EAGLE Series: From Feature Prediction to Training-Time Test

Evolution of Training Mechanisms in Speculative Sampling

Contents

1 Introduction: Why Focus on Training Details?

The core of Speculative Sampling is the draft model — its quality directly determines the acceleration effect. The three generations of EAGLE essentially answer one question:

Core Question: How to train a draft model that is both lightweight and accurate?

EAGLE-1 proposed feature-level autoregression, EAGLE-2 optimized the draft tree structure, and EAGLE-3 completely redesigned the training paradigm.

2 EAGLE-1: Feature-Level Autoregression

2.1 Core Idea

EAGLE-1's starting point: **predicting features is easier than predicting tokens**.

Tokens are discrete and one-hot, while features are continuous and semantically rich. The second-to-top layer feature of the Target LLM (hidden state before the LM Head) already encodes rich contextual information. Predicting the next feature based on it is simpler than predicting tokens from scratch.

2.2 Draft Model Architecture

EAGLE-1's draft model is very lightweight:

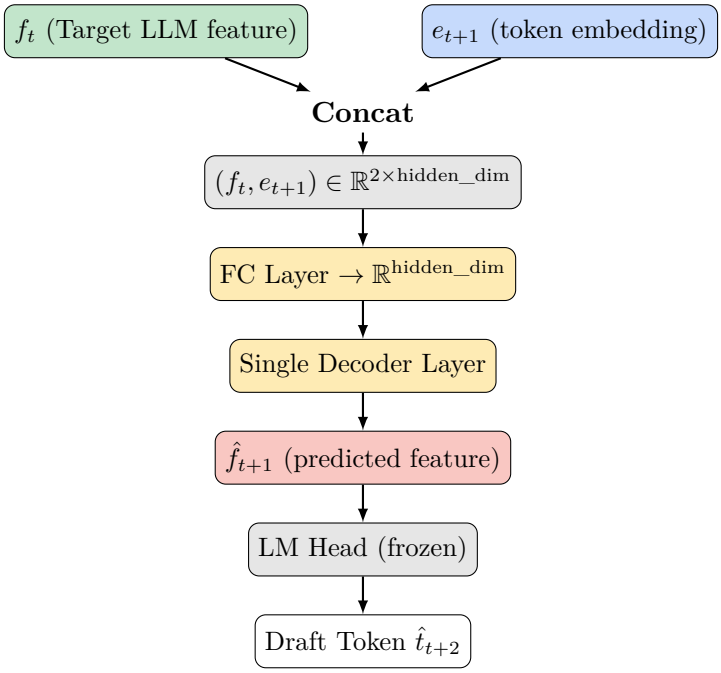

Figure 1: EAGLE-1 Draft Model Architecture. Note: The Embedding layer and LM Head directly reuse Target LLM parameters; only the FC and Decoder Layer need training.

Key Design: The input uses **concatenation** rather than summation. f_t and e_{t+1} are concatenated into a $2 \times$ hidden_dim vector, then reduced via FC layer.

Why is e_{t+1} needed? This is EAGLE-1's solution to "sampling uncertainty" (contrast with Medusa). The feature f_t may correspond to multiple possible next tokens (e.g., "am" or "always"), each leading to different subsequent features. By inputting the embedding of the already-sampled token t_{t+1}, the draft model knows which path to predict along.

2.3 Training: Dual Loss Design

EAGLE-1 uses two loss functions:

$$\mathcal{L} = \underbrace{\text{SmoothL1}(\hat{f}_{t+1}, f_{t+1})}_{\mathcal{L}_{\text{reg}}:\text{Feature Prediction}} + 0.1 \times \underbrace{\text{CrossEntropy}(\hat{p}_{t+2}, p_{t+2})}_{\mathcal{L}_{\text{cls}}:\text{Token Prediction}} \tag{1}$$

Role of $\mathcal{L}_{\text{reg}}$: Forces the draft model output $\hat{f}$ to approximate the Target LLM's true feature f.

Role of $\mathcal{L}_{\text{cls}}$: Directly optimizes the final objective — token prediction accuracy.

Reason for 0.1 weight: The classification loss is numerically an order of magnitude larger than the regression loss.

2.4 Train-Test Gap at Inference

There is a critical issue here. Let's examine the input differences between training and inference:

Phase	Step 1 Input	Step 2 Input
Training	f_t (from Target LLM)	f_{t+1} (from Target LLM)
Inference	f_t (from Target LLM)	$\hat{f}_{t+1}$ (from Draft Model!)

Table 1: EAGLE-1's Train-Test Gap

The problem: **At inference, Step 2's input $\hat{f}_{t+1}$ is predicted by the draft model itself**, while training uses the exact f_{t+1}. If $\hat{f}_{t+1}$ and f_{t+1} differ significantly, Step 2's predictions will be severely inaccurate.

EAGLE-1's Solution: Use $\mathcal{L}_{\text{reg}}$ to force $\hat{f} \approx f$, so even when using predicted values at inference, the distribution won't deviate too much.

Why This Approach Has Limitations

$\mathcal{L}_{\text{reg}}$ is essentially an **additional constraint**: requiring a small model (single Decoder layer) to fit the large model's high-dimensional feature space (4096 dimensions).
This regression task is inherently difficult. It limits the model's expressive capacity and is the core bottleneck that EAGLE-3 aims to break through.

2.5 Data Augmentation: Noise Injection

To make the model more robust in handling imprecise inputs, EAGLE-1 adds noise to features during training:

$$f_{\text{noisy}} = f + \epsilon, \quad \epsilon \sim U(-0.1, 0.1) \tag{2}$$

This enables the model to handle erroneous feature inputs.

3 EAGLE-2: Dynamic Draft Tree (Training-Independent)

EAGLE-2's improvements are unrelated to training; it focuses on **draft tree construction strategy at inference**.

3.1 Core Discovery

EAGLE-2 discovered that the draft model's confidence score is highly correlated with actual acceptance rate:

Confidence Score	Actual Acceptance Rate
< 0.05	≈ 0.04
$0.45 - 0.55$	≈ 0.50
> 0.95	≈ 0.98

This means: **Without calling the Target LLM, we can estimate which draft tokens are more likely to be accepted based solely on the draft model's confidence**.

3.2 Dynamic Tree Construction

Based on this discovery, EAGLE-2 introduces the concept of Value:

$$V_i = \prod_{t_j \in \text{Path}(\text{root}, t_i)} c_j \tag{3}$$

i.e., a node's Value equals the product of all confidences along the path from root to that node.

Expansion Phase: Select top-k nodes with highest Value at each layer for further expansion.

Reranking Phase: Sort all nodes by Value and select top-m as the final draft.

Key Advantage: EAGLE-2 **requires no additional training**; it directly reuses EAGLE-1's weights while achieving 20%-40% acceleration improvement.

4 EAGLE-3: Breaking Free from Feature Prediction

4.1 Problem Diagnosis: Why Can't EAGLE Scale?

EAGLE-3's starting point is an experimental observation: **increasing training data provides very limited improvement for EAGLE-1/2**.

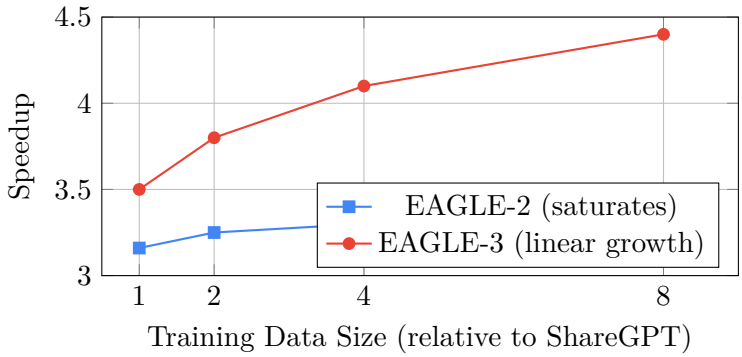

Figure 2: Scaling Law Comparison. EAGLE-2's performance quickly saturates, while EAGLE-3 continues to benefit from more data.

Why is this? EAGLE-3 team's analysis: $\mathcal{L}_{\text{reg}}$ **is the bottleneck**.

Feature prediction essentially asks a small model to "imitate" the large model's internal representations. This constraint is too strong — the small model has limited capacity and cannot precisely fit the large model's 4096-dimensional feature space regardless of data amount.

4.2 What Happens If We Simply Remove $\mathcal{L}_{\text{reg}}$?

A natural idea: since $\mathcal{L}_{\text{reg}}$ is the bottleneck, just remove it and keep only $\mathcal{L}_{\text{cls}}$.

Experimental Results:

- 0-α (first draft token acceptance rate): significantly improved (0.78 $\rightarrow$ 0.80)
- 1-α (second draft token acceptance rate): **dramatically dropped** (0.68 $\rightarrow$ 0.35)

Why? Without $\mathcal{L}_{\text{reg}}$'s constraint, the draft model's output $\hat{a}_{t+1}$ differs greatly from the true f_{t+1}. Training Step 2 sees f_{t+1}, but inference sees $\hat{a}_{t+1}$ — severe distribution mismatch.

4.3 Solution: Training-Time Test (TTT)

EAGLE-3's core innovation: **simulate the multi-step inference process during training**.

Core Idea

Since inference Step 2 uses its own output $\hat{a}_{t+1}$ as input, do the same during training! Let the model directly learn to handle its own (possibly inaccurate) hidden states.

4.3.1 TTT Training Process

Using the sequence "How can I" as an example:

Step 1 (Standard Training):

- Input: g_{can} (from Target LLM) + e_{I}
- Output: hidden state a_{I}, predicted token distribution
- Compute $\mathcal{L}_{\text{cls}}^{(1)}$

Step 2 (Simulated Inference):

- Input: a_{I} (from Step 1's output!) + e_{do}
- Output: hidden state a_{do}, predicted token distribution
- Compute $\mathcal{L}_{\text{cls}}^{(2)}$
- **Key**: gradients backpropagate to Step 1

Step 3 and beyond: Continue using own outputs as inputs, simulating deeper inference.

Total Loss: $\mathcal{L} = \mathcal{L}_{\text{cls}}^{(1)} + \mathcal{L}_{\text{cls}}^{(2)} + \mathcal{L}_{\text{cls}}^{(3)} + \ldots$

4.3.2 TTT Attention Mask

Since Step 2+ inputs form a tree structure (each position may have multiple candidates), the attention mask requires special handling:

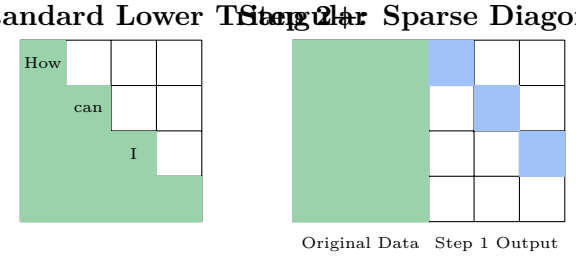

Figure 3: TTT Attention Mask. In Step 2+, each query only attends to original data and its corresponding position (ancestor path).

Implementation Optimization: Step 2+'s mask is diagonal, allowing vector dot products instead of matrix multiplication for attention score computation.

4.3.3 Deeper Implications of TTT

Through TTT, gradients backpropagate from Step 2, Step 3 all the way to Step 1. This forces the model to learn:

> *Compress the key information needed for subsequent token prediction into the hidden state a for passing forward.*

a no longer needs to "look like" the Target LLM's feature; it only needs to serve as the draft model's own "notes," recording information useful for subsequent predictions.

4.4 Multi-Layer Feature Fusion

EAGLE-1 only uses the Target LLM's top-layer feature (hidden state before LM Head). But top-layer features mainly encode "next token" information; predicting "next-next token" may require information from middle/lower layers.

EAGLE-3 fuses multi-layer features:

$$g = \mathrm{FC}\left([l; m; h]\right) \in \mathbb{R}^k \tag{4}$$

where l, m, h are hidden states from low, middle, and high layers respectively.

Why couldn't EAGLE-1 do this?

Because EAGLE-1 has $\mathcal{L}_{\text{reg}}$, requiring outputs to approximate a **specific layer**'s (top layer) feature. After multi-layer fusion, outputs cannot correspond to any single layer, making the loss incomputable.

After EAGLE-3 removes $\mathcal{L}_{\text{reg}}$, input design becomes completely free.

Significance of Multi-Layer Fusion:

Although only Step 1 can use g (Step 2+ uses a):

1. Step 1 is the root of the entire draft tree; its quality affects all branches
2. Step 1's KV cache is reused by subsequent steps; information in g propagates through attention
3. Experiments show (EAGLE-3 Table 2) multi-layer fusion brings an additional 0.58x speedup improvement

4.5 EAGLE-3 Complete Inference Pipeline

Using Prompt "How can" with Target LLM just generating "I" as an example:

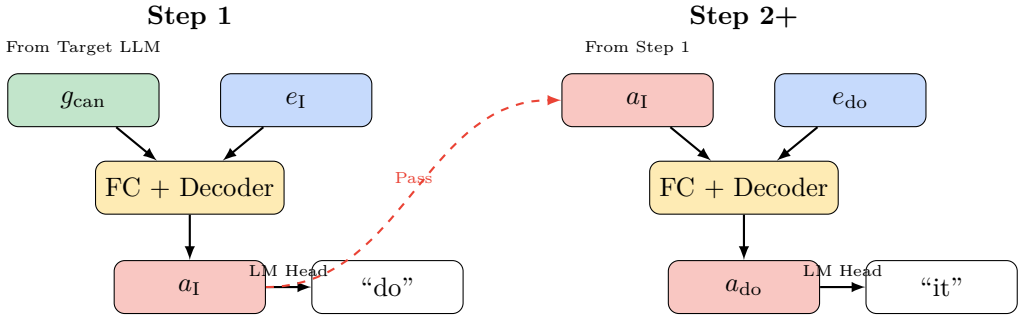

Figure 4: EAGLE-3 Inference Pipeline. Step 1 uses Target LLM's fused feature g; Step 2+ uses its own output a.

Key Distinction:

- Step 1: Input comes from Target LLM's multi-layer fused feature g
- Step 2+: Input comes from draft model's own hidden state a

This is the inherent constraint of speculative sampling: the draft phase cannot call the Target LLM (otherwise there's no acceleration), so subsequent steps can only use their own outputs.

5 Training Configuration Comparison

Table 2: EAGLE-1 vs EAGLE-3 Training Comparison

Configuration	EAGLE-1	EAGLE-3
Loss Function	$\mathcal{L}_{\text{reg}} + 0.1 \times \mathcal{L}_{\text{cls}}$	Only $\mathcal{L}_{\text{cls}}$
Training Input	All from Target LLM	Step 1 from Target, Step 2+ from self
Input Feature	Top-layer feature f	Multi-layer fused feature g
Output Constraint	Must approximate Target's f	Unconstrained, free vector a
Dataset	ShareGPT (68K)	ShareGPT + UltraChat-200K (≈8x)
Data Generation	Fixed dataset	Call Target Model to generate
Learning Rate	3e-5	5e-5
Optimizer	AdamW	AdamW
(β_1, β_2)	(0.9, 0.95)	(0.9, 0.95)
Gradient Clipping	0.5	0.5
Scaling Law	No (saturates)	Yes (linear growth)

6 Acceptance Rate Analysis

The n-α metric measures: acceptance rate when the input contains n hidden states from the draft model itself.

Table 3: Acceptance Rate Comparison (MT-bench, LLaMA-Instruct 3.1 8B)

Method	0-α	1-α	2-α	3-α	4-α	5-α
EAGLE	0.78	0.68	0.64	0.62	0.60	0.58
EAGLE-3	**0.80**	**0.75**	**0.73**	**0.72**	**0.71**	**0.70**

Key Observations:

- EAGLE: Acceptance rate **drops rapidly** as n increases (0.78 to 0.58)
- EAGLE-3: Acceptance rate **remains nearly constant** (0.80 to 0.70)

This demonstrates that TTT effectively resolves the train-test gap: the model learns how to handle its own outputs.

7 Summary: Evolution of Design Philosophy

Table 4: EAGLE Series Technical Evolution

Feature	EAGLE-1	EAGLE-2	EAGLE-3
Core Innovation	Feature Autoregression	Dynamic Draft Tree	Training-Time Test
Feature Prediction Loss	✓	✓	Removed
Input Feature	Top-layer	Top-layer	Multi-layer Fusion
Use Own Output in Training	×	×	✓
Requires Additional Training	-	No	Yes
Scaling Law	×	×	✓
Speedup (Vicuna 13B)	3.07x	4.26x	**5.58x**

Core Insight

EAGLE-1: Make draft model "imitate" Target LLM's feature space → Limited by regression task difficulty
EAGLE-3: Let draft model "freely express," only caring about final token prediction → Unleashes model potential, unlocks Scaling Law
From "an imitator that must mimic the large model's features" to "a successor that only borrows from the large model initially, then develops independent optimized thinking."

8 Appendix: Draft Tree Overhead Analysis

8.1 The Cost of Token Embedding

The introduction of e_{t+1} (token embedding) brings computational overhead. Since each branch has a different token embedding, they **cannot share computation**.

Table 5: Draft Tree Node Count (assuming top_k=3)

Depth	Nodes at This Layer	Cumulative Nodes
1	1	1
2	3	4
3	9	13
4	27	40
5	81	121
6	243	364

Fully expanded, this would yield 364 nodes — clearly too many.

8.2 EAGLE's Solution: Pruning

EAGLE-1: Uses static pruning with predefined tree structure, limiting total nodes to around 40-60.

EAGLE-2: Uses dynamic pruning:

- Expansion phase: Only expand top-k (e.g., 10) nodes with highest Value per layer
- Reranking phase: Keep only top-m (e.g., 48-60) nodes total

Table 6: EAGLE-2 Hyperparameter Configuration

Target LLM Size	Total Nodes (m)	Tree Depth	Expansion Top-k
7B / 8B	60	6	10
13B	50	6	10
70B	48	6	10

8.3 Comparison with Medusa

Table 7: EAGLE vs Medusa: Draft Overhead Trade-off

Feature	Medusa	EAGLE
Draft Method	Parallel independent heads	Autoregressive + token embedding
Forward Passes for Draft	1	depth times
Solves Sampling Uncertainty	×	✓
Acceptance Rate	∼0.6	∼0.8

Trade-off: EAGLE exchanges more draft overhead for higher acceptance rate. Experiments show this trade-off is worthwhile — EAGLE's overall speedup still outperforms Medusa.

This also explains why EAGLE's draft model must be very lightweight (single Decoder layer) — it needs multiple forward passes, each processing multiple branches.