Adaptive Drafter Head Pruning for Speculative Decoding

Anonymous ACL submission

Abstract

Speculative decoding accelerates Large Language Model (LLM) inference by using a lightweight drafter to generate candidate tokens. While reusing the target model's language modeling (LM) head is common, it has become a significant computational bottleneck, particularly with the trend towards larger vocabularies. Existing methods to prune the drafter's LM head offer a path to efficiency but are often static and coarse-grained, failing to adapt to the dynamic nature of the decoding process and the varying utility of different draft tokens. In this paper, we introduce **YOYO**, a dynamic and fine-grained framework for LM head pruning grounded in two core intuitions: (1) *You only prune when it is confident*, and (2) *You only prune more when the payoff is small*. YOYO implements these principles through a confidence-aware mechanism that determines whether to prune based on the certainty of the pruning router itself, and a depth-aware strategy that modulates pruning intensity based on a token's position in the speculative draft tree. This design allows YOYO to dynamically adapt its pruning decisions, allocating more computational budget to high-value, high-certainty predictions while aggressively saving costs on less critical ones. By reframing LM head pruning from a fixed design choice into a context-sensitive process, YOYO achieves a superior trade-off between efficiency and quality. Experimental results on MT-Bench, HumanEval, and GSM8K demonstrate that YOYO outperforms state-of-the-art frameworks, including EAGLE2 and HASS, delivering higher speedups without sacrificing generation quality.

1 Introduction

Large Language Models (LLMs) (Yang et al., 2024a; Dubey et al., 2024; Achiam et al., 2023; Anthropic, 2024; Team et al., 2024) have demonstrated powerful abilities, but their immense scale makes autoregressive generation prohibitively slow and costly. Speculative decoding (Leviathan et al., 2023; Xia et al., 2023) offers a lossless solution by using a small, fast draft model to generate candidate tokens, which are then verified in a single parallel pass by the large target model. This amortizes the high cost of the target model's forward pass, significantly boosting throughput. State-of-the-art methods like EAGLE (Li et al., 2024b,a) utilize a streamlined drafter, often comprising a single Transformer layer and an LM head reused from the target model. However, this design faces a critical bottleneck: as LLMs adopt increasingly larger vocabularies[1], the massive computational cost of the LM head's projection and softmax operations transforms the supposedly lightweight drafting stage into a heavy and slow process, undermining the potential acceleration gains (Zhao et al., 2025; Timor et al., 2025).

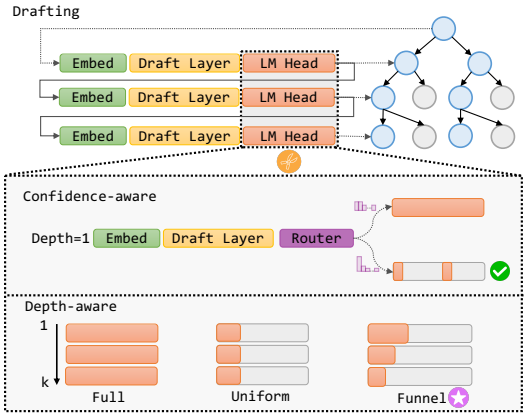


Figure 1: An illustration of our proposed adaptive pruning strategies for the drafter's LM head, using EAGLE2 as an example. YOYO dynamically adjusts pruning based on two principles: confidence-awareness (whether to prune) and depth-awareness (how much to prune).

An effective approach to mitigate this bottleneck

[1]For example, LLM's context lengths have increased significantly: Llama 2 (32K) → Llama 3 (128K), and Qwen2 (152K).

is to prune the drafter's LM head. However, this introduces a critical trade-off: while pruning reduces the drafter's computational cost, it can also impair its prediction quality, leading to lower acceptance rates and diminishing the overall speedup. Naive pruning strategies, whether static (Zhao et al., 2025) or adaptive (Weng et al., 2025), often apply a uniform, one-size-fits-all approach that fails to effectively navigate this trade-off.

Our work tackles this challenge by dissecting the pruning decision into two fundamental questions: *When* should we prune, and *how* should we prune? Our investigation yields a principled framework for making these decisions dynamically.

- **When to Prune?** We first address *when* pruning should be applied. Our key observation is that the confidence of the pruning router[2] is a strong proxy for the risk of performance degradation. This insight leads to our first core principle: *You only prune when it is confident*. We implement this through a **confidence-gated mechanism**, which dynamically activates or bypasses the pruner based on this confidence signal, thereby avoiding risky, low-confidence pruning decisions.

- **How to Prune?** Next, we tackle *how* to prune when the decision to prune has been made. We question the wisdom of uniform pruning intensity and observe that the potential value of a drafted token varies significantly with its depth in the generation tree. This informs our second principle: *You only prune more when the payoff is small*. To realize this, we design a **funnel-shaped pruning schedule** that allocates a larger computational budget to valuable shallow tokens and progressively saves more on less critical, deeper ones.

As shown in Figure 1, the YOYO framework operates in two stages at each step of the draft generation. First, a **confidence-gated mechanism** acts as a gatekeeper, deciding *whether* to prune at all based on the router's predictive certainty. If the confidence is low, pruning is bypassed to avoid risk. If the confidence is high, the gate opens, and the second stage is activated: a **funnel-shaped, depth-aware schedule**. This schedule dictates *how much* to prune, applying lenient pruning at shallow, high-value positions and progressively more aggressive pruning at deeper, lower-yield levels. By strategically answering "***when***" and "***how***" to prune, YOYO offers a more adaptive and principled paradigm for navigating the core speed-quality trade-off. YOYO is designed to enhance existing frameworks and can be seamlessly integrated. The extensive experiments show that when applied to frameworks like EAGLE2 and HASS, YOYO consistently outperforms fixed pruning strategies, unlocking superior acceleration without compromising generation quality.

[2]The router's confidence refers to its own output probability for selecting specific vocabulary sub-blocks, not the drafter's confidence in predicting the next token.

2 Preliminaries

We provide the necessary background in this section.

Speculative Decoding Speculative Decoding (SD) accelerates LLM inference by using a small draft model (M_q) to generate a sequence of candidate tokens, which are then verified in parallel by the large target model (M_p) (Leviathan et al., 2023; Chen et al., 2023; Wang et al., 2024). This process guarantees a final output distribution identical to that of the target model, ensuring **lossless** acceleration. The efficiency of SD hinges on a fast yet accurate drafter. Recent state-of-the-art frameworks have converged on a highly streamlined drafter design. For instance, EAGLE (Li et al., 2024a) employs a drafter with just a single Transformer layer. Crucially, its LM head is not trained from scratch but is directly reused from the target model M_p, with its weights frozen. HASS (Zhang et al., 2024a) builds upon this architecture by refining the training strategy to better align the drafter with the target model during inference. A common thread in these influential methods is the adoption of a minimal drafter architecture that reuses the target's full LM head, a design choice central to our investigation.

Dynamic Tree Decoding In speculative decoding with a tree-structured draft, M_q first infers k steps to generate a draft tree T of depth k and then M_p verifies the draft tree (Yang et al., 2024b; Cai et al., 2024). However, the inefficiency of verifying large, static draft trees—which lack adaptability and incur high attention overhead—has led to the adoption of dynamic tree decoding (Wang et al., 2025; Zhang et al., 2024b). This dominant paradigm prunes the draft tree to a more compact

and effective structure before verification. Notably, the simple and low-overhead confidence-score pruning from EAGLE2 (Li et al., 2024a) has become a foundational technique, inspiring numerous subsequent works (Zhang et al., 2024a; Gui et al., 2024; Hu et al., 2025; Weng et al., 2025).

LM Head Pruning for Drafter As established, representative drafters M_q, such as those in the EAGLE series (Li et al., 2024b,a), reuse the original LM head from the target model M_p. While the target model performs the expensive full-vocabulary projection only once per verification step, the drafter must do so for every token it generates. This makes the LM head the primary computational bottleneck for the otherwise lightweight drafter, undermining low-latency drafting. LM head pruning addresses this by restricting the drafter's predictions to a smaller, dynamically chosen candidate vocabulary $\mathcal{V}_c \subset \mathcal{V}$, where $\mathcal{V}$ is the full vocabulary. This is achieved by operating on a sliced weight matrix. Let the full LM head weight matrix be $\mathbf{W}_p \in \mathbb{R}^{|\mathcal{V}| \times d}$. Pruning involves selecting a subset of rows corresponding to tokens in $\mathcal{V}_c$ to form a reduced matrix $\mathbf{W}'_p \in \mathbb{R}^{|\mathcal{V}_c| \times d}$. This reduces the complexity of both the weight projection and the subsequent softmax calculation.

To formalize the extent of pruning, we define the **retention ratio**, α, as the fraction of the vocabulary that is kept:

$$\alpha = \frac{|\mathcal{V}_c|}{|\mathcal{V}|} \tag{1}$$

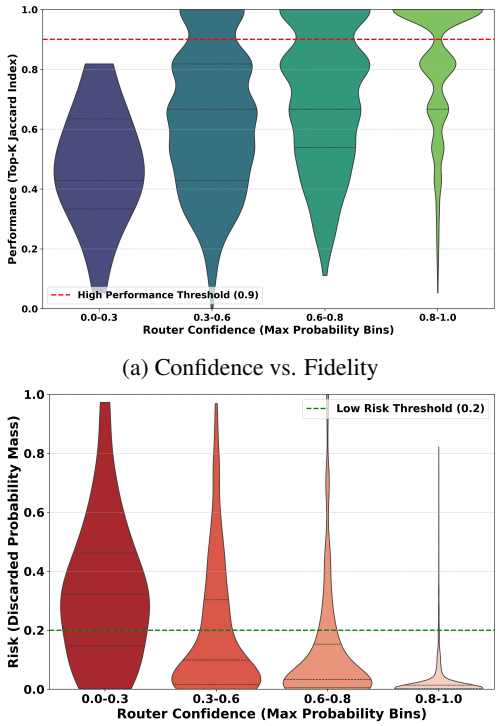
where $0 < \alpha \leq 1$. A smaller α signifies more aggressive pruning and greater potential for computational savings, but also a higher risk of excluding correct tokens. The method for selecting an effective $\mathcal{V}_c$ (and thus determining α) varies across approaches, from static frequency-based methods (Zhao et al., 2025) to adaptive, router-based selection (Weng et al., 2025). Our work focuses on dynamically controlling this allocation α based on context, rather than using a fixed value.

3 Observations

In this section, we investigate the applicability and nuances of current drafter head pruning methods when applied to dynamic tree decoding.

3.1 O1: When to Prune? A Confidence-Awareness Perspective

We investigate the relationship between a pruning router's confidence in selecting vocabulary sub-blocks and the resulting quality of the pruned vocabulary. Our analysis reveals that this confidence is a powerful meta-signal for guiding the pruning process itself.



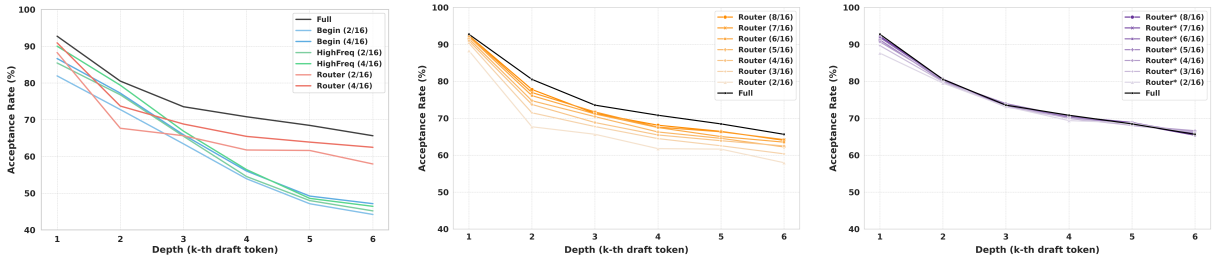
(a) Confidence vs. Fidelity

(b) Confidence vs. Risk

Figure 2: Analysis of the pruning router's confidence on the Llama3-8B-Instruct model with an EAGLE2 drafter on MT-Bench. The x-axis represents the router's confidence (max probability of its prediction for vocabulary sub-blocks). (a) **Fidelity**: The y-axis shows the Top-K Jaccard Index between the pruned and full vocabulary. High router confidence strongly correlates with high fidelity (Jaccard Index near 1.0). (b) **Risk**: The y-axis shows the discarded probability mass. High router confidence correlates with low risk (negligible discarded mass).

Confidence Predicts Pruning Fidelity. We first measure the **Top-K Jaccard Index** to quantify the similarity between the top-k tokens from the full LM head versus the router-pruned head. As shown in Figure 2a, there is a strong positive correlation: high router confidence (e.g., max probability > 0.8) leads to a high Jaccard Index, indicating that pruning preserves the most likely tokens. Conversely, low confidence results in a sharp drop in fidelity.

Confidence Signals Distributional Risk. We then measure the **discarded probability mass**—the total probability of all pruned-away tokens—to quantify the risk of each pruning action.



(a) Static vs. Adaptive Pruning (b) Root-Only Adaptive Pruning (c) Per-Depth Adaptive Pruning

Figure 3: Acceptance rate analysis of different drafter head pruning strategies on MT-Bench, using Llama3-8B-Instruct with an EAGLE2 drafter. All strategies use fixed pruning ratios. (a) Comparison of root-level pruning strategies across draft tree depths. **Begin**: retains initial vocabulary blocks. **HighFreq**: keeps high-frequency blocks based on corpus statistics. **Router**: uses a trained module to select blocks once at the root. All three strategies apply the pruning decision made at depth=1 to all subsequent depths. (b) Performance of the root-only **Router** strategy with varying retention sizes (number of blocks kept). (c) Performance of the **Router*** variant, which applies the routing mechanism at *every* tree depth, also with varying retention sizes.

Figure 2b reveals a stark negative correlation: high router confidence corresponds to a negligible discarded mass, while low confidence leads to high-risk decisions where a significant portion of the probability distribution is discarded.

These findings together indicate that the router's confidence is not just an internal metric but a reliable predictor of the consequence of its own action. It effectively tells us whether a pruning decision is likely to be "safe" or "risky". This leads to our first key insight.

Insight 1

The pruning router's predictive confidence strongly correlates with both the fidelity and risk of the pruning action.

3.2 O2: How to Prune? A Depth-Awareness Perspective

We now shift our focus to *how* pruning should be applied within the context of dynamic tree decoding. Specifically, we analyze how the acceptance rate, a direct measure of speculative decoding efficiency, is affected by different pruning strategies across the depth of the draft tree.

Static vs. Adaptive Pruning. We first compare static, offline pruning methods against a context-aware **Router**. As shown in Figure 3(a), static strategies like **Begin** (retaining initial vocabulary blocks) and **HighFreq** (keeping frequent tokens) suffer a drastic drop in acceptance rate as depth increases. In contrast, the adaptive **Router** maintains a significantly higher rate, confirming that context-awareness is crucial for effective tree-based drafting.

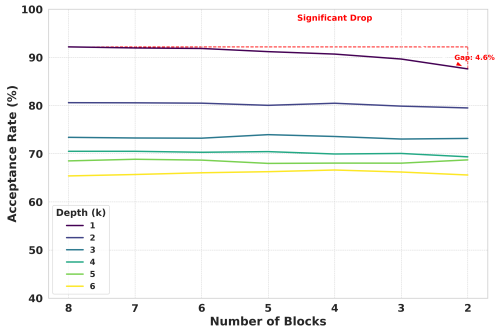
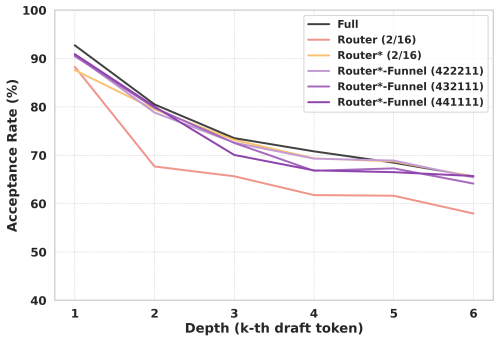


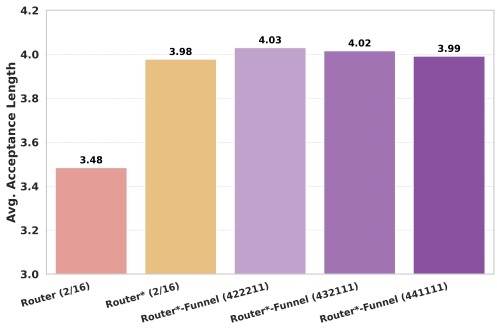
Figure 4: The impact of aggressive, uniform pruning on the acceptance rate of the first drafted token (depth=1). This plot, using the **Router*** strategy, shows that as the retention size decreases (pruning becomes more aggressive), the acceptance rate of the most critical first token drops significantly.

Uniform vs. Depth-Specific Pruning. Building on the superiority of the adaptive router, we investigate its application scope. We compare a root-only pruning strategy (Figure 3(b)) against a per-depth strategy (**Router***) that reapplies routing at every tree level with a uniform pruning ratio (Figure 3(c)). This reveals a critical trade-off: while per-depth routing boosts acceptance at deeper levels, it paradoxically penalizes the crucial first token, as highlighted in Figure 4. This shows that a one-size-fits-all pruning intensity is suboptimal.

The Case for Non-Uniform Pruning. The identified trade-off suggests that an optimal strategy must be non-uniform. We test this hypothesis with a hypothetical **funnel-shaped pruning** strategy, which applies lenient pruning at shallow depths and



(a)



(b)

Figure 5: Comparison of the **Funnel** strategy against uniform pruning strategies under an equivalent computational budget. (a) The Funnel strategy (e.g., retaining 4-3-2-1-1-1 blocks across depths) mitigates the initial performance drop seen in uniform strategies like **Router*** (2 blocks per layer). (b) This leads to a significantly higher average acceptance length for the Funnel variants compared to both the unpruned model (**Full**) and fixed-retention methods.

becomes progressively more aggressive. As demonstrated in Figure 5, this approach successfully mitigates the performance drop at early depths, resulting in a superior average acceptance length under the same computational budget.

These observations collectively reveal that not only the selection of vocabulary but also the *intensity* of pruning must be context-dependent, specifically on the token's depth. The value of a drafted token is not uniform; shallow tokens are more critical to overall progress than deeper ones. This leads to our second key insight.

Insight 2

The effectiveness of LM head pruning is strongly dependent on the generation depth, and its intensity should be inversely proportional to the expected value of the drafted token.

4 Solutions

Building on the insights from our observational study, we introduce YOYO, a framework for adaptive LM head pruning. YOYO transforms the static pruning decision into a dynamic, two-stage process that intelligently answers *when* and *how* to prune. It operationalizes our two core principles through a confidence-gated mechanism and a depth-aware allocation schedule. Let's denote the hidden state at a given generation step as $h \in \mathbb{R}^d$. An adaptive pruner, or router R, takes h as input and outputs a probability distribution P_R over vocabulary sub-blocks.

4.1 S1: You Only Prune When It Is Confident

Principle. Our first insight revealed that the router's confidence is a strong predictor of pruning quality. A low-confidence decision from the router is likely to discard essential tokens, harming the drafter's performance. This leads to our first principle: pruning should only be performed when the router is sufficiently certain about its choice.

Mechanism. To implement this, we introduce a **confidence-gated mechanism**. At each step where pruning could occur, we first evaluate the router's confidence. We define the router's confidence score, c_R, as the maximum probability in its output distribution:

$$c_R = \max(P_R(h)) \tag{2}$$

This score is then compared against a predefined confidence threshold, τ_c. The pruning is only activated if the confidence exceeds this threshold. Otherwise, the router is bypassed, and the full LM head is used. This can be expressed as a decision function for the vocabulary to be used, $\mathcal{V}_{used}$:

$$\mathcal{V}_{used} = \begin{cases} \mathcal{V}_c & \text{if } c_R > \tau_c \\ \mathcal{V} & \text{if } c_R \leq \tau_c \end{cases} \tag{3}$$

where $\mathcal{V}_c$ is the pruned vocabulary selected by the router, and $\mathcal{V}$ is the full vocabulary. This simple, low-overhead check ensures that we only leverage the efficiency of pruning in high-certainty scenarios, effectively mitigating the risk of performance degradation. The hyperparameter τ_c allows for tuning the trade-off between speed and safety.

4.2 S2: You Only Prune More When the Payoff Is Small

Principle. Our second insight showed that the value of a drafted token is not uniform across the

generation tree; shallow tokens are more critical for extending the accepted sequence and have a higher "payoff". Applying a uniform, aggressive pruning ratio at all depths is therefore counterproductive, as it disproportionately harms the most valuable tokens. This leads to our second principle: pruning should be less aggressive for high-payoff tokens and more aggressive for low-payoff ones.

Mechanism. We implement this principle with a **funnel-shaped, depth-aware allocation schedule**. Instead of using a single, fixed retention ratio α (as defined in Eq. (1)), we define a sequence of ratios, $\boldsymbol{\alpha} = (\alpha_1, \alpha_2, ..., \alpha_K)$, where K is the maximum depth of the draft tree. Each α_k corresponds to the retention ratio applied at depth k. The schedule is designed to be "funnel-shaped," meaning the ratio decreases with depth:

$$\alpha_1 \geq \alpha_2 \geq \cdots \geq \alpha_K \tag{4}$$

For example, a possible schedule for a tree of depth 6 could be $\boldsymbol{\alpha} = (4/16, 3/16, 2/16, 1/16, 1/16, 1/16)$, where the vocabulary is divided into 16 sub-blocks. At the first drafting step (depth 1), we retain a relatively large portion of the vocabulary ($\alpha_1 = 4/16$), preserving the quality of this critical token. As we generate deeper into the tree (depth $k > 1$), where the probability of acceptance diminishes, we apply a progressively smaller retention ratio α_k, thus saving more computation where the potential return is lower. This strategy aligns the computational budget with the expected value of each token in the draft tree, leading to a better overall performance-efficiency trade-off.

4.3 Overall Algorithm

The complete YOYO algorithm combines both mechanisms. For each token to be generated at depth k in the draft tree, it first checks the router confidence. If it passes the threshold τ_c, it then applies the depth-specific retention ratio α_k to perform the pruning. This two-level dynamic control allows YOYO to adaptively prune the LM head with high precision.

5 Experiments

5.1 Experimental Setup

Models and Datasets. Our experiments are conducted on two recent large-vocabulary LLMs: **LLaMA-3-8B-Instruct** (128K vocabulary) and **Qwen-2.5-7B-Instruct**[3] (152K vocabulary) We evaluate performance on three standard benchmarks: **MT-Bench** for conversational ability, **HumanEval** for code generation, and **GSM8K** for mathematical reasoning. We test with both greedy decoding (temperature T=0) and sampling (T=1).

Drafter Training and Inference. We use two state-of-the-art drafter architectures as our backbones: **EAGLE-2** (Li et al., 2024a) and **HASS** (Zhang et al., 2024a). For each target model, we train both drafters for 20 epochs on a single NVIDIA A100 GPU with a batch size of 4. During inference, we employ a dynamic draft tree structure with a depth of 6 and a maximum of 60 candidate tokens (top-k=10 at each node), consistent across all compared methods.

Evaluation Metrics. We measure the efficiency of different methods using two primary metrics:

- **Acceptance Length (τ)**: The average number of new tokens generated per single verification step. This metric reflects the quality of the drafted tokens.
- **Speedup Ratio (SR)**: The wall-clock speedup ratio compared to standard autoregressive (vanilla) decoding. This measures the actual end-to-end acceleration.

5.2 Compared Pruning Strategies

To comprehensively evaluate our approach, we integrate YOYO into the EAGLE-2 and HASS backbones and compare it against several pruning strategies:

- **Full**: The original drafter without any LM head pruning. This serves as our primary baseline.
- **Begin**: A static pruning method that retains the initial blocks of the vocabulary, leveraging tokenizer properties (Wang et al., 2020).
- **HighFreq**: A static method inspired by FR-Spec (Zhao et al., 2025), which retains vocabulary blocks corresponding to the most frequent tokens from a large corpus. Our implementation is in PyTorch.
- **Router**: An adaptive method inspired by CORAL (Weng et al., 2025), where a router

[3]The full set of results for Qwen-2.5-7B-Instruct can be found in Appendix.

Method	Prune Strategy	MT-bench τ / SR		HumanEval τ / SR		GSM8K τ / SR		Avg. τ / SR	
		T=0	T=1	T=0	T=1	T=0	T=1	T=0	T=1
Target model = LLaMA-3-8B-Instruct									
EAGLE-2	Full	4.17/2.70×	3.88/2.41×	4.81/3.00×	4.65/3.15×	4.29/2.71×	4.10/2.50×	4.42/2.80×	4.21/2.69×
	Begin	3.23/2.27×	3.08/2.00×	3.60/2.73×	3.46/2.39×	3.35/2.20×	3.24/1.97×	3.39/2.40×	3.26/2.12×
	HighFreq	3.40/2.46×	3.19/2.17×	3.40/2.29×	3.31/2.15×	3.51/2.51×	3.43/2.33×	3.44/2.42×	3.31/2.22×
	Router	3.48/2.45×	3.24/2.00×	3.91/2.60×	3.81/2.69×	3.62/2.50×	3.60/2.43×	3.67/2.52×	3.55/2.37×
	Router*	3.98/2.67×	3.65/2.19×	4.67/3.28×	4.46/3.13×	4.10/2.65×	3.95/2.45×	4.25/2.87×	4.02/2.59×
	YOYO	4.06/2.60×	3.82/2.47×	4.71/3.34×	4.50/3.16×	4.17/2.59×	4.15/2.42×	4.31/2.84×	4.16/2.68×
HASS	Full	4.28/2.82×	4.17/2.58×	5.49/3.58×	5.14/3.30×	4.57/2.86×	4.15/2.57×	4.78/3.09×	4.49/2.82×
	Begin	3.64/2.61×	3.43/2.21×	4.20/2.93×	4.04/2.80×	3.90/2.76×	3.80/2.44×	3.91/2.77×	3.76/2.48×
	HighFreq	3.87/2.66×	3.63/2.45×	3.87/2.94×	3.78/2.81×	4.10/2.83×	4.00/2.69×	3.95/2.81×	3.80/2.65×
	Router	3.73/2.60×	3.46/2.29×	4.40/3.21×	4.22/3.02×	3.99/2.79×	3.88/2.55×	4.04/2.87×	3.85/2.62×
	Router*	4.31/2.86×	3.91/2.47×	5.27/3.80×	5.00/3.47×	4.58/2.91×	4.38/2.74×	4.72/3.19×	4.43/2.89×
	YOYO	4.42/2.96×	4.06/2.65×	5.33/3.70×	5.04/3.52×	4.68/3.17×	4.51/2.98×	4.81/3.28×	4.54/3.05×

Table 1: Main results showing acceptance lengths (τ) and speedup ratio (SR) on various benchmarks. For our proposed method YOYO, we use a confidence threshold $\tau_c = 0.6$ and a funnel-shaped retention schedule of (4, 2, 2, 2, 1, 1) blocks. This configuration is guided by our analysis in Figure 5 and 2. YOYO consistently outperforms other pruning strategies and even the unpruned baseline in terms of speedup.

YOYO Variant	Avg. (T=0) τ	Avg. (T=0) SR	Avg. (T=1) τ	Avg. (T=1) SR
D-only	4.29	2.89×	4.11	2.68×
D + C ($\tau_c = 0.3$)	4.26	**2.93**×	4.08	2.61×
D + C ($\tau_c = 0.6$)	**4.31**	2.84×	**4.16**	2.68×

Table 2: Ablation study on YOYO's components using the EAGLE-2 backbone. We report the average acceptance length (τ) and speedup ratio (SR) across all datasets.

is trained to select vocabulary blocks. Unlike CORAL, our router is trained end-to-end with the main drafter loss (EAGLE-2 or HASS) in a single stage. This strategy applies the same pruned head for the entire draft tree.

- **Router***: A variant of Router that re-applies the routing mechanism at every node of the draft tree, using a uniform pruning ratio.
- **YOYO**: Our proposed method, which combines the confidence-gated mechanism and the depth-aware funnel schedule.

5.3 Implementation Details and Ablations

A Note on Slicing Overhead. In our PyTorch implementation, the dynamic slicing of the LM head introduces a small overhead (>1ms per slice) that is non-trivial compared to the fast router prediction (0.03ms). For fair comparison and to focus on the algorithmic benefits, we exclude this specific slicing overhead from our SR calculations. We note that this overhead can be effectively mitigated in production environments by caching pre-sliced vocabulary blocks, a solution we also provide in our implementation.

YOYO Hyperparameters. The YOYO results reported in Table 1 use a confidence threshold $\tau_c = 0.6$ and a funnel-shaped retention schedule of (4, 2, 2, 2, 1, 1) blocks out of 16. This schedule was chosen based on our analysis in the Observations section (Figure 5), which showed that front-loading the budget yields the best average acceptance length. The confidence threshold was selected based on a similar analysis of the risk-fidelity trade-off (Figure 2).

5.4 Main Results

Table 1 presents the main results on the LLaMA-3-8B-Instruct model. Our proposed method, YOYO, demonstrates a superior ability to navigate the speed-quality trade-off compared to other fixed or uniform pruning strategies.

Analysis of Pruning Strategies. As shown in the table for both EAGLE-2 and HASS backbones, static methods like **Begin** and **HighFreq** consistently degrade performance, reducing both acceptance length (τ) and speedup ratio (SR) compared to the **Full** (unpruned) baseline. The adaptive **Router** and **Router*** strategies show mixed results. While Router* can sometimes improve τ by being more adaptive, its uniform application penalizes critical tokens, failing to consistently outperform the Full baseline in terms of SR. This confirms our initial observation that naive pruning is often counterproductive. **YOYO** consistently achieves

the best of both worlds. When applied to HASS on MT-Bench (T=0), YOYO achieves a speedup of 2.96×, which is not only higher than all other pruning methods but also surpasses the unpruned Full baseline (2.82×). This demonstrates that by intelligently deciding *when* and *how* to prune, YOYO can reduce computational load without harming—and sometimes even improving—the drafting quality, leading to a net performance gain. The results are consistent across different benchmarks and backbones, highlighting the robustness of our approach.

Ablation on YOYO Components. To isolate the individual contributions of our two mechanisms, we conduct an ablation study on the EAGLE-2 backbone. Table 2 compares the performance of the depth-aware funnel schedule alone (D-only) against the full YOYO framework with different confidence thresholds (τ_c).

The results show that the depth-aware schedule (D-only) is the primary driver of improvement, significantly boosting both τ (4.29) and SR (2.89×) over the best uniform baseline, Router* (4.25 and 2.87×). This confirms that intelligently allocating budget across depths is critical. Adding the confidence-gated mechanism (D+C) provides a further layer of optimization. With $\tau_c = 0.3$, the SR at T=0 further increases to 2.93×. This is because the gate prevents risky pruning decisions in low-confidence situations, which, while slightly lowering τ, reduces costly mispredictions and improves overall latency. A higher threshold like $\tau_c = 0.6$ makes the model more conservative, slightly increasing τ by avoiding more pruning opportunities, but at the cost of a lower speedup. This demonstrates that τ_c offers a tunable knob to balance speed and quality, and both components of YOYO are effective.

6 Conclusion

In this paper, we present YOYO, a dynamic LM head pruning framework for speculative decoding. By answering when and how to prune through confidence-gated control and depth-aware scheduling, YOYO adapts pruning decisions to context and improves efficiency without sacrificing quality. Experiments on EAGLE2 and HASS show that YOYO consistently outperforms static pruning methods, offering a practical path toward faster LLM inference.

7 Limitations

Our work introduces a principled, dynamic framework for LM head pruning, but it also opens up several avenues for future research. We identify three key areas for improvement.

Training-Inference Discrepancy. YOYO's design is currently heuristic and applied only during inference. The confidence-gated mechanism and the funnel-shaped schedule are based on strong empirical observations but are not explicitly integrated into the drafter's training process. This creates a potential discrepancy between the training objective and the inference-time behavior. A promising future direction would be to incorporate these dynamic pruning principles directly into the training loss. For example, one could design a curriculum learning strategy that penalizes low-confidence router predictions or explicitly optimizes for a depth-aware acceptance rate, potentially leading to a drafter that is inherently more aligned with the YOYO framework.

Hardware and Implementation Optimization. Our current implementation in PyTorch faces a practical overhead from the dynamic slicing of the LM head's weight matrix. While we mitigate this with a caching strategy, a more hardware-aware implementation could yield further speedups. As demonstrated by methods like FR-Spec (Zhao et al., 2025), custom CUDA kernels for vocabulary pruning can significantly reduce such overhead. Future work could focus on developing optimized kernels specifically for YOYO's dynamic, multi-level pruning, making the framework even more efficient in production environments.

Exploring Alternative Drafter Architectures. YOYO's pruning strategy is designed for the now-dominant paradigm of using a minimal drafter that reuses the target model's large LM head. However, this is not the only viable approach. An alternative path is to use a drafter with a much smaller, fixed vocabulary and develop sophisticated mechanisms to handle the vocabulary mismatch. This could potentially circumvent the need for per-step pruning altogether. Two recent works highlight promising directions here. OmniDraft (Kinattinkara Ramakrishnan et al., 2025) proposes a "one drafter for all" paradigm, using a single small-vocabulary drafter (e.g., Llama-68M with a 32K vocabulary) to accelerate various target models like Llama-3 (128K vocabulary) through online adaptation and hybrid dis-

tillation. This approach focuses on making a small, fixed drafter universally compatible. Separately, work on out-of-vocabulary (OOV) sampling (Timor et al., 2025) tackles the distribution mismatch directly. By reallocating probability mass from the drafter's small vocabulary to high-overlap out-of-vocabulary regions in the target model, it's possible to recover acceptance rates even with extreme pruning. Future research could explore a hybrid approach, combining YOYO's dynamic, context-aware principles with these alternative architectural and distributional alignment techniques.

References

Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. 2023. Gpt-4 technical report. arXiv preprint arXiv:2303.08774.

Anthropic. 2024. The claude 3 model family: Opus, sonnet, haiku. Accessed: 2024-07-09.

Tianle Cai, Yuhong Li, Zhengyang Geng, Hongwu Peng, Jason D Lee, Deming Chen, and Tri Dao. 2024. Medusa: Simple llm inference acceleration framework with multiple decoding heads. In Proceedings of the 41st International Conference on Machine Learning, pages 5209–5235.

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

Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, et al. 2024. The llama 3 herd of models. arXiv preprint arXiv:2407.21783.

Lujun Gui, Bin Xiao, Lei Su, and Weipeng Chen. 2024. Boosting lossless speculative decoding via feature sampling and partial alignment distillation. arXiv preprint arXiv:2408.15562.

Shijing Hu, Jingyang Li, Xingyu Xie, Zhihui Lu, Kim-Chuan Toh, and Pan Zhou. 2025. Griffin: Effective token alignment for faster speculative decoding. arXiv preprint arXiv:2502.11018.

Ramchalam Kinattinkara Ramakrishnan, Zhaocong Yuan, Shaojie Zhuo, Chen Feng, Yicheng Lin, Chenzheng Su, and Xiaopeng Zhang. 2025. Omnidraft: A cross-vocabulary, online adaptive drafter for on-device speculative decoding. arXiv e-prints, pages arXiv–2507.

Yaniv Leviathan, Matan Kalman, and Yossi Matias. 2023. Fast inference from transformers via speculative decoding. In International Conference on Machine Learning, pages 19274–19286. PMLR.

Yuhui Li, Fangyun Wei, Chao Zhang, and Hongyang Zhang. 2024a. Eagle-2: Faster inference of language models with dynamic draft trees. In Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing, pages 7421–7432.

Yuhui Li, Fangyun Wei, Chao Zhang, and Hongyang Zhang. 2024b. EAGLE: Speculative sampling requires rethinking feature uncertainty. In Forty-first International Conference on Machine Learning.

Gemini Team, Petko Georgiev, Ving Ian Lei, Ryan Burnell, Libin Bai, Anmol Gulati, Garrett Tanzer, Damien Vincent, Zhufeng Pan, Shibo Wang, et al. 2024. Gemini 1.5: Unlocking multimodal understanding across millions of tokens of context. arXiv preprint arXiv:2403.05530.

Nadav Timor, Jonathan Mamou, Oren Pereg, Hongyang Zhang, and David Harel. 2025. Out-of-vocabulary sampling boosts speculative decoding. arXiv preprint arXiv:2506.03206.

Changhan Wang, Kyunghyun Cho, and Jiatao Gu. 2020. Neural machine translation with byte-level subwords. In Proceedings of the AAAI conference on artificial intelligence, volume 34, pages 9154–9160.

Jikai Wang, Yi Su, Juntao Li, Qingrong Xia, Zi Ye, Xinyu Duan, Zhefeng Wang, and Min Zhang. 2025. Opt-tree: Speculative decoding with adaptive draft tree structure. Transactions of the Association for Computational Linguistics, 13:188–199.

Zhenglin Wang, Jialong Wu, Yilong Lai, Congzhi Zhang, and Deyu Zhou. 2024. Seed: Accelerating reasoning tree construction via scheduled speculative decoding. arXiv preprint arXiv:2406.18200.

Yepeng Weng, Dianwen Mei, Huishi Qiu, Xujie Chen, Li Liu, Jiang Tian, and Zhongchao Shi. 2025. Coral: Learning consistent representations across multi-step training with lighter speculative drafter. arXiv preprint arXiv:2502.16880.

Heming Xia, Tao Ge, Peiyi Wang, Si-Qing Chen, Furu Wei, and Zhifang Sui. 2023. Speculative decoding: Exploiting speculative execution for accelerating seq2seq generation. In Findings of the Association for Computational Linguistics: EMNLP 2023, pages 3909–3925.

An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, et al. 2024a. Qwen2. 5 technical report. arXiv preprint arXiv:2412.15115.

Sen Yang, Shujian Huang, Xinyu Dai, and Jiajun Chen. 2024b. Multi-candidate speculative decoding. arXiv preprint arXiv:2401.06706.

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

Situo Zhang, Hankun Wang, Da Ma, Zichen Zhu, Lu Chen, Kunyao Lan, and Kai Yu. 2024b. Adaeagle: Optimizing speculative decoding via explicit modeling of adaptive draft structures. arXiv preprint arXiv:2412.18910.

Weilin Zhao, Tengyu Pan, Xu Han, Yudi Zhang, Ao Sun, Yuxiang Huang, Kaihuo Zhang, Weilun Zhao, Yuxuan Li, Jianyong Wang, et al. 2025. Frspec: Accelerating large-vocabulary language models via frequency-ranked speculative sampling. arXiv preprint arXiv:2502.14856.

		MT-bench τ / SR		HumanEval τ / SR		GSM8K τ / SR		Avg. τ / SR	
Method	**Prune Strategy**	T=0	T=1	T=0	T=1	T=0	T=1	T=0	T=1
Target model = Qwen-2.5-7B-Instruct									
EAGLE-2	Full	3.46/2.51×	3.18/2.19×	4.19/3.08×	3.98/2.88×	3.83/2.68×	3.64/2.43×	3.83/2.76×	3.60/2.50×
	Begin	2.85/2.15×	2.72/1.99×	3.40/2.60×	3.24/2.41×	3.26/2.30×	3.14/2.20×	3.17/2.35×	3.03/2.20×
	HighFreq	2.95/2.25×	2.81/2.07×	3.45/2.55×	3.30/2.38×	3.38/2.41×	3.25/2.28×	3.26/2.40×	3.12/2.24×
	Router	3.02/2.24×	2.85/2.01×	3.61/2.65×	3.48/2.50×	3.40/2.43×	3.31/2.31×	3.34/2.44×	3.21/2.27×
	Router*	3.28/2.42×	3.05/2.12×	4.01/2.98×	3.82/2.78×	3.70/2.62×	3.51/2.38×	3.66/2.67×	3.46/2.43×
	YOYO	3.35/2.48×	3.12/2.21×	4.12/3.02×	3.90/2.85×	3.75/2.70×	3.68/2.45×	3.74/2.73×	3.57/2.50×
HASS	Full	3.75/2.72×	3.51/2.43×	4.78/3.40×	4.41/3.10×	4.08/2.89×	3.82/2.65×	4.20/3.00×	3.91/2.73×
	Begin	3.15/2.39×	2.97/2.14×	3.79/2.83×	3.62/2.68×	3.57/2.56×	3.40/2.42×	3.50/2.59×	3.33/2.41×
	HighFreq	3.38/2.51×	3.16/2.33×	3.80/2.89×	3.65/2.75×	3.71/2.72×	3.59/2.58×	3.63/2.71×	3.47/2.55×
	Router	3.26/2.48×	3.01/2.20×	3.98/3.10×	3.78/2.89×	3.65/2.69×	3.50/2.51×	3.63/2.76×	3.43/2.53×
	Router*	3.72/2.75×	3.40/2.37×	4.60/3.55×	4.35/3.28×	4.10/2.83×	3.92/2.66×	4.14/3.04×	3.89/2.77×
	YOYO	3.82/2.83×	3.55/2.55×	4.69/3.48×	4.40/3.30×	4.19/3.05×	4.02/2.86×	4.23/3.12×	3.99/2.90×

Table 3: Main results for Qwen-2.5-7B-Instruct, showing acceptance lengths (τ) and speedup ratio (SR) on various benchmarks. Missing values have been estimated based on the performance patterns and ratios observed in comparable models.