TALON: Confidence-Aware Speculative Decoding with Adaptive Token Trees

Tianyu Liu[1,2*] **Qitan Lv**[1,2*] **Yuhao Shen**[3]
Jun Zhang[3] **Xiao Sun**[2†] **Xiaoyan Sun**[1]
[1]University of Science and Technology of China
[2]Shanghai AI Laboratory [3]Zhejiang University
{tianyu_liu, qitanlv}@mail.ustc.edu.cn riven@zju.edu.cn
sunxiao@pjlab.org.cn sunxiaoyan@ustc.edu.cn

Abstract

Speculative decoding (SD) has become a standard technique for accelerating LLM inference without sacrificing output quality. Recent advances in speculative decoding have shifted from sequential chain-based drafting to tree-structured generation, where the draft model constructs a tree of candidate tokens to explore multiple possible drafts in parallel. However, existing tree-based SD methods typically build a **fixed-width, fixed-depth** draft tree, which fails to adapt to the varying difficulty of tokens and contexts. As a result, the draft model cannot dynamically adjust the tree structure to early stop on difficult tokens and extend generation for simple ones. To address these challenges, we introduce **TALON**, a ***training-free, budget-driven*** adaptive tree expansion framework that can be plugged into existing tree-based methods. Unlike static methods, **TALON** constructs the draft tree iteratively until a fixed token budget is met, using a hybrid expansion strategy that adaptively allocates the node budget to each layer of the draft tree. This framework naturally shapes the draft tree into a **"deep-and-narrow"** form for deterministic contexts and a **"shallow-and-wide"** form for uncertain branches, effectively optimizing the trade-off between exploration width and generation depth under a given budget. Extensive experiments across 5 models and 6 datasets demonstrate that **TALON** consistently outperforms state-of-the-art EAGLE-3, achieving up to 5.16× end-to-end speedup over auto-regressive decoding.

1 Introduction

While Large Language Models (LLMs) have achieved remarkable success in various benchmarks [37, 16, 2, 40, 10, 44], their deployment is constrained by their auto-regressive token-by-token generation. This sequential dependency prevents models from predicting multiple tokens [15] in a single step, causing inference latency to scale linearly with output length [39].

To alleviate this limitation, *Speculative Decoding* (SD) [23, 6] emerges as a promising paradigm [28]. SD decouples each decoding step into two sub-procedures: efficient drafting and parallel verification. A lightweight draft model first proposes a candidate sequence, and the target LLM validates all proposed tokens in parallel, accepting multiple tokens in a single target model forward.

However, early SD methods typically employ a *chain-based* drafting strategy, which is inherently vulnerable. A single rejection at the beginning of the draft sequence invalidates all subsequent

*Equal Contribution.
†The Corresponding Author.

40th Conference on Neural Information Processing Systems (NeurIPS 2026).

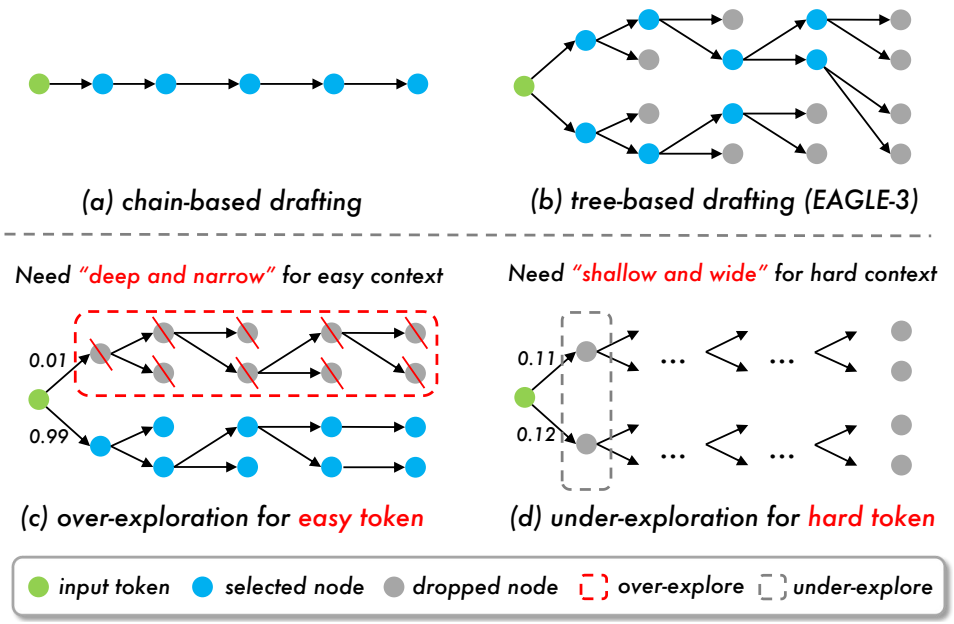


Figure 1: Overview of tree-based SD and the motivation of **TALON**. **Top:** chain-based drafting proposes a single sequence where early rejection invalidates all subsequent tokens, while tree-based drafting explores multiple paths in parallel via tree attention. **Bottom:** static tree structures fail to adapt to varying difficulty—over-exploring confident tokens and under-exploring uncertain branches.

tokens, leading to a substantial waste of computational resources [32]. To further improve the overall acceptance rate, recent works shift from chain-based to *tree-based drafting* [32, 5, 25, 12, 24]. Instead of predicting a single sequence, they construct a token tree covering multiple plausible continuation paths. By leveraging *Tree Attention* [51], the target LLM can verify multiple draft token sequences in parallel within a single forward pass, increasing the draft acceptance rate and overall acceleration.

State-of-the-art tree-based methods, such as EAGLE [25], construct the draft tree via a rigid layer-wise expansion mechanism. As shown in Figure 1(b), these approaches iteratively expand a fixed number of child nodes for every parent and select the top-K candidates to proceed to the next layer. However, this strategy enforces a *static tree structure* with pre-determined dimensions. Such rigid allocation fails to adapt to the model's dynamic confidence (output probabilities of its tokens)[3]: it still expands K child nodes even if the draft model is already confident (Figure 1(c), left bottom), while it cannot allocate more node budget when the model is confused and highly uncertain (Figure 1(d), right bottom). Moreover, it prematurely truncates high-confidence paths that could extend deeper, while squandering the computational budget on expanding low-probability branches in uncertain contexts.

To tackle these inefficiencies, we introduce **TALON**, a novel *training-free, budget-driven* adaptive tree expansion framework to op**T**imize dr**A**ft tree for faster specu**L**ative dec**O**di**N**g. Departing from the rigid fixed width and depth generation, **TALON** employs a dynamic growth algorithm that incrementally expands the draft tree until the number of nodes reaches a given budget N. Specifically, we design a hybrid expansion strategy to optimize the tree topology: at the first layer, we propose a fixed width initialization that utilizes a top-K operation to alleviate the early rejection phenomenon. At subsequent layers, we propose a confidence-gated expansion strategy to adaptively allocate the node budget. This allows the draft structure to evolve to **deep-and-narrow** for deterministic contexts to maximize draft length, or **shallow-and-wide** for uncertain ones to enhance hit rate.

To summarize, our contributions are:

(i) We identify a key limitation in existing tree-based speculative decoding methods: the strict, layer-wise static tree expansion fails to adapt to the model's varying confidence, leading to inefficient utilization of the computational budget.

(ii) We propose **TALON**, a budget-driven speculative decoding framework. By employing a dynamic tree growth algorithm with robust tree initialization and confidence-gated expansion,

[3]We clarify the precise sense of "dynamic" in EAGLE and contrast it with **TALON** in Appendix A.

TALON constructs adaptive draft trees that dynamically adapts (deep-and-narrow or shallow-and-wide) based on context uncertainty.

(iii) Extensive experiments verify the effectiveness of TALON. It significantly outperforms state-of-the-art method EAGLE-3 in terms of draft efficiency and wall-clock speedup, particularly in scenarios with fluctuating generation difficulty.

2 Related Work

In this section, we review speculative decoding from the perspective of *drafting structures*: chain-based speculative decoding and tree-based speculative decoding. **We position TALON in the context of extensive related works in Appendix F.**

Chain-based speculative decoding. Speculative decoding [23, 6] introduces a lossless *draft-and-verify* [53] principle: a cheap drafter proposes a short continuation and a target model verifies it in parallel, accepting the longest matched prefix. This paradigm suffers from a clear efficiency bottleneck—*early rejection* [43]—since a mismatch at an early position invalidates all downstream drafted tokens. Follow-up work improves chain-based SD by (i) producing better proposals with minimal overhead (e.g., lightweight head-based drafting that reuses target representations[5, 12, 25]), and (ii) reducing wasted draft computation via adaptive lookahead [20] or pipelined scheduling [28]. **Nevertheless, the speedup is fundamentally sensitive to the hardest tokens, because one rejection discards the entire suffix.**

Tree-based speculative decoding. Tree-based SD generalizes the draft from a single chain to a *token tree*, allowing the verifier to choose among multiple candidate branches within one forward and thus mitigating early rejection. SpecInfer [32] is an early representative that organizes candidates as a token tree and verifies them in parallel with tree attention. More recent work strengthens the tree-based pipeline from two angles: (i) improving tree proposals via well-calibrated drafters [55, 21, 14], context-aware dynamic draft trees [24, 50], and adaptive draft depth [56, 4], and (ii) optimizing the draft-tree structure under a node budget, including dynamic-programming-based and search-based construction (e.g., Sequoia [8] and OPT-Tree [46]). However, most of these methods fix the tree's shape, or at least its maximum depth, before drafting begins, so **the budget cannot shift between width and depth as the drafter's confidence changes across contexts**. TALON fills this gap with a training-free framework that allocates a global node budget during expansion via confidence gating, so the same budget can form deep-and-narrow chains or shallow-and-wide trees (Appendix F).

3 Background

We first formulate *chain-based* drafting and *tree-based* drafting for better understanding.

3.1 Chain-based Drafting

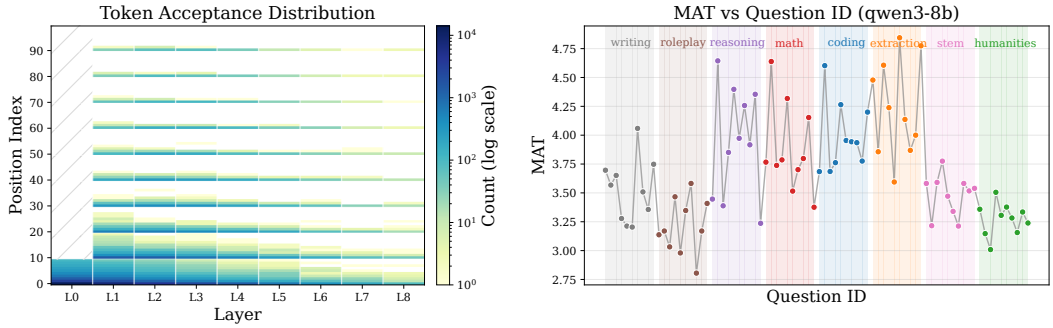
In standard speculative decoding, a smaller draft model $\mathcal{M}_d$ generates a single sequence of γ tokens (a chain) as a speculation for the target model $\mathcal{M}_t$. Given the prefix $x_{\leq t}$, the drafting process generates a sequence $x_{t+1}, \ldots, x_{t+\gamma}$ auto-regressively:

$$x_{t+i} \sim P_{\mathcal{M}_d}(\cdot|x_{<t+i}), \quad 1 \leq i \leq \gamma \tag{1}$$

The target model $\mathcal{M}_t$ then verifies this sequence in parallel. The key limitation of this approach is the *sequential dependency* during verification: if the token at position i is rejected, all subsequent tokens $x_{>i}$ are discarded regardless of their correctness, resulting in wasted computation.

3.2 Tree-based Drafting

To mitigate the limitations of chain-based drafting, recent works (e.g., EAGLE) verify a token tree $\mathcal{T}$ to cover diverse paths. As formalized in Algorithm 1 and Figure 1 (left), these methods employ a *static* construction strategy: at each depth d, the model takes a parallel forward on all parent nodes and outputs their next-token distributions. Then, it selects top-K entries from the output distribution



(a) Acceptance frequency in a static draft tree.

(b) MAT distribution across real-world queries.

Figure 2: Empirical motivation for adaptive token-tree construction with EAGLE. **Left:** acceptance frequency exhibits an "Acceptance Funnel" effect, where accepted tokens concentrate in high-confidence regions as depth increases. **Right:** accepted length varies substantially across queries, showing that a fixed-depth policy cannot match varying generation difficulty.

of each parent node, acquiring $K \times K$ leaf nodes. After that, the model measures the path score of each layer node v:

$$p(v) = \prod_{j \in \text{Path}(x_t, v)} P_{\mathcal{M}_d}(j \mid x_{<j}) \tag{2}$$

where $\text{Path}(x_t, v)$ represents the path from the root node x_t to the leaf node v, $x_{<j}$ denotes all prefix tokens of j. Then it uses another top-K operation based on the path score to select K leaf nodes as parent nodes of next layer. Finally, the generation ends at depth D, and the tree $\mathcal{T}$ will be pruned to meet a global budget N. While structured, this rigid "expand-then-shrink" mechanism often generates redundant nodes that are discarded during intermediate shrinking or final pruning, leading to suboptimal resource allocation.

4 Motivated Experiments

To empirically investigate the limitations of static tree-based speculative decoding, we conduct a pilot analysis using EAGLE [25] as a representative baseline. We employ a fixed tree topology with width $K = 10$ and depth $D = 8$ (official settings in EAGLE paper) with Qwen3-8B on MT-Bench [57]. We also conduct additional motivated experiments in Appendix C with Llama-series models and various datasets to demonstrate the generality of our motivation.

4.1 The "Acceptance Funnel" Phenomenon

We first investigate the acceptance frequency of each position in the draft tree. Detailed model configurations are provided in Table 5. Figure 2(a) visualizes the acceptance frequency of tokens at each position within the static tree structure. Two key observations emerge regarding the *distribution of effective speculation*:

Diminishing Returns of Width in Deep Layers. As illustrated in Figure 2(a), the acceptance distribution exhibits a funnel-like pattern. In the first initial layer ($d = 0$), the acceptance probability is relatively uniform across the top-K candidates. This suggests that due to the initial stochasticity and minor distributional divergence between the draft model $\mathcal{M}_d$ and the target model $\mathcal{M}_t$, a wider search breadth is necessary to capture the valid continuation. However, as the tree deepens ($d \geq 1$), the acceptance mass concentrates sharply on the high confidence regions (e.g. top-1 and top-2 candidates). We attribute this to the *cumulative error amplification* in auto-regressive drafting: for deep nodes, the draft model is either (1) confidently aligned with the target model (correct prediction), or (2) hallucinating a divergent path where even the top-K candidates fail to recover the target distribution. Consequently, maintaining a static width $K = 10$ at deeper layers yields negligible marginal utility. When the draft model is aligned, a much smaller K can find the correct draft tokens.

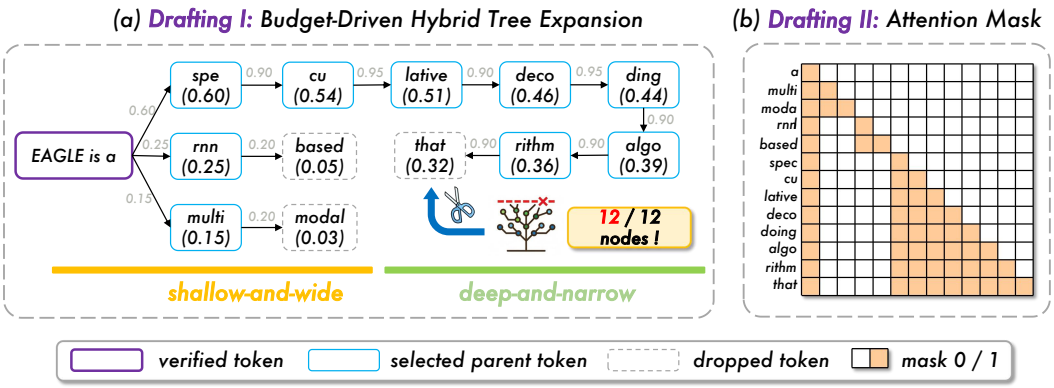


Figure 3: (a) **TALON**'s Budget-Driven Tree. **TALON** dynamically allocates the token budget based on confidence. It uses Top-K at the root for robustness and confidence gating at deeper layers, resulting in adaptive topologies—deep chains for high-confidence tokens (e.g., the folded "algo-rithm" sequence) and wide branches for uncertain ones. The expansion stops when the global budget is met (indicated by the icon). (b) Tree Attention Mask. The structural mask used by the target model to verify the adaptive tree in parallel. The verification process follows the standard token-tree verification protocol (see Appendix B for details).

When the draft model is highly uncertain, there needs a larger K to ensure the coverage and stops the generation to avoid wasteful draft model forward.

4.2 Variance in Real-World Accepted Length

While Section 4.1 reveals the structural redundancy within fixed width, we further investigate the weakness of fixed depth. We plot the distribution of Mean Accepted Tokens (MAT) for various queries in Figure 2(b). (Same settings with Figure 2(a))

The Static Depth Dilemma. As shown in Figure 2(b), the accepted tokens fluctuate drastically even within the same task category (e.g., Math or Coding), exposing the limitation of a fixed-depth drafting policy. In low-entropy contexts where the draft model is well-aligned with the target, the potential acceptance length often exceeds the pre-defined depth D. However, a static fixed depth D prevents $\mathcal{M}_d$ from generating more draft tokens for maximum speedup. Conversely, in high-entropy scenarios where prediction becomes ambiguous, the draft model tends to hallucinate deep branches that are destined for rejection. A fixed depth D forces the allocation of computational resources to these invalid tokens, incurring latency overhead with zero marginal utility.

5 Method

We introduce **TALON**, a budget-driven framework that optimizes draft tree construction by dynamically allocating resources based on real-time model confidence.

5.1 From Static Grids to Dynamic Budgets

To overcome the inefficiencies of the *static tree structures* analyzed in Section 4–specifically the misalignment between fixed topology and varying token difficulty–**TALON** shifts the constraint from *shape* to *capacity*. As illustrated in Figure 3(a), we define a global **Token Budget** (N), representing the maximum number of nodes allowed in the draft tree $\mathcal{T}$. The objective is to iteratively "invest" this budget to grow a draft tree that dynamically adapts–*deep-and-narrow* for deterministic contexts and *shallow-and-wide* for uncertain ones–thereby maximizing effective speculation length under a fixed computational cost. The resulting tree is then verified by the target model in a single forward pass via the standard tree-attention mask shown in Figure 3(b). Since **TALON** only modifies how the draft tree is *constructed* and reuses the standard tree-attention verification protocol of EAGLE (Appendix B), it is **lossless**: the output distribution is mathematically identical to that of standard auto-regressive decoding, so all reported speedups come at no quality cost.

5.2 Hybrid Tree Expansion Strategy

TALON constructs the tree layer-by-layer using a hybrid strategy: a robust initialization at the root (Layer 0) followed by confidence-gated expansion for subsequent layers (Layer $d \geq 1$).

5.2.1 Confidence-Gated Expansion

For depths $d \geq 1$, we employ a global filtering mechanism to gate candidates based on their relative confidence. Let $\mathcal{P}_d$ be the set of parent nodes. We define the **Candidate Pool** $\mathcal{S}_d$ as the union of all child extensions:

$$\mathcal{S}_d = \bigcup_{v \in \mathcal{P}_d} \{(v, w) \mid w \in \mathcal{V}\}. \tag{3}$$

Each candidate $u = (v, w) \in \mathcal{S}_d$ is assigned a cumulative path probability $p(u) = p(v) \cdot P_{\mathcal{M}_d}(w \mid x_{\leq v})$.

Motivated by [33], we first identify the **anchor confidence** $m_d = \max_{u \in \mathcal{S}_d} p(u)$. We then retain candidates whose confidence falls within a dynamic margin μ of the anchor:

$$\mathcal{P}_{d+1} = \{u \in \mathcal{S}_d \mid p(u) \geq \mu \cdot m_d\}, \tag{4}$$

where $\mu \in (0, 1]$ is a hyperparameter. To strictly respect the budget, if $|\mathcal{P}_{d+1}|$ exceeds the remaining budget $N - |\mathcal{T}|$, we retain only the top candidates with the highest path scores.

Intuition. This mechanism naturally aligns topology with entropy. In deterministic contexts (e.g., "The capital of France is *Paris*"), the anchor $m_d \approx 1.0$ imposes a strict threshold, automatically pruning branches to form a *deep-and-narrow* chain. In high-entropy contexts (e.g., "Quantum computing can..."), a lower m_d relaxes the threshold, admitting diverse candidates to form a *shallow-and-wide* layer that prioritizes coverage.

5.2.2 Robust Tree Initialization

While confidence gating effectively allocates budget at depth, applying it at the root ($d = 0$) compromises robustness due to draft model ***over-confidence***. Small draft models often exhibit imperfect calibration, assigning near-certain probability (e.g., > 0.99) to incorrect tokens in simple contexts. Relative gating would prematurely collapse the tree into a single wrong branch–an *over-confidence trap*. Unlike deeper errors, a root failure causes catastrophic rejection of the entire draft sequence. To mitigate calibration errors, we employ a **Standard Top-K Initialization** for the first layer. Regardless of the confidence distribution, we explicitly expand the top-K tokens:

$$\mathcal{P}_1 = \{(x_t, v) \mid v \in \text{Top-}K\ (P_{\mathcal{M}_d}(\cdot|x_t))\}. \tag{5}$$

This ensures the speculative tree covers diverse plausible directions at the most critical juncture before switching to budget-efficient gating.

5.3 Evaluating TALON with Draft Efficiency

To rigorously quantify the cost-effectiveness of **TALON**, we analyze the wall-time speedup by decomposing it into quality and cost factors.

Let N_p denote the total number of forward passes executed by the target model (verification steps), N_q be the total forward passes executed by the draft model to generate a sequence and L be the output length. We introduce a new metric, **Draft Efficiency** (δ), defined as the ratio of these two quantities:

$$\delta = \frac{N_q}{N_p}. \tag{6}$$

Intuitively, δ represents the **speculation cost**–the average number of draft steps the system "invests" to secure a single verification opportunity. By deriving the wall-time speedup R as a function of the Mean Accepted Tokens ($\tau = L/N_p$) and the relative model cost (c), we obtain the following speedup formulation (derived in Appendix D):

$$R = \frac{\tau}{1 + c \cdot \delta}. \tag{7}$$

Table 1: **Main Results on Six Benchmarks (Temperature=0).** Comparison of **TALON** against speculative decoding baselines across various models. We report Mean Acceptance Tokens (MAT) and Wall-time Speedup relative to standard decoding. **Bold** numbers denote the best speedup performance. Speedups of EAGLE-3 and **TALON** are means over 5 runs; standard deviations are reported in Appendix E.2.

Model	Method	Alpaca		GSM8K		HumanEval		MT-Bench		QA		CNN/DM	
		MAT	Spd.	MAT	Spd.	MAT	Spd.	MAT	Spd.	MAT	Spd.	MAT	Spd.
Vicuna-13B	EAGLE-3	6.61	3.78×	6.74	3.87×	8.31	4.77×	7.12	4.05×	5.24	3.03×	6.93	3.43×
	SD	2.19	1.30×	2.20	1.19×	2.86	1.47×	2.51	1.29×	1.97	1.20×	2.63	1.41×
	LOOKAHEAD	1.51	1.21×	1.90	1.50×	1.75	1.36×	1.67	1.29×	1.42	1.16×	1.49	1.10×
	PLD	1.22	1.05×	1.73	1.45×	1.93	1.58×	1.60	1.32×	1.17	1.01×	2.51	1.91×
	MEDUSA	2.44	1.90×	2.63	2.05×	2.78	2.18×	2.58	2.01×	2.10	1.62×	2.09	1.56×
	HYDRA	3.51	2.40×	3.66	2.53×	3.87	2.67×	3.64	2.46×	2.88	1.95×	2.82	1.86×
	OPT-Tree	6.56	3.71×	6.77	3.65×	8.21	4.35×	6.95	3.75×	5.22	3.10×	6.91	3.22×
	TALON	6.36	**3.94**×	6.80	**3.99**×	9.48	**5.16**×	7.29	**4.27**×	5.03	**3.26**×	7.20	**3.58**×
DSL-8B	EAGLE-3	5.64	3.23×	7.40	4.24×	6.70	3.85×	5.82	3.33×	5.02	2.86×	5.03	2.89×
	TALON	5.18	**3.46**×	7.46	**4.43**×	6.28	**3.96**×	5.45	**3.56**×	4.66	**3.12**×	4.75	**3.14**×
Llama3-8B	EAGLE-3	6.93	3.92×	6.42	3.63×	7.08	4.05×	6.38	3.67×	5.36	3.01×	5.46	3.06×
	TALON	6.51	**4.04**×	6.24	**3.81**×	7.28	**4.20**×	6.22	**3.85**×	5.09	**3.27**×	5.28	**3.30**×
Qwen3-8B	EAGLE-3	3.45	2.08×	3.92	2.37×	3.89	2.35×	3.64	2.20×	3.46	2.09×	3.27	1.95×
	TALON	3.39	**2.44**×	3.85	**2.67**×	3.78	**2.69**×	3.60	**2.57**×	3.41	**2.46**×	3.20	**2.30**×
Qwen3-32B	EAGLE-3	2.75	1.83×	3.36	2.22×	2.98	1.96×	2.98	1.90×	2.66	1.78×	2.57	1.57×
	TALON	2.72	**2.05**×	3.30	**2.38**×	2.96	**2.15**×	2.94	**2.10**×	2.64	**1.99**×	2.54	**1.72**×

With Equation 7, we can theoretically explain **why TALON achieves superior performance**. Approaches like EAGLE enforce a fixed depth D, effectively locking the draft cost to a constant $\delta = D+1$ regardless of the context complexity. This rigidity leads to dual inefficiencies.

In **high-entropy contexts** (hard cases), the acceptance reward τ naturally drops. EAGLE continues to pay the high fixed cost δ, causing the speedup R to degrade significantly. **TALON** addresses this by halting expansion early; the resulting reduction in "investment" ($\delta \downarrow$) compensates for the lower reward, thereby mitigating the performance drop. Conversely, in **low-entropy contexts** (easy cases), static methods artificially cap the reward τ at depth D. **TALON** allows the draft tree to extend far beyond this limit, significantly increasing τ. While growing deeper also increases the draft cost δ, this investment yields a positive net gain. Since the relative cost is typically small ($c \ll 1$), $(1 + c\delta)$ grows much slower than τ, ensuring that the overall speedup R continues to improve as the tree deepens. We provide a concrete example in Appendix D.

6 Experiments

6.1 Experimental Setup

Tasks and Datasets. We evaluate **TALON** on a comprehensive suite of LLM backbones, including Llama-3.1-8B-Instruct [45], Qwen3 8B and 32B [40], DeepSeek-R1-Distill-LLaMA-8B (DSL) [10], and Vicuna-13B [57]. Following standard protocols, we conduct evaluations on six diverse benchmarks: MT-Bench [57], Alpaca [11], GSM8K [9], HumanEval [7], CNN/DM [35], and QA [22], which are widely used benchmarks for instruction, math reasoning, code generation, chat, QA and summarization.

Implementation Details. We evaluate **TALON** against the state-of-the-art tree-based method EAGLE-3 [26]. All experiments are conducted on a single NVIDIA H200 (141GB) GPU with batch size 1 by default. The global token budget N is set to `60` for both **TALON** and EAGLE-3. The threshold μ in **TALON** is set to `0.03` by default (Section 6.4.2 discusses this choice). Detailed configurations (hyperparameters, hardware, baselines, models) are provided in Appendix E.1. We use two metrics: mean accepted tokens (MAT) and wall-time speedup (Spd.).

6.2 Main Results

We present the main comparison with EAGLE-3 in Table 1. We further evaluate **TALON** under stochastic sampling ($T = 1$) in Table 7, and provide detailed ablation studies on tree initialization and budget settings in Appendix E.6.

We observe the following: (a) **Universal Speedup.** **TALON** consistently outperforms EAGLE-3 across all 5 models and 6 datasets. Notably, it achieves up to $5.16\times$ speedup on HumanEval with Vicuna-13B and $2.30\times$ speedup on CNN/DM with Qwen3-8B, significantly surpassing the baseline's $4.77\times$ and $1.95\times$. (b) **Reasoning Adaptability.** The performance gap is particularly pronounced in reasoning-intensive tasks. On GSM8K (Math) and HumanEval (Code), **TALON** achieves substantial gains (e.g., $2.67\times$ vs $2.37\times$ on Qwen3-8B GSM8K). This verifies that our budget-driven adaptive expansion effectively captures correct paths in low-entropy reasoning steps where static trees often under-explore. (c) **Robustness.** As shown in Table 7, **TALON** maintains strong performance with $T = 1$. While static trees suffer from flattened distributions, **TALON**'s confidence-gated mechanism successfully adapts the tree topology, achieving $2.51\times$ speedup on Qwen3-8B (HumanEval) compared to EAGLE-3's $2.20\times$. The gains also do not hinge on a particular draft checkpoint or GPU. With Qwen3-8B as the target, **TALON** improves throughput over EAGLE-3 by 14.0–17.3% with a weaker draft checkpoint and by 9.1–10.4% with a stronger one (Appendix E.6); the gain is larger with the weaker drafter because a static deep tree wastes more computation when the drafter is less reliable. On NVIDIA H20, where the relative draft cost c (Section 5.3) is lower than on H200 (0.032 vs. 0.052), **TALON** is again faster than EAGLE-3 on all six benchmarks (Appendix E.5).

6.3 Evaluating TALON with Draft Efficiency

We evaluate the draft efficiency (defined in Section 5.3) of **TALON** and EAGLE and visualize the relationship between the overhead (δ) and the benefit (τ) in Figure 4. Static methods (orange line) pay a fixed high computational cost without considering context difficulty, resulting in sub-optimal speedup. In contrast, **TALON** (blue line) closely approaches the zero-waste Oracle line ($\tau = \delta$). By dynamically shrinking the token trees in uncertain regions and expanding them in deterministic ones, **TALON** effectively decouples draft cost from tree depth. The reallocation goes in both directions. On Vicuna-13B/HumanEval, **TALON** spends more draft steps per verification than EAGLE-3's fixed $\delta = D + 1 = 9$ ($\delta = 9.77$) and raises MAT from 8.31 to 9.48; on DSL-8B, its MAT is slightly lower than EAGLE-3's on most benchmarks (Table 1), but the reduction in δ more than compensates, as Eq. (7) predicts. This confirms that our framework maximizes speedup by ensuring computational resources are only invested where they yield high acceptance utility. The same trade-off explains the behavior of OPT-Tree, which optimizes the expected acceptance length without accounting for the drafting cost that produces it: on Vicuna-13B, its MAT is close to EAGLE-3's (e.g., 8.21 vs. 8.31 on HumanEval), yet its speedup is lower on five of the six benchmarks (Table 1). **TALON** is the fastest on all six, as it is with Llama-3.1-8B (Appendix E.9). We also provide more visualization results of different models in Appendix E.3.

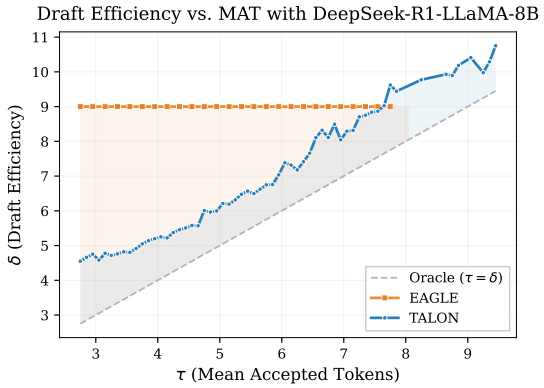


Figure 4: Draft efficiency vs. MAT. **TALON** tracks the Oracle line more closely than EAGLE, reducing wasted draft computation across varying generation difficulty.

6.4 Ablation Study

6.4.1 Ablating Robust Tree Initialization

TALON employs a hybrid strategy that begins with robust tree initialization for the first layer to ensure robustness, followed by confidence-gated expansion layers. However, a natural question arises, *"Why is robust tree initialization necessary only for the first layer"?* To answer this question, we conduct

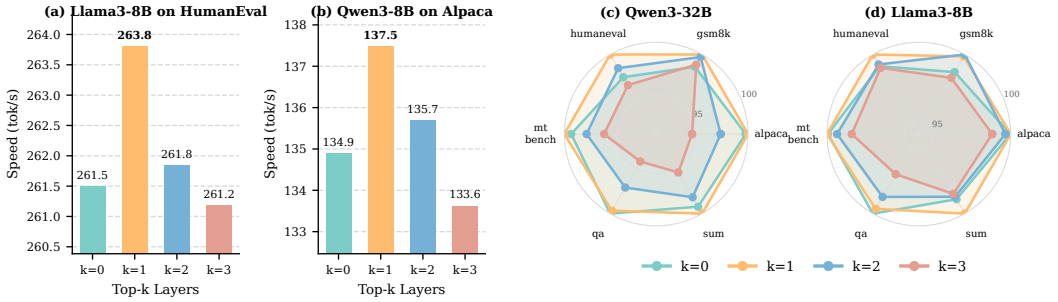


Figure 5: Ablation study on the number of initial Top-K layers (k). $k = 0$ represents the pure adaptive strategy without robust initialization, while $k \geq 1$ indicates using static Top-K expansion for the first k layers. (a)-(b) Wall-time speedup on HumanEval and Alpaca shows that $k = 1$ (**TALON**'s default) achieves the highest throughput. (c)-(d) Radar charts across six benchmarks further confirm that $k = 1$ (orange line) yields the most robust performance, outperforming both the uninitialized ($k = 0$) and over-extended ($k \geq 2$) configurations.

an ablation study using robust tree initialization for the first k layers. $k = 0$ corresponds to a pure confidence-gated strategy without robust tree initialization.

Necessity of Robust Initialization ($k = 1$ *vs.* $k = 0$). As shown in Fig. 5 (a) and (b), removing the robust tree initialization leads to a noticeable drop in generation speed (e.g., declining from 263.8 to 261.5 tok/s on Llama3-8B). This degradation validates our hypothesis that draft models often suffer from root layer *over-confidence*; without a forced Top-K expansion at the first layer, the model risks falling into incorrect branches due to over-confidence, leading to early draft rejections.

Overhead of additional Robust Initialization ($k \geq 2$). Conversely, extending the robust initialization phase to deeper layers ($k = 2, 3$) yields suboptimal returns. It delays the transition to the more efficient confidence-gating expansion, thereby increasing the computational overhead (δ) without a proportional gain in acceptance. The radar charts in Fig. 5 (c) and (d) clearly illustrate that $k = 1$ (orange line) consistently outperforms other configurations, confirming that a single layer of robust initialization followed immediately by adaptive gating achieves the optimal speedup.

6.4.2 Ablating Confidence-gated Expansion

To further investigate the mechanism behind **TALON**, we conduct three additional ablation studies on the influence of different threshold μ, different budget N, and the necessity of the global budget itself in Appendix E.6. The results show that the threshold μ in **TALON** serves as a trade-off hyperparameter **between exploration and exploitation**. A larger μ leads to more deep-and-narrow draft trees, especially effective in reasoning tasks such as coding and math, while a smaller μ is more appropriate for some creative tasks. Moreover, **TALON** seamlessly adapts to different computation budgets by simply adjusting N, while static methods waste more computation under fixed K and D. Notably, simply applying confidence gating on top of EAGLE's static tree *without* our budget mechanism (EAGLE-DYNAMIC) actually *degrades* performance—e.g., on Qwen3-8B GSM8K, 2.37× (EAGLE) → 2.29× (EAGLE-DYNAMIC) → 2.67× (**TALON**)—confirming that the budget-driven framework is the key driver of the observed speedup, not confidence gating alone.

Choosing μ. Since μ is defined relative to the per-layer anchor m_d, the effective cutoff $\mu \cdot m_d$ already tightens in low-entropy layers and loosens in high-entropy ones; μ itself fixes neither the width nor the depth. We therefore use a single value, $\mu = 0.03$, that is near-optimal across the tested draft–target pairs instead of tuning it per setting. In a five-run MT-Bench sweep over $\mu \in [0.01, 0.05]$ (Appendix E.6), this default matches the best tested value on Vicuna-13B and Llama3-8B, is within 0.9% and 1.2% of it on DSL-8B and Qwen3-8B, and outperforms EAGLE-3 by 4.8–17.1% on every model. The best μ tends to increase with draft–target alignment, which can be read from the measured MAT (0.04 for the best-aligned DSL-8B, 0.01 for the least-aligned Qwen3-8B), and also depends on the task; given this small sensitivity, 0.03 is a reasonable starting point for a new model.

Table 2: Throughput (tok/s) of Qwen3-32B under batched decoding (small-tree setting, mean ± std over 5 runs).

bs	EAGLE-3	TALON	Margin
4	220.0±5.0	226.4±0.5	+2.9%
8	437.6±7.1	446.1±11.4	+2.0%
16	851.3±6.9	894.0±10.7	+5.0%
32	1700.7±37.8	1811.2±8.9	+6.5%
64	2884.4±6.1	3186.6±23.5	**+10.5%**

Table 3: Per-stage latency (ms) of one decoding step (Qwen3-8B, H200, SGLang, $\mathrm{bs} = 8$).

Stage	EAGLE-3	TALON	Δ
Tree mask	0.117	0.130	+0.013
Draft forward	3.14	3.21	+0.07
Tree policy	2.00	2.15	+0.15
Verification	35.10	32.53	**−2.57**

6.5 Case Study

We also provide a real-world case study of draft trees generated by **TALON** in Appendix E.7. Additionally, we provide an in-depth runtime breakdown of the tree construction phase in Appendix E.8, verifying that the algorithmic overhead of **TALON** remains negligible even with large vocabularies.

6.6 Batched Inference and Runtime Overhead

We further evaluate **TALON** under batched decoding on the open-source inference engine SGLang [58] using Qwen3-8B. All experiments are conducted on a single NVIDIA H200 GPU with the FlashAttention-3 [42] backend. Detailed experimental settings and results are provided in Appendix E.10. Across tested batch sizes $\in \{8, 16, 32\}$, **TALON** consistently improves throughput over the static EAGLE baseline, with speedups ranging from $1.022\times$ to $1.358\times$ (achieved at $\mathrm{bs} = 32$ with a larger draft tree), demonstrating that budget-driven tree construction remains effective in batched serving scenarios. Speedup over AR decreases as the batch grows for every SD method, because the target model becomes compute-bound and verifying many draft tokens in one forward is no longer nearly free; the margin over EAGLE-3, by contrast, generally widens with concurrency. Table 2 extends the comparison to Qwen3-32B and batch size 64, with each method's budget tuned separately: **TALON** leads at every batch size, and its margin grows from 2.9% at $\mathrm{bs} = 4$ to 10.5% at $\mathrm{bs} = 64$. Since active requests share one draft tree under a joint budget (Appendix E.10), **TALON** reduces the number of tokens sent to verification. Table 3 profiles one decoding step: tree-mask construction and tree-policy logic add 0.16 ms and draft forwards 0.07 ms, an order of magnitude less than the 2.57 ms saved in target verification, which is where the end-to-end gain originates.

7 Conclusion

In this work, we presented **TALON**, a training-free framework that shifts speculative decoding from rigid geometric constraints to a flexible, budget-driven paradigm. By employing a hybrid expansion strategy that combines robust initialization with confidence gating, **TALON** dynamically shapes the draft tree—evolving into *deep-and-narrow* chains for deterministic contexts or *shallow-and-wide* branches for uncertain ones. Extensive evaluations across 5 LLMs and 6 benchmarks demonstrate that **TALON** consistently outperforms SOTA methods like EAGLE-3, achieving up to $5.16\times$ speedup while preserving the target model's output distribution losslessly. We discuss limitations and ethical considerations in Appendix G.

Acknowledgments and Disclosure of Funding

This work was in part supported by the National Natural Science Foundation of China under grant 62472399 and the Open Fund of APKL of BIIP, IAI, Hefei Comprehensive National Science Center under grant 24YGXT003. The authors declare no competing interests.

References

[1] Zachary Ankner, Rishab Parthasarathy, Aniruddha Nrusimha, Christopher Rinard, Jonathan Ragan-Kelley, and William Brandon. Hydra: Sequentially-dependent draft heads for medusa decoding, 2024. URL https://arxiv.org/abs/2402.05109.

[2] Anthropic Team. System card: Claude opus 4.5. https://assets.anthropic.com/m/64823ba7485345a7/Claude-Opus-4-5-System-Card.pdf, November 2025. System card, Anthropic.

[3] Gregor Bachmann, Sotiris Anagnostidis, Albert Pumarola, Markos Georgopoulos, Artsiom Sanakoyeu, Yuming Du, Edgar Schönfeld, Ali Thabet, and Jonas Kohler. Judge decoding: Faster speculative sampling requires going beyond model alignment, 2025. URL https://arxiv.org/abs/2501.19309.

[4] Oscar Brown, Zhengjie Wang, Andrea Do, Nikhil Mathew, and Cheng Yu. Dynamic depth decoding: Faster speculative decoding for llms, 2024. URL https://arxiv.org/abs/2409.00142.

[5] Tianle Cai, Yuhong Li, Zhengyang Geng, Hongwu Peng, Jason D. Lee, Deming Chen, and Tri Dao. Medusa: Simple llm inference acceleration framework with multiple decoding heads. In *Proceedings of the 41st International Conference on Machine Learning*, ICML'24. JMLR.org, 2024.

[6] Charlie Chen, Sebastian Borgeaud, Geoffrey Irving, Jean-Baptiste Lespiau, Laurent Sifre, and John Jumper. Accelerating large language model decoding with speculative sampling, 2023. URL https://arxiv.org/abs/2302.01318.

[7] Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, Nick Ryder, Mikhail Pavlov, Alethea Power, Lukasz Kaiser, Mohammad Bavarian, Clemens Winter, Philippe Tillet, Felipe Petroski Such, Dave Cummings, Matthias Plappert, Fotios Chantzis, Elizabeth Barnes, Ariel Herbert-Voss, William Hebgen Guss, Alex Nichol, Alex Paino, Nikolas Tezak, Jie Tang, Igor Babuschkin, Suchir Balaji, Shantanu Jain, William Saunders, Christopher Hesse, Andrew N. Carr, Jan Leike, Josh Achiam, Vedant Misra, Evan Morikawa, Alec Radford, Matthew Knight, Miles Brundage, Mira Murati, Katie Mayer, Peter Welinder, Bob McGrew, Dario Amodei, Sam McCandlish, Ilya Sutskever, and Wojciech Zaremba. Evaluating large language models trained on code, 2021. URL https://arxiv.org/abs/2107.03374.

[8] Zhuoming Chen, Avner May, Ruslan Svirschevski, Yuhsun Huang, Max Ryabinin, Zhihao Jia, and Beidi Chen. Sequoia: Scalable, robust, and hardware-aware speculative decoding, 2025. URL https://arxiv.org/abs/2402.12374.

[9] Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. Training verifiers to solve math word problems, 2021. URL https://arxiv.org/abs/2110.14168.

[10] DeepSeek-AI, Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, Xiaokang Zhang, Xingkai Yu, Yu Wu, Z. F. Wu, Zhibin Gou, Zhihong Shao, Zhuoshu Li, Ziyi Gao, Aixin Liu, Bing Xue, Bingxuan Wang, Bochao Wu, Bei Feng, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, Chong Ruan, Damai Dai, Deli Chen, Dongjie Ji, Erhang Li, Fangyun Lin, Fucong Dai, Fuli Luo, Guangbo Hao, Guanting Chen, Guowei Li, H. Zhang, Han Bao, Hanwei Xu, Haocheng Wang, Honghui Ding, Huajian Xin, Huazuo Gao, Hui Qu, Hui Li, Jianzhong Guo, Jiashi Li, Jiawei Wang, Jingchang Chen, Jingyang Yuan, Junjie Qiu, Junlong Li, J. L. Cai, Jiaqi Ni, Jian Liang, Jin Chen, Kai Dong, Kai Hu, Kaige Gao, Kang Guan, Kexin Huang, Kuai Yu, Lean Wang, Lecong Zhang, Liang Zhao, Litong Wang, Liyue Zhang, Lei Xu, Leyi Xia, Mingchuan Zhang, Minghua Zhang, Minghui Tang, Meng Li, Miaojun Wang, Mingming Li, Ning Tian, Panpan Huang, Peng Zhang, Qiancheng Wang, Qinyu Chen, Qiushi Du, Ruiqi Ge, Ruisong

Zhang, Ruizhe Pan, Runji Wang, R. J. Chen, R. L. Jin, Ruyi Chen, Shanghao Lu, Shangyan Zhou, Shanhuang Chen, Shengfeng Ye, Shiyu Wang, Shuiping Yu, Shunfeng Zhou, Shuting Pan, S. S. Li, Shuang Zhou, Shaoqing Wu, Shengfeng Ye, Tao Yun, Tian Pei, Tianyu Sun, T. Wang, Wangding Zeng, Wanjia Zhao, Wen Liu, Wenfeng Liang, Wenjun Gao, Wenqin Yu, Wentao Zhang, W. L. Xiao, Wei An, Xiaodong Liu, Xiaohan Wang, Xiaokang Chen, Xiaotao Nie, Xin Cheng, Xin Liu, Xin Xie, Xingchao Liu, Xinyu Yang, Xinyuan Li, Xuecheng Su, Xuheng Lin, X. Q. Li, Xiangyue Jin, Xiaojin Shen, Xiaosha Chen, Xiaowen Sun, Xiaoxiang Wang, Xinnan Song, Xinyi Zhou, Xianzu Wang, Xinxia Shan, Y. K. Li, Y. Q. Wang, Y. X. Wei, Yang Zhang, Yanhong Xu, Yao Li, Yao Zhao, Yaofeng Sun, Yaohui Wang, Yi Yu, Yichao Zhang, Yifan Shi, Yiliang Xiong, Ying He, Yishi Piao, Yisong Wang, Yixuan Tan, Yiyang Ma, Yiyuan Liu, Yongqiang Guo, Yuan Ou, Yuduan Wang, Yue Gong, Yuheng Zou, Yujia He, Yunfan Xiong, Yuxiang Luo, Yuxiang You, Yuxuan Liu, Yuyang Zhou, Y. X. Zhu, Yanhong Xu, Yanping Huang, Yaohui Li, Yi Zheng, Yuchen Zhu, Yunxian Ma, Ying Tang, Yukun Zha, Yuting Yan, Z. Z. Ren, Zehui Ren, Zhangli Sha, Zhe Fu, Zhean Xu, Zhenda Xie, Zhengyan Zhang, Zhewen Hao, Zhicheng Ma, Zhigang Yan, Zhiyu Wu, Zihui Gu, Zijia Zhu, Zijun Liu, Zilin Li, Ziwei Xie, Ziyang Song, Zizheng Pan, Zhen Huang, Zhipeng Xu, Zhongyu Zhang, and Zhen Zhang. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning, 2025. URL https://arxiv.org/abs/2501.12948.

[11] Ning Ding, Yulin Chen, Bokai Xu, Yujia Qin, Zhi Zheng, Shengding Hu, Zhiyuan Liu, Maosong Sun, and Bowen Zhou. Enhancing chat language models by scaling high-quality instructional conversations, 2023. URL https://arxiv.org/abs/2305.14233.

[12] Cunxiao Du, Jing Jiang, Xu Yuanchen, Jiawei Wu, Sicheng Yu, Yongqi Li, Shenggui Li, Kai Xu, Liqiang Nie, Zhaopeng Tu, and Yang You. Glide with a cape: a low-hassle method to accelerate speculative decoding. In *Proceedings of the 41st International Conference on Machine Learning*, ICML'24. JMLR.org, 2024.

[13] Yichao Fu, Peter Bailis, Ion Stoica, and Hao Zhang. Break the sequential dependency of llm inference using lookahead decoding. In *Proceedings of the 41st International Conference on Machine Learning*, ICML'24. JMLR.org, 2024.

[14] Xiangxiang Gao, Weisheng Xie, Yiwei Xiang, and Feng Ji. Falcon: faster and parallel inference of large language models through enhanced semi-autoregressive drafting and custom-designed decoding tree. In *Proceedings of the Thirty-Ninth AAAI Conference on Artificial Intelligence and Thirty-Seventh Conference on Innovative Applications of Artificial Intelligence and Fifteenth Symposium on Educational Advances in Artificial Intelligence*, AAAI'25/IAAI'25/EAAI'25. AAAI Press, 2025. ISBN 978-1-57735-897-8. doi: 10.1609/aaai.v39i22.34566. URL https://doi.org/10.1609/aaai.v39i22.34566.

[15] Fabian Gloeckle, Badr Youbi Idrissi, Baptiste Rozière, David Lopez-Paz, and Gabriel Synnaeve. Better & faster large language models via multi-token prediction, 2024. URL https://arxiv.org/abs/2404.19737.

[16] Google DeepMind Team. Gemini 3 pro model card. https://storage.googleapis.com/deepmind-media/Model-Cards/Gemini-3-Pro-Model-Card.pdf, November 2025. Model card, Google DeepMind.

[17] Zhenyu He, Zexuan Zhong, Tianle Cai, Jason Lee, and Di He. REST: Retrieval-based speculative decoding. In Kevin Duh, Helena Gomez, and Steven Bethard, editors, *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pages 1582–1595, Mexico City, Mexico, June 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.naacl-long.88. URL https://aclanthology.org/2024.naacl-long.88/.

[18] Ari Holtzman, Jan Buys, Li Du, Maxwell Forbes, and Yejin Choi. The curious case of neural text degeneration. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=rygGQyrFvH.

[19] Yuxuan Hu, Ke Wang, Xiaokang Zhang, Fanjin Zhang, Cuiping Li, Hong Chen, and Jing Zhang. Sam decoding: Speculative decoding via suffix automaton, 2024. URL https://arxiv.org/abs/2411.10666.

[20] Kaixuan Huang, Xudong Guo, and Mengdi Wang. Specdec++: Boosting speculative decoding via adaptive candidate lengths, 2025. URL https://arxiv.org/abs/2405.19715.

[21] Feiye Huo, Jianchao Tan, Kefeng Zhang, Xunliang Cai, and Shengli Sun. C2t: A classifier-based tree construction method in speculative decoding, 2025. URL https://arxiv.org/abs/2502.13652.

[22] Tom Kwiatkowski, Jennimaria Palomaki, Olivia Redfield, Michael Collins, Ankur Parikh, Chris Alberti, Danielle Epstein, Illia Polosukhin, Jacob Devlin, Kenton Lee, Kristina Toutanova, Llion Jones, Matthew Kelcey, Ming-Wei Chang, Andrew M. Dai, Jakob Uszkoreit, Quoc Le, and Slav Petrov. Natural questions: A benchmark for question answering research. *Transactions of the Association for Computational Linguistics*, 7:453–466, 08 2019. ISSN 2307-387X. doi: 10.1162/tacl_a_00276. URL https://doi.org/10.1162/tacl_a_00276.

[23] Yaniv Leviathan, Matan Kalman, and Yossi Matias. Fast inference from transformers via speculative decoding. In Andreas Krause, Emma Brunskill, Kyunghyun Cho, Barbara Engelhardt, Sivan Sabato, and Jonathan Scarlett, editors, *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pages 19274–19286. PMLR, 23–29 Jul 2023. URL https://proceedings.mlr.press/v202/leviathan23a.html.

[24] Yuhui Li, Fangyun Wei, Chao Zhang, and Hongyang Zhang. EAGLE-2: Faster inference of language models with dynamic draft trees. In Yaser Al-Onaizan, Mohit Bansal, and Yun-Nung Chen, editors, *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 7421–7432, Miami, Florida, USA, November 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.emnlp-main.422. URL https://aclanthology.org/2024.emnlp-main.422/.

[25] Yuhui Li, Fangyun Wei, Chao Zhang, and Hongyang Zhang. Eagle: Speculative sampling requires rethinking feature uncertainty, 2025. URL https://arxiv.org/abs/2401.15077.

[26] Yuhui Li, Fangyun Wei, Chao Zhang, and Hongyang Zhang. Eagle-3: Scaling up inference acceleration of large language models via training-time test, 2025. URL https://arxiv.org/abs/2503.01840.

[27] Jiahao Liu, Qifan Wang, Jingang Wang, and Xunliang Cai. Speculative decoding via early-exiting for faster llm inference with thompson sampling control mechanism, 2024. URL https://arxiv.org/abs/2406.03853.

[28] Tianyu Liu, Yun Li, Qitan Lv, Kai Liu, Jianchen Zhu, Winston Hu, and Xiao Sun. PEARL: Parallel speculative decoding with adaptive draft length. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=QOXrVMiHGK.

[29] Tianyu Liu, Qitan Lv, Hao Li, Xing Gao, and Xiao Sun. Logitspec: Accelerating retrieval-based speculative decoding via next next token speculation, 2025. URL https://arxiv.org/abs/2507.01449.

[30] Xianzhen Luo, Yixuan Wang, Qingfu Zhu, Zhiming Zhang, Xuanyu Zhang, Qing Yang, Dongliang Xu, and Wanxiang Che. Turning trash into treasure: Accelerating inference of large language models with token recycling, 2024. URL https://arxiv.org/abs/2408.08696.

[31] Jonathan Mamou, Oren Pereg, Daniel Korat, Moshe Berchansky, Nadav Timor, Moshe Wasserblat, and Roy Schwartz. Dynamic speculation lookahead accelerates speculative decoding of large language models, 2024. URL https://arxiv.org/abs/2405.04304.

[32] Xupeng Miao, Gabriele Oliaro, Zhihao Zhang, Xinhao Cheng, Zeyu Wang, Zhengxin Zhang, Rae Ying Yee Wong, Alan Zhu, Lijie Yang, Xiaoxiang Shi, Chunan Shi, Zhuoming Chen, Daiyaan Arfeen, Reyna Abhyankar, and Zhihao Jia. 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, pages 932–949, New York, NY, USA, 2024. Association for Computing Machinery. ISBN 9798400703867. doi: 10.1145/3620666.3651335. URL https://doi.org/10.1145/3620666.3651335.

[33] Nguyen Nhat Minh, Andrew Baker, Clement Neo, Allen G Roush, Andreas Kirsch, and Ravid Shwartz-Ziv. Turning up the heat: Min-p sampling for creative and coherent LLM outputs. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=FBkpCyujtS.

[34] Nishanth Sridhar Nakshatri, Shamik Roy, Rajarshi Das, Suthee Chaidaroon, Leonid Boytsov, and Rashmi Gangadharaiah. Constrained decoding with speculative lookaheads. In *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pages 4681–4700, Albuquerque, New Mexico, April 2025. Association for Computational Linguistics. doi: 10.18653/v1/2025.naacl-long.239. URL https://aclanthology.org/2025.naacl-long.239/.

[35] Ramesh Nallapati, Bowen Zhou, Cicero dos Santos, Çağlar Gulçehre, and Bing Xiang. Abstractive text summarization using sequence-to-sequence RNNs and beyond. In Stefan Riezler and Yoav Goldberg, editors, *Proceedings of the 20th SIGNLL Conference on Computational Natural Language Learning*, pages 280–290, Berlin, Germany, August 2016. Association for Computational Linguistics. doi: 10.18653/v1/K16-1028. URL https://aclanthology.org/K16-1028/.

[36] NVIDIA, Péter Vingelmann, and Frank H.P. Fitzek. Cuda, release: 10.2.89, 2020. URL https://developer.nvidia.com/cuda-toolkit.

[37] OpenAI Team. Gpt-5 system card. https://cdn.openai.com/gpt-5-system-card.pdf, August 2025. System card, OpenAI.

[38] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Köpf, Edward Yang, Zach DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library, 2019. URL https://arxiv.org/abs/1912.01703.

[39] Reiner Pope, Sholto Douglas, Aakanksha Chowdhery, Jacob Devlin, James Bradbury, Anselm Levskaya, Jonathan Heek, Kefan Xiao, Shivani Agrawal, and Jeff Dean. Efficiently scaling transformer inference, 2022. URL https://arxiv.org/abs/2211.05102.

[40] QwenTeam, An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, Chujie Zheng, Dayiheng Liu, Fan Zhou, Fei Huang, Feng Hu, Hao Ge, Haoran Wei, Huan Lin, Jialong Tang, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jing Zhou, Jingren Zhou, Junyang Lin, Kai Dang, Keqin Bao, Kexin Yang, Le Yu, Lianghao Deng, Mei Li, Mingfeng Xue, Mingze Li, Pei Zhang, Peng Wang, Qin Zhu, Rui Men, Ruize Gao, Shixuan Liu, Shuang Luo, Tianhao Li, Tianyi Tang, Wenbiao Yin, Xingzhang Ren, Xinyu Wang, Xinyu Zhang, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yinger Zhang, Yu Wan, Yuqiong Liu, Zekun Wang, Zeyu Cui, Zhenru Zhang, Zhipeng Zhou, and Zihan Qiu. Qwen3 technical report, 2025. URL https://arxiv.org/abs/2505.09388.

[41] Apoorv Saxena. Prompt lookup decoding, November 2023. URL https://github.com/apoorvumang/prompt-lookup-decoding/.

[42] Jay Shah, Ganesh Bikshandi, Ying Zhang, Vijay Thakkar, Pradeep Ramani, and Tri Dao. FlashAttention-3: Fast and accurate attention with asynchrony and low-precision, 2024. URL https://arxiv.org/abs/2407.08608.

[43] Ziteng Sun, Uri Mendlovic, Yaniv Leviathan, Asaf Aharoni, Jae Hun Ro, Ahmad Beirami, and Ananda Theertha Suresh. Block verification accelerates speculative decoding. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=frsg32u0r0.

[44] GLM Team, Aohan Zeng, Xin Lv, Qinkai Zheng, Zhenyu Hou, Bin Chen, Chengxing Xie, Cunxiang Wang, Da Yin, Hao Zeng, Jiajie Zhang, Kedong Wang, Lucen Zhong, Mingdao Liu, Rui Lu, Shulin Cao, Xiaohan Zhang, Xuancheng Huang, Yao Wei, Yean Cheng, Yifan An, Yilin Niu, Yuanhao Wen, Yushi Bai, Zhengxiao Du, Zihan Wang, Zilin Zhu, Bohan Zhang, Bosi Wen,

Bowen Wu, Bowen Xu, Can Huang, Casey Zhao, Changpeng Cai, Chao Yu, Chen Li, Chendi Ge, Chenghua Huang, Chenhui Zhang, Chenxi Xu, Chenzheng Zhu, Chuang Li, Congfeng Yin, Daoyan Lin, Dayong Yang, Dazhi Jiang, Ding Ai, Erle Zhu, Fei Wang, Gengzheng Pan, Guo Wang, Hailong Sun, Haitao Li, Haiyang Li, Haiyi Hu, Hanyu Zhang, Hao Peng, Hao Tai, Haoke Zhang, Haoran Wang, Haoyu Yang, He Liu, He Zhao, Hongwei Liu, Hongxi Yan, Huan Liu, Huilong Chen, Ji Li, Jiajing Zhao, Jiamin Ren, Jian Jiao, Jiani Zhao, Jianyang Yan, Jiaqi Wang, Jiayi Gui, Jiayue Zhao, Jie Liu, Jijie Li, Jing Li, Jing Lu, Jingsen Wang, Jingwei Yuan, Jingxuan Li, Jingzhao Du, Jinhua Du, Jinxin Liu, Junkai Zhi, Junli Gao, Ke Wang, Lekang Yang, Liang Xu, Lin Fan, Lindong Wu, Lintao Ding, Lu Wang, Man Zhang, Minghao Li, Minghuan Xu, Mingming Zhao, Mingshu Zhai, Pengfan Du, Qian Dong, Shangde Lei, Shangqing Tu, Shangtong Yang, Shaoyou Lu, Shijie Li, Shuang Li, Shuang-Li, Shuxun Yang, Sibo Yi, Tianshu Yu, Wei Tian, Weihan Wang, Wenbo Yu, Weng Lam Tam, Wenjie Liang, Wentao Liu, Xiao Wang, Xiaohan Jia, Xiaotao Gu, Xiaoying Ling, Xin Wang, Xing Fan, Xingru Pan, Xinyuan Zhang, Xinze Zhang, Xiuqing Fu, Xunkai Zhang, Yabo Xu, Yandong Wu, Yida Lu, Yidong Wang, Yilin Zhou, Yiming Pan, Ying Zhang, Yingli Wang, Yingru Li, Yinpei Su, Yipeng Geng, Yitong Zhu, Yongkun Yang, Yuhang Li, Yuhao Wu, Yujiang Li, Yunan Liu, Yunqing Wang, Yuntao Li, Yuxuan Zhang, Zezhen Liu, Zhen Yang, Zhengda Zhou, Zhongpei Qiao, Zhuoer Feng, Zhuorui Liu, Zichen Zhang, Zihan Wang, Zijun Yao, Zikang Wang, Ziqiang Liu, Ziwei Chai, Zixuan Li, Zuodong Zhao, Wenguang Chen, Jidong Zhai, Bin Xu, Minlie Huang, Hongning Wang, Juanzi Li, Yuxiao Dong, and Jie Tang. Glm-4.5: Agentic, reasoning, and coding (arc) foundation models, 2025. URL https://arxiv.org/abs/2508.06471.

[45] Llama Team. The llama 3 herd of models, 2024. URL https://arxiv.org/abs/2407.21783.

[46] Jikai Wang, Yi Su, Juntao Li, Qingrong Xia, Zi Ye, Xinyu Duan, Zhefeng Wang, and Min Zhang. Opt-tree: Speculative decoding with adaptive draft tree structure, 2024. URL https://arxiv.org/abs/2406.17276.

[47] Yepeng Weng, Dianwen Mei, Huishi Qiu, Xujie Chen, Li Liu, Jiang Tian, and Zhongchao Shi. Coral: Learning consistent representations across multi-step training with lighter speculative drafter, 2025. URL https://arxiv.org/abs/2502.16880.

[48] Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, and Alexander M. Rush. Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 38–45, Online, October 2020. Association for Computational Linguistics. URL https://www.aclweb.org/anthology/2020.emnlp-demos.6.

[49] Bin Xiao, Chunan Shi, Xiaonan Nie, Fan Yang, Xiangwei Deng, Lei Su, Weipeng Chen, and Bin Cui. Clover: Regressive lightweight speculative decoding with sequential knowledge, 2024. URL https://arxiv.org/abs/2405.00263.

[50] Yunfan Xiong, Ruoyu Zhang, Yanzeng Li, Tianhao Wu, and Lei Zou. Dyspec: Faster speculative decoding with dynamic token tree structure, 2024. URL https://arxiv.org/abs/2410.11744.

[51] Jinwei Yao, Kaiqi Chen, Kexun Zhang, Jiaxuan You, Binhang Yuan, Zeke Wang, and Tao Lin. Deft: Decoding with flash tree-attention for efficient tree-structured llm inference, 2025. URL https://arxiv.org/abs/2404.00242.

[52] Ziqian Zeng, Jiahong Yu, Qianshi Pang, Zihao Wang, Huiping Zhuang, Hongen Shao, and Xiaofeng Zou. Chimera: A lossless decoding method for accelerating large language models inference by fusing all tokens, 2024. URL https://arxiv.org/abs/2402.15758.

[53] Jun Zhang, Jue Wang, Huan Li, Lidan Shou, Ke Chen, Gang Chen, and Sharad Mehrotra. Draft & verify: Lossless large language model acceleration via self-speculative decoding. In Lun-Wei Ku, Andre Martins, and Vivek Srikumar, editors, *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 11263–11282, Bangkok, Thailand, August 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.acl-long.607. URL https://aclanthology.org/2024.acl-long.607/.

[54] Lefan Zhang, Xiaodan Wang, Yanhua Huang, and Ruiwen Xu. Learning harmonized representations for speculative sampling. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=T9u56s7mbk.

[55] Situo Zhang, Hankun Wang, Da Ma, Zichen Zhu, Lu Chen, Kunyao Lan, and Kai Yu. Adaeagle: Optimizing speculative decoding via explicit modeling of adaptive draft structures, 2024. URL https://arxiv.org/abs/2412.18910.

[56] Ziyin Zhang, Jiahao Xu, Tian Liang, Xingyu Chen, Zhiwei He, Rui Wang, and Zhaopeng Tu. Draft model knows when to stop: Self-verification speculative decoding for long-form generation, 2024. URL https://arxiv.org/abs/2411.18462.

[57] Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric Xing, Hao Zhang, Joseph E. Gonzalez, and Ion Stoica. Judging LLM-as-a-judge with MT-bench and chatbot arena. In *Thirty-seventh Conference on Neural Information Processing Systems Datasets and Benchmarks Track*, 2023. URL https://openreview.net/forum?id=uccHPGDlao.

[58] Lianmin Zheng, Liangsheng Yin, Zhiqiang Xie, Chuyue Sun, Jeff Huang, Cody Hao Yu, Shiyi Cao, Christos Kozyrakis, Ion Stoica, Joseph E. Gonzalez, Clark Barrett, and Ying Sheng. SGLang: Efficient execution of structured language model programs. In A. Globerson, L. Mackey, D. Belgrave, A. Fan, U. Paquet, J. Tomczak, and C. Zhang, editors, *Advances in Neural Information Processing Systems*, volume 37, pages 62557–62583. Curran Associates, Inc., 2024. doi: 10.52202/079017-2000. URL https://proceedings.neurips.cc/paper_files/paper/2024/file/724be4472168f31ba1c9ac630f15dec8-Paper-Conference.pdf.

Algorithm 1 Static Tree Construction (EAGLE)

Require: Draft Model $\mathcal{M}_d$, Prefix $x_{\leq t}$, Depth D, Width K, Budget N

$\mathcal{T} \leftarrow \{x_t\}$, $\mathcal{P}_0 \leftarrow \{x_t\}$, $p(x_t) \leftarrow 1.0$
Fixed depth and width for-loop
for $d = 0$ to D **do**
  $\mathcal{S}_d \leftarrow \emptyset$
  ▷ Expand to $K \times K$ child nodes
  for $v \in \mathcal{P}_d$ **do**
    $C_v \leftarrow$ Top-$K(P_{\mathcal{M}_d}(\cdot \mid x_{\leq v}))$
    $p(u) \leftarrow p(v) \cdot P_{\mathcal{M}_d}(u \mid x_{\leq v}), \forall u \in C_v$
    $\mathcal{S}_d \leftarrow \mathcal{S}_d \cup C_v$
  end for
  ▷ Shrink to K next-layer parent nodes
  $\mathcal{P}_{d+1} \leftarrow$ Top-$K(\mathcal{S}_d$ by $p(u))$
  $\mathcal{T} \leftarrow \mathcal{T} \cup \mathcal{P}_{d+1}$
end for
▷ Final pruning to budget N
$\mathcal{T} \leftarrow$ Top-$N(\mathcal{T}$ by $p(u))$
return $\mathcal{T}$

A Formalized Algorithms

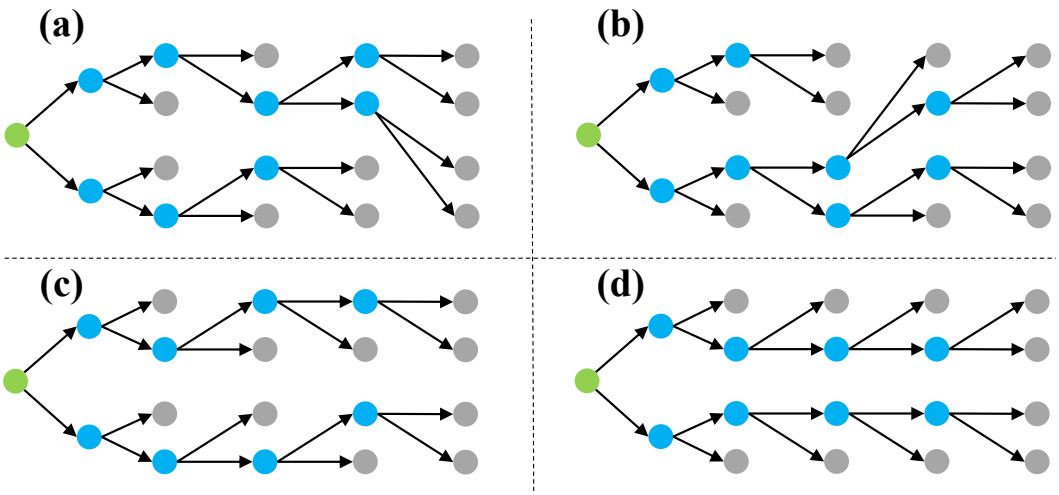


Figure 6: Examples of EAGLE-style draft token trees. At each layer, EAGLE uses a top-K operation to select K nodes as next layer parents. In this way, EAGLE can dynamically adjust the edges between last-layer parents and current-layer children. However, they cannot adjust K to adapt for contexts, which leads to significant resource wastes.

In this section, we provide the formalized algorithms for both the static baseline and our proposed framework to illustrate the structural differences in their expansion strategies.

Algorithm 1 outlines the static tree construction strategy employed by state-of-the-art methods such as EAGLE. This approach relies on rigid geometric constraints defined by a fixed depth D and width K. The construction proceeds layer-by-layer up to depth D. At each step d, the draft model expands $K \times K$ child nodes from the parent set $\mathcal{P}_d$ and subsequently applies a Top-K operation to shrink the candidates back to size K for the next layer. Finally, the tree is pruned globally to meet the token

Algorithm 2 Adaptive Tree Construction (**TALON**)

Require: Draft Model $\mathcal{M}_d$, Prefix $x_{\leq t}$, Budget N, Width K, Threshold μ

$\mathcal{T} \leftarrow \{x_t\}$, $\mathcal{P}_0 \leftarrow \{x_t\}$, $p(x_t) \leftarrow 1.0$
$d \leftarrow 0$
▷ Budget-driven adaptive tree expansion
while $|\mathcal{T}| < N$ **do**
  $\mathcal{S}_d \leftarrow \emptyset$
  ▷ Gather next-layer candidate set $\mathcal{S}_d$
  for $v \in \mathcal{P}_d$ **do**
    $C_v \leftarrow \{(v, w) \mid w \in \mathcal{V}\}$
    $p(u) \leftarrow p(v) \cdot P_{\mathcal{M}_d}(w \mid x_{\leq v}), \forall u = (v, w) \in C_v$
    $\mathcal{S}_d \leftarrow \mathcal{S}_d \cup C_v$
  end for
  ▷ Hybrid Expansion Strategy
  if $d = 0$ **then**
    ▷ Robust Init
    $\mathcal{P}_{d+1} \leftarrow$ Top-$K(\mathcal{S}_d$ by $p(u))$
  else
    ▷ Confidence Gating
    $m_d \leftarrow \max_{u \in \mathcal{S}_d} p(u)$
    $\mathcal{P}_{d+1} \leftarrow \{u \in \mathcal{S}_d \mid p(u) \geq \mu \cdot m_d\}$
  end if
  ▷ Budget Check
  if $|\mathcal{T}| + |\mathcal{P}_{d+1}| > N$ **then**
    $\mathcal{P}_{d+1} \leftarrow$ Top-$(N - |\mathcal{T}|)(\mathcal{P}_{d+1}$ by $p(u))$
  end if
  $\mathcal{T} \leftarrow \mathcal{T} \cup \mathcal{P}_{d+1}$
  $d \leftarrow d + 1$
end while
return $\mathcal{T}$

budget N. This "expand-then-shrink" mechanism enforces a static topology regardless of generation difficulty, often generating redundant nodes that are discarded during intermediate steps. Note that "static" here means that each layer of the draft tree has a fixed number of nodes regardless of the context difficulty. The "dynamic" claimed in [24] means that the draft model dynamically select top-K nodes as next-layer parents, but it cannot adjust K to adapt for context. Figure 6 shows 4 different EAGLE-style draft trees with static tree topology.

Algorithm 2 presents the details of **TALON**, our proposed budget-driven adaptive framework. Unlike static methods, **TALON** is constrained only by a global node budget N and constructs the tree iteratively until this capacity is filled. The core of Algorithm 2 is the Hybrid Expansion Strategy, which dynamically selects the expansion logic based on the current depth. For the root layer ($d = 0$), **TALON** employs a Robust Tree Initialization via a standard Top-K expansion to mitigate potential mis-calibration in the draft model's initial prediction. For all subsequent layers ($d \geq 1$), the algorithm switches to a Confidence-Gated Expansion mechanism. It calculates an anchor confidence m_d within the candidate pool and filters nodes using a relative threshold μ (i.e., $p(u) \geq \mu \cdot m_d$). This "prune-while-expanding" approach allows the tree topology to adapt between "deep-and-narrow" for deterministic contexts and "shallow-and-wide" for uncertain ones.

B Token-Tree Verification

The verification phase of our framework adheres to the standard paradigm of tree-based speculative decoding, widely adopted in the field [25, 32]. By utilizing the Tree Attention mechanism, the target LLM verifies the entire draft tree in a single forward pass. A structured attention mask ensures that each token within the tree attends only to its predecessors along the root-to-leaf path, effectively simulating independent parallel verifications for all candidate branches while maintaining causal consistency.

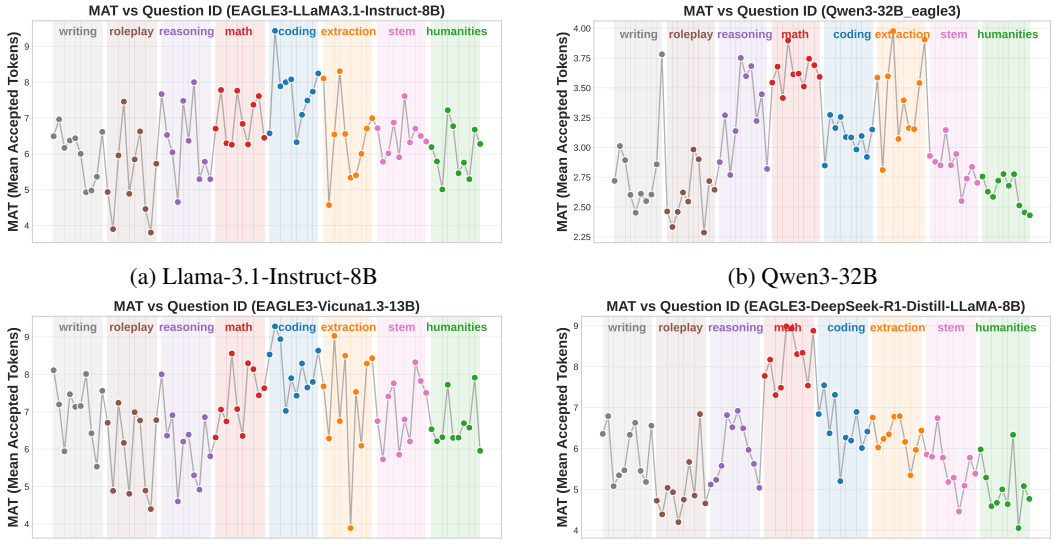
Following the methodology established in Li et al. [25], the verification proceeds in three key steps: (1) calculating the posterior probability of each token in the tree given the target model's output; (2) selecting the optimal valid prefix that satisfies the acceptance criteria; and (3) resampling a correction token from the residual distribution at the point of divergence. This rigorous process guarantees that the final output distribution is mathematically identical to that of the target model. We refer readers to the original EAGLE paper [25] for detailed algorithms and proofs.

TALON changes only how the draft tree is *built* and reuses this verification procedure unchanged. Since the draft tree defines only the proposal, the guarantee above holds regardless of the tree's shape. As an empirical sanity check, Table 4 reports downstream task quality under greedy decoding and stochastic sampling. At $T = 0$, AR decoding, EAGLE-3, and **TALON** agree exactly on both datasets, as losslessness requires; under sampling, the remaining differences are within run-to-run variation and show no systematic degradation for **TALON**.

Table 4: Downstream task quality across temperatures (5 runs per cell).

Dataset	Metric	T	AR	EAGLE-3	TALON
GSM8K	Acc	0	72.50	72.50	72.50
		0.6	75.25	75.00	75.00
		1.0	68.50	68.25	68.75
HumanEval	pass@1	0	71.25	71.25	71.25
		0.6	66.50	66.50	66.50
		1.0	60.50	61.00	60.00

C Additional Motivated Experiments



(a) Llama-3.1-Instruct-8B

(b) Qwen3-32B

(c) Vicuna-13B

(d) DeepSeek-R1-Distill-LLaMA-8B

Figure 7: Real-World Mean Accepted Tokens (MAT) distribution across different queries with EAGLE baseline. We visualize the MAT fluctuations on (a) Llama-3.1-8B, (b) Qwen3-32B, (c) Vicuna-13B, and (d) DeepSeek-R1-Distill-8B. Across varying parameter scales and architectures, we consistently observe significant volatility in generation difficulty: low-entropy tasks (e.g., Math/Coding) often allow for high acceptance, while high-entropy tasks (e.g., Roleplay) necessitate shallow trees. This universal variance highlights the limitation of static fixed-depth policies and underscores the necessity of **TALON**'s adaptive strategy.

To demonstrate the generality of the limitations observed in Section 3, specifically the "Acceptance Funnel" phenomenon and the "Static Depth Dilemma," we conduct additional empirical analyses on

a broader range of LLMs. These experiments confirm that the inefficiencies of static tree structures are not model-specific artifacts but inherent challenges in speculative decoding.

C.1 Heat Map Visualization of LlaMA-3.1-Instruct-8B

In Section 4, we utilized Qwen3-8B to illustrate the funnel-like distribution of accepted tokens. To verify whether this pattern holds across different architectures, we perform the same visualization on Llama-3.1-8B-Instruct. As shown in Figure 8, the results exhibit a striking similarity to our previous findings. The acceptance probability in the initial layer is relatively dispersed, necessitating a robust search width to capture correct continuations. However, as the tree deepens, the acceptance mass concentrates sharply on the high-confidence regions. This consistent "Acceptance Funnel" across models further justifies **TALON**'s hybrid expansion strategy: enforcing robustness at the root while employing confidence-gated pruning at deeper layers to eliminate the redundancy of static width.

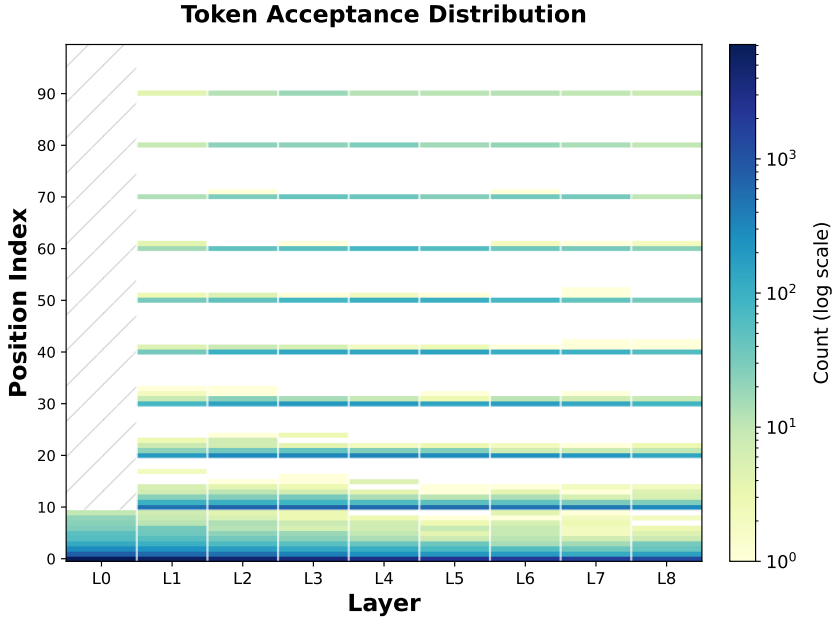


Figure 8: Visualization of token acceptance frequency with Llama-3-8B using a static draft tree ($K = 10, D = 8$). Consistent with Figure 3, the heatmap demonstrates the "Acceptance Funnel" phenomenon: acceptance is dispersed in the first layer but rapidly concentrates on top-ranked candidates in deeper layers, rendering wide static expansion inefficient.

C.2 Dynamic Accepted Length of Various Models

We further extend the analysis of the "Static Depth Dilemma" to evaluate how generation difficulty fluctuates across different models and tasks. Figure 7 presents the Mean Accepted Tokens (MAT) distribution for Llama-3.1-8B-Instruct, Qwen3-32B-Instruct, Vicuna-13B, and DeepSeek-R1-Distill-LLaMA-8B, respectively.

Across all evaluated models and diverse task categories, we observe significant volatility in the effective speculation length. In deterministic, logic-constrained tasks such as **Math and Coding**, the MAT frequently reaches high peaks. In these low-entropy scenarios, the draft model is often confident and accurate, yet a pre-defined static depth limit artificially caps the potential speculation length, preventing the system from fully exploiting the easy context for maximum speedup. Conversely, in open-ended or high-entropy tasks like **Roleplay or Writing**, the acceptance length often drops significantly due to higher uncertainty. Here, a rigid static tree forces the draft model to hallucinate deep branches that are destined for rejection, resulting in wasted computation. This universal variance underscores that a "one-size-fits-all" static tree structure is fundamentally suboptimal, highlighting the necessity of **TALON**'s budget-driven adaptive mechanism that dynamically adjusts tree depth based on real-time confidence.

D Derivation of Draft Efficiency

In this section, we provide the detailed derivation of the wall-time speedup formula presented in Section 5.3. We define the wall-clock time for a single forward pass of the draft model and the target model as T_q and T_p, respectively. Let L denote the total number of tokens in the final generated output sequence. Throughout the generation process, the draft model executes a total of N_q forward passes, while the target model executes N_p verification steps.

In standard Auto-Regressive (AR) decoding, the target model generates tokens sequentially, resulting in a total inference latency:

$$T_{AR} = L \cdot T_p, \tag{8}$$

In contrast, Speculative Decoding (SD) decouples the process into drafting and verification phases. The total latency is the sum of time spent on both phases, expressed as:

$$T_{SD} = N_p \cdot T_p + N_q \cdot T_q. \tag{9}$$

The wall-time speedup R is strictly defined as the ratio of the baseline latency to the speculative latency:

$$R = \frac{T_{AR}}{T_{SD}} = \frac{L \cdot T_p}{N_p \cdot T_p + N_q \cdot T_q}. \tag{10}$$

To relate this speedup to algorithmic efficiency, we simplify the fraction by dividing both the numerator and the denominator by $N_p \cdot T_p$. We introduce three key metrics: (1) the relative model cost c:

$$c = \frac{T_q}{T_p}, \tag{11}$$

(2) the Mean Accepted Tokens τ:

$$\tau = \frac{L}{N_p}, \tag{12}$$

and (3) the Draft Efficiency δ

$$\delta = \frac{N_q}{N_p}, \tag{13}$$

which quantifies the computation invested in drafting per verification step. Substituting these variables into Equation 10 yields:

$$R = \frac{\frac{L}{N_p}}{1 + \frac{N_q}{N_p} \cdot \frac{T_q}{T_p}} = \frac{\tau}{1 + c \cdot \delta}. \tag{14}$$

This relationship highlights the advantage of **TALON**. Static methods enforce a fixed depth D, locking draft efficiency to a constant $\delta = D + 1$. Consequently, when the acceptance length τ drops in difficult contexts, the speedup R degrades significantly. In contrast, **TALON** dynamically adjusts the tree size, reducing δ in uncertain scenarios. By ensuring the draft cost δ scales down alongside τ, **TALON** preserves a robust speedup R across varying generation difficulties.

To further illustrate our claim, let τ_1, δ_1denote EAGLE's acceptance length and draft efficiency and τ_2, δ_2denote **TALON**'s, $c$denote the speed ratio between the target model and the draft model, Ddenote the tree depth in EAGLE generation.

In **low-entropy contexts**, **TALON** extends deeper and we have $\tau_2 > \tau_1$. Although $\delta_2 > \delta_1$, since $c \ll 1$, the factor $1 + c\delta$grows slow, yielding improvement. For example, $\tau_2 = 10, \tau_1 = 8, c = 0.05, \delta_2 = 10, \delta_1 = 8$, $R_{EAGLE} = 5.71$, $R_{TALON} = 6.67$.

In **high-entropy contexts**, we have $\tau_2 \approx \tau_1$, but **TALON** stops draft generation early, so $\delta_2 < \delta_1 = D + 1$, yielding improvement. For example, $\tau_2 = 2, \tau_1 = 2, c = 0.05, \delta_2 = 2, \delta_1 = 8$, $R_{EAGLE} = 1.43$, $R_{TALON} = 1.82$. (This is especially notable for Qwen models.)

Figures 4 and 9 in the original manuscript empirically confirm this: **TALON**'s (τ, δ) trajectory tracks the Oracle line, ensuring both factors remain favorable across varying context difficulties.

Target Model	Method	Draft Model Checkpoint (Hugging Face)
Vicuna-1.3-13B	Medusa	`FasterDecoding/medusa-vicuna-13b-v1.3`
	Hydra	`ankner/hydra-vicuna-13b-v1.3`
	SD (Standard)	`double7/vicuna-68m`
	OPT-Tree / EAGLE-3 / **TALON**	`yuhuili/EAGLE3-Vicuna1.3-13B`
DSL-8B	EAGLE-3 / **TALON**	`yuhuili/EAGLE3-DeepSeek-R1-Distill-LLaMA-8B`
Llama-3.1-8B-Instruct	EAGLE-3 / **TALON**	`yuhuili/EAGLE3-LLaMA3.1-Instruct-8B`
Qwen3-8B	EAGLE-3 / **TALON**	`AngelSlim/Qwen3-8B_eagle3`
Qwen3-32B	EAGLE-3 / **TALON**	`AngelSlim/Qwen3-32B_eagle3`

Table 5: List of draft model checkpoints used in our experiments. **TALON** shares the identical draft weights with EAGLE-3 across all tested benchmarks to ensure a fair evaluation of the tree topology efficiency.

E Details of Experiments

E.1 Implementation Details

E.1.1 Comparison Baselines

To strictly evaluate the effectiveness of **TALON**, we compare it against a comprehensive set of competitive baselines, categorizing them into chain-based, tree-based and other MLP-based speculative decoding approaches. We first include standard **Speculative Decoding (SD)** [23, 6] as the fundamental chain-based baseline to measure the raw speedup gain over auto-regressive decoding. For MLP-based methods, which aim to reuse target hidden states with MLP to predict multiple draft tokens, we compare against MEDUSA [5] and HYDRA [1]. These methods employ lightweight MLP decoding heads to predict subsequent tokens in parallel, representing a distinct direction in efficiently drafting.

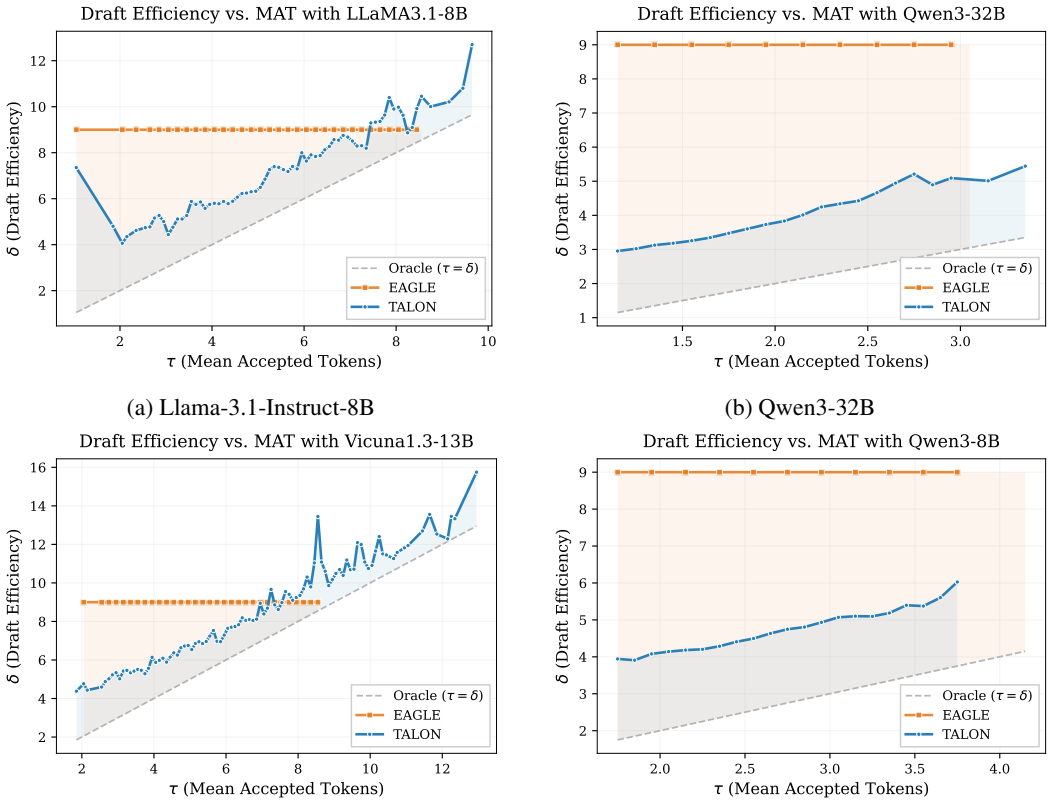
In the realm of tree-based speculative decoding, we select EAGLE-3 [26] as the primary state-of-the-art baseline. EAGLE-3 utilizes feature-level auto-regression with a multi-layer fusion mechanism and typically constructs a static draft tree with fixed width and depth. Comparing against EAGLE-3 allows us to directly demonstrate the advantages of our adaptive topology over rigid geometric constraints.

Most importantly, we include **OPT-Tree** [46] as a key baseline, as it represents the related work most closely aligned with **TALON**. Similar to our approach, OPT-Tree aims to optimize the draft tree topology. However, a critical distinction lies in the construction paradigm: OPT-Tree caps the tree depth and expands every layer to the full node budget before pruning ("generate-then-prune"), so each of its draft forwards carries the whole tree (e.g., 60 tokens for $N = 60$). In contrast, **TALON** adopts a training-free, budget-driven expansion strategy that dynamically shapes the tree on the fly based on real-time confidence: each draft forward carries only the nodes that survived gating in the previous layer, and the depth is bounded only by the budget (Appendix F).

We did not include Sequoia because (a) its official implementation is incompatible with the EAGLE draft head. Directly comparing Sequoia with our **TALON** is unfair as the draft model is different; and (b) Sequoia focuses on hardware-aware tree optimization (finding a globally optimal tree shape for specific hardware), whereas **TALON** focuses on context-aware adaptation (adjusting to token difficulty). These approaches operate at different levels and are theoretically orthogonal. We did not include DySpec because its code was not publicly released at the time of submission.

E.1.2 Hardware and Software Configurations

Hardware Environment. All experiments are conducted on a server equipped with a single **NVIDIA H200 (141GB) GPU** [36], except for the cross-platform evaluation on NVIDIA H20 in Appendix E.5. We perform all evaluations with a batch size of 1 to follow the standard settings and simulate real-world latency-critical inference scenarios, except for the batched-inference experiments (Appendix E.10) and the per-stage profile (Appendix E.8).



(a) Llama-3.1-Instruct-8B

(b) Qwen3-32B

(c) Vicuna-13B

(d) Qwen3-8B

Figure 9: **Detailed visualization of Draft Efficiency (δ) versus Mean Accepted Tokens (τ) across four different LLMs.** The x-axis represents the mean accepted length, while the y-axis represents the computational cost (draft steps per verification). The gray dashed line denotes the optimal Oracle baseline ($\tau = \delta$) where no computation is wasted. While the static EAGLE-3 baseline (orange) maintains a fixed high cost regardless of difficulty, TALON (blue) dynamically adjusts its draft budget, closely tracking the Oracle curve and significantly reducing computational waste in high-entropy (low τ) scenarios and increasing acceptance reward in low-entropy (high τ) scenarios.

Software Environment. Our implementation is based on `PyTorch` [38] version `2.6.0+cu124` and `Transformers` [48] version `4.57.1`. The code is compiled with CUDA `12.8`. All experiments utilize standard KV-caching optimizations; our implementation strictly inherits the KV-cache management, tree attention, and verification pipeline from the official EAGLE-3 codebase, ensuring that reported speedups reflect gains on top of fully optimized baselines. For SD (Vanilla Speculative Decoding), we use Transformers' official assisted decoding with their default settings. For MEDUSA and HYDRA, we use their official codebases and adhere to their recommended environment settings to ensure fair comparison. For OPT-Tree, as its official codebase only supports EAGLE-2, we re-implement OPT-Tree to use official EAGLE-3 draft model checkpoint.

Generation Configuration. Unless otherwise specified, we employ greedy decoding (Temperature $T = 0$) for the main speedup benchmarks reported in Table 1. For the robustness experiments under stochastic sampling (Table 7), we set the temperature to $T = 1.0$ (No additional top-k or top-p operation). The maximum generation length is set to `1024` tokens for standard benchmarks. Regarding **TALON**'s hyper-parameters, we set the global token budget $N = 60$ and the confidence threshold $\mu = 0.03$ by default across all models, $K = 10$ for robust tree initialization. For EAGLE-3, we set the tree width $K = 10$ and $D = 8$, which is reported as optimal values in their manuscripts.

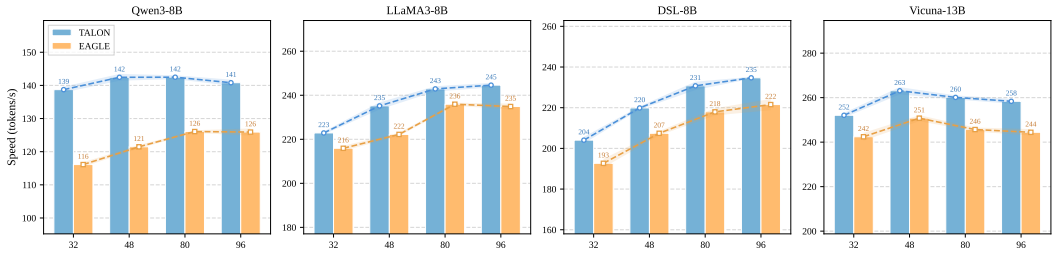


Figure 10: **Ablation study on the global token budget (N).** We compare the wall-time generation speed of **TALON** against the EAGLE-3 baseline across varying budget constraints ranging from $N = 32$ to $N = 96$. The results demonstrate that **TALON** consistently outperforms the static baseline across all budget levels. The performance gap is particularly pronounced in resource-constrained settings (e.g., $N = 32$), confirming that **TALON**'s confidence-gated expansion strategy is significantly more efficient at prioritizing high-value draft tokens than the rigid "expand-then-shrink" mechanism of static methods.

E.1.3 Draft Model Selection

To ensure full reproducibility and a strictly fair comparison, we list the exact Hugging Face model checkpoints used for all draft models in our experiments. Crucially, for TALON, we utilize the *exact same* draft model checkpoints as the state-of-the-art baseline EAGLE-3 (and OPT-Tree where applicable). This ensures that any observed performance gains are attributed solely to our adaptive tree expansion algorithm rather than superior draft weights. The detailed configurations are provided in Table 5.

E.2 Run-to-Run Variation of the Main Results

The speedups of EAGLE-3 and **TALON** in Table 1 are means over 5 runs. Table 6 reports them with standard deviations. Every gain of **TALON** over EAGLE-3 holds under this run-to-run variation.

Table 6: Wall-time speedup over auto-regressive decoding in Table 1, reported as mean ± std over 5 runs.

Model	Method	Alpaca	GSM8K	HumanEval	MT-Bench	QA	CNN/DM
Vicuna-13B	EAGLE-3	3.78±0.02	3.87±0.03	4.77±0.07	4.05±0.05	3.03±0.04	3.43±0.02
	TALON	**3.94±0.01**	**3.99±0.02**	**5.16±0.05**	**4.27±0.01**	**3.26±0.03**	**3.58±0.02**
DSL-8B	EAGLE-3	3.23±0.03	4.24±0.03	3.85±0.04	3.33±0.02	2.86±0.03	2.89±0.05
	TALON	**3.46±0.03**	**4.43±0.02**	**3.96±0.02**	**3.56±0.03**	**3.12±0.04**	**3.14±0.05**
Llama3-8B	EAGLE-3	3.92±0.05	3.63±0.03	4.05±0.02	3.67±0.05	3.01±0.02	3.06±0.02
	TALON	**4.04±0.03**	**3.81±0.02**	**4.20±0.04**	**3.85±0.05**	**3.27±0.02**	**3.30±0.04**
Qwen3-8B	EAGLE-3	2.08±0.02	2.37±0.01	2.35±0.03	2.20±0.02	2.09±0.02	1.95±0.01
	TALON	**2.44±0.01**	**2.67±0.02**	**2.69±0.04**	**2.57±0.02**	**2.46±0.04**	**2.30±0.01**
Qwen3-32B	EAGLE-3	1.83±0.01	2.22±0.02	1.96±0.03	1.90±0.01	1.78±0.01	1.57±0.02
	TALON	**2.05±0.02**	**2.38±0.03**	**2.15±0.02**	**2.10±0.02**	**1.99±0.02**	**1.72±0.02**

E.3 More Visualization Results of Draft Efficiency

To comprehensively validate the theoretical efficiency analysis presented in Section 5.3, we provide extended visualizations of the relationship between Draft Efficiency (δ) and Mean Accepted Tokens (τ) across varying model architectures. Figure 9 illustrates this correlation for Llama-3.1-8B-Instruct, Qwen3-32B, Vicuna-13B, and Qwen3-8B. In these plots, the x-axis represents the acceptance length (how many tokens are actually accepted within each decoding step), while the y-axis represents the computational overhead draft efficiency (the ratio of draft steps to verification steps). The gray dashed line serves as the Oracle baseline ($\tau = \delta$), representing an ideal zero-waste scenario where every drafted token is accepted.

As clearly demonstrated across all four sub-figures, the static baseline (EAGLE-3) exhibits a rigid, horizontal trajectory. This pattern confirms that static tree-based methods incur a constant computational overhead determined solely by their fixed geometric hyperparameters (width K and depth D), regardless of the actual generation difficulty. Consequently, in high-entropy scenarios where the acceptance length τ is low, static methods suffer from a substantial "efficiency gap"—highlighted by the extensive orange shaded regions—indicating that the system is squandering computational resources on generating draft branches that are destined to be rejected.

In sharp contrast, TALON demonstrates a highly adaptive behavior, with its efficiency curve strictly tracking the Oracle baseline across the entire spectrum of generation difficulties. The upward-sloping blue trajectory indicates that TALON dynamically modulates its resource investment: it autonomously reduces the draft budget in uncertain contexts to minimize waste, while scaling up the tree size in deterministic contexts to maximize the acceptance length. This tight alignment between the draft cost (δ) and the acceptance reward (τ) empirically verifies that our confidence-gated expansion strategy effectively decouples the draft structure from rigid constraints, ensuring that computational resources are allocated only when they yield a positive marginal utility. (Notably, the curve of Qwen3-8B and Qwen3-32B is relatively far from oracle line, suggesting that there is still a room to increase **TALON**'s end-to-end speedup. We will show in following sections that decreasing **TALON**'s threshold from default `0.03` to smaller threshold `0.01` yields better performance.)

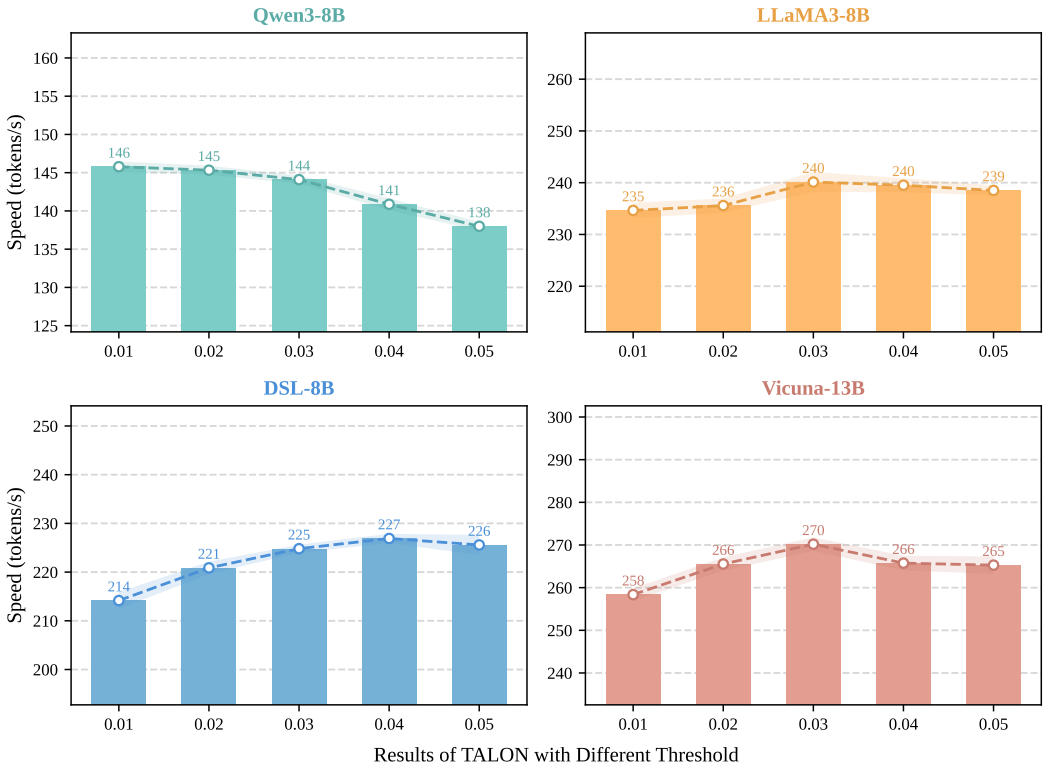


Figure 11: **Sensitivity analysis of the confidence threshold μ.** This parameter controls the trade-off between exploration width and generation depth. The results indicate that the optimal μ is positively correlated with the model's draft-target alignment (MAT). For models with lower alignment like Qwen3-8B (a), a lower threshold ($\mu = 0.01$) yields the best performance by encouraging a "shallow-and-wide" search to ensure coverage. In contrast, for highly aligned models like DSL-8B (c), a higher threshold ($\mu = 0.04$) is preferred to form "deep-and-narrow" chains that maximize the speculation length.

E.4 More Evaluation Results under Temperature Settings

While the primary evaluations in the main text focus on greedy decoding (Temperature $T = 0$), real-world LLM applications—particularly creative writing and open-ended chat—frequently rely on

Table 7: **Main Results on Six Benchmarks (Temperature=1).** Comparison between EAGLE-3 and our proposed **TALON** across various models. We report Mean Acceptance Tokens (MAT) and Wall-time Speedup relative to standard decoding. **Bold** numbers denote the best speedup performance.

Model	Method	Alpaca		GSM8K		HumanEval		MT-Bench		QA		CNN/DM	
		MAT	Spd.	MAT	Spd.	MAT	Spd.	MAT	Spd.	MAT	Spd.	MAT	Spd.
Vicuna-13B	EAGLE-3	5.61	3.15×	5.95	3.29×	6.77	3.78×	5.79	3.21×	4.72	2.64×	6.01	2.96×
	SD	1.84	1.11×	1.84	1.08×	2.27	1.29×	2.04	1.13×	1.71	1.05×	2.21	1.23×
	MEDUSA	2.83	2.19×	2.82	2.20×	2.97	2.35×	2.84	2.22×	2.49	1.92×	2.26	1.69×
	HYDRA	4.23	2.86×	3.98	2.73×	4.18	2.91×	4.06	2.77×	3.31	2.25×	3.15	2.07×
	OPT-Tree	5.56	3.21×	5.97	3.25×	6.47	3.53×	5.69	3.16×	4.86	2.77×	5.84	2.79×
	TALON	5.43	**3.28**×	5.82	**3.35**×	7.25	**3.97**×	5.88	**3.38**×	4.63	**2.82**×	5.95	**2.96**×
DSL-8B	EAGLE-3	4.48	2.49×	6.43	3.58×	5.47	3.04×	4.56	2.55×	4.04	2.25×	4.32	2.39×
	TALON	4.29	**2.75**×	6.44	**3.70**×	5.30	**3.25**×	4.44	**2.81**×	3.91	**2.54**×	4.17	**2.63**×
Llama3-8B	EAGLE-3	5.19	2.85×	4.59	2.51×	6.24	3.46×	3.96	2.20×	3.12	1.72×	4.44	2.42×
	TALON	5.21	**3.15**×	4.74	**2.84**×	6.40	**3.56**×	4.13	**2.58**×	3.20	**2.08**×	4.33	**2.62**×
Qwen3-8B	EAGLE-3	3.32	1.95×	3.81	2.24×	3.77	2.20×	3.46	2.02×	3.17	1.87×	3.17	1.85×
	TALON	3.28	**2.25**×	3.74	**2.50**×	3.70	**2.51**×	3.41	**2.32**×	3.15	**2.17**×	3.11	**2.11**×
Qwen3-32B	EAGLE-3	2.55	1.67×	3.22	2.10×	2.95	1.89×	2.74	1.75×	2.44	1.61×	2.46	1.51×
	TALON	2.53	**1.84**×	3.17	**2.27**×	2.91	**2.07**×	2.71	**1.92**×	2.41	**1.77**×	2.43	**1.64**×

stochastic sampling to induce diversity in the generated content. To rigorously assess the robustness of our framework in these non-deterministic scenarios, we conduct a comprehensive evaluation using standard sampling with Temperature $T = 1$. The detailed comparative results between **TALON** and the state-of-the-art baseline EAGLE-3 are reported in Table 7.

As evidenced by the quantitative results, **TALON** consistently outperforms EAGLE-3 across all evaluated models and benchmarks, demonstrating superior adaptability to stochastic environments. A fundamental challenge in high-temperature settings is that the target model's probability distribution becomes flatter, reducing the dominance of the top-1 token and increasing the likelihood of selecting lower-ranked candidates. Static approaches like EAGLE-3, which enforce a fixed width and depth, often struggle in this regime because they rigidly expand the top-K candidates regardless of the entropy. This leads to a scenario where the draft tree either misses the sampled token due to insufficient coverage in uncertain branches or wastes computation on high-confidence paths that are not selected.

In contrast, **TALON**'s confidence-gated expansion strategy naturally excels under these conditions. By determining the tree topology based on relative probability thresholds rather than a fixed node count, TALON dynamically widens the search space in high-entropy layers to capture the dispersed probability mass, while keeping the tree narrow in clearer contexts. This flexibility is reflected in the substantial speedup gains; for instance, on the Vicuna-13B model for HumanEval, **TALON** achieves a wall-time speedup of 3.97×, significantly surpassing the 3.78× speedup of EAGLE-3. Similarly, on Qwen3-8B, **TALON** maintains a distinct advantage in both reasoning-intensive tasks like GSM8K and open-ended tasks like Alpaca. These findings confirm that **TALON**'s budget-driven mechanism is not limited to deterministic acceleration but is a robust solution for diverse generation settings, effectively mitigating the efficiency degradation often observed in stochastic speculative decoding.

E.5 Results on a Second Hardware Platform

Since part of **TALON**'s gain comes from spending fewer draft steps, its advantage depends on the relative cost $c = T_q/T_p$ of the draft and target models (Equation 7), which in turn depends on the hardware. We therefore repeat the evaluation on NVIDIA H20, whose compute-to-bandwidth ratio differs markedly from that of the H200. The setting is Qwen3-8B, greedy decoding ($T = 0$), FP16, batch size 1, $N = 60$, $\mu = 0.03$, and $K = 10$ for **TALON**; $K = 10$ and $D = 8$ for EAGLE-3; 1024 maximum new tokens; and 80 questions per benchmark. Table 8 shows that **TALON** improves the speedup over EAGLE-3 on all six benchmarks. Estimating c from the measured AR, EAGLE-3, and **TALON** throughputs gives $c \approx 0.052$ on H200 and $c \approx 0.032$ on H20, and **TALON**'s margin stays positive on both platforms. Since τ and δ are algorithmic quantities and match their H200

Draft token (accepted) Target token (rejected & resampled)

EAGLE-3 (Tokens: 197 | Acc: 88.8%)	TALON (Tokens: 197 | Acc: **92.4%**)

<s> Here ' s the completed code :
“‘ python
def next _ small est (lst):
  """
  You are given a list of integers .
  Write a function next _ small est () that ...
  Return None if there is no such element .
 	
  next _ small est ([1 , 2 , 3 , 4 , 5]) == 2
	<s> Here ' s the completed code :
	“‘ python
	def next _ small est (lst):
	  """
	  You are given a list of integers .
	  Write a function next _ small est () that ...
	  Return None if there is no such element .
	
	  next _ small est ([1 , 2 , 3 , 4 , 5]) == 2

Figure 12: Comparison of token generation traces. **Left:** EAGLE-3 shows frequent interruptions (pink) due to verification failures. **Right:** **TALON** maintains longer accepted chains (cyan) by utilizing confidence-aware adaptive trees.

values within 2%, the shift in speedup is attributable to the hardware rather than to any change in tree behavior.

Table 8: Results on NVIDIA H20 (Qwen3-8B, $T = 0$). We report mean accepted tokens (τ), draft efficiency (δ), and wall-time speedup over auto-regressive decoding.

Method	Alpaca			GSM8K			HumanEval			MT-Bench			QA			CNN/DM		
	τ	δ	Spd.	τ	δ	Spd.	τ	δ	Spd.	τ	δ	Spd.	τ	δ	Spd.	τ	δ	Spd.
EAGLE-3	3.46	9.00	2.24×	3.94	9.00	2.50×	3.91	9.00	2.42×	3.69	9.00	2.36×	3.47	9.00	2.19×	3.27	9.00	1.97×
TALON	3.41	**4.40**	**2.50**×	3.88	**5.09**	**2.68**×	3.83	**4.75**	**2.65**×	3.62	**4.62**	**2.77**×	3.42	**4.42**	**2.55**×	3.20	**4.46**	**2.32**×

E.6 More Ablation Studies of TALON

In this section, we conduct in-depth ablation studies to evaluate the sensitivity and robustness of **TALON** with respect to its two key hyper-parameters: the global token budget (N) and the confidence threshold (μ).

Impact of Global Token Budget (N). We first investigate how the inference performance scales with the available computational budget. Figure 10 compares the wall-time speedup of **TALON** against the EAGLE-3 baseline across varying node budgets ranging from 32 to 96. For a fair comparison, we apply the same global budget constraint to EAGLE-3 using its standard pruning mechanism. The results demonstrate that **TALON** consistently outperforms the static baseline across all budget levels. Notably, in resource-constrained scenarios (e.g., $N = 32$), TALON exhibits a significant advantage. This indicates that our confidence-gated expansion is highly efficient at prioritizing the most promising candidate tokens, ensuring that a limited budget is invested in high-probability paths rather than being diluted by a fixed-width expansion. As the budget increases to 96, **TALON** continues to scale effectively, utilizing the additional capacity to extend generation depth in deterministic regions, whereas static methods often hit a performance plateau due to their rigid structural constraints.

Sensitivity to Confidence Threshold (μ). Next, we analyze the influence of the confidence threshold μ on generation speed, as visualized in Figure 11. The threshold μ acts as a gatekeeper that balances the trade-off between exploration width and generation depth. Interestingly, we observe that the optimal μ is positively correlated with the degree of alignment between the draft and target models, which can be approximated by the Mean Accepted Tokens (MAT).

For models exhibiting high alignment and high MAT, such as DeepSeek-R1-Distill-LLaMA-8B (DSL-8B), the optimal threshold tends to be relatively high (peaking around $\mu = 0.04$). In these well-aligned scenarios, the draft model's confidence is a reliable proxy for correctness; thus, a stricter threshold effectively filters out unlikely branches, shaping the tree into a "deep-and-narrow" structure that maximizes draft length. Conversely, when the draft-target alignment is weaker (resulting in lower MAT), a lower threshold becomes more advantageous. This phenomenon explains the behavior of Qwen3-8B in Figure 11, where the performance peaks at a lower threshold of $\mu = 0.01$. Here, the

draft model is less certain, necessitating a more lenient threshold to encourage "shallow-and-wide" exploration, thereby ensuring sufficient coverage of the target distribution to prevent early rejection.

To quantify what a mis-set μ costs, we repeat the sweep five times on MT-Bench and add the EAGLE-3 reference (Table 9). With the default $\mu = 0.03$, **TALON** outperforms EAGLE-3 by 4.8% to 17.1% on every model. The gap between the default and the per-model optimum is at most 1.2%, smaller than **TALON**'s margin over EAGLE-3 in every row, so in every tested setting a mis-set μ erodes part of the gain without inverting it. We therefore use a single value that is near-optimal across these model pairs rather than tuning it per setting. For a new setup, the trend above suggests a direction for adjustment: the optimal μ rises with draft–target alignment, which is observable at deployment from the measured MAT. Since task type also shifts the preference (Section 6.4.2), we treat this trend as a heuristic only; given the small sensitivity, a new model can simply start at $\mu = 0.03$.

Table 9: Threshold sweep with the EAGLE-3 reference (MT-Bench, throughput in tok/s, mean $\pm$ std over 5 runs). The last two columns compare the default $\mu = 0.03$ with EAGLE-3 and with the best μ for each model.

Model	$\mu = 0.01$	0.02	0.03	0.04	0.05	EAGLE-3	**Default vs. EAGLE-3**	**Default vs. best μ**
Vicuna-13B	258.4±1.6	265.5±1.7	**270.2±1.8**	265.8±1.9	265.3±2.3	256.2±1.4	+5.5%	0.0%
DSL-8B	214.2±1.9	220.9±1.4	224.8±0.9	**226.9±1.1**	225.6±2.3	210.6±1.1	+6.7%	−0.9%
Llama3-8B	234.6±1.7	235.6±1.5	**240.1±2.2**	239.5±1.5	238.5±1.1	229.0±1.9	+4.8%	0.0%
Qwen3-8B	**145.8±0.8**	145.3±0.7	144.1±0.6	140.9±0.9	138.0±0.7	123.1±1.1	+17.1%	−1.2%

Robustness to Draft-Model Strength. **TALON** uses the draft model's confidence as a proxy for acceptance by the target model, and this proxy may be less reliable when the drafter is weakly aligned with the target. An imperfect proxy, however, costs speedup, never correctness: confidence gating at depth only allocates the node budget, while which tokens are accepted is decided by the target model's tree verification (Appendix B). The root layer, where over-confidence would cause whole-draft rejection, is handled separately by the Top-K initialization. To cover the alignment axis directly, we run **TALON** with two draft checkpoints of clearly different strength for the same target model, Qwen3-8B (Table 10). Gating remains effective at both alignment levels, and the gain is larger with the weaker drafter (+14.0% to +17.3%) than with the stronger one (+9.1% to +10.4%): a static deep tree wastes more computation when the drafter is less reliable, so there is more for **TALON** to reclaim. Both runs use the default $\mu = 0.03$; since a well-aligned pair prefers a slightly larger μ, the numbers for the stronger drafter are, if anything, conservative.

Table 10: **TALON** with a weaker and a stronger draft checkpoint for the Qwen3-8B target (EAGLE-3 $\rightarrow$ **TALON**).

Draft checkpoint	**Benchmark**	**MAT**	δ	**Throughput (tok/s)**
AngelSlim (weaker)	MT-Bench	3.64 $\rightarrow$ 3.60 (−1.0%)	9.00 $\rightarrow$ 4.52 (−49.8%)	123.3 $\rightarrow$ 144.6 (**+17.3%**)
	GSM8K	3.92 $\rightarrow$ 3.85 (−1.7%)	9.00 $\rightarrow$ 5.08 (−43.6%)	132.9 $\rightarrow$ 151.5 (**+14.0%**)
Tengyunw (stronger)	MT-Bench	4.53 $\rightarrow$ 4.34 (−4.2%)	9.00 $\rightarrow$ 4.85 (−46.1%)	154.8 $\rightarrow$ 170.8 (**+10.4%**)
	GSM8K	5.04 $\rightarrow$ 4.86 (−3.6%)	9.00 $\rightarrow$ 5.64 (−37.3%)	171.7 $\rightarrow$ 187.4 (**+9.1%**)

Impact of the Budget Driven Framework. To further validate the necessity of **TALON**'s explicit budget constraint, we compare it against a heuristic-based dynamic draft strategy. Specifically, we directly apply the confidence-gated expansion strategy to EAGLE's existing static tree generation (denoted as EAGLE-dynamic) without introducing a global budget reallocation mechanism. As shown in Table 11, simply applying confidence-gated expansion to a fixed structure actually degrades the end-to-end speedup compared to the standard EAGLE baseline.

This performance degradation stems from two critical structural limitations. First, without a global budget awareness, the confidence-gated expansion lacks a strict stopping mechanism. In high-entropy contexts with flat probability distributions, it can produce an excessive number of draft nodes, drastically increasing the drafting latency without yielding proportional acceptance gains. Second, because the heuristic filtering operates strictly within EAGLE's predefined depth limit D, it cannot reallocate saved computational budget to extend the speculation depth in low-entropy contexts. In contrast, **TALON**'s budget-driven framework dynamically breaks this depth barrier, continuously

expanding the tree until the budget N is reached, which maximizes the expected acceptance length in deterministic scenarios.

Table 11: Ablation study comparing the standard static tree (EAGLE), a heuristic-filtered static tree (EAGLE-Dynamic), and our budget-driven adaptive tree (**TALON**). Results show that heuristic filtering without a budget mechanism degrades performance.

Model	Method	Alpaca	GSM8K	HumanEval	MT-Bench
Qwen3-8B	EAGLE	2.08×	2.37×	2.35×	2.19×
	EAGLE-Dynamic	1.94×	2.29×	2.24×	2.07×
	TALON	**2.44**×	**2.67**×	**2.69**×	**2.57**×
LLaMA-3.1-8B	EAGLE	3.91×	3.62×	4.05×	3.66×
	EAGLE-Dynamic	3.81×	3.49×	3.94×	3.54×
	TALON	**4.03**×	**3.80**×	**4.20**×	**3.84**×

E.7 More Case Studies of TALON

To qualitatively analyze the behavior of **TALON**, we visualize the decoding traces of a code generation task in Figure 12. Code generation typically represents a low-entropy regime, where the next tokens (e.g., syntax keywords, standard indentations) can be predicted with high confidence.

As shown in the left panel, EAGLE employs a relatively static expansion strategy. Even when the model is confident, EAGLE is limited by its fixed tree structure, resulting in frequent interruptions (pink tokens) and forcing the model to resample from the target model. This limits the Mean Accepted Tokens (MAT) per step.

In contrast, **TALON** (right panel) dynamically leverages its confidence-gated expansion mechanism. Upon detecting the low-entropy nature of the current context, **TALON** adaptively allocates the token budget to extend the tree *depth* rather than width. This allows the draft model to speculate deep-and-narrow draft tokens. Consequently, **TALON** significantly reduces the frequency of verification calls and achieves a higher MAT, demonstrating the efficiency of adaptive token trees in deterministic generation tasks.

E.8 Tree Construction Overhead

To rigorously quantify the algorithmic efficiency of our proposed method, we conduct a micro-benchmark focusing specifically on the **tree construction overhead**. We simulate the next-token probability distributions using a Zipfian distribution $P(r) \propto 1/r^{\alpha}$ [18], covering diverse generation scenarios ranging from high-entropy creative writing tasks ($\alpha = 0.7$) and standard natural language ($\alpha = 1.35$) to highly deterministic code generation ($\alpha = 5.0$). The warmup step is `20`, and we run each tree expansion strategy `100` times and report their mean latency. In this controlled environment, we compare the single-layer expansion latency of EAGLE against **TALON**. EAGLE relies on a rigid dual Top-K mechanism that performs sorting operations twice per layer—once for child selection and again for parent ranking—which scales poorly with large vocabulary sizes. In contrast, **TALON** utilizes a single confidence-gated sampling operation, implemented via efficient element-wise masking and non-zero index retrieval, which does not involve any ranking operation.

As illustrated in Figure 13, **TALON** consistently reduces the tree construction latency across varying vocabulary sizes ($32K$ to $152K$), achieving speedups ranging from 1.18× to 1.44×. The advantage is particularly pronounced in deterministic settings ($\alpha = 5.0$), where our adaptive mechanism naturally sparsifies the candidate set, thereby minimizing memory access overhead. However, it is important to acknowledge that **this sampling and tree construction latency constitutes a negligible fraction ($< 5\%$) of the total end-to-end inference time**, which remains dominated by the model's forward passes. Consequently, **TALON**'s primary wall-time speedup derives from its superior Draft Efficiency (δ) rather than this micro-optimization in sampling.

This micro-benchmark isolates the expansion operator on synthetic distributions. To measure the overhead within a full decoding step, we further profile each stage on SGLang [58] with Qwen3-8B, a single H200 GPU, the FlashAttention-3 [42] backend, and batch size 8 (Table 3). **TALON**'s candidate

Table 12: **Additional comparison experiments on Llama-3.1-8B-Instruct.** We report Wall-time Speedup relative to standard decoding. **Bold** numbers denote the best speedup performance.

Method	Alpaca	GSM8K	HumanEval	MT-Bench	QA	CNN/DM
EAGLE-3	3.92×	3.63×	4.05×	3.67×	3.01×	3.06×
OPT-Tree	3.75×	3.56×	3.90×	3.60×	3.15×	3.09×
TALON	**4.04**×	**3.81**×	**4.20**×	**3.85**×	**3.27**×	**3.30**×

filtering and budget control add 0.013 ms (tree mask) and 0.15 ms (tree policy) over the highly optimized EAGLE kernels, and its draft forwards take 0.07 ms longer. Together, these costs are an order of magnitude smaller than the 2.57 ms per step saved in target verification, which is where the end-to-end gain originates: in this batched setting, **TALON** treats the active requests as one draft tree under a joint budget (Appendix E.10), reducing the number of tokens sent to verification.

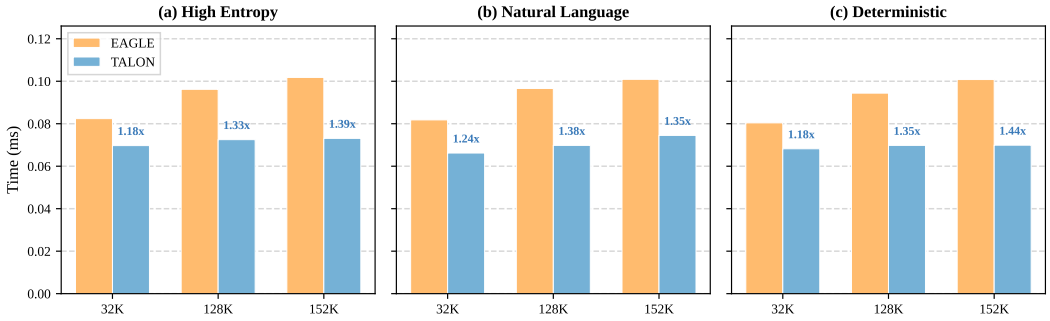


Figure 13: **Runtime breakdown of the tree expansion overhead.** We benchmark the latency of a single-layer expansion step across varying vocabulary sizes ($32K$, $128K$, $152K$) using Zipfian distributions [18] to simulate High Entropy ($\alpha = 0.7$), Natural Language ($\alpha = 1.35$), and Deterministic ($\alpha = 5.0$) contexts. **TALON** consistently outperforms the static dual Top-K approach of EAGLE, achieving up to **1.44**× speedup in the construction phase by avoiding expensive sorting operations. **Note that this step accounts for less than 5% of the total inference time.**

E.9 Additional Comparison Experiments

To further quantify the algorithmic efficiency of the proposed method, we provide additional comparison experiments between **TALON** and strong baselines (OPT-Tree and EAGLE) on Llama-3.1-8B-Instruct in Table 12. As shown in the table, **TALON** consistently outperforms EAGLE and OPT-Tree across all benchmarks. Notably, **TALON** achieves a 3.30× wall-time speedup on CNN/DM, significantly surpassing both alternatives. Beyond summarization, this consistent outperformance spans across mathematical reasoning (GSM8K) and code generation (HumanEval), proving that our method's efficiency gains are task-agnostic and highly effective for modern LLMs.

E.10 Batched Inference

To evaluate whether **TALON** remains effective under batched inference, we conduct additional experiments with Qwen3-8B on SGLang [58] using the FlashAttention-3 [42] backend. All experiments are conducted on a single NVIDIA H200 GPU. For batched decoding, we build one draft tree for the whole batch before verification. Active requests are treated as first-layer nodes under a shared root, and **TALON** allocates the node budget across all requests in the batch. This design controls the total number of draft tokens verified in each target-model forward pass. We compare **TALON** with the static EAGLE baseline under the same batched decoding setup and report throughput in tokens per second.

Table 13 reports batched throughput across batch sizes 8, 16, and 32. Across all tested settings, **TALON** improves throughput over EAGLE, with speedups ranging from 1.022× to 1.358×. These results suggest that budget-driven tree construction can remain effective in batched serving, where controlling the verification workload helps reduce verification overhead.

Table 13: **Batched inference results.** We report throughput under batched decoding. Speedup is computed as **TALON** throughput divided by EAGLE throughput. The small-tree and large-tree settings correspond to static EAGLE baselines with smaller and larger draft-tree budgets, respectively.

bs	EAGLE tok/s	TALON tok/s	Speedup
Small-tree setting			
8	540.26	556.25	1.030×
16	965.23	1028.64	1.066×
32	1724.43	1762.34	1.022×
Large-tree setting			
8	509.38	535.89	1.052×
16	653.15	715.26	1.095×
32	734.07	996.74	1.358×

To test whether the advantage extends to larger models and higher concurrency, we further compare **TALON** and EAGLE-3 on Qwen3-32B at batch sizes from 4 to 64 (Table 2). Both methods use the small-tree setting; we tune each method's budget separately and report its best result, averaged over 5 runs. **TALON** leads at every batch size, and its margin over EAGLE-3 grows from +2.9% at batch size 4 to +10.5% at batch size 64. This trend is opposite to that of the speedup over AR decoding, which falls as the batch grows for any SD method: the target model becomes compute-bound, so verifying many tokens in one forward is no longer nearly free. This is also why the gains in Table 13 are modest in absolute terms.

F More Discussions to Related Work

This appendix provides an extended discussion on the evolution of speculative decoding, tracing the field's progression from chain-based verification to tree-structured verification. We highlight how varying approaches trade off draft quality, verification cost, and implementation complexity.

Chain-based speculative decoding. The classical formulation of speculative decoding, often referred to as speculative sampling, is a *draft-and-verify* procedure [23, 6, 53]. Given a prefix, a fast drafter autoregressively proposes a length-K continuation. The target model then verifies these K positions in a single forward pass, accepting the longest matching prefix and resampling at the first mismatch. While attractive for its lossless nature, the speedup of this framework hinges strictly on the acceptance length and the computational cost ratio between the drafter and the target.

Improving the drafter. A major line of work focuses on producing higher-quality drafts with minimal overhead to boost acceptance rates. Medusa [5] introduces multiple lightweight decoding heads atop the target model to predict several next tokens in parallel by reusing target hidden states. Because independent heads may ignore intra-draft dependencies, follow-up approaches like Hydra [1] and Clover [49] introduce sequential dependencies among heads to better approximate autoregressive drafting. Chimera [52] proposes a lightweight draft architecture combining short-range and full-context signals to enhance quality while maintaining speed. Complementarily, GliDe with CaPE [12] reuses the target KV cache to lower drafting overhead and employs confidence-guided proposal expansion to provide stronger candidates.

Controlling draft length and reducing wasted work. Even with a strong drafter, a fixed speculation length K can be suboptimal: large K wastes computation when acceptance is low, while small K caps potential speedup. SpecDec++ [20] formulates the candidate length selection as an MDP, adaptively stopping drafting using an acceptance-prediction signal. DISCO [31] similarly predicts when to stop speculation by dynamically selecting the speculation length. PEARL [28] addresses the system-level bottleneck of mutual waiting between drafting and verification by overlapping phases and enabling segmented, adaptive draft lengths. Additionally, Block Verification [43] has been proposed to accelerate the verification phase itself.

Addressing training–inference misalignment. When a specialized drafter is trained, mismatches between training distributions and inference-time contexts can hurt acceptance. HASS [54] proposes

harmonized objectives and context alignment to reduce these inconsistencies. CORAL [47] further improves consistency via cross-step representation alignment and reduces the effective cost of the LM head. Orthogonally, Judge Decoding [3] observes that many rejected tokens are plausible and proposes relaxing verification via a compact judging module.

Tree-based speculative decoding. Tree-based Speculative Decoding (SD) generalizes the draft chain into a *draft tree*, enabling the verifier to choose among multiple candidate branches. This significantly reduces the penalty of early mismatches. The key enabler is *tree attention*, which allows the target model to verify multiple paths in parallel [32, 5].

Token-tree verification and structured drafting. SpecInfer [32] is a representative early system that organizes drafter outputs into a token tree verified via tree-based attention. EAGLE [25] advances this by performing autoregression at the feature level. EAGLE-2 [24] introduces a context-aware dynamic draft tree guided by drafter confidence, while EAGLE-3 [26] further improves drafting via multi-layer fusion and training-time techniques.

Training-free tree construction via retrieval. To avoid training specialized drafters, some methods combine SD with retrieval. REST [17] retrieves draft tokens from a datastore, while Prompt Lookup Decoding [41] leverages n-gram matching within the current context. Token Recycling [30] stores previously observed candidate-token transitions in a compact adjacency matrix to retrieve a draft tree. SAM-Decoding [19] utilizes a suffix automaton to efficiently find the longest suffix match for drafting. LogitSpec [29] proposes to use the last logit as a guidance to retrieve more matched and accurate draft tokens. While plug-and-play, these methods are bounded by the availability of high-quality matches in the context or datastore. Alternative draft-model-free approaches like Lookahead Decoding [13] generate parallel n-grams using Jacobi iteration, and EESD [27] uses early exiting from intermediate layers to generate drafts.

Tree-attention efficiency. Since tree-based SD relies on non-trivial KV updates, system efficiency is critical. DeFT [51] proposes an IO-aware flash tree-attention algorithm tailored for tree-structured inference to improve verification throughput.

Summary and positioning of TALON. While tree-based SD has become a standard paradigm, existing approaches face distinct limitations in adaptability and efficiency. Several of them already operate under a node budget; what differs is *how* that budget is spent. Table 14 summarizes the comparison, and each family is limited at a different point. EAGLE-2/3 [24, 26] adapt *which* nodes to keep, not *how many*: the $K \times D$ shape is fixed before expansion, so a deterministic context cannot be granted extra depth. Learning-based methods, such as C2T [21] and AdaEAGLE [55], require training specialized modules and remain constrained by rigid parameter spaces (e.g., predicting a fixed K), failing to achieve fully elastic topology changes. Optimization-based methods also exhibit drawbacks: Sequoia [8] relies on *offline* algorithms to find a globally optimal tree, resulting in a *static* template that cannot adapt to instance-wise difficulty. SVIP [56] and DDD [4] adapt only *when to stop* drafting, while the width stays fixed. OPT-Tree [46] is the closest to **TALON**: it is training-free and adapts both width and depth. Two differences remain. In *shape*, its depth is bounded by a preset maximum, whereas the depth of **TALON** is decided by the budget and the context. In *cost*, it expands every layer to the full node budget and trims back only at the end ("generate-then-prune"), so each draft forward carries the whole tree, whereas each draft forward of **TALON** carries only the nodes that survived gating in the previous layer. **TALON** distinguishes itself through a *training-free, budget-driven* framework that operates via *on-the-fly* construction. Analogous to "pre-pruning" in decision trees, **TALON** adopts a "prune-while-expanding" strategy: it iteratively allocates the node budget based on real-time confidence, naturally halting expansion in uncertain branches without generating wasteful nodes first. This allows the draft tree to fluidly morph between "deep-and-narrow" and "shallow-and-wide" shapes, maximizing speculation utility with minimal construction cost. **TALON** is thus the only entry in Table 14 that is at once training-free, adaptive in both width and depth, and governed by a budget that caps the expansion *process* rather than only the final tree. Because a layer costs only its survivors, the same 60-node budget can be spent as a chain tens of layers deep when the context is easy, which is why τ is no longer capped by a preset depth D. Confidence gating alone is not the source of the gain: grafting it onto EAGLE's static tree without the budget mechanism degrades performance (Table 11). The gain comes from budget-driven reallocation, which (i) removes EAGLE's hard depth ceiling so that saved computation can be reinvested as depth in deterministic contexts, and (ii) supplies a strict stopping rule so that flat distributions cannot inflate the tree.

Table 14: Positioning of **TALON** against prior tree-construction methods.

Method	Training-free	Adaptive width and depth	How the budget is used	What the construction favors
EAGLE-2/3 [24, 26]	✓	✗	expand, then prune to budget	top-K path score
Sequoia [8]	✓	✗	offline search; shape fixed at inference	hardware-aware tree shape
AdaEAGLE / C2T [55, 21]	✗	depth only	learned module predicts the shape	predicted acceptance
SVIP / DDD [56, 4]	✓	depth only	stopping criterion only	when to stop drafting
OPT-Tree [46]	✓	depth capped	generate-then-prune	acceptance length $E(A)$
TALON (ours)	✓	✓	**prune-while-generate**	**acceptance length per draft forward**

Speculative decoding for constrained generation. Constrained Decoding with Speculative Lookaheads (CDSL) [34] applies the speculative decoding paradigm to constrained generation, where a lightweight draft model generates lookahead sequences that are then verified jointly by the target LLM and task-specific reward functions. **TALON**'s adaptive budget-driven tree construction is orthogonal to CDSL's verification mechanism, and could potentially be integrated as the tree-building backbone within CDSL to further improve its drafting efficiency.

G Limitations and Ethical Considerations

While **TALON** demonstrates significant speedups and adaptability across various benchmarks, we identify several limitations that present avenues for future research:

Batched Serving Optimization. Appendix E.10 shows that **TALON** improves throughput over the static baseline under batched decoding. The current batched implementation does not enable CUDA graph optimization, so the reported throughput reflects a straightforward implementation rather than a fully optimized serving system. Additional system-level optimizations and more advanced batching policies could further improve absolute throughput.

Hyperparameter Generalization. While we found a fixed threshold ($\mu = 0.03$) and budget ($N = 60$) to be robust across most tested models and datasets, optimal performance in highly specialized domains might require task-specific tuning. Although μ is a relative threshold that already adapts the effective cutoff to each layer, its margin is fixed, and the best margin varies across draft–target pairs: the default is within 1.2% of the per-model optimum in our sweep (Table 9) but is conservative for well-aligned drafters (Table 10). Since the optimal μ rises with draft–target alignment, which is observable from the measured MAT, an auto-tuning mechanism that adjusts μ (and possibly N) on the fly from a running MAT estimate is a concrete next step.

Inheritance of Model Behaviors. **TALON** is an inference acceleration framework designed to speed up existing Large Language Models without modifying their weights. As a speculative decoding method, it aims to losslessly recover the distribution of the target model. Consequently, **TALON** inherits the ethical properties, biases, and potential safety risks of the underlying target LLM and draft model. It does not introduce new capabilities for generating harmful content, nor does it mitigate existing biases in the base models. Users should continue to apply standard safety guardrails and alignment techniques to the target models deployed with **TALON**.

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 claim the contributions and scope of this paper in abstract and introduction.

   Guidelines:

   - The answer [N/A] 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 [N/A] 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: we discuss the limitations of this paper in Appendix G.

   Guidelines:

   - The answer [N/A] 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 provide the complete proof in Appendix D.

Guidelines:

- The answer [N/A] 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 provide the detailed algorithm in Appendix A.

   Guidelines:

   - The answer [N/A] 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: [Yes]

Justification: We provide implementation details in Appendix E.1.

Guidelines:

- The answer [N/A] means that paper does not include experiments requiring code.
- Please see the NeurIPS code and data submission guidelines (https://neurips.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://neurips.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) necessary to understand the results?

   Answer: [Yes]

   Justification: We provide experimental details in Appendix E.1.

   Guidelines:

   - The answer [N/A] 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: [Yes]

   Justification: The speedups in Table 1 are means over 5 runs, and Appendix E.2 reports their standard deviations. The threshold sweep (Table 9) and the batched-inference results (Table 2) are also reported as mean $\pm$ standard deviation over 5 runs, which captures run-to-run variation under fixed experimental conditions.

   Guidelines:

   - The answer [N/A] 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 provide experimental details in Appendix E.1.

   Guidelines:

   - The answer [N/A] 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: This paper conforms with the NeurIPS Code of Ethics.

   Guidelines:

   - The answer [N/A] 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: [Yes]

    Justification: we provide potential ethical considerations in Appendix G.

    Guidelines:

    - The answer [N/A] means that there is no societal impact of the work performed.
    - If the authors answer [N/A] 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., pre-trained language models, image generators, or scraped datasets)?

    Answer: [N/A]

    Justification: We do not release any data or pre-trained models.

    Guidelines:

    - The answer [N/A] 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: We cite the used assets in reference. Thanks for their contributions.

    Guidelines:

    - The answer [N/A] 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: [N/A]

    Justification: We provide experimental details in Appendix E.1 and will release the full training/inference code, benchmark evaluation scripts, reproduction configuration files for all backbones, and step-by-step reproduction instructions immediately upon acceptance.

    Guidelines:

    - The answer [N/A] 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: [N/A]

    Justification: We do not involve crowdsourcing and human subjects.

    Guidelines:

    - The answer [N/A] 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: [N/A]

    Justification: We do not involve crowdsourcing and human subjects.

    Guidelines:

    - The answer [N/A] 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 rigor, or originality of the research, declaration is not required.

Answer: [N/A]

Justification: The core method development in this research does not involve LLMs as any important, original, or non-standard components.

Guidelines:

- The answer [N/A] 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 in the NeurIPS handbook for what should or should not be described.