. We present full results in Table 2 and this chart is only a subset of all comparisons.

1 Introduction

Large Language Models (LLMs), such as GPT (OpenAI, 2023) and Llama series (Touvron et al., 2023a,b; Grattafiori et al., 2024), have demonstrated exceptional capabilities in various natural language processing tasks. However, achieving stronger model performance often depends on increasing the number of model parameters (Kaplan et al., 2020; Hoffmann et al., 2022), which leads to higher costs in both training and inference. Thus, achieving strong performance while maintaining quick response is a crucial part in LLM implementations. Under common hardware conditions, transformer decoder-based LLMs are memory-bound (Dao et al., 2022), which means that the generation speed is mainly determined by memory access and bandwidth, rather than arithmetic computations. This allows for the acceleration of generation using speculative decoding (Chen et al., 2023; Leviathan et al., 2023). The general idea of speculative decoding is to utilize one or multiple lightweight draft models to predict the output of target LLM for several upcoming timesteps, and then verify the drafted predictions in parallel using the target model. The memory-bound characteristic guarantees that the parallel verification of multiple tokens does not incur a significant increase in latency compared to generating a single token.

Recently, autoregressive draft models, such as EAGLE (Li et al., 2024b), have received widespread attention for their excellent speedup performance. For training, EAGLE uses not only the output tokens but also the last hidden states from target LLM as input to the draft model, while during the drafting phase, the draft model uses its own hidden states from the previous timestep,

Model	Hidden	Inter. size	Vocab	W_d / W_t	L_d / L_t
Llama2-7B	4096	11008	32000	350M/6301M(5.6%)	1.36ms/23.65ms(5.8%)
Llama3-8B	4096	14336	128256	741M/7157M(10.4%)	2.58ms/26.06ms(9.9%)
Qwen2.5-7B	3584	18944	152064	767M/6743M(11.4%)	2.69ms/24.58ms(10.9%)

Table 1: Parameters and latencies of Llama3-8B, Llama2-7B, and Qwen2.5-7B draft and target models. W_d, W_t and L_d, L_t denote the parameter counts and latency of draft and target model. In the table, M represents 1024×1024. Parameters of the embedding layer are not calculated because they do not participate in general matrix multiplication (GEMM). Latencies are tested with one token on a single NVIDIA A6000 GPU.

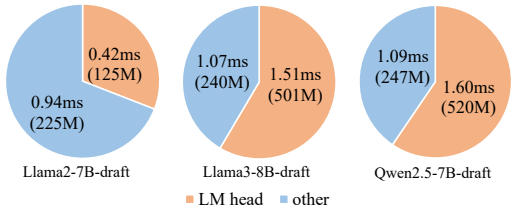

Figure 2: Parameters and latencies of Llama3-8B, Llama2-7B, Qwen2.5-7B draft model. For a model with large vocabulary, the LM head takes the majority of the drafting latency.

which may contain biases. This misalignment leads to a decrease in the prediction accuracy of the draft model. HASS (Zhang et al., 2024) proposes a multi-step training strategy, where the hidden states output by the draft model are fed back into itself multiple times during training, allowing the draft model to learn the feature distribution of the inference phase. In Section 2 we will provide more detailed discussions on them.

Although HASS exhibits impressive performance, there are still some limitations to multi-step training. Specifically, their design causes the input features at different training steps to vary, which might be challenging for a lightweight draft model to adapt to. The discrepancy of each training step may also introduce potential gradient conflicts. Furthermore, modern LLMs are increasingly moving towards large vocabularies to obtain better performance (Tao et al., 2024). For example, previous model such as Llama2 has a small vocabulary size of only 32000 (Touvron et al., 2023b), while the vocabulary size of Llama3 (Grattafiori et al., 2024) is 128256, and that of Qwen2.5 (Yang et al., 2024) is 152064. Such large vocabularies lead to an increase in the parameter size of the Language Model head (LM head), resulting in increased overhead of drafting, which is presented in Table 1. As demonstrated in Figure 2, the heavy LM head could potentially dominate the latency of draft model. However, few studies have focused on this aspect.

In this paper, we introduce CORAL (learning COnsistent Representations Across multi-step training with Lighter speculative drafter), a speculative decoding method that improves the alignment between the draft model and the target model while maintaining high drafting speed. We first propose Cross-Step Representation Alignment (CSRA), which leverages the idea of contrastive learning to enforce consistency among the output features of each training step. The constraint on features makes them more stable, and thus improves the training efficiency and the performance of the draft model. Furthermore, by grouping the LM heads, we significantly reduce the activated parameters of the draft model with large vocabulary size, thereby decreasing the wall time of speculative decoding.

We evaluate acceleration capability of CORAL on multi-turn conversation, code generation, and mathematical reasoning tasks using the MT-Bench, HumanEval and GSM8K datasets, respectively. The results show that our method achieves 2.50×-4.07× speedup over vanilla decoding at a temperature of 0, surpassing state-of-the-art methods such as EAGLE-2 and HASS.

Our key contributions can be summarized as follows.

1. We propose Cross-Step Representation Alignment, a technique that enables the draft model to learn consistent representations across multiple timesteps.

2. We find that the vocabulary size can significantly influence the latency of the draft model, and propose a novel method which selectively activates a subset of LM head parameters during inference using a router.

3. CORAL achieves speedup ratios of 2.50×-4.07× on various LLMs and datasets, outperforming existing speculative decoding methods such as EAGLE-2 and HASS.

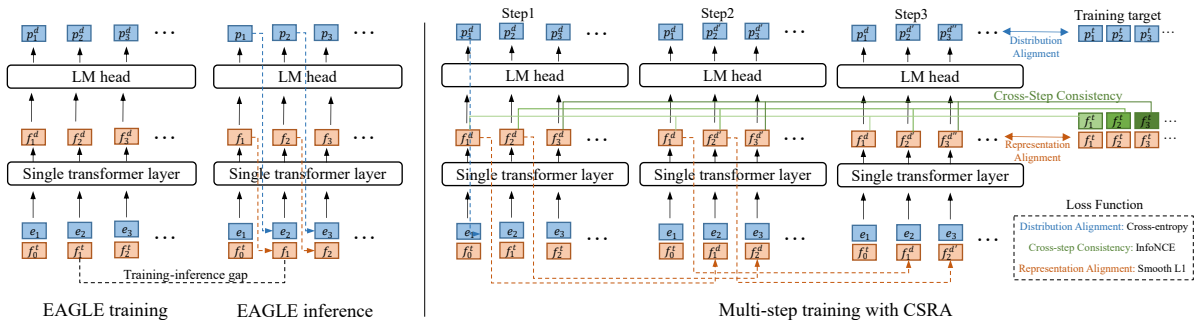

Figure 3: Demonstration of EAGLE training / inference and multi-step training with CSRA. f denotes feature and e denotes embedding. Superscripts indicate the source of the variable, with t and d denoting the target model and draft model. Subscripts index the position of a feature or embedding. For example, f_3^t means the feature in position 3 and comes from the target model. For multi-step training, we use apostrophes to distinguish the outputs of different training steps. Specifically, we denote the output feature of step 1 as f^d, and for step 2 and 3 we use $f^{d'}$ and $f^{d''}$, respectively. Compared to HASS, CSRA introduces additional constraints on feature consistency. The training target is applied at each step, and we only illustrate it once for the sake of clarity.

2 Preliminaries

In this section, we provide some background information related to speculative decoding and review some existing methods, including EAGLE and HASS.

2.1 Speculative Decoding

Speculative decoding (Chen et al., 2023; Leviathan et al., 2023) aims to accelerate the generation speed of autoregressive LLMs. Vanilla speculative decoding employs a lightweight model (draft model) to generate a chain of candidate tokens for the next γ timesteps, which are then verified in parallel by the original LLM (target model) and decide whether to accept them or not. Since the latency of LLM generation mainly lies in the memory access, parallel verification of multiple tokens does not significantly impact the latency of the target LLM, although the computational cost is multiplied.

The acceleration capability of speculative decoding is typically evaluated using two metrics: average acceptance length τ and the actual Speedup Ratio (SR). A drafting-verification cycle consists of one token provided by the target model and multiple candidates generated by the draft model over γ time steps. The average acceptance length τ is defined as the number of new tokens generated in a single drafting-verification cycle.

Ideally, we can estimate the speedup ratio using τ and the latencies of draft and target model:

$$SR \approx \tau \times \frac{L_t'}{\gamma \times L_d + L_t}, \tag{1}$$

where L_t and L_d denote the latency of the target model and draft model, respectively. L_t' denotes the latency for evaluating multiple tokens one time, it could be slightly different from L_t depending on the hardware. Some additional overheads might also contribute to latency, such as comparing the probabilities of tokens from draft and target models to determine acceptance. However, since these overheads typically do not dominate the overall latency, it is a good choice to ignore them when estimating the speedup ratio.

From Equation (1) we can see the speedup ratio is primarily influenced by two factors: the alignment between the draft model and the target model, which mainly influences τ, and the ratio of their latencies. Specifically, the lower the latency of the draft model and the better alignment between the two models, the higher the speedup ratio will be achieved by speculative decoding.

2.2 EAGLE

EAGLE (Li et al., 2024b) is a lightweight autoregressive draft model that leverages a single transformer layer identical to that of the target model. The LM head of draft model is reused directly from the target model, with its parameters frozen. EAGLE discovers that utilizing the feature (*i.e.*, the last hidden states) of the target model can effectively enhance the alignment between the draft and target model. For training, the input of the draft model at position s is the current token t_s and the feature of the target model at position $s-1$. The token t_s will first be transformed into embedding e_s, and then concatenated with the feature. A linear layer is adopted to reduce the dimensions before

the single transformer layer.

The training target of EAGLE is to align the feature (regression) and probability distribution (classification) of the draft and target model. EAGLE uses smooth L1 as the regression loss and cross-entropy as the classification loss.

EAGLE selects multiple candidates at each timestep during drafting, resulting in a tree-shaped structure rather than a chain. Tree decoding offers more possible trajectories than chain decoding, leading to a higher acceptance length. EAGLE-2 (Li et al., 2024a) improves the fixed tree structure to a dynamic one and achieves better performance.

2.3 HASS

HASS (Zhang et al., 2024) addresses the inconsistency between the training and inference phases of EAGLE by introducing a multi-step training strategy. As demonstrated in Figure 3, EAGLE uses the feature of the target model for training, whereas in inference, the draft model uses its own feature. HASS solves this problem by feeding the output feature of draft model back into itself for multiple times. To expose the draft model to inference-time conditions during training, attention masks from different training steps require careful adjustment. HASS also incorporates other improvements on EAGLE, but they are orthogonal to multi-step alignment. In this paper, we focus mainly on HASS alignment, and all references to HASS in the remainder of this paper denote HASS alignment unless otherwise specified.

While HASS improves the accuracy of draft models in autoregressive generation, we argue that there are still unresolved issues due to the discrepancies between representations from multiple training steps (*i.e.*, f^d, $f^{d'}$ and $f^{d''}$ in Figure 3). It is harder for the draft model to adapt to more complex inputs and the conflicting gradients from multiple steps may hinder convergence speed.

3 Method

In this section, we first introduce Cross-Step Representation Alignment, a method designed to strengthen the alignment between the draft model and the target model. We then analyze the speedup ratio and identify the LM head of the draft model as a bottleneck. To address this issue, we propose the LM head router, a novel solution that aims to reduce the latency of the draft model.

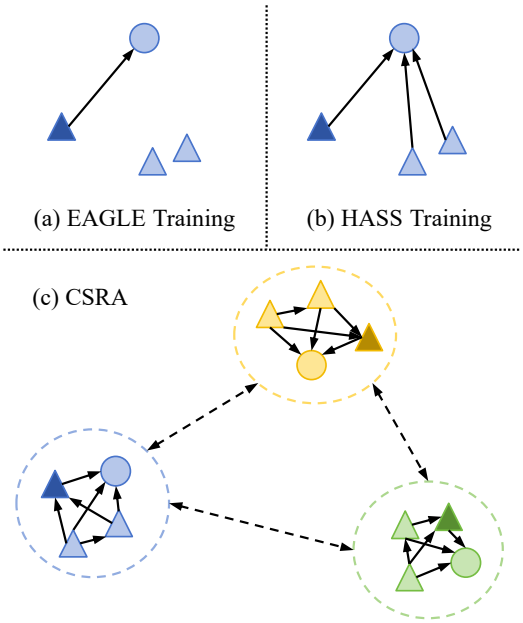

Figure 4: Comparison of EAGLE training, HASS training and CSRA. Here ◯ denotes training target, △ denotes output features from different steps. Triangles filled with darker colors represent the first step's output. Different colors represent outputs or targets of different positions. Optimization direction is marked as $\rightarrow$, and the dashed $\leftrightarrow$ means repulsion.

3.1 Cross-Step Representation Alignment

Cross-Step Representation Alignment (CSRA) leverages the idea of contrastive learning (Chopra et al., 2005; Schroff et al., 2015). Specifically, in multi-step training, we treat the output features at the same position in a sentence as positive views of the same sample, while all other features are considered negative samples.

Assuming current training step is t, the output features of current step are $F_t \in \mathbb{R}^{B\times S\times D}$, where B, S, and D represent the batch size, sequence length, and hidden dimension, respectively. Naturally, we regard them as $B \times S$ samples, and each sample has t positive views, while all other features are considered negative samples.

For each output feature f in current training step, our objective is to minimize its distance to other positive views while maximizing the distance to negative samples. To achieve this, we normalize the features and compute the InfoNCE loss (van den Oord et al., 2018) as the objective function, which encourages the feature to be closer to its positive views and away from negative samples:

$$\mathcal{L}_{CSRA} = -\log \frac{\exp(\text{sim}(q, f^+)/\tau)}{\sum_{f\in F} \exp(\text{sim}(q, f)/\tau)}, \quad (2)$$

where q and f^+ denotes the query feature and positive views, and F is the set of all features along with the targets. The similarity function $\mathrm{sim}(\cdot,\cdot)$ is defined as cosine similarity. Here τ is the temperature hyperparameter. Figure 4 shows the differences between EAGLE / HASS training and CSRA.

The training loss can be defined as:

$$\mathcal{L} = w_{reg}\mathcal{L}_{reg} + w_{cls}\mathcal{L}_{cls} + w_{CSRA}\mathcal{L}_{CSRA}, \quad (3)$$

where $\mathcal{L}_{reg}$ and $\mathcal{L}_{cls}$ represent the regression loss and classification loss, respectively. Since $\mathcal{L}_{CSRA}$ primarily affects representation learning, we maintain w_{cls} consistent with EAGLE and adjust another two weights according to different target models. For detailed parameter settings, please refer to Appendix A.

3.2 Estimation of Speedup Ratio

As discussed in Section 2.1, the generation speed is primarily constrained by memory bandwidth. Therefore, the theoretical latency $L_{theo.}$ in generation phase is proportional to the LLM's parameter count W_{LLM}:

$$L_{theo.} \propto W_{LLM}. \quad (4)$$

However, this estimation is not always accurate due to the following factors: 1) Not all operators and computing graphs are fully optimized. 2) The latency of some element-wise operators (*e.g.*, activation, norm) is not reflected in the parameter count. This issue is particularly noticeable for PyTorch, because it is not a framework optimized for inference.

Luckily, the draft model and target one share the same transformer structure, and the extra latency caused by the aforementioned factors is relatively consistent in both models. This allows us to estimate the wall time and speedup ratio of speculative decoding based on the parameters of draft model and target model:

$$\frac{L_d}{L_t} \approx \frac{W_d}{W_t}, \quad (5)$$

$$SR \approx \tau \times \frac{W_t}{\gamma \times W_d + W_t}, \quad (6)$$

where W_d, W_t and L_d, L_t denote the parameter counts and latency of draft and target model, respectively. Note that the embedding layer does not participate in general matrix multiplication (GEMM), therefore its parameters should not be included in latency estimation. Table 1 presents the latencies and parameters of different LLMs, along with their corresponding draft models. The results suggest that estimating the latency ratio between the draft and target models based on their parameter counts is relatively accurate. Notably, for Llama3-8B and Qwen2.5-7B, the latency of draft model is approximately 10% of that of target model. As the depth of drafting increases, the latency of draft model is expected to contribute significantly to the overall wall time.

Furthermore, it is also possible to estimate the latency of each component of the draft model based on their parameter count. As shown in Figure 2, in cases with large vocabularies, the latency of LM head accounts for a significant proportion of the total latency, which provides us with a valuable insight: If we can reduce the activated weights of the LM head, the overall speedup will be substantially improved.

3.3 LM Head Router

As mentioned in Section 3.2, for draft models with large vocabularies, LM head constitutes the major part of drafting latency. We propose the LM head router, aiming to group the LM head and then activate only a subset of LM head parameters during drafting, as demonstrated in Figure 5.

Assuming a LLM with a vocabulary size V, we divide the LM head equally into N groups, each with a vocabulary size of $v = V/N$. We utilize a router to select which group to activate. The output of router can be outlined as follows:

$$p_{router} = \mathrm{Softmax}(W_2(\mathrm{act}(W_1 h) + h)), \quad W_2 \in \mathbb{R}^{N\times d}, W_1 \in \mathbb{R}^{d\times d}, \quad (7)$$

where h denotes the hidden states of draft model, d is the hidden size.

Let $p(x)$, $q(x)$ denote the predicted and target distribution, and $p_{\text{group}}(x^n)$ denote the probability distribution within a specific group n. After selecting a particular group, the softmax probability is calculated by logits in this group, independent of the logits in other groups.

Then the final distribution with router should be

$$p(x) = p_{\text{router}}(n) \cdot p_{\text{group}}(x^n). \quad (8)$$

For each group, $\sum p_{\text{group}}(x^n) = 1$, and for router we have $\sum p_{\text{router}}(n) = 1$. Therefore, the final $p(x)$ is normalized.

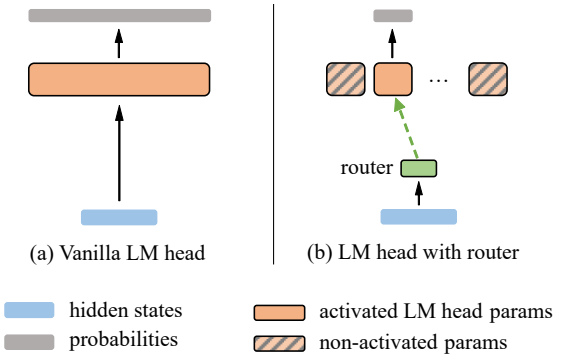

Figure 5: Demonstration of LM head router in draft model. With the router, we only output probabilities of one or multiple subsets of vocabulary.

The training target of LM head router is the sum of target probabilities in each group, namely $q_{\text{router}}(n) = \sum q_{\text{group}}(x^n)$. We use cross-entropy as the loss function:

$$\mathcal{L}_{\text{router}} = -\sum q_{\text{router}}(n) \log p_{\text{router}}(n). \quad (9)$$

It is evident that, although the LM head router reduces the latency of the draft model, it comes at the cost of a slight decrease in acceptance length τ due to imperfect routing accuracy. Based on Equation (5) and (6), the LM head router gets its best performance when 1) the LM head accounts for a significant portion of the latency of draft model 2) the latency ratio between the draft model and the target model is substantial. Therefore, we only apply the LM head router to models with large vocabularies (Qwen2.5, Llama3) and relatively small sizes (7B, 14B).

We adopt a two-stage training strategy, where we first train the draft model following the standard training procedure (either single-step or multi-step), and then fix the weights of draft model and train the router separately. For further discussion, please refer to Appendix D.

4 Experiments

In this section, we first introduce the experimental setup, then discuss the overall effectiveness of our method, and finally present the ablation studies on CSRA and LM head router.

4.1 Experimental Setup

Target LLMs. We choose Llama3-Instruct-8B/70B(Grattafiori et al., 2024), Llama2-chat-7B/13B(Touvron et al., 2023b) and Qwen2.5-Instruct-7B/14B(Yang et al., 2024) as our target models.

Tasks. We choose multiple datasets covering three tasks, including MT-Bench(Zheng et al., 2023) for multi-turn dialogue, GSM8K(Cobbe et al., 2021) for mathematical reasoning, and HumanEval(Chen et al., 2021) for code generation. For 7B/14B models, experiments are conducted with batch size of 1 on a single NVIDIA A6000 48G GPU. For Llama3-70B, we use 4×A6000 GPUs due to memory requirements.

Metrics. Since CORAL is a lossless speculative decoding strategy, it is not necessary to measure the generation quality. For acceleration, we use two metrics to evaluate the performance:

- **Speedup Ratio**: the actual speedup ratio compared to vanilla decoding.
- **Acceptance Length τ**: the average number of new tokens generated per drafting-verification cycle.

Comparisons. We use vanilla decoding as the baseline (1.00×) to measure the speedup ratio. We primarily compare CORAL with the latest lossless speculative decoding methods, including EAGLE, EAGLE-2, and HASS. Since EAGLE is already one of the fastest speculative decoding methods, we choose EAGLE as the speculative decoding baseline and do not compare with other methods with lower speedup ratios.

Implementation. Our implementation is based on the open source repositories of HASS[1] and EAGLE-2[2], and the settings are primarily identical to those of them. All models are trained with ShareGPT dataset for 20 epochs with batch size of 2 per GPU. For HASS and CORAL, the default step for training is set to 3. Our system prompt for Llama3 is slightly different from that of EAGLE, please refer to Appendix E for detailed discussion. For inference, we employ a tree depth of 6 and select 60 candidate tokens for all models.

4.2 Effectiveness and Ablation Studies

4.2.1 Effectiveness

We present the acceptance lengths τ and speedup ratios of three datasets in Table 2. The results show that CSRA achieves the best performance in both τ and speedup ratio (SR) in all experiments we have tested, surpassing EAGLE, EAGLE-2, and HASS. The advantages of CSRA are more pronounced for

[1]https://github.com/HArmonizedSS/HASS
[2]https://github.com/SafeAILab/EAGLE

		MT-bench τ / SR		HumanEval τ / SR		GSM8K τ / SR		Average τ / SR
model	method	T=0	T=1	T=0	T=1	T=0	T=1	T=0
L2-13B	EAGLE	3.93/3.04×	3.23/2.35×	4.51/3.47×	3.47/2.56×	4.01/3.10×	3.51/2.59×	4.15/3.20×
	EAGLE-2	4.80/3.16×	4.68/3.06×	5.59/3.75×	5.41/3.60×	4.98/3.38×	4.84/3.25×	5.12/3.43×
	HASS	5.20/3.42×	5.02/3.26×	5.99/4.01×	5.79/3.86×	5.32/3.60×	5.24/3.51×	5.50/3.68×
	CORAL	**5.25/3.45**×	**5.10/3.32**×	**6.06/4.07**×	**5.90/3.93**×	**5.39/3.65**×	**5.25/3.51**×	**5.57/3.72**×
L2-7B	EAGLE	3.80/2.67×	3.21/2.10×	4.29/3.04×	3.55/2.33×	3.84/2.73×	3.48/2.30×	3.87/2.81×
	EAGLE-2	4.68/2.89×	4.45/2.70×	5.34/3.35×	5.02/3.11×	4.70/2.98×	4.67/2.89×	4.91/3.07×
	HASS	5.02/3.09×	4.77/2.88×	5.71/3.58×	5.35/3.30×	5.11/3.25×	4.99/3.10×	5.28/3.31×
	CORAL	**5.09/3.13**×	**4.86/2.94**×	**5.73/3.58**×	**5.48/3.40**×	**5.12/3.25**×	**5.05/3.13**×	**5.31/3.32**×
L3-70B	EAGLE	2.87/2.24×	2.62/2.02×	3.73/2.93×	3.45/2.67×	3.46/2.71×	3.23/2.50×	3.35/2.63×
	EAGLE-2	4.08/2.70×	3.91/2.61×	4.95/3.31×	4.89/3.27×	4.03/2.70×	3.73/2.50×	4.35/2.90×
	HASS	4.10/2.71×	4.00/2.65×	5.23/3.49×	5.10/3.40×	4.12/2.76×	3.83/2.56×	4.48/2.99×
	CORAL	**4.23/2.79**×	**4.13/2.72**×	**5.31/3.54**×	**5.19/3.46**×	**4.34/2.90**×	**3.91/2.61**×	**4.63/3.08**×
L3-8B	EAGLE	2.63/1.65×	2.30/1.35×	3.65/2.29×	3.13/1.85×	3.47/2.18×	3.05/1.78×	3.25/2.04×
	EAGLE-2	4.16/2.28×	3.84/2.08×	4.78/2.61×	4.64/2.50×	4.21/2.32×	3.94/2.13×	4.38/2.40×
	HASS	4.48/2.45×	4.12/2.21×	5.31/2.89×	5.12/2.76×	4.56/2.51×	4.18/2.28×	4.78/2.62×
	CORAL	**4.57/2.50**×	**4.15/2.24**×	**5.43/2.95**×	**5.28/2.83**×	**4.70/2.58**×	**4.39/2.38**×	**4.90/2.68**×
	CORAL w/ r.	4.26/**2.63**×	3.92/**2.39**×	5.22/**3.21**×	5.03/**3.07**×	4.42/**2.76**×	4.12/**2.53**×	4.63/**2.87**×
Q2.5-14B	EAGLE	2.63/1.83×	2.33/1.55×	3.31/2.31×	2.82/1.88×	3.62/2.52×	3.21/2.16×	3.19/2.22×
	EAGLE-2	4.08/2.36×	3.76/2.15×	5.01/2.89×	4.85/2.78×	4.62/2.69×	4.58/2.65×	4.57/2.65×
	HASS	4.52/2.59×	4.12/2.35×	5.50/3.18×	5.37/3.07×	5.03/2.92×	4.91/2.83×	5.02/2.90×
	CORAL	**4.56/2.62**×	**4.13/2.35**×	**5.64/3.26**×	**5.40/3.09**×	**5.16/3.00**×	**5.12/2.95**×	**5.12/2.96**×
	CORAL w/ r.	4.26/**2.74**×	3.88/**2.46**×	5.31/**3.44**×	5.12/**3.28**×	4.80/**3.14**×	4.72/**3.05**×	4.79/**3.11**×
Q2.5-7B	EAGLE	2.53/1.56×	2.17/1.23×	3.04/1.87×	2.62/1.49×	3.32/2.05×	2.86/1.63×	2.96/1.83×
	EAGLE-2	3.91/2.13×	3.45/1.86×	4.62/2.53×	4.36/2.35×	4.23/2.33×	4.07/2.21×	4.25/2.33×
	HASS	4.15/2.26×	3.65/1.96×	4.96/2.71×	4.74/2.55×	4.53/2.49×	4.35/2.35×	4.55/2.49×
	CORAL	**4.22/2.30**×	**3.83/2.05**×	**5.09/2.78**×	**4.86/2.62**×	**4.67/2.57**×	**4.50/2.44**×	**4.66/2.55**×
	CORAL w/ r.	4.02/**2.50**×	3.62/**2.21**×	4.86/**3.05**×	4.57/**2.81**×	4.38/**2.76**×	4.16/**2.58**×	4.42/**2.77**×

Table 2: Acceptance lengths τ and speedup ratio (SR) of different methods on MT-bench, HumanEval, and GSM8K datasets with temperature $T \in \{0, 1\}$. The best results are in **bold**, and some minor advantages may be obscured due to rounding. We also calculate the average τ and SR under $T = 0$ for a more direct comparison. L2, L3, Q2.5 represents Llama2-Chat, Llama3-Instruct, and Qwen2.5-Instruct, respectively. As clarified in Section 3.3, we apply LM head router for relatively small LLMs with large vocabularies (denoted as CORAL w/ r.), such as Qwen2.5-7B/14B and Llama3-8B. For Llama2 series and Llama3-70B, we use CSRA only.

LLMs with larger vocabularies, whereas the benefits are less significant for earlier models such as Llama2. For LM head router, we set the group number to 16 and choose the top-2 groups for the best performance. Although the router sacrifices some acceptance length, the overall speedup ratio benefits from reduced latency and shows a considerable increase.

4.2.2 Ablation Study on CSRA

We adjust the number of training steps and make a more detailed comparison with HASS. Since CSRA and HASS employ the same draft model, the inference overheads are identical, we therefore compare the acceptance length only. The results in Table 3 show that CSRA consistently outperforms HASS under different training steps.

To provide a more intuitive measure of the alignment between the draft model and the target model, we compare the acceptance rates α of HASS and CSRA at different timesteps during inference, as shown in Figure 6. The results show that CSRA generally outperforms HASS at different timesteps.

	MT-bench		HumanEval		GSM8K	
step	HASS	CSRA	HASS	CSRA	HASS	CSRA
2	4.41	4.53	5.24	5.35	4.50	4.60
3	4.48	4.57	5.31	5.43	4.56	4.70
4	4.46	4.58	5.39	5.55	4.58	4.70

Table 3: Acceptance length of Llama3-8B under different alignment steps. Step-3 is the default setting.

4.2.3 Ablation Study on LM Head Router

The LM head router has two hyperparameters: the total number of groups N, and the number of top-

n groups to activate during inference. A larger group number, although leading to activating fewer parameters, would increase the difficulty of training and damage accuracy. Similarly, how many groups to activate is also a trade-off between speed and accuracy. We perform a grid search over these two hyperparameters in the MT-bench dataset with Llama3-8B, and the results are shown in Table 4.

CORAL T=0						
N	top1	top2	top3	top4	top6	top8
N/A	2.50×	-	-	-	-	-
4	2.60×	2.46×	-	-	-	-
8	2.62×	2.61×	2.54×	-	-	-
16	2.53×	**2.63**×	2.60×	2.57×	-	-
32	2.41×	2.59×	2.60×	2.61×	2.56×	-
64	2.33×	2.51×	2.55×	2.57×	2.57×	2.53×
EAGLE-2 T=0						
N	top1	top2	top3	top4	top6	top8
N/A	2.28×	-	-	-	-	-
4	**2.44**×	2.29×	-	-	-	-
8	2.40×	2.39×	2.33×	-	-	-
16	2.30×	2.41×	2.39×	2.36×	-	-
32	2.24×	2.37×	2.40×	2.38×	2.35×	-
64	2.18×	2.33×	2.37×	2.37×	2.37×	2.33×

Table 4: Speedup of Llama3-8B with LM head router on MT-bench dataset. We group the LM head parameters into N groups and selectively activate top-n of them. N/A denotes the results without LM head router.

The results show that our method consistently yields significant improvements, regardless of whether multi-step training is employed. For CORAL, dividing the LM head into 16 groups and activating the top-2 groups during inference brings the best speedup performance. Since the optimal setting may vary across different LLMs and cannot be easily estimated, we recommend empirical studies to identify the optimal configuration.

5 Related Work

There has been a significant amount of work in accelerating LLMs. Some methods focus on reducing the number of parameters, such as low-bit quantization (Dettmers et al., 2022; Frantar et al., 2023; Xiao et al., 2023; Lin et al., 2024), and model distillation (Gu et al., 2024; Ko et al., 2024; Zhong et al., 2024). Recently, some studies have also explored activating only a subset of model parameters during inference to reduce memory access cost (Du et al., 2022; Fedus et al., 2022). Speculative decoding

Figure 6: Acceptance rates in MT-bench dataset. Here n-α denotes the acceptance rate of the n-th token.

(Chen et al., 2023; Leviathan et al., 2023) leverages the memory-bound nature of decoder-only LLMs and achieves lossless acceleration using a drafting-verification framework.

Research on speculative decoding has primarily focused on two areas: 1) drafter design, 2) verification strategy. For drafter design, Medusa (Cai et al., 2024) attaches multiple heads to the original LLM and predict multiple subsequent tokens one time. Hydra (Ankner et al., 2024) improves Medusa by enhancing correlations between draft heads. Clover (Xiao et al., 2024) introduces an RNN-based draft head. Some methods utilize more information from target model to improve alignment, EAGLE (Li et al., 2024b) combines the output token and last hidden states of target LLMs to resolve the uncertainty in drafter's prediction. GLIDE (Du et al., 2024) reuses the KV cache of target LLMs. For the verification strategy, Hu and Huang (2024); Sun et al. (2024) find that the acceptance length of speculative sampling is not optimal and take into account the probability of subsequent tokens. SpecInfer (Miao et al., 2024) proposes decoding tree for verification. Sequoia (Chen et al., 2024), EAGLE-2 (Li et al., 2024a), and OPT-tree (Wang et al., 2024) adopts a dynamic tree structure.

6 Conclusion

This paper proposes CORAL, an efficient speculative decoding method. We introduce Cross-Step Representation Alignment, which effectively mitigates training-inference misalignment and improves the accuracy of speculation. Additionally, we propose the LM head router, a plug-and-play module designed to reduce the latency of the draft model. We compare CORAL with other state-of-the-art methods on various LLMs and datasets, and the results show that CORAL achieves the best speedup performance.

Limitations

There are mainly two limitations in this work. Firstly, the introduction of CSRA loss may lead to a slight increase in regression loss, which results in a decrease in the acceptance length if the draft model is trained with single step. This issue can be addressed by multi-step training. Secondly, adopting a large vocabulary is a trend in the development of modern LLMs, and our LM head router is specifically designed for LLMs with large vocabularies. It might not be suitable for models with small vocabularies, as the computational overhead of LM head is limited in the overall wall time of speculative decoding. In this case, the time saved by the draft model cannot compensate for the loss in acceptance length.

References

Zachary Ankner, Rishab Parthasarathy, Aniruddha Nrusimha, Christopher Rinard, Jonathan Ragan-Kelley, and William Brandon. 2024. Hydra: Sequentially-dependent draft heads for medusa decoding. *arXiv preprint arXiv:2402.05109*.

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*.

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

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Pondé de Oliveira Pinto, et al. 2021. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*.

Zhuoming Chen, Avner May, Ruslan Svirschevski, Yuhsun Huang, Max Ryabinin, Zhihao Jia, and Beidi Chen. 2024. Sequoia: Scalable, robust, and hardware-aware speculative decoding. *arXiv preprint arXiv:2402.12374*.

Sumit Chopra, Raia Hadsell, and Yann LeCun. 2005. Learning a similarity metric discriminatively, with application to face verification. In *Proceedings of the Conference on Computer Vision and Pattern Recognition*.

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 preprint arXiv:2110.14168*.

Tri Dao, Daniel Y. Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. 2022. Flashattention: Fast and memory-efficient exact attention with io-awareness. In *Advances in Neural Information Processing System*.

Tim Dettmers, Mike Lewis, Younes Belkada, and Luke Zettlemoyer. 2022. LLM.int8(): 8-bit matrix multiplication for transformers at scale.

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

Nan Du, Yanping Huang, Andrew M. Dai, Simon Tong, Dmitry Lepikhin, Yuanzhong Xu, Maxim Krikun, et al. 2022. GLaM: Efficient scaling of language models with mixture-of-experts. In *Proceedings of the International Conference on Machine Learning*.

William Fedus, Barret Zoph, and Noam Shazeer. 2022. Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity. *Journal of Machine Learning Research*.

Elias Frantar, Saleh Ashkboos, Torsten Hoefler, and Dan Alistarh. 2023. GPTQ: Accurate post-training quantization for generative pre-trained transformers.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, et al. 2024. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*.

Yuxian Gu, Li Dong, Furu Wei, and Minlie Huang. 2024. MiniLLM: Knowledge distillation of large language models. In *Proceedings of the International Conference on Learning Representations*.

Jordan Hoffmann, Sebastian Borgeaud, Arthur Mensch, Elena Buchatskaya, et al. 2022. Training compute-optimal large language models. *arXiv preprint arXiv:2203.15556*.

Zhengmian Hu and Heng Huang. 2024. Accelerated speculative sampling based on tree monte carlo. In *Proceedings of the International Conference on Machine Learning*.

Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B. Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. 2020. Scaling laws for neural language models. *arXiv preprint arXiv:2001.08361*.

Jongwoo Ko, Sungnyun Kim, Tianyi Chen, and Se-Young Yun. 2024. Distillm: Towards streamlined distillation for large language models. In *Proceedings of the International Conference on Machine Learning*.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph Gonzalez, Hao Zhang, and Ion Stoica. 2023. Efficient

memory management for large language model serving with pagedattention. In *Proceedings of the29th 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*.

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

Yuhui Li, Fangyun Wei, Chao Zhang, and Hongyang Zhang. 2024b. EAGLE: Speculative sampling requires rethinking feature uncertainty. 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 the Annual Conference on Machine Learning and Systems*.

Xupeng Miao, Gabriele Oliaro, Zhihao Zhang, Xinhao Cheng, Zeyu Wang, engxin 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 ACM International Conference on Architectural Support for Programming Languages and Operating Systems*.

OpenAI. 2023. GPT-4 technical report. *arXiv preprint arXiv:2303.08774*.

Florian Schroff, Dmitry Kalenichenko, and James Philbin. 2015. FaceNet: A unified embedding for face recognition and clustering. In *Proceedings of the Conference on Computer Vision and Pattern Recognition*.

Rico Sennrich, Barry Haddow, and Alexandra Birch. 2016. Neural machine translation of rare words with subword units. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics*.

Ziteng Sun, Uri Mendlovic, Yaniv Leviathan, Asaf Aharoni, Ahmad Beirami, Jae Hun Ro, and Ananda Theertha Suresh. 2024. Block verification accelerates speculative decoding. *arXiv preprint arXiv:2403.10444*.

Chaofan Tao, Qian Liu, Longxu Dou, Niklas Muennighoff, Zhongwei Wan, Ping Luo, Min Lin, and Ngai Wong. 2024. Scaling laws with vocabulary: Larger models deserve larger vocabularies.

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, Aurelien Rodriguez, Armand Joulin, Edouard Grave, and Guillaume Lample. 2023a. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*.

Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, et al. 2023b. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*.

Aäron van den Oord, Yazhe Li, and Oriol Vinyals. 2018. Representation learning with contrastive predictive coding. *arXiv preprint arXiv:1807.03748*.

Changhan Wang, Kyunghyun Cho, and Jiatao Gu. 2020. Neural machine translation with byte-level subwords. In *Proceedings of the AAAI Conference on Artificial Intelligence*.

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

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

Guangxuan Xiao, Ji Lin, Mickaël Seznec, Hao Wu, Julien Demouth, and Song Han. 2023. SmoothQuant: Accurate and efficient post-training quantization for large language models. In *Proceedings of the International Conference on Machine Learning*.

An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, et al. 2024. Qwen2.5 technical report. *arXiv preprint arXiv:2412.15115*.

Lefan Zhang, Xiaodan Wang, Yanhua Huang, and Ruiwen Xu. 2024. Learning harmonized representations for speculative sampling. *arXiv preprint arXiv:2408.15766*.

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric P. Xing, Hao Zhang, Joseph E. Gonzalez, and Ion Stoica. 2023. Judging llm-as-a-judge with mt-bench and chatbot arena. In *Advances in Neural Information Processing Systems*.

Qihuang Zhong, Liang Ding, Li Shen, Juhua Liu, Bo Du, and Dacheng Tao. 2024. Revisiting knowledge distillation for autoregressive language models. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics*.

A Hyperparameters in CSRA Loss

The temperature of $\mathcal{L}_{CSRA}$ is set to 0.07, consistent with some previous works such as CLIP (Zheng et al., 2023).

Then we set w_{reg} to 0.5, half of EAGLE's original setting. The weight of CSRA loss is adjusted according to different target models, making the values of $w_{CSRA}\mathcal{L}_{CSRA}$ and $w_{reg}\mathcal{L}_{reg}$ roughly the same. In this way, the loss imposed on representation is approximately the same as EAGLE/HASS training.

Based on the values of $w_{reg}\mathcal{L}_{reg}$, we choose $w_{CSRA} = 0.2$ for Qwen2.5-7B, $w_{CSRA} = 0.15$ for Llama3-8B, $w_{CSRA} = 0.1$ for Llama3-70B, Qwen2.5-14B and Llama2-7B, and 0.05 for Llama2-13B.

B Training Details

We utilize a fixed dataset of 68,000 examples from ShareGPT[3] as our training set, which is identical to EAGLE and HASS. CORAL requires approximately 2 days to train a 7B draft model under default settings (training step=3, epoch=20). It is worth noting that draft models with large vocabularies such as Llama3 and Qwen2.5 require more GPU memory compared to Llama2, so we use 4×NVIDIA H20-96G GPUs for training. Training large draft models such as Llama3-70B on A100-40G GPU may result in out-of-memory issues under our experimental settings. We recommend using GPUs with larger memory capacities or choosing other alternatives (*e.g.*, reducing the batch size, model parallelism).

C Single-step Training with CSRA

We do not recommend using the CSRA loss in the context of single-step training. Our empirical findings suggest that introducing the CSRA loss may lead to a slight increase in regression loss, likely due to the mismatch between the two optimization objectives. Specifically, the CSRA loss focuses solely on the angular relationships between the output features, without imposing any constraints on the feature norm, whereas the regression loss aims to learn features that are identical to the target. The increase in regression loss may damage the acceptance length. We present the results of CSRA with single-step training in Table 5.

A plausible explanation for this phenomenon is that in single-step training, the draft model lacks exposure to subsequent steps, therefore the L1 distance between the prediction and target feature is relatively more critical. In contrast, for multi-step training, the draft model learns to adapt to subsequent steps, making the discriminative power of different representations and the multi-step consistency more crucial.

[3]https://huggingface.co/datasets/Aeala/ShareGPT_Vicuna_unfiltered

	MT-bench	HumanEval	GSM8K
EAGLE-2	4.16	4.78	4.21
CSRA Step1	4.10	4.70	4.10

Table 5: Acceptance length of Llama3-8B EAGLE-2 and CORAL model with single-step training.

D Discussion on LM Head Router

In this section, we will discuss some issues of LM head router.

Tree decoding. In tree decoding, each timestep contains multiple candidate tokens. Since each candidate requires a different set of LM head groups, we need to activate all the involved groups, which may bring additional latency. In some cases, we even need to activate the entire LM head parameters (*e.g.*, if we take the top two groups and top 10 candidates, the worst-case scenario might require activating 20 groups).

This issue can be addressed through appropriate grouping strategies. First, dividing the tokens into more groups helps alleviate the problem. For instance, with a total of 32 groups, selecting the top 10 candidates from the top 2 groups ensures that the LM head parameters are not fully activated, even in the worst-case scenario. Second, modern LLMs utilize BPE (Sennrich et al., 2016) or BBPE (Wang et al., 2020) for tokenization, where higher-frequency tokens tend to be concentrated in groups with smaller indices. As a result, such an extreme scenario is unlikely to occur in practice.

Two-stage training. There are mainly two reasons for adopting two-stage training. Firstly, the two-stage training strategy ensures that the router serves as a plug-and-play module, without affecting the standalone usage of the first-stage model, thereby providing greater flexibility. Secondly, since the number of groups is a hyperparameter that may require multiple experiments to determine the optimal setting, two-stage training allows us to store the output of draft model and train the router only, making it easier for parameter tuning.

Backends. Although many researches on speculative decoding measure the speedup ratio on PyTorch, we do not consider PyTorch to be a good backend. For example, as shown in Table 2, the

Train	Test	MT-bench	HumanEval	GSM8K
sys_p2	sys_p2	4.16	4.78	4.21
	sys_p1	4.11(-0.05)	4.73(-0.05)	4.27(+0.06)
sys_p1	sys_p1	4.18	4.78	4.38
	sys_p2	3.87(-0.31)	4.17(-0.61)	3.93(-0.45)
open source (sys_p1)	sys_p1	4.24	4.92	4.34
	sys_p2	3.94(-0.30)	4.67(-0.25)	3.91(-0.43)

Table 6: Acceptance lengths of EAGLE-2 for Llama3-8B-Instruct with different system prompts.

FP16 latency of Llama3-8B-draft head on RTX A6000 GPU is 1.51ms, which is close to the theoretical time of 1.3ms (1002M memory access with 768GB/s bandwidth). However, for other parts, which mainly consists of transformer, the actual time is much higher than the theoretical time (1.07ms vs 0.63ms), achieving only about 60% of the theoretical performance.

This is a problem inherent to PyTorch. For instance, in Qwen2 speed benchmark[4], the inference speed of 7B model on A100 80G GPU is only 38 token/s (*i.e.*, 26ms/token), which is far from the theoretical time of about 7ms (estimated by 14G memory access with 2TB/s bandwidth). This problem can be mitigated by using a more optimized backend, such as vLLM (Kwon et al., 2023).

Therefore, the performance of the LM head router may be affected by the hardware and backend conditions. In a well-optimized backend, the router's performance will be better than reported in this paper, as the latency of the LM head will occupy a larger proportion in the draft model.

E Discussion on System Prompt

EAGLE utilizes the system prompt from the official Llama2-chat example[5]:

sys_p1 = You are a helpful, respectful and honest assistant. Always answer as helpfully as possible, while being safe. Your answers should not include any harmful, unethical, racist, sexist, toxic, dangerous, or illegal content. Please ensure that your responses are socially unbiased and positive in nature.\n\nIf a question does not make any sense, or is not factually coherent, explain why instead of answering something not correct. If you don't know the answer to a question, please don't share false information.

The same system prompt is also used in Llama3 drafter training. However, it appears that Llama3 does not have a default system prompt. Nevertheless, we find the system prompt in the official Llama3.3 example[6] is simpler and also widely adopted:

sys_p2 = You are a helpful assistant

The system prompt has a certain impact on the acceptance length and speedup ratio. To investigate this, we compared the open-source Llama3-8B-Instrct draft model in EAGLE official repository (trained with sys_p1) and draft models trained by ourselves using sys_p1 and sys_p2. Our results in Table 6 show that switching between different system prompts might lead to a decrease in speedup and acceptance length on the MT-Bench and Humaneval datasets, while GSM8K is an exception.

Upon closer inspection of the GSM8K results, we find that when using sys_p1, most responses start with a sentence similar to "Let's break this down step by step", whereas when using sys_p2, the beginning if outputs will be more diverse. This suggests that the speedup ratio using sys_p1 might be artificially inflated in some cases.

Furthermore, since longer system prompts provide the draft model with more context, we suppose that detailed prompts and increased information could potentially improve the performance of draft model when the system prompt of training and inference is aligned. However, when the system prompts are not consistent, training the model with a more detailed system prompt may lead to greater performance degradation.

To obtain a more generalizable draft model, we use sys_p2 in all experiments with Llama3-Instruct 8B/70B. We believe a more general and simple system prompt would reflect the draft model's true capabilities more accurately.

[4]https://qwen.readthedocs.io/en/v2.0/benchmark/speed_benchmark.html

[5]https://huggingface.co/blog/llama2

[6]https://github.com/meta-llama/llama-models/blob/main/models/llama3_3/prompt_format.md

F Licenses of Artifacts

We present the licenses of artifacts related to this paper in table 7.

models	Llama3	llama3 license
	Llama2	llama2 license
	Qwen2.5	apache-2.0
datasets	ShareGPT	apache-2.0
	MT-bench	CC-BY-4.0
	HumanEval	MIT
	GSM8K	MIT
codes	EAGLE/EAGLE2	apache-2.0
	HASS	not provided

Table 7: Licenses of artifacts

G Use of AI Assistants

We use Llama3.2-90B to assist with grammar checks and text polishing in the writing of this paper.