Cross-Attention Speculative Decoding

Anonymous Author(s)
Affiliation
Address
email

Abstract

Speculative decoding (SD) is a widely adopted approach for accelerating inference in large language models (LLMs), particularly when the draft and target models are well aligned. However, state-of-the-art SD methods typically rely on tightly coupled, self-attention-based Transformer decoders, often augmented with auxiliary pooling or fusion layers. This coupling makes them increasingly complex and harder to generalize across different models. We present Budget EAGLE (Beagle), the first, to our knowledge, cross-attention-based Transformer decoder SD model that achieves performance on par with leading self-attention SD models (EAGLE-v2) while eliminating the need for pooling or auxiliary components, simplifying the architecture, improving training efficiency, and maintaining stable memory usage during training-time simulation. To enable effective training of this novel architecture, we propose Two-Stage Block-Attention Training, a new method that achieves training stability and convergence efficiency in block-level attention scenarios. Extensive experiments across multiple LLMs and datasets show that Beagle achieves competitive inference speedups and higher training efficiency than EAGLE-v2, offering a strong alternative for architectures in speculative decoding.

1 Introduction

Speculative decoding (SD) (Stern et al. 2018; Sun et al. 2021; Xia, Ge, et al. 2022; Leviathan et al. 2023; Chen et al. 2023; Xia, Z. Yang, et al. 2024) is an effective method for accelerating inference in large language models (LLMs), where a lightweight draft model proposes the next n tokens in advance, reducing the need for multiple target model invocations. The adoption of SD has been growing in industry due to its ability to deliver lossless latency improvements in both greedy and sampling-based decoding (Leviathan et al. 2023; Chen et al. 2023), while also improving utilization of otherwise idle compute during memory-bound decoding phases.

Implementing SD efficiently typically requires replacing generic, often misaligned draft models with dedicated ones that are co-trained alongside the target model to match its output distribution better. As a result, the design and integration of SD models pose both research and engineering challenges. From a research standpoint, identifying an effective draft model remains an open problem, nearly as difficult as designing the target LLM itself. A draft model must closely approximate the target model's token predictions while remaining much smaller for practical deployment. Toward this goal, a range of architectures has been explored: from lightweight MLPs or FFNs (Stern et al. 2018; Cai et al. 2024; Ankner et al. 2024), to RNN-based designs (Cheng et al. 2024), to more expressive Transformer-based decoding heads (Yuhui Li et al. 2025b; Yuhui Li et al. 2024), which have demonstrated superior performance over simpler FFNs (Stern et al. 2018; Cai et al. 2024; Gloeckle et al. 2024).

More recently, self-attention-based autoregressive draft heads have gained traction due to their strong performance in future token prediction. Most of the latest high-performing SD systems (Yuhui Li et al. 2024; Ankner et al. 2024; Yuhui Li et al. 2025a; B. Xiao et al. 2024; L. Zhang et al. 2025) adopt

Submitted to 39th Conference on Neural Information Processing Systems (NeurIPS 2025). Do not distribute.

a similar architecture: a self-attention layer that pools token embeddings and prior target states, often using similar design patterns and even shared code bases. Moreover, speculative decoding support remains limited in current LLM inference frameworks. At the time of writing, SGLang (Zheng, Yin, et al. 2024) is the only publicly available framework that supports one of the leading SD methods, EAGLE-v2, but only within the EAGLE model family (Yuhui Li et al. 2024; Yuhui Li et al. 2025a). Other frameworks (Joao Gante 2023; LM Studio Team 2025) are either still under development or rely on disconnected (and often misaligned) draft and target models, resulting in marginal or even negative speedups. Much of the complexity of importing advanced SD models stems from non-standard architectural components and extensive customization for each new model integration.

Inspired by the success of the deep-shallow configuration in translation (Kasai et al. 2020), which reinterprets decoder-only LLMs as an encoder followed by a single cross-attention decoder, we explore whether a similarly minimal cross-attention structure can serve as a viable alternative to self-attention-based draft models.

Unlike most existing self-attention-based SD solutions, we reduce a standard Transformer decoder (Vaswani et al. 2023) to a minimal cross-attention structure without auxiliary layers. It is shown to match the performance of state-of-the-art SD models for the same training data while maintaining architectural simplicity. On the other hand, self-attention-based draft models generate queries and keys from the same hidden state, which complicates integration with autoregressive inputs and target outputs. To address this, existing work often require auxiliary pooling layers (e.g., of size $2d \times d$, or even $3d \times d$ in EAGLE-v3 (Yuhui Li et al. 2025a) where d is the hidden state dimension) to handle heterogeneous features. In contrast, our cross-attention architecture naturally handles different autoregressive states without pooling or custom fusion layers. Moreover, our cross-attention design enables efficient multi-token prediction during training, akin to "condensing" (Gao et al. 2021a; Gao et al. 2021b) future token information into the draft representation. To fully exploit this, we introduce a novel Two-Stage Block-Attention Training method that makes our architecture not only simple but effective. Unlike prior Training-Time Test (TTT) based on self-attention (Yuhui Li et al. 2025a; L. Zhang et al. 2025), our training scheme maintains constant memory usage and avoids the need to scale hidden states with the number of simulated steps, enabling full training of a 7B model on a single 24GiB GPU. We believe that our method offers a strong alternative to existing SD architectures, combining simplicity, familiarity, and practical efficiency.

2 Related Work

Multi-token prediction training and SD: Initial speculative decoding work (Stern et al. 2018; Sun et al. 2021; Xia, Ge, et al. 2022) concentrated on tasks such as machine translation, where parallel mappings in word space make it easier to realize substantial speed gains. These task-specific advances laid the foundation for more general-purpose SD methods (Cai et al. 2024; Ankner et al. 2024; Gloeckle et al. 2024), which leverage parallel decoding heads for broader LLM applications. A recent study (Lindsey et al. 2025) demonstrates that, in poetic text, hidden states—even from early positions—may already contain information about several upcoming tokens. This insight aligns with empirical findings showing that LLM performance can be enhanced by multi-token prediction (MTP) (DeepSeek-AI et al. 2025) or by adopting more challenging objectives that inject additional contextual signals into model states(Gao et al. 2021a).

While multi-token prediction has shown promise in prior work, state-of-the-art SD methods (L. Zhang et al. 2025; Yuhui Li et al. 2025a) continue to use autoregressive next-token prediction, aligning next-k tokens via step-by-step simulation during training, i.e., Training-Time Test, or TTT (Yuhui Li et al. 2025a). Furthermore, SD adaptation to every new LLM often entails training the SD model from scratch, amplifying the cost of training. Although self-speculative methods (J. Zhang et al. 2024) address the co-training overhead by reusing the target model as the draft, they generally fall short in delivering good speed improvements (Zhong et al. 2024).

Cross-attention heads for SD: While it may seem natural to integrate draft and target model states through cross-attention, only limited prior work (Du et al. 2024; Zimmer et al. 2024; B. Xiao et al. 2024) have investigated this approach in the context of speculative decoding. GLIDE with a CAPE (Du et al. 2024) employs a conventional cross-attention decoder that includes a self-attention sublayer. In contrast, our approach eliminates half of the attention parameters by using a single-layer cross-attention module followed by an MLP. Combined with an effective two-stage training

scheme, this design advances cross-attention-based SD to achieve state-of-the-art speedups on the same training data scale (Yuhui Li et al. 2024), doubling the performance reported in (Du et al. 2024). MoA (Zimmer et al. 2024), which adds self-attention and mean aggregation layers on top of cross-attention to extract keys from the target model's hidden states, further increases the complexity of the classic cross-attention module in the draft model. As a result, MoA's speedup remains limited, and their evaluation of EAGLE-v2 (Yuhui Li et al. 2024) does not fully reflect the state-of-the-art speedup potential achievable at that data scale. Clover-2 (B. Xiao et al. 2024) achieves effective speedups by incorporating cross-attention into one of several auxiliary layers. Notably, its *augment block* is solely used to improve first-token prediction. In this work, we directly compare against Clover-2 and demonstrate that improved training efficiency alone allows us to surpass its speedups without introducing any inference-time overhead.

3 Preliminaries

Let V be a discrete space over all possible tokens in the vocabulary, we model a target LLM of parameter Θ^* by the conditional distribution $p_n(t) = \Pr(t_{n+1} = t \mid t_1, ..., t_n; \Theta^*)$ given context sequence $C = t_1, t_2, ..., t_n$ where $t_i \in V$ and the subscript i denotes token positions. In Transformer-based LLMs, the sampling of the next token $t_{n+1} \sim p_n$ is conditioned on the preceding context C through the use of causal attention masks. By modifying the attention masks, the dependency can be restricted to only a subset of tokens in C, enabling partial conditioning (Beltagy et al. 2020; Child et al. 2019).

In SD, the draft model $q_n(t) = \Pr(\hat{t}_{n+1} = t \mid t_1, \ldots, t_n; \Theta)$ is optimized to approximate the target distribution, with Θ optionally incorporating Θ^* to enhance alignment. During each SD iteration, a sequence of γ draft tokens $\hat{t}_{n+1}, \hat{t}_{n+2}, \ldots, \hat{t}_{n+\gamma}$ is *proposed*, and in lossless SD, only a contiguous prefix can be accepted. In the *verify* step, each proposed token $\hat{t}_{n+i}$ is accepted with probability $\min(1, p_{n+i-1}(\hat{t}_{n+i})/q_{n+i-1}(\hat{t}_{n+i}))$ for $i = 1, \ldots, \gamma$. At the first rejection position j, or when $j = \gamma + 1$ without encountering any rejections, one additional token is sampled from normalized $\max(0, p_{n+j-1} - q_{n+j-1})$. As a result, each SD iteration produces at least one new token, and here we denote the total number of accepted tokens $\tau \geq 1$. The above method ensures that accepted tokens are equivalent to those sampled from the target distribution (Leviathan et al. 2023). In the case of greedy decoding, this strategy effectively matches the top-1 tokens from the target and draft distributions, ensuring that the generated tokens exactly replicate the target model outputs.

The speedup potential comes from the fact that Transformers can verify multiple tokens in parallel in one forward pass with a time cost T_v (often assumed to be constant within a small window). Because the speed of SD-assisted generation is reflected by $\mathbb{E}[\tau]$ divided by the average iteration time cost $T = T_d + T_v$ where T_d is the cost for drafting tokens, the per-iteration speedup, or *improvement factor* (Leviathan et al. 2023), will be $\mathbb{E}[\tau]/(T_d/T_v + 1)$, which is seen as a proxy for the overall speedups. Therefore, a high speedup requires both better acceptance lengths (when draft and target distributions align well) and low draft cost.

To reduce T_d, parallel multi-token SD methods proposes draft tokens in parallel (Cai et al. 2024; Gloeckle et al. 2024; Zhong et al. 2024; Lin et al. 2024; Z. Xiao et al. 2024; Monea et al. 2023). However, the acceptance lengths of these models are generally worse than autoregressive SD methods (Yuhui Li et al. 2025b; Yuhui Li et al. 2024; L. Zhang et al. 2025), although in the latter case T_d is linearly proportional to γ. Because many autoregressive models need only a single-layer draft model (compared to 32 layers when Llama 7B is the target model, essentially $T_d \ll T_v$) to be able to achieve $\mathbb{E}[\tau] > 3$ or even more, the speedups are generally more sensitive to the accuracy of predictions rather than iteration overheads. To this end, most state-of-the-art draft models are autoregressive and are trained with the highest effective precisions (i.e., TF32) (Yuhui Li et al. 2025b; Yuhui Li et al. 2024; L. Zhang et al. 2025). To further maximize alignment between the draft and target models, these systems are also uniformly trained from scratch. Together, these factors underscore the importance of addressing the overall training cost of speculative decoding.

4 Methodology

In this work, we propose an SD method, Budget EAGLE (Beagle), which does more accurate autoregressive predictions at inference time but utilizes multi-token parallel predictions during

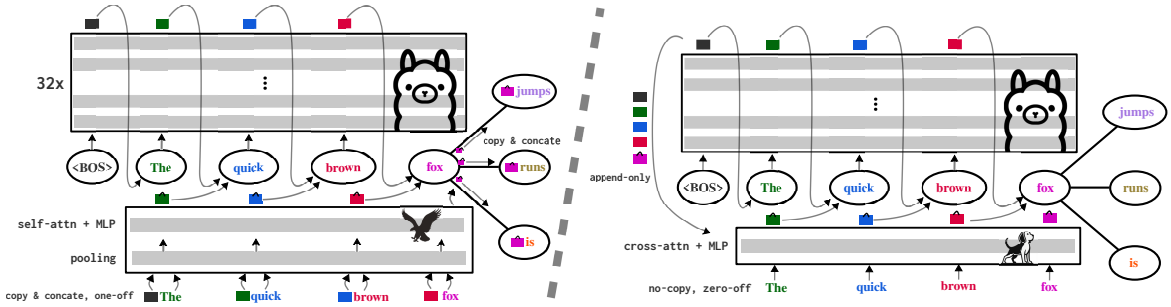

Figure 1: Comparison between EAGLE (Yuhui Li et al. 2025b; Yuhui Li et al. 2024) **(Left)** and our Beagle architecture **(Right)**. Square boxes denote higher-level states; a hat on top indicates states predicted by the draft model. Embedding layers are omitted for clarity, and colored words represent tokens generated at different positions. The right-side trees represent branched prediction via tree attention (Miao et al. 2024; Cai et al. 2024). Using self attention, EAGLE requires auxiliary pooling layers and explicit copying of higher-level states for concatenation. In contrast, Beagle adopts a standard training pipeline without offsets and avoids copying, simplifying draft modeling.

training to improve training efficiency and to condense more future information into draft model states. Figure 1 illustrates our model architecture at a high level and highlights its differences to a representative autoregressive SD method, EAGLE (Yuhui Li et al. 2025b; Yuhui Li et al. 2024).

4.1 Cross-Attention Draft Modeling

Our draft model architecture, more specifically, is composed of a single-layer cross-attention Transformer block that maps lower-level context $t_1, ..., t_n$ to higher-level state $\mathbf{h}_n$ (for clarity, we omit layer details such as normalization, GQA, or positional embeddings):

$$\begin{aligned} \mathbf{h}_n &= \mathrm{MLP}(\mathbf{y}_n) + \mathbf{y}_n \\ \mathbf{y}_n &= \mathrm{CrossAttn}(\mathbf{h}_{1:n-1}, \mathbf{e}(t_n)) + \mathbf{e}(t_n) \end{aligned} \tag{1}$$

where $\mathbf{h}_i$ at any context position $i = 1, ..., n-1$ is expected to allow either target model top hidden states (we name them *true states*) or the autoregressively generated states from the draft model itself. Furthermore, $\mathbf{e} : t \to \mathbb{R}^d$ is the embedding layer, and MLP is a point-wise feed-forward layer.

For the cross-attention sublayer, more specifically, the query, key, and values are processed as follows:

$$\begin{aligned} \mathrm{Q}_i^{(h)}, \mathrm{K}_j^{(h)}, \mathrm{V}_j^{(h)} &= W_{h,Q}^T \mathbf{e}(t_i), W_{h,K}^T \mathbf{h}_j, W_{h,V}^T \mathbf{h}_j \\ s_{i,j}^{(h)} &= \mathrm{Softmax}_j(\langle \mathrm{Q}_i^{(h)}, \mathrm{K}_j^{(h)} \rangle / \sqrt{d_h}) \\ \mathbf{o}_i^{(h)} &= \sum_j \mathrm{Mask}_{i,j}^{(h)} \cdot s_{i,j}^{(h)} \cdot V_j^{(h)} \\ \mathbf{y}_i &= W_O^T [\mathbf{o}_i^{(1)}; \mathbf{o}_i^{(2)}; ...; \mathbf{o}_i^{(H)}]_{d \times 1} \end{aligned} \tag{2}$$

where weights $W_{h,Q}, W_{h,K}, W_{h,V} \in \mathbb{R}^{d \times d_h}$ and $W_O^T \in \mathbb{R}^{d \times d}$ where d and d_h are model and head hidden dimensions, respectively. Unlike causal self attention, the cross-attention mask here has to ensure "diagonal scores" are also masked, i.e., $\mathrm{Mask}_{i,j} \to -\infty$ for $j \geq i$ given query at position i. Using constant-space cross-attention masks, we can allow predicting multiple future tokens during training (see Section 4.2).

Finally, our draft model can be defined by $q_n(t) = \mathrm{Softmax}(\mathbf{z}_n)$ where logits $\mathbf{z}_n = \mathbf{e}^{-1}(\mathbf{h}_n)$ and the language model head is a linear mapping $\mathbf{e}^{-1} : \mathbb{R}^d \to \mathbb{R}^V$.

As seen in Eq. 1, we replace the commonly used self-attention layer with a single cross-attention layer. However, this cross-attention Transformer block is used in a *non-standard* causal fashion to decode draft tokens autoregressively during inference: At query position i, the computation of $s_{i,j}$ and $\mathbf{o}_i$ in Eq. 2 can reuse existing K_j, V_j if they are cached for all $j < i$, and we only need to append

new states K_i, V_i to KV-cache when we query state $\mathbf{h}_{i+1}$. Since we need to reset KV-cache to only contain true states at the end of SD iteration, the above lazy-appending cache can effectively skip and save the last memory operation in every SD iteration.

On the other hand, these changes also eliminate the need to use any auxiliary pooling layers because we can handle low- and high-level states via different queries and key/value space. As a result, it also avoids copying and concatenating high-level states to feed the draft model as next inputs (see Figure 1), leading to greater memory locality. We will show, with effective training, this simplified architecture can perform evenly or better than more complex architectures commonly seen in recently developed SD models (Yuhui Li et al. 2024; B. Xiao et al. 2024).

4.2 Two-Stage Block-Attention Training

Many autoregressive speculative decoding methods are trained to predict only the immediate next token following the training-data token, which is effectively equivalent to training a draft LLM using the Next Token Prediction (NTP) objective, conditioned on the target model's runtime hidden states. However, one-step NTP does not explicitly capture the actual inference dynamics in speculative decoding, particularly when the draft model starts to rely on its own predicted hidden states during autoregressive inference. Within a draft window, prediction errors and accumulated noise can cause the draft model's behavior to diverge from the target distribution used during training, leading to a suboptimal speed.

As a result, many recent speculative training methods have adopted the Training-Time Testing (TTT) scheme to explicitly allow potentially inaccurate predictions and to train on the simulated inference data, effectively "unrolling" for multiple steps during training. However, this is at the expense of much longer training time, which we believe would not be suitable for the entire training cycle.

Instead, during the **early stage**, we propose to predict multiple future tokens $\hat{t}_{n+1}, \hat{t}_{n+2}, ..., \hat{t}_{n+k}$ and feed them to Transformer in parallel. And only in the **late stage**, we apply training-time simulation. Compared to self-attention heads, we will show this leads to reduced training overheads as well.

Denote the model prediction distribution to a i-step ahead future token t_{n+i} as

$$q_n^{(i)}(t) = \Pr(\hat{t}_{n+i} = t \mid t_1, ..., t_n; \Theta). \tag{3}$$

where $q_n^{(1)}(t) = q_n(t)$. We mask out continuous windows of tokens in the attention mask (starting at a random minor offset). Specifically, at a window of size k starting at n, and for a query at position $n + i, i = 1, ..., k$, the corresponding local future keys at $n + j, 1 \leq j \leq k$ are all masked out. This masking results in a block attention matrix as shown in Figure 2 (the left-most attention mask), different from usual block attentions where only local tokens are seen (A. Q. Jiang et al. 2023), we may dub it *inverse* block attentions because local tokens are masked out.

Other than encouraging the draft model to contain representations for multiple future tokens, we also make sure each query state will be utilized to backpropagate losses, thus maximizing sampling efficiency in block attentions. As such, we define our early-stage loss as

$$\mathcal{L}_{early} = -\frac{1}{N} \sum_{n \in w_0} \sum_{j=1}^{k} \mathbb{E}_{t \sim p_{n+j}} [\log q_n^{(j)}(t)] \tag{4}$$

where k is the window size and j represents the query position relative to window start positions $n \in w_0 = \{\epsilon + w \cdot k : w = 0, 1, ..., N\}$ with a random offset $\epsilon \in [0, k)$. Additionally, the maximum number of windows N is selected to cover all training inputs for maximum sample efficiency.

For late-stage training, we start to simulate multiple steps of inference (illustrated in the right two attentions in Figure 2). Specifically, the predicted states at the start of the window ($i = 1$) are saved and expanded to the next step, where the initial attention mask is also extended and allows next queries in the same window to access updated states. And after the second step, the newly predicted states are again saved, but via in-place modifications with a correspondingly modified attention mask. This process is repeated in a single training iteration until a total simulation steps s. Unlike the linear expansions seen in self-attention training (L. Zhang et al. 2025; Yuhui Li et al. 2025a), here our inference simulation consumes constant space and does not need to unroll queries during training.

At simulation step i, denote the j-step ahead future draft distribution as

$$q_n^{(i,j)}(t) = \Pr(\hat{t}_{n+j} = t \mid t_1, ..., t_n, \hat{t}_{n+1}, ..., \hat{t}_{n+i-1}; \Theta) \tag{5}$$

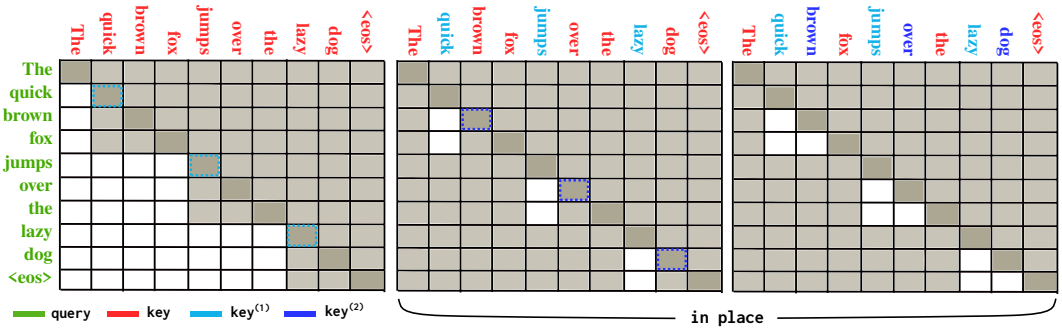

Figure 2: The cross-attention masks used during training for draft model heads. **Left one (early stage block attention):** Query states are derived directly from token embeddings, and keys are from high-level states. An *inverse* block attention starting at a random offset with a fixed window hides local keys from a query, encouraging the model to condense more information on future tokens. **Right two (after simulation step 1 and step 2, late stage):** In the late-stage training, we unroll newly predicted states to accurately simulate inference during training. Unlike Training-Time Test with self attentions, this method requires no new queries to be generated, and only needs one-step attention memory allocation for in-place adding of next-predicted keys.

where we have predicted tokens up to $n+i-1$, and are predicting token at $n+j$ where $j \geq i$. The true context of states from training data is still fixed back at position n. The general form of loss considering simulation is, for $1 \leq s \leq l \leq k$,

$$\mathcal{L}(s, l, \beta) = -\frac{1}{N} \sum_{n \in w_0} \sum_{i=1}^{s} \sum_{j=i}^{l} \beta_{i,j} \cdot \mathbb{E}_{t \sim p_{n+j}} [\log q_n^{(i,j)}(t)]. \tag{6}$$

Here, s denotes the maximum number of simulated steps, and l indicates the maximum lookahead position. The associated weight $\beta_{i,j} > 0$ is used to re-weight loss terms.

We select $s = k$, $l = i$, and $\beta^*_{i,j} = k - i + 1$ to define our late-stage loss as $\mathcal{L}_{\text{late}} = \mathcal{L}(k, i, \beta^*)$. We provide justifications in Appendix A.2 showing that this formulation serves as a surrogate loss that approximates the acceptance length during SD inference, and offers a better approximation than the early-stage loss toward the end of training.

5 Experiments

5.1 Experimental Setups

In this work, we limit our baselines to lossless decoding methods and focus on single-batch greedy decoding. The optimization for throughput and speculative sampling is left for future work.

Baselines: We consider popular target models in this domain: Vicuna (7B), LLaMA-2 (7B), and LLaMA-3 (7B). We use HuggingFace TGI (Text Generation Inference) for baseline SD, paired with JackFram LLaMA-68M and Vicuna-68M as used in other work (Miao et al. 2024; S. Yang et al. 2024). We consider a popular inference-time parallel decoding SD method, Medusa (Cai et al. 2024), with official Vicuna weights. Moreover, a representative zero-memory-overhead self-speculative method (J. Zhang et al. 2024) with official LLaMA-2 weights is also added. Importantly, we include Clover 2 (B. Xiao et al. 2024), which is reportedly the most efficient model based on cross attention. EAGLE v1 and v2 series represent the best open models using the same training data scales. EAGLE v2, which adopts a dynamic draft attention tree, constantly performs better (Yuhui Li et al. 2024), so we always include EAGLE-v2 and adopt the same dynamic tree method. We do not include EAGLE-v3 (which is trained on 8× more data) or much more expensive full-stage TTT training approaches (L. Zhang et al. 2025), as our focus is on exploring efficient training strategies and architectural alternatives under comparable data scales.

Datasets: We limit our training dataset to only ShareGPT (Aeala 2023), which is composed of over 60K conversational dialogues from ChatGPT and its users. This is to align with other baselines (Cai et al. 2024; Yuhui Li et al. 2025b; Yuhui Li et al. 2024; B. Xiao et al. 2024) with the same amount

Table 1: Speed and memory comparisons among different models (A6000 Ada). The left two columns specify Target and Draft models, where V, L2, and L3 represent Vicuna, LLaMA-2, and LLaMA-3, respectively. The metrics Spu, τ, and M represent Speedup, acceptance length, and GPU peak Memory usage.

T	D	MT-Bench				GSM-8K				CNN-Daily			
		Speed	Spu	τ	M↓	Speed	Spu	τ	M↓	Speed	Spu	τ	M↓
V	None	35.2	1.0	1.0	13.7	35.3	1.0	1.0	13.5	34.2	1.0	1.0	14.1
	TGI	56.8	1.6	3.9	13.1	60.1	1.7	2.9	12.9	56.4	1.6	2.9	13.6
	Medusa	71.2	2.0	2.1	15.0	83.5	2.4	2.6	14.9	63.0	1.8	2.0	15.3
	Clover 2	53.3	1.5	**4.1**	16.0	57.2	1.6	4.2	16.0	47.2	1.4	3.6	16.4
	EAGLE v2	103.4	2.9	4.0	14.5	**118.4**	**3.4**	**4.6**	14.3	**87.6**	**2.6**	**3.6**	14.8
	Beagle (Ours)	**104.6**	**3.0**	**4.1**	13.5	108.5	3.1	4.3	13.3	82.0	2.4	3.4	14.0
L2	None	34.9	1.0	1.0	13.8	35.3	1.0	1.0	13.3	34.8	1.0	1.0	14.2
	Self-Spec	34.8	1.0	1.7	13.7	34.1	1.0	1.7	13.1	34.6	1.0	1.9	14.2
	TGI	46.0	1.3	3.3	13.1	47.6	1.3	2.6	12.9	40.8	1.2	2.6	13.6
	EAGLE v2	104.9	**3.0**	4.0	15.5	**111.6**	**3.2**	4.4	15.4	**89.8**	**2.6**	**3.7**	15.9
	Beagle (Ours)	**106.2**	**3.0**	**4.1**	13.5	111.4	**3.2**	**4.5**	13.2	85.4	2.5	3.6	14.0
L3	None	32.3	1.0	1.0	15.6	32.5	1.0	1.0	15.5	33.3	1.0	1.0	15.8
	EAGLE v2	**80.2**	**2.5**	**3.6**	17.8	83.0	**2.6**	3.9	17.7	**69.6**	**2.1**	**3.4**	18.1
	Beagle (Ours)	79.2	**2.5**	3.5	15.7	**83.5**	**2.6**	**4.0**	15.5	67.2	2.0	3.2	16.0

of training data. Our inference datasets cover multi-turn general conversational benchmark MT-Bench (Zheng, Chiang, et al. 2023), reasoning task GSM-8K (Cobbe et al. 2021), and summarization task CNN-Daily (Hermann et al. 2015) with a subset of 1,000 samples following J. Zhang et al. 2024. In total, a full evaluation run for one system covers more than 2,100 inputs (including conversation turns), and our measurement values (other than peak memory) are aggregated averages.

Inference and Training: All implementations are based on HuggingFace Transformers (Wolf et al. 2020) contained in the same Docker environment[1] and are running in PyTorch eager mode with BF16 precisions during inference. Our evaluation framework also makes sure each system is running inference against the same data, but we follow the training prompt formats of each baseline for maximum speed. The inference jobs are run on two grades of servers using single-threaded executions: an NVIDIA A6000 Ada node and an Amazon AWS instance with A10G GPUs.

Similar to trained baseline models, our model is trained with mixed precisions where model weights are TF32 while target model states are preserved in half precisions, which are generated offline to minimize training time and GPU memory usage. If not specified otherwise, we train 20 epochs maximum, where the first 10 epochs we use the early-stage training strategy and the rest use the late-stage training strategy. The detailed training configurations of our different settings can be found in Appendix A.3. As official EAGLE weights are trained from an unknown number of epochs, we replicate EAGLE-v2 with the same number of training epochs to ensure comparable results. Also, we align the EAGLE dynamic attention tree with the same hyperparameters (i.e., depth=5, topk=10, and candidates to be accepted per draft iteration is set to 60).

5.2 Results

Table 1 (and 2 in the Appendix) compares both speeds (in tokens per second) and peak memory usage among systems across two different grades of GPUs. Our system, Beagle, has shown a similar efficiency level to EAGLE v2, where both are trained with the same training data for 20 epochs. However, our memory overhead on top of the target model is minimal, while EAGLE consumes 10% to 15% more GPU memory.

On the other hand, Self-Spec using self-speculative decoding consumes no additional memory and does generate greater-than-one acceptance lengths (thus better than no-SD), but it leads to no speed improvements because its draft model consists of multiple layers of overheads. For the same reason, Clover 2 – with various augment layers which help to achieve much better acceptance rates – obtains only around 1.5x speedups. Due to the lack of co-training and model alignment, baseline SD (TGI)

[1]We use the official `pytorch:2.6.0-cuda11.8-cudnn9-runtime` as our base Docker image.

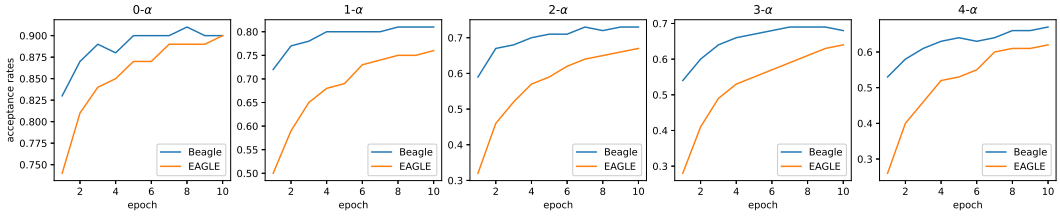

Figure 3: **Early-stage** acceptance rates at different draft steps (step-α) during the first 10 epoch training process (evaluated on MT-Bench). Our model (Beagle) uses the early-stage loss based on multi-token predictions. At this stage, our training efficiency is consistently better than EAGLE (v1/v2) (Yuhui Li et al. 2025b; Yuhui Li et al. 2024)

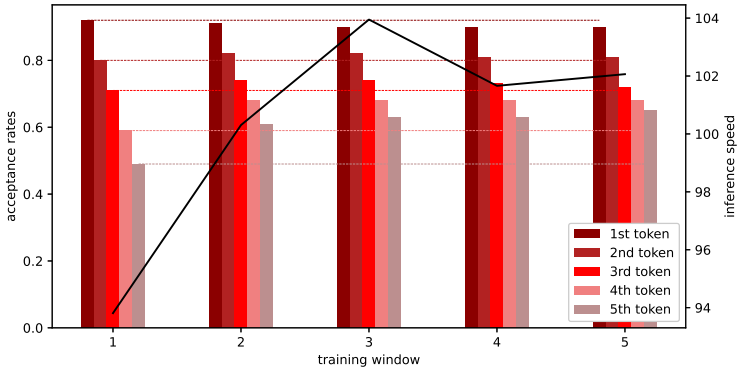

Figure 4: **Early-stage** token acceptance rates at different positions and corresponding inference speeds (evaluated on MT-Bench). We vary the window length from 1 to 5 for five early-stage training settings. Multi-token prediction (using $\mathcal{L}_{early}$) with a proper window width (optimal width achieved at 3) improves further-step token acceptance rates, generally enhancing inference speeds.

adds little speedup as well. Finally, Medusa, using parallel decoding at inference, has suboptimal acceptance lengths compared to autoregressive models such as EAGLE v2 and ours.

Other than time and memory efficiency, our having fewer parameters than EAGLE also enables much greater training efficiency in the early stage. As shown in Figure 3, although we converge to a similar acceptance rate for the first token, our system shows consistently better performance during the early stage of training. Additionally, according to Figure 4, the multi-token prediction loss (Eq. 4) used in the early stage leads to better training results and improved future token predictions. Moreover, it consumes no more data than EAGLE, utilizing the Transformer's parallel forwards advantage.

5.3 Justifications for Two-Stage Training

We conduct experiments to verify our interpretations for two-stage losses in Appendix A.2, i.e., (1) the early-stage loss is a worse surrogate (but trains more efficiently); (2) and the late-stage loss corresponds to inference efficiency more precisely (although spending more compute on each training iteration).

As shown in Figure 4, agnostic to training window sizes, the future token acceptance rates constantly show further degradations over distances. But unlike the strict assumption we have in Appendix A.2, the multi-token training loss $\mathcal{L}_{early}$ can actually "bend" the decline curves to form a slower slop as window size enlarges, which does not necessarily improve the 1st token acceptance rates but instead enhancing the overall acceptance among all future tokens – leading to a better speed than single-token prediction training baseline (window=1) and also justifying our early-stage training loss.

However, there is an optimal window size (at 3) which can lead to the best end speed number. This is likely a trade-off between focusing on early tokens or on late tokens – although a large window helps preserve degradation on future token predictions, it does hurt the 1st token acceptance rates beyond a

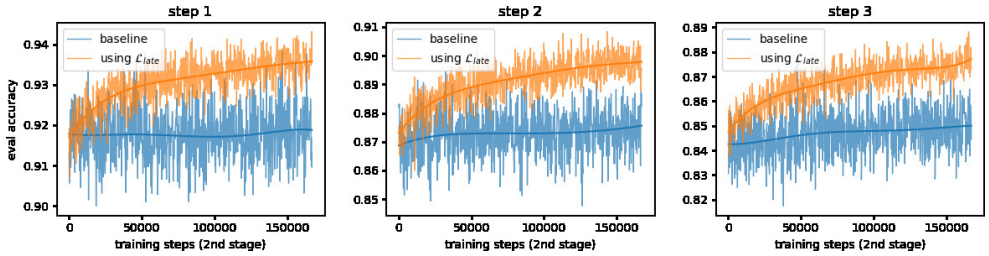

Figure 5: The **late-stage** (10th- to 20th-epoch) draft model prediction accuracy changes using different training losses (the validation set during training is a partial MT-Bench data). The orange lines correspond to the model trained with our proposed late-stage loss $\mathcal{L}_{late}$, and the blue baselines are when the model continues to be trained with early-stage loss $\mathcal{L}_{early}$. Due to the high variance of accuracy changes during late-stage training, we also highlight the interpolated smooth curves.

window size of 2, at the meantime, the 1st token acceptance rate is crucial (as it is weighted the most in the expected acceptance length as shown in Appendix A.2).

Figure 5 justifies the necessity of our late-stage loss. Although our late-stage training loss requires simulating each query token for multiple steps to train on policy (thus obviously adding linear time overheads with respect to steps), it is necessary to obtain better prediction capabilities after training. Towards the end of the training, the first-stage loss offers minimal accuracy improvements over time (blue lines in Figure 5), while the late-stage training loss can keep advancing prediction accuracies notably for all shown unrolled steps. As a result, we consider our late-stage training to be a complementary and necessary addition to the early-stage training.

Finally, at the end of training, our late-stage surrogate loss $\mathcal{L}_{late}$ is shown (in Appendix A.2) to have an almost constant bound w.r.t. the approximated acceptance length. In contrast, many existing SD training approaches apply uniform weighting to predicted tokens across different steps.

6 Conclusion

In this work, we present a novel cross-attention-based speculative decoding (SD) modeling along with an effective, well-grounded two-stage training scheme built on block attention mechanisms. Our method employs a simpler and less tailored architecture without auxiliary layers, having an effectively improved early-stage training efficiency and constant GPU memory usage during simulated inference. With improved training strategies, we demonstrate – for the first time – that cross-attention models can match the performance of state-of-the-art EAGLE-v2 self-attention architecture on the same training data. We believe this work opens new research directions for exploring more diverse architectures and applications in SD, e.g., optimizing vision-language models (VLMs) for vision tasks.

7 Limitations

Our hypothesis that the i-th ahead token follows a geometric degradation in accuracies *if* these tokens are predicted in parallel (Eq. 9) may not strictly reflect real observations. However, this does not undermine our major conclusions and the necessity of our proposed two-stage training because we have provided sufficient arguments and empirical results in Section 5.3. Secondly, our work is limited to exploring efficient training strategies and architectural alternatives under comparable data scales. As a result, systems achieving greater speedups by training with a different scale of data (e.g., EAGLE-v3 using 8x more data) or with more expenses are not compared in this work. Finally, we have conducted training and experiments only on smaller-scale models because we are limited by resources to train larger LLMs with permissive hardware, and the selection of target models is also largely restricted by commonly used model checkpoints shared among our evaluated baselines. We believe scaling our effectiveness to different model sizes is an orthogonal topic and can be left to future work.

References

Hermann, Karl Moritz et al. (2015). "Teaching machines to read and comprehend". In: *NIPS*.

Stern, Mitchell, Noam Shazeer, and Jakob Uszkoreit (2018). *Blockwise Parallel Decoding for Deep Autoregressive Models*. arXiv: 1811.03115 [cs.LG]. URL: https://arxiv.org/abs/1811.03115.

Child, Rewon et al. (2019). *Generating Long Sequences with Sparse Transformers*. arXiv: 1904.10509 [cs.LG]. URL: https://arxiv.org/abs/1904.10509.

Loshchilov, Ilya and Frank Hutter (2019). "Decoupled Weight Decay Regularization". In: *ICLR*. URL: https://openreview.net/forum?id=Bkg6RiCqY7.

Beltagy, Iz, Matthew E. Peters, and Arman Cohan (2020). *Longformer: The Long-Document Transformer*. arXiv: 2004.05150 [cs.CL]. URL: https://arxiv.org/abs/2004.05150.

Kasai, Jungo et al. (2020). "Deep encoder, shallow decoder: Reevaluating non-autoregressive machine translation". In: *arXiv preprint arXiv:2006.10369*.

Wolf, Thomas et al. (2020). *HuggingFace's Transformers: State-of-the-art Natural Language Processing*. arXiv: 1910.03771 [cs.CL].

Cobbe, Karl et al. (2021). *Training Verifiers to Solve Math Word Problems*. arXiv: 2110.14168 [cs.LG]. URL: https://arxiv.org/abs/2110.14168.

Gao, Luyu and Jamie Callan (2021a). *Condenser: a Pre-training Architecture for Dense Retrieval*. arXiv: 2104.08253 [cs.CL]. URL: https://arxiv.org/abs/2104.08253.

– (2021b). *Unsupervised Corpus Aware Language Model Pre-training for Dense Passage Retrieval*. arXiv: 2108.05540 [cs.IR]. URL: https://arxiv.org/abs/2108.05540.

Sun, Xin et al. (2021). "Instantaneous grammatical error correction with shallow aggressive decoding". In: *arXiv preprint arXiv:2106.04970*.

Xia, Heming, Tao Ge, et al. (2022). "Speculative decoding: Exploiting speculative execution for accelerating seq2seq generation". In: *arXiv preprint arXiv:2203.16487*.

Aeala (2023). *ShareGPT*. https://huggingface.co/datasets/Aeala/ShareGPT_Vicuna_unfiltered.

Chen, Charlie et al. (2023). "Accelerating large language model decoding with speculative sampling". In: *arXiv preprint arXiv:2302.01318*.

Jiang, Albert Q. et al. (2023). *Mistral 7B*. arXiv: 2310.06825 [cs.CL]. URL: https://arxiv.org/abs/2310.06825.

Joao Gante (2023). *Assisted Generation: a new direction toward low-latency text generation*. DOI: 10.57967/hf/0638. URL: https://huggingface.co/blog/assisted-generation.

Leviathan, Yaniv, Matan Kalman, and Yossi Matias (2023). *Fast Inference from Transformers via Speculative Decoding*. arXiv: 2211.17192 [cs.LG]. URL: https://arxiv.org/abs/2211.17192.

Monea, Giovanni, Armand Joulin, and Edouard Grave (2023). *PaSS: Parallel Speculative Sampling*. arXiv: 2311.13581 [cs.CL]. URL: https://arxiv.org/abs/2311.13581.

Vaswani, Ashish et al. (2023). *Attention Is All You Need*. arXiv: 1706.03762 [cs.CL]. URL: https://arxiv.org/abs/1706.03762.

Zheng, Lianmin, Wei-Lin Chiang, et al. (2023). "Judging LLM-as-a-Judge with MT-Bench and Chatbot Arena". In: *NeurIPS*. URL: https://openreview.net/forum?id=uccHPGDlao.

Ankner, Zachary et al. (2024). *Hydra: Sequentially-Dependent Draft Heads for Medusa Decoding*. arXiv: 2402.05109 [cs.LG].

Cai, Tianle et al. (2024). "Medusa: Simple llm inference acceleration framework with multiple decoding heads". In: *arXiv preprint arXiv:2401.10774*.

Cheng, Yunfei et al. (2024). *Recurrent Drafter for Fast Speculative Decoding in Large Language Models*. arXiv: 2403.09919 [cs.CL]. URL: https://arxiv.org/abs/2403.09919.

Dai, Damai et al. (2024). *DeepSeekMoE: Towards Ultimate Expert Specialization in Mixture-of-Experts Language Models*. arXiv: 2401.06066 [cs.CL]. URL: https://arxiv.org/abs/2401.06066.

Du, Cunxiao et al. (2024). "Glide with a cape: A low-hassle method to accelerate speculative decoding". In: *arXiv preprint arXiv:2402.02082*.

Gloeckle, Fabian et al. (2024). *Better & Faster Large Language Models via Multi-token Prediction*. arXiv: 2404.19737 [cs.CL]. URL: https://arxiv.org/abs/2404.19737.

Li, Yuhui et al. (2024). *EAGLE-2: Faster Inference of Language Models with Dynamic Draft Trees*. arXiv: 2406.16858 [cs.CL]. URL: https://arxiv.org/abs/2406.16858.

Lin, Feng et al. (2024). *BiTA: Bi-Directional Tuning for Lossless Acceleration in Large Language Models*. arXiv: 2401.12522 [cs.CL]. URL: https://arxiv.org/abs/2401.12522.

Miao, Xupeng et al. (Apr. 2024). "SpecInfer: Accelerating Large Language Model Serving with Tree-based Speculative Inference and Verification". In: *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 3*. ASPLOS '24. ACM, pp. 932–949. DOI: 10.1145/3620666.3651335. URL: http://dx.doi.org/10.1145/3620666.3651335.

Xia, Heming, Zhe Yang, et al. (2024). *Unlocking Efficiency in Large Language Model Inference: A Comprehensive Survey of Speculative Decoding*. arXiv: 2401.07851 [cs.CL]. URL: https://arxiv.org/abs/2401.07851.

Xiao, Bin et al. (2024). *Clover-2: Accurate Inference for Regressive Lightweight Speculative Decoding*. arXiv: 2408.00264 [cs.CL]. URL: https://arxiv.org/abs/2408.00264.

Xiao, Zilin et al. (2024). *ParallelSpec: Parallel Drafter for Efficient Speculative Decoding*. arXiv: 2410.05589 [cs.CL]. URL: https://arxiv.org/abs/2410.05589.

Yang, Sen et al. (2024). *Multi-Candidate Speculative Decoding*. arXiv: 2401.06706 [cs.CL].

Zhang, Jun et al. (2024). "Draft & Verify: Lossless Large Language Model Acceleration via Self-Speculative Decoding". In: *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Association for Computational Linguistics, pp. 11263–11282. DOI: 10.18653/v1/2024.acl-long.607. URL: http://dx.doi.org/10.18653/v1/2024.acl-long.607.

Zheng, Lianmin, Liangsheng Yin, et al. (2024). *SGLang: Efficient Execution of Structured Language Model Programs*. arXiv: 2312.07104 [cs.AI]. URL: https://arxiv.org/abs/2312.07104.

Zhong, Wei and Manasa Bharadwaj (2024). *S3D: A Simple and Cost-Effective Self-Speculative Decoding Scheme for Low-Memory GPUs*. arXiv: 2405.20314 [cs.CL]. URL: https://arxiv.org/abs/2405.20314.

Zimmer, Matthieu et al. (2024). "Mixture of Attentions For Speculative Decoding". In: *arXiv preprint arXiv:2410.03804*.

DeepSeek-AI et al. (2025). *DeepSeek-V3 Technical Report*. arXiv: 2412.19437 [cs.CL]. URL: https://arxiv.org/abs/2412.19437.

Li, Yuhui et al. (2025a). *EAGLE-3: Scaling up Inference Acceleration of Large Language Models via Training-Time Test*. arXiv: 2503.01840 [cs.CL]. URL: https://arxiv.org/abs/2503.01840.

– (2025b). *EAGLE: Speculative Sampling Requires Rethinking Feature Uncertainty*. arXiv: 2401.15077 [cs.LG]. URL: https://arxiv.org/abs/2401.15077.

Lindsey, Jack et al. (2025). "On the Biology of a Large Language Model". In: *Transformer Circuits Thread*. URL: https://transformer-circuits.pub/2025/attribution-graphs/biology.html.

LM Studio Team (2025). *LM Studio 0.3.10: Speculative Decoding*. URL: https://lmstudio.ai/blog/lmstudio-v0.3.10.

Zhang, Lefan et al. (2025). *Learning Harmonized Representations for Speculative Sampling*. arXiv: 2408.15766 [cs.LG]. URL: https://arxiv.org/abs/2408.15766.

A Appendix

A.1 Supplementary Evaluation Table (A10G)

Table 2: Speed and memory comparisons among different models (A10G). The left two columns specify Target and Draft models, where V, L2, and L3 represent Vicuna, LLaMA-2, and LLaMA-3, respectively. The metrics Spu, τ, and M represent Speedup, acceptance length, and GPU peak Memory usage.

T	D	MT-Bench				GSM-8K				CNN-Daily			
		Speed	Spu	τ	M↓	Speed	Spu	τ	M↓	Speed	Spu	τ	M↓
V	None	13.3	1.0	1.0	13.7	13.1	1.0	1.0	13.5	13.0	1.0	1.0	14.1
	TGI	23.6	1.8	4.0	13.1	23.8	1.8	2.9	12.9	21.7	1.7	2.7	13.6
	Medusa	27.2	2.0	2.1	15.0	32.3	2.5	2.6	14.9	24.6	1.9	2.0	15.3
	Clover 2	21.6	1.6	4.0	16.0	21.8	1.7	4.2	16.0	16.8	1.3	3.5	16.4
	EAGLE v2	**40.0**	**3.0**	3.9	14.5	**46.4**	**3.5**	**4.7**	14.3	30.7	**2.4**	**3.6**	14.8
	Beagle (ours)	39.8	**3.0**	**4.1**	13.5	41.9	3.2	4.3	13.3	**31.6**	**2.4**	3.4	14.0
L2	None	13.2	1.0	1.0	13.8	13.3	1.0	1.0	13.3	13.0	1.0	1.0	14.2
	Self-Spec	13.8	1.0	1.7	13.7	13.2	1.0	1.7	13.1	13.7	1.0	1.9	14.2
	TGI	17.9	1.4	3.2	13.1	19.0	1.4	2.6	12.9	15.8	1.2	2.8	13.6
	EAGLE v2	41.7	**3.2**	**4.1**	15.5	42.8	3.2	**4.4**	15.4	**34.7**	**2.7**	**3.7**	15.9
	Beagle (ours)	**41.9**	**3.2**	**4.1**	13.5	**43.5**	**3.3**	**4.4**	13.2	32.7	2.5	3.6	14.0
L3	None	12.4	1.0	1.0	15.6	12.7	1.0	1.0	15.5	12.5	1.0	1.0	15.8
	EAGLE v2	**33.7**	**2.7**	**3.6**	17.8	**34.9**	**2.7**	3.9	17.7	**28.9**	**2.3**	**3.4**	18.1
	Beagle (ours)	33.2	**2.7**	**3.6**	15.7	**34.9**	**2.7**	**4.0**	15.5	27.9	2.2	3.2	16.0

A.2 Interpretations of Two Stage Losses

For better SD inference performance, we are essentially optimizing the expected acceptance length L within a window k, that is,

$$\begin{aligned}\mathbb{E}[L] &= \sum_{\ell=1}^{k} \Pr(L \geq \ell) \qquad \text{(tail-sum formula)} \\ &= \sum_{i=1}^{k} \exp(\sum_{j=1}^{i} \log \alpha^{(j)})\end{aligned} \tag{7}$$

where $\alpha^{(i)}$ denotes the expected acceptance rate at position i. Although this objective can be directly modeled as a loss function using negative $\log \mathbb{E}[L]$ and calculating $\mathrm{logsumexp}$ of accumulated $\log \alpha^{(i)}$ values for each simulated step, the numerical issue arise due to the large differences of magnitudes of values in different steps. However, as long as the training is effective (it is a reasonable assumption because cross-entropy loss terms are also maximizing the data log likelihoods), we may assume $\log \alpha^{(i)} \to 0$ towards the end of training. Then, by first-order Taylor expansion,

$$\begin{aligned}\mathbb{E}[L] \approx J &= \sum_{i=1}^{k} (1 + \sum_{j=1}^{i} \log \alpha^{(j)}) \\ &= \sum_{i=1}^{k} (k - i + 1) \log \alpha^{(i)} + k.\end{aligned} \tag{8}$$

This time, the RHS objective J in Eq. 8 is a more numerically stable objective.

On the other hand, we hypothesize that the i-th ahead token follows a geometric degradation in accuracies *if* these tokens are predicted in parallel:[2]

$$\alpha^{(i)} = r^{\,i-1} \alpha^{(1)} \tag{9}$$

[2]We do not hypothesize a degradation in autoregressive predictions because Yuhui Li et al. 2025a show that the acceptance rates can be maintained very effectively given enough training data.

where the degradation rate $r = r(n, \Theta) < 1$ is a variable depending on the draft model of parameters Θ given a context prior to position n.

With the draft distribution notation $q_n^{(i,j)}(t)$ in Eq. 5, our general loss function in Eq. 6 can be rewritten as (for one window with non-branching prediction)

$$\begin{aligned}
\mathcal{L}(s, l, \beta) &= -\sum_{i=1}^{s}\sum_{j=i}^{l} \beta_{i,j} \cdot \mathbb{E}_{t\sim p_{n+j}}[\log q_n^{(i,j)}(t)] \\
&\geq -\sum_{i=1}^{s}\sum_{j=i}^{l} \beta_{i,j} \cdot \log \mathbb{E}_{t\sim p_{n+j}}[q_n^{(i,j)}(t)] \qquad \text{(Jensen's inequality)} \\
&= -\sum_{i=1}^{s}\sum_{j=i}^{l} \beta_{i,j} \cdot \log(r^{(j-i)}\alpha^{(i)}) \qquad (10) \\
&= -\sum_{i=1}^{s}\sum_{j=i}^{l} \beta_{i,j} \cdot [(j-i)\log r + \log \alpha^{(i)}] \\
&\geq -\sum_{i=1}^{s}\sum_{j=i}^{l} \beta_{i,j} \log \alpha^{(i)} \qquad (s, l \text{ are hyperparameters})
\end{aligned}$$

where equality holds when $i = j$.

We may denote $q_n^{(j)}(t) = q_n^{(1,j)}(t)$ according to Eq. 3 and 5, the early loss is then

$$\mathcal{L}_{early} = \mathcal{L}(1, k, \mathbf{1}) \geq -k \log \alpha^{(i)} = -J + \sum_{i=2}^{k}(k - i + 1)\log \alpha^{(i)} + k \geq -J \qquad (11)$$

which can be seen as a surrogate to the objective in Eq. 8.

Alternatively, when we simulate the exact autoregressive decoding (predicting the immediate next token in many steps) by assigning $s = k, l = i$ and $\beta^*_{i,j} = k - i + 1$, it becomes our proposed late-stage loss $\mathcal{L}_{late}$ and it is also a surrogate to the objective because

$$\mathcal{L}_{late} = \mathcal{L}(k, i, \beta^*) \geq -\sum_{i=1}^{k}(k - i + 1)\log \alpha^{(i)} = -J + k. \qquad (12)$$

Compared to Eq. 12 where there is an almost constant gap (up to a *Jensen's Gap* due to Eqs. 10), the surrogate gap in Eq. 11 depends on future tokens ($\alpha^{(i)}, i \geq 2$), and is expected to have a higher variance and thus is a worse surrogate loss. Intuitively, $\mathcal{L}_{late}$ is a better choice for late-stage training because it is simulating the exact SD autoregressive inference, where we essentially avoid parallel predictions and only predict the immediate next token.

However, $\mathcal{L}_{late}$ is *not a good choice* for early-stage training because it requires "unrolling" the data multiple times during training, notably adding training time with almost linear increments. In contrast, setting single-step $s = 1$ and assigning maximum parallel predictions $l = k$ as in our early-stage loss will substantially utilize the Transformer architecture by forwarding multiple tokens and improving sample efficiencies.

A.3 Training Configurations and Observations

We use a context length of 2048 and a training precision of TF32 in different training settings. We have considered using batch sizes of 4, 8, 16, and 32. Our pivot training in Figure 6 shows the model can converge to a similar level, less sensitive to batch size. Nevertheless, we choose to use a batch size of 16 for the majority of experiments. Our optimization uses the PyTorch fused version of AdamW (Loshchilov et al. 2019) kernel with betas (0.9, 0.95), and we use a constant learning rate of 3e-5 with a warm-up of 2000 steps. In addition, we adopt a maximum gradient norm of 0.5.

To align our training when replicating EAGLE (Yuhui Li et al. 2025b), we enforce the same training settings in EAGLE in addition to aforementioned settings. This also includes adding a Gaussian

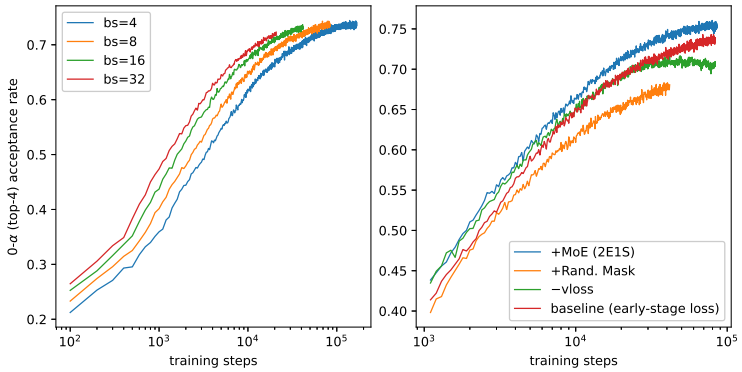

Figure 6: The averaged first-token top-4 acceptance rates using MT-Bench as evaluation set during 10-epoch trainings for the Llama2 7B target model. **Left:** Training with different batch sizes (bs); **Right:** pilot training experiments using different modeling methods with a fixed bs=8 **(Right)**.

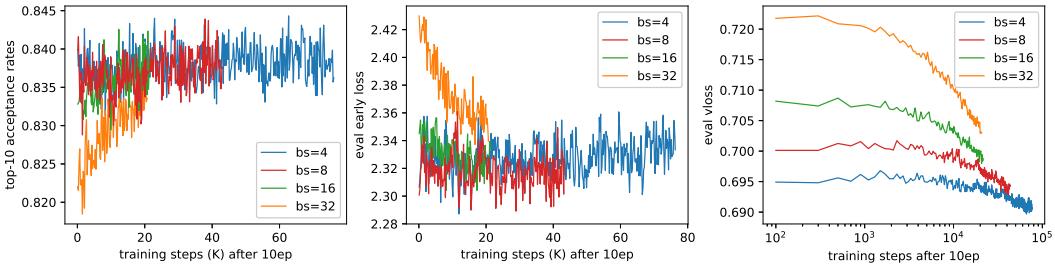

Figure 7: The convergence of training (top-10 averaged acceptance rates, $\mathcal{L}_{early}$, and *vloss* or the regression loss (Yuhui Li et al. 2025b) after 10 epochs for the Llama2 7B target model. For different batch sizes, the acceptance rates can converge to a similar level.

noise $N(0, 0.2)$ to target model states, and importantly, adding a hidden state distillation loss (i.e., *vloss* or regression loss) with a coefficient of 10 in both stages to regularize the training. In our pilot training experiments (Figure 6, right), without adding this distillation loss will cause the model to even degrade at the end of the training.

Specific to our modeling, we set the early-stage window $k = 5$, and due to time and expense budgets, we limit the simulation steps $s = 4$ unless described otherwise. For $s = 3$, a single GPU with 24GiB memory is sufficient to train a 7B model using an unit batch size, although we choose to use the A6000 Ada to train our most competitive models with $s = 4$ and a larger batch size of 16 in most of experiments (according to Figure 10 and our pivot trainings in Figure 7, the differences of the final model should be minimal). After 10 epochs, we find that our early-stage loss starts to converge (Figure 7), although the incorporated EAGLE vloss can still improve.

Our final model training processes for both stages can be found in Figure 8 and 9.

A.4 Ablation Study

We have done additional studies to break down the improvements in both effectiveness and efficiency.

For effectiveness, we study the impact of simulated steps in late-stage training. As shown in Figure 10, both token-wise acceptance rates and inference speeds can be improved consistently by running more simulated steps. The further a token is in the prediction window, the more potential it has to benefit from a larger number of simulated steps. And methodology-wise, the two proposed training stages greatly improve the acceptance length almost linearly, i.e., multi-token prediction has around 0.3 average acceptance length improvements over single-step NTP, and training-time exact simulation further adds a similar improvement after the late-stage training.

Moreover, our effectiveness improvements in the late stage, compared to EAGLE-v2, mainly comes from mid-range tokens (2nd to 3rd tokens) where we maintain notably higher numbers in terms of

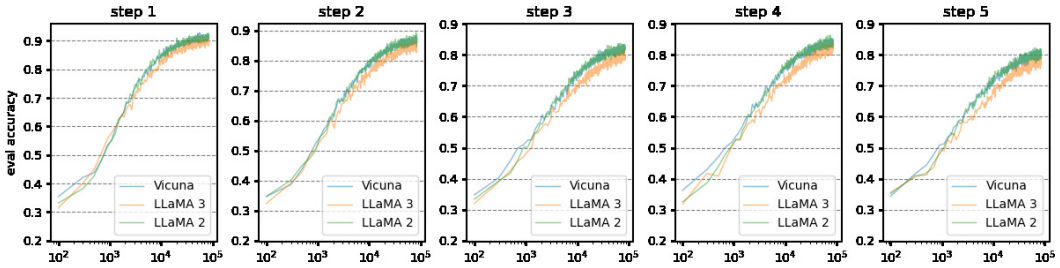

Figure 8: Early-stage (1 – 10 epochs) training of target models using a window size of 5.

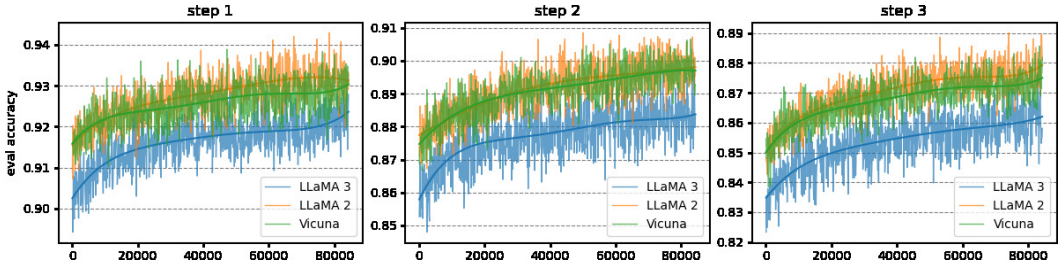

Figure 9: Late-stage training (10 – 20 epochs) of target models using 3 simulation steps.

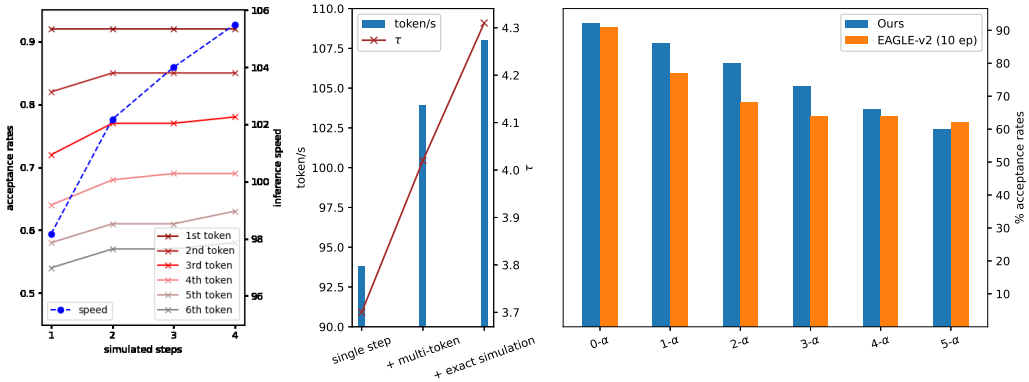

Figure 10: Ablations of our methods evaluated on MT-Bench. **Left:** Late-stage training effectiveness at different token positions using different training settings; **Middle:** The end speed (in token/s) and acceptance length ablations for the major methods proposed in this work; **Right:** Our model after two-stage training compared to EAGLE-v2 trained for only the early stage.

per-token acceptance rates (Figure 10, right). This may be attributed to our cross-attention architecture and the condensing of future token information in the early stage of the training.

In terms of efficiency, we provide a decomposition of the time costs for the overall generation process, projected over the first 256 tokens, as well as the per-iteration cost of proposing draft tokens within a single SD iteration (Figure 11). Our different configurations may contribute to runtime costs differently. Particularly, we denote our method *dynamic* if we apply a dynamic drafting attention tree similar to EAGLE, and we denote our method *concate* when we do concatenation with newly generated hidden stages to be used for generating the next token.

As shown in Figure 11, although our optimal configuration is "dynamic+concate", we find that concatenation can be disabled with only up to 3% loss in speed. This option is potentially valuable for deployment on devices with high memory movement penalties, where the efficiency gains from avoiding the copying and concatenation of dynamically generated states may outweigh the accuracy

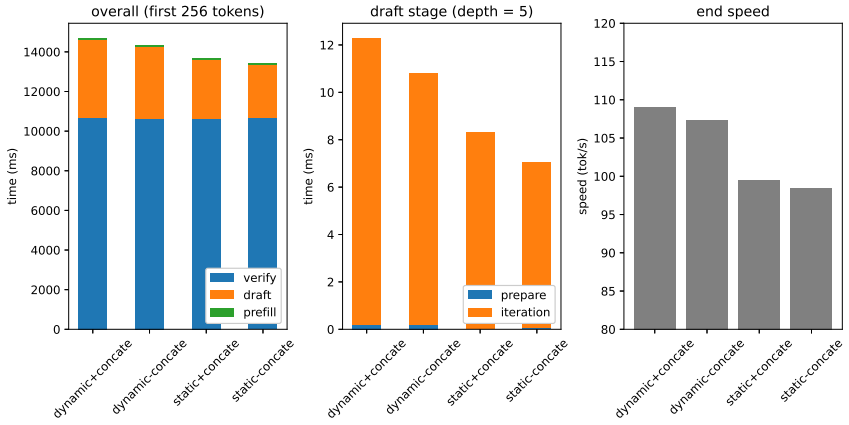

Figure 11: The decoding cost effectiveness analysis on partial MT-Bench (LLaMA 7B). We compare different settings including whether using dynamic tree attention or concatenating key/value states during the drafting stage in cross-attention heads. **Left:** overall time cost for first 256 tokens assuming constant compositional costs; **Middle:** the detailed draft costs; **Right:** and corresponding final inference speeds.

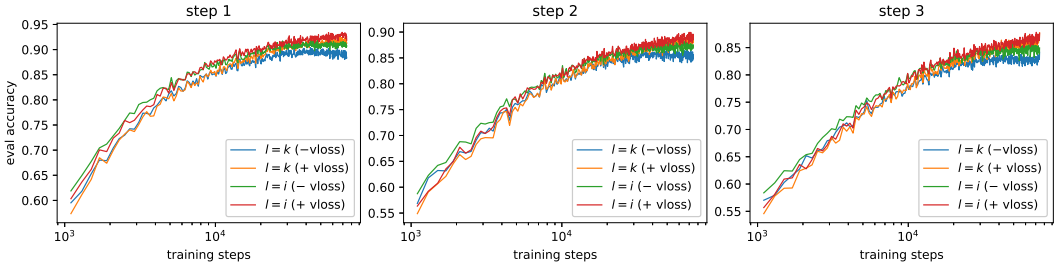

Figure 12: Various loss alternatives (trained from scratch with a window of 3 and simulated for all 3 steps at each training iteration).

degradation. However, achieving such flexibility is more challenging in self-attention heads where autoregressively generated states must be appended to predict subsequent tokens.

However, we observe a substantial speed loss when drafting tokens using static trees compared to dynamic trees (Figure 11, right). This further underscores the importance of prediction accuracy in the speculative decoding trade-off, as static trees incur greater losses in accuracy than the gains they offer in iteration efficiency.

A.5 Supplementary Explorations

During the pilot training, we have also explored adding random masks and replacing the MLP layer with MoE (Dai et al. 2024)[3] as shown in Figure 7. However, random masks lead to underperforming our proposed early-stage training, and although MoE improves acceptance rates, its added overheads make the resulting draft model less efficient. Nevertheless, we believe a specialized MoE kernel that optimizes inference speed may greatly reduce these overheads, but we leave this to future work.

In addition, we have tried various general loss forms in Eq. 6, but trained for the initial 10 epochs. As shown in Figure 12, we can verify that the proposed exact simulation (when $l = i$) achieves better evaluation accuracies compared to alternative combinations. The $l = k$ combines both multi-token prediction with multi-step simulations, but causes suboptimal accuracies for all 3 steps. Lastly, we find that the regularization loss (vloss) is crucial to the converged performance in both cases.

[3]Due to GPU memory constraints, we have reduced the default routed experts to the minimal: only 2 routed experts and 1 shared experts.

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: We have accurately described our work contributions and scope in a clear layout.

   Guidelines:

   - The answer NA means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A No or NA answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: See our Section 7.

   Guidelines:

   - The answer NA means that the paper has no limitation while the answer No means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

   Answer: [Yes]

Justification: We have provided some theoretical insights with assumptions and derivations in Appendix A.2. These are meant to provide insights and justifications for our proposed two-stage losses.

Guidelines:

- The answer NA means that the paper does not include theoretical results.
- All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
- All assumptions should be clearly stated or referenced in the statement of any theorems.
- The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
- Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
- Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

Answer: [Yes]

Justification: we have provided main experimental setups in Section 5, along with supplemental and detailed training setups and study in Appendix A.3. We align and specify the system environment for inference in the same (Docker) container to ensure reproducibility.

Guidelines:

- The answer NA means that the paper does not include experiments.
- If the paper includes experiments, a No answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
- If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
- Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
- While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
  (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
  (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
  (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
  (d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

   Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

   Answer: [No]

   Justification: Per our company policy and commonly followed procedures, unfortunately, we default to not disclosing the code. However, we may respond to model checkpoint requests for any research requests.

   Guidelines:

   - The answer NA means that paper does not include experiments requiring code.
   - Please see the NeurIPS code and data submission guidelines (`https://nips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - While we encourage the release of code and data, we understand that this might not be possible, so "No" is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
   - The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://nips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
   - The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
   - At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
   - Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer, etc.) necessary to understand the results?

   Answer: [Yes]

   Justification: We provide extensive dataset details in both Section 5.1 and Appendix A.3.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
   - The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [NA]

   Justification: Our claims in this paper do not include the word *significant* or *significantly*, nor do they rely on statistical significance. Our contributions are primarily supported by theoretical justifications and consistently improved training efficiency in early-stage training. Matching EAGLE-v2 performance, in itself, does not necessitate any claims of significance. Most importantly, our evaluated datasets contain more than 2,100 model inference examples, and the reported metrics are aggregated to reflect and approximate mean values.

   Guidelines:

   - The answer NA means that the paper does not include experiments.

- The authors should answer "Yes" if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
- The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
- The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
- The assumptions made should be given (e.g., Normally distributed errors).
- It should be clear whether the error bar is the standard deviation or the standard error of the mean.
- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g. negative error rates).
- If error bars are reported in tables or plots, The authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: We have provided sufficient information on computer resources in both Section 5.1 and Appendix A.3.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes]

   Justification: We are using less-capable and licensed LLMs for experiments, and we do not see any potential harmful consequences incurred from our research.

   Guidelines:

   - The answer NA means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer No, they should explain the special circumstances that require a deviation from the Code of Ethics.
   - The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [NA]

Justification: We are using less-capable and licensed LLMs for experiments, and we do not see any negative societal impact of this work being performed.

Guidelines:

- The answer NA means that there is no societal impact of the work performed.
- If the authors answer NA or No, they should explain why their work has no societal impact or why the paper does not address societal impact.
- Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
- The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
- The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
- If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pretrained language models, image generators, or scraped datasets)?

Answer: [NA]

Justification: To our knowledge, this paper does not pose such risks.

Guidelines:

- The answer NA means that the paper poses no such risks.
- Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
- Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
- We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

Answer: [Yes]

Justification: All used existing model checkpoints in this work are accessed from HuggingFace with permissive licenses. All frameworks and platforms (including HuggingFace) are properly credited and cited.

Guidelines:

- The answer NA means that the paper does not use existing assets.
- The authors should cite the original paper that produced the code package or dataset.

- The authors should state which version of the asset is used and, if possible, include a URL.
- The name of the license (e.g., CC-BY 4.0) should be included for each asset.
- For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
- If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
- For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

    Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

    Answer: [NA]

    Justification: This paper will not release new assets publicly.

    Guidelines:

    - The answer NA means that the paper does not release new assets.
    - Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
    - The paper should discuss whether and how consent was obtained from people whose asset is used.
    - At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

    Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

    Answer: [NA]

    Justification: this work does not involve crowdsourcing nor research with human subjects.

    Guidelines:

    - The answer NA means that the paper does not involve crowdsourcing nor research with human subjects.
    - Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
    - According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

    Answer: [NA]

    Justification: This paper will not involve crowdsourcing nor research with human subjects.

    - The answer NA means that the paper does not involve crowdsourcing nor research with human subjects.

- Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
- We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
- For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

    Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does not impact the core methodology, scientific rigorousness, or originality of the research, declaration is not required.

    Answer: [NA]

    Justification: Our usage of LLMs does not impact the core methodology and thus does not require declarations.

    Guidelines:

    - The answer NA means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
    - Please refer to our LLM policy (`https://neurips.cc/Conferences/2025/LLM`) for what should or should not be described.