SMART: When is it Actually Worth Expanding a Speculative Tree?

Lifu Wang[1] and Pan Zhou[1] ⋆

Singapore Management University, Singapore
{lifuwang, panzhou}@smu.edu.sg

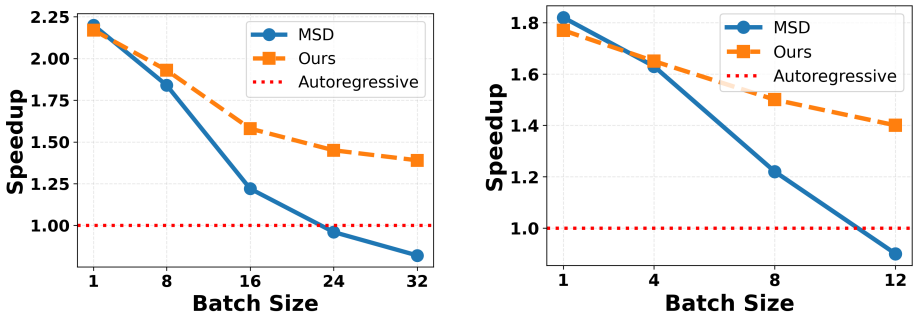
Abstract. Tree-based speculative decoding accelerates autoregressive generation by verifying a branching tree of draft tokens in a single target-model forward pass. However, existing methods prioritize maximizing token-level likelihood or the number of accepted tokens while ignoring a critical "efficiency paradox": the computational overhead of drafting and verifying big trees can grow super-linearly, particularly at scale. This often leads to negative wall-clock speedup when batch sizes increase or hardware saturation limits are reached. To address this, we propose **SMART**, a **S**ystem-aware **M**arginal **A**nalysis framework for **R**untime **T**ree construction. SMART reformulates tree expansion as a hardware-aware optimization problem that directly maximizes end-to-end speedup. By applying a principled marginal benefit–cost rule at inference time, SMART expands a node only when its marginal benefit–cost ratio exceeds the tree-level speedup. SMART is training-free and serves as a plug-and-play controller for existing frameworks like MSD and EAGLE. Extensive evaluations across three MLLMs (e.g., LLaVA, Qwen2-VL) and four LLMs (e.g., Llama-3.1, DeepSeek-R1) demonstrate that SMART consistently outperforms state-of-the-art baselines. It delivers an average additional speedup of **20.0%** for MLLMs and **15.4%** for LLMs across compute-bound batching regimes and diverse GPU architectures without performance loss.

Keywords: speculative decoding · (multimodal) large language models

1 Introduction

Autoregressive decoding is the computational workhorse of modern generative AI, underpinning both Large Language Models (LLMs) [1,7,9,31,32] and Multimodal LLMs (MLLMs) [17,22,37] for tasks ranging from (visual) reasoning [4,11] to high-fidelity image synthesis [29]. Yet its core mechanism is intrinsically sequential: tokens must be generated one after another. As model sizes grow and outputs become longer, this strict dependency chain turns into a dominant latency bottleneck, throttling throughput and inflating serving cost in real deployments.

⋆ Corresponding Author



(a) RTX Pro 6000 **(b)** L40S

Fig. 1: Speedup over autoregressive decoding across batch sizes (data from Table 3). Likelihood-maximizing tree methods such as Multimodal Speculative Decoding (MSD) exhibit severe performance degradation at large batch sizes, dropping below 1× at batch 12 on L40S and reaching only 0.82× at batch 32 on RTX Pro 6000. In contrast, our speedup-maximizing approach maintains consistent speedup by constructing trees based on the verification budget and device-specific cost models. This demonstrates that tree construction must be scalable with batch size and hardware-aware.

Speculative decoding [5,16] has emerged as a practical strategy to break this sequential barrier. It uses a lightweight *draft* model to propose multiple candidate tokens, which are then *verified* by the target model in parallel. Tree-based speculative decoding [20,21,26] further extends this by constructing a **draft tree**—a branching structure of multiple candidate continuations. By verifying an entire tree in a single forward pass, the system increases the expected acceptance length (number of accepted tokens) per target model forward. Conventional methods [20, 21] typically drive tree expansion via token-level likelihood, selecting candidates with the highest cumulative probability. More recently, GTO [12] argues that token-level likelihood maximization during training is a poor proxy for the actual speculative-decoding goal, and proposed training objectives that directly maximize the *expected acceptance length* of the draft tree, aligning training with inference behavior and improving over vanilla speculative decoding.

Unfortunately, the ultimate goal of speculative decoding is not to maximize the number of tokens accepted, but to maximize **end-to-end wall-clock speedup**. Existing designs suffer from a fundamental misalignment with system-level costs. Specifically, speedup is a function of both the acceptance length and the computational overhead of drafting and verification. Greedily expanding a tree to capture more tokens can be counterproductive; if the marginal cost of verifying a larger tree outweighs the gains in acceptance length, the system may experience "negative speedup," performing worse than vanilla autoregressive decoding. Experienced inference practitioners are well aware of this issue and routinely profile latency to tune speculative-decoding configurations. However, current practice relies on offline grid search over static hyperparameters that

remain constant across all inputs and must be re-tuned whenever the hardware, batch size, or workload distribution changes.

As illustrated in Fig. 1, this mismatch is exacerbated by two critical factors in production environments: **batch-size scalability** and **hardware heterogeneity**. First, as batch sizes increase, the computational overhead of verifying big draft trees grows super-linearly. In memory-bandwidth bound regimes (e.g., $b = 1$), verifying a large tree is beneficial as it amortizes the high cost of weight loading. However, once the batch size exceeds a hardware-specific threshold—approximately $b \geq 8$ for an RTX Pro 6000—the GPU shifts into a **compute-bound regime**. In this state, the arithmetic intensity of verifying a large tree for every sequence in the batch exceeds the device's peak throughput, causing the verification cost to outweigh the gains in acceptance length. Second, the "pivot point" where this bottleneck occurs is highly device-specific. For instance, a likelihood-maximizing tree constructed by MSD yielding 1.8× speedup on an RTX Pro 6000 may drop to 1.2× on an L40S at the same batch size of 8 because the latter saturates its compute units earlier. These results underscore that a likelihood-maximizing tree is inherently suboptimal. Effective speculative decoding requires hardware-aware draft tree construction that adapts to the specific **arithmetic intensity** and **saturation limits** of the underlying GPU.

In this paper, we adopt a **system-oriented** view of speculative decoding. Instead of asking *how to maximize acceptance length*, we ask: *When is it computationally worth expanding the draft tree, and which expansions measurably improve end-to-end speedup under the current hardware and batching regime?* Our goal is to automate the manual, latency-aware deployment tuning that practitioners currently perform offline into an *online*, *context-adaptive* marginal decision rule.

Contributions. We propose **SMART**, a **S**ystem-aware **M**arginal **A**nalysis framework for **R**untime **T**ree construction in speculative decoding. SMART constructs a speedup-maximizing draft tree *at inference time* using a principled marginal benefit–cost rule. Importantly, SMART is **training-free**: it requires no changes to the draft model or target model weights, making it an *out-of-the-box* drop-in improvement for existing speculative decoding pipelines (e.g. MSD [21] and EAGLE-3 [20]). Our main contributions are three-fold.

First, we define a **system-level speedup objective.** Given a draft tree $\mathcal{T}$, we explicitly model the end-to-end speedup as $\mathcal{R}(\mathcal{T}) = \frac{c_T \cdot L^{\text{tree}}}{C_{\text{draft}} + C_{\text{verify}}}$, where L^{tree} is the expected number of accepted tokens (i.e., acceptance length) of the tree $\mathcal{T}$, c_T is the per-token cost of vanilla autoregressive decoding under the target model, and C_{draft} and C_{verify} are the total drafting and verification costs induced by the tree. Intuitively, the $c_T \cdot L^{\text{tree}}$ denotes the cost of vanilla sequential decoding for generating L^{tree} tokens, while $C_{\text{draft}} + C_{\text{verify}}$ is the cost of speculative decoding for getting L^{tree} accepted tokens. Therefore, this formulation makes the central trade-off explicit: maximizing L^{tree} alone can be suboptimal if it increases $C_{\text{draft}} + C_{\text{verify}}$ disproportionately. By directly optimizing the ratio $\mathcal{R}(\mathcal{T})$, SMART targets the metric that matters in deployment: wall-clock speedup relative to vanilla decoding.

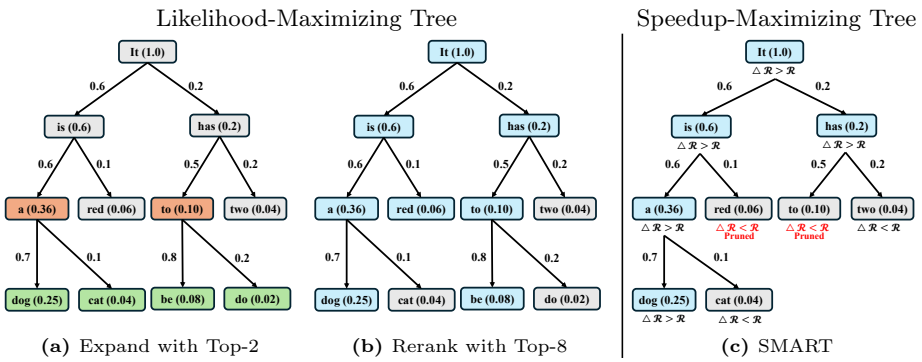
Second, SMART builds upon the reward $\mathcal{R}(\mathcal{T})$ to propose a speedup-maximizing tree expansion framework. To maximize the reward efficiently, SMART formulates tree construction as a sequence of speedup-maximizing expansion decisions. At each layer, we estimate the marginal gain and marginal cost of expanding each candidate node, and expand a node only when its marginal benefit–cost ratio exceeds the current tree's global ratio. This criterion ensures that local expansions improve the global speedup objective, allowing the tree shape to adapt to both the context difficulty and the available hardware budget.

Finally, SMART is training-free and applicable for plug-and-play deployment. SMART does not modify the draft model, the target model, or the verification mechanism. Instead, it replaces the likelihood-maximizing tree-construction policy with a speedup-maximizing policy. As a result, SMART is immediately compatible with existing speculative decoding systems and can be integrated as a lightweight inference-time controller.

Extensive evaluations across three MLLMs (LLaVA-1.5-7B and LLaVA-1.5-13B [22], and Qwen2-VL-7B-Instruct [33]) and four LLMs (LLaMA-3.1-Instruct-8B and LLaMA-3.3-70B [9], Vicuna-1.3-13B [7], and DeepSeek-R1-Distill-LLaMA-8B [11]) demonstrate that SMART consistently improves end-to-end speedup over strong baselines such as MSD [21] and EAGLE-3 [20], yielding an average of **20.0%** additional acceleration on MLLMs and **15.4%** on LLMs, while remaining robust across diverse hardware and batching scenarios.

2 Related Work

Speculative decoding accelerates autoregressive generation by proposing draft tokens with a lightweight model and verifying them in parallel using the target model. [5,16,26,30]. One line of work focuses on the design and training of draft models. Medusa [3] trains multiple prediction heads to generate draft tokens in parallel. EAGLE [19] predicts future representations in the feature space, while EAGLE-3 [20] later returns to token-level prediction with scaled training data. HASS [34] explicitly mitigates feature-level draft–target mismatches, and GRIFFIN [13] further reduces token-level draft–target mismatches. GTO [12] treats the expected acceptance length of draft trees as a reward signal and updates the draft model using a PPO-style surrogate objective. Another line of work focuses on constructing draft trees, which is also the focus of our work. These methods can be divided based on whether they require training extra modules. On the training side, SpecDec++ [14] and DISCO [24] learn classifiers to predict optimal draft lengths. On the inference side, EAGLE-2 [18] proposes a context-aware dynamic draft tree to increase acceptance length. TapOut [28] and SVIP [35] rely on per-token heuristics such as entropy or confidence scores to determine when to stop drafting. However, although these token-level heuristics are simple and system-friendly, they often suffer from threshold sensitivity and limited transferability. In contrast, SMART makes expansion decisions through a marginal benefit–cost ratio analysis and does not rely on externally tuned thresholds.



(a) Expand with Top-2 **(b)** Rerank with Top-8 **(c)** SMART

Fig. 2: Comparison of likelihood-maximizing ((a)–(b)) and speedup-maximizing tree construction (c). **(a) Expansion phase.** At each layer, the method selects the top-2 nodes with the highest cumulative probability predicted by the draft model (orange) and generating their top-2 children (green) using the draft model. **(b) Rerank phase.** After reaching the maximum depth, all nodes in the tree are globally reranked by confidence and the top-8 nodes (blue) are retained for verification. **(c) SMART.** Instead of expanding all candidates, SMART evaluates each node's marginal benefit–cost ratio ($\Delta R = \frac{\Delta C_{\text{target}}(u)}{\Delta C_{\text{spec}}(u)}$, with marginal terms defined in Eqn. 10) online and only expands nodes (blue) that improve the overall speedup, producing a smaller and more efficient draft tree.

3 Methodology

Motivated by the mismatch between likelihood-driven tree heuristics and *end-to-end* speedup, we propose a speedup-maximizing framework that constructs draft trees by directly optimizing a device-specific speedup objective. Our method has two components. First, we define an end-to-end speedup metric (Sec. 3.1) that captures true wall-clock efficiency by balancing expected accepted tokens against the measured costs of drafting and verification on a given device. Second, we formulate tree construction as a sequential decision problem (Sec. 3.2) and greedily expand tree nodes only when their marginal benefit–cost ratio exceeds the current tree's global ratio. This ensures each tree expansion improves the global speedup objective while preserving linear-time construction.

3.1 Expected Speedup of A Draft Tree

We start by introducing how likelihood-maximizing draft trees are constructed. State-of-the-art approaches such as EAGLE-3 [20] and MSD [21] construct a depth-d draft tree with a fixed two-stage policy: (i) as shown in Fig. 2a, at each layer, we select the global top-k nodes by computing cumulative probability $P(u)$ from token probability given by the draft model; (ii) as shown in Fig. 2b, after reaching the maximal depth d, namely, finishing tree construction, we re-rank tree nodes by $P(\cdot)$ and keep the top-g nodes for verification. These methods optimize the acceptance rate of draft tokens. However, higher acceptance does

not necessarily imply higher speedup. As discussed in Sec. 1, this assumption can fail because speedup depends on *both* the number of tokens accepted and the computation cost required to produce and verify them, which varies with batch size and hardware.

To resolve this issue, we evaluate the quality of a draft tree using a system-level speedup that directly compares the generation cost (i.e., wall-clock time) of vanilla autoregressive decoding and speculative decoding. Formally, given draft tree $\mathcal{T}$, we define its **expected speedup** as

$$\mathcal{R}(\mathcal{T}) = \frac{c_T \cdot L^{\text{tree}}}{C_{\text{draft}}(\mathcal{T}) + C_{\text{verify}}(\mathcal{T})} \tag{1}$$

where c_T is the per-token autoregressive decoding cost (i.e., wall-clock time) of the target model, L^{tree} is the expected number of tokens accepted from $\mathcal{T}$, and $C_{\text{draft}}(\mathcal{T})$ and $C_{\text{verify}}(\mathcal{T})$ are the measured costs to generate and verify the draft tree, respectively. To generate L^{tree} tokens, target model needs to forward L^{tree} times and thus needs the total cost $c_T \cdot L^{\text{tree}}$ which corresponds to the numerator in Eqn. (1). Meanwhile, to obtain an expected acceptance length of L^{tree}, speculative decoding must generate a draft tree and verify it with the target model. The total cost is the sum of drafting and verification costs, corresponding to the denominator $C_{\text{draft}}(\mathcal{T}) + C_{\text{verify}}(\mathcal{T})$ in Eqn. (1). Unlike likelihood- or acceptance-length-only objectives, $\mathcal{R}(\mathcal{T})$ directly measures the speedup by comparing the cost of vanilla autoregressive decoding of the target model and the cost of speculative decoding under the same acceptance length. This new metric is hardware-aware that adapts to the specific arithmetic intensity and saturation limits of the underlying GPU.

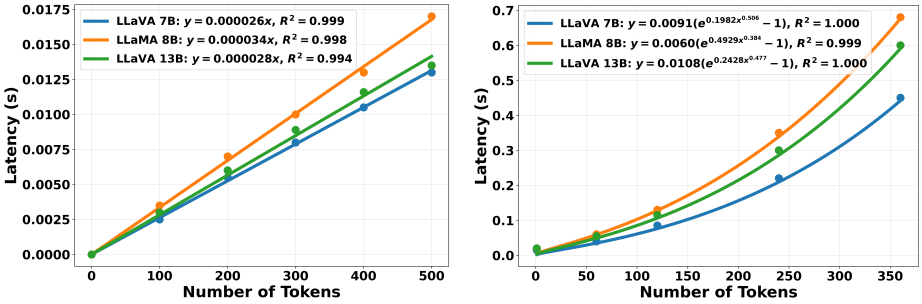
Estimation of Expected Accepted Tokens L^{tree}. To evaluate Eqn. (1), we first estimate the tree acceptance length L^{tree} of the draft tree $\mathcal{T}$, which in turn yields an estimate of the target model's sequential decoding cost: $c_T \cdot L^{\text{tree}}$. To this end, we follow the spirit of GTO [12] and compute the expected acceptance length L^{tree} as the mean across all paths in the tree $\mathcal{T}$. Specifically, given a context $\mathbf{x}$, the draft model generates draft tokens to construct a tree $\mathcal{T}$ which contains $|\mathcal{P}|$ draft sequences, and the corresponding acceptance length L^{tree} can be estimated as

$$L^{\text{tree}} = \frac{1}{|\mathcal{P}|} \sum\nolimits_{\tilde{\mathbf{x}}^{(i)} \in \mathcal{P}} L_i = \frac{1}{|\mathcal{P}|} \sum\nolimits_{\tilde{\mathbf{x}}^{(i)} \in \mathcal{P}} \sum\nolimits_{j=1}^{|\tilde{\mathbf{x}}^{(i)}|} P\left(\tilde{\mathbf{x}}_{1:j}^{(i)} \mid \mathbf{x}\right), \tag{2}$$

where $\mathcal{P}$ denotes the set of all root-to-leaf paths in the tree $\mathcal{T}$ and $P\left(\tilde{\mathbf{x}}_{1:j}^{(i)} \mid \mathbf{x}\right)$ is the accumulated probability of sequence $\tilde{\mathbf{x}}_{1:j}^{(i)}$:

$$P\left(\tilde{\mathbf{x}}_{1:j}^{(i)} \mid \mathbf{x}\right) = \prod\nolimits_{k=1}^{j} p\left(\tilde{\mathbf{x}}_{k}^{(i)} \mid \mathbf{x}, \tilde{\mathbf{x}}_{1:k-1}^{(i)}\right), \tag{3}$$

where $p\left(\tilde{\mathbf{x}}_{k}^{(i)} \mid \mathbf{x}, \tilde{\mathbf{x}}_{1:k-1}^{(i)}\right)$ denotes the probability of generating token $\tilde{\mathbf{x}}_{k}^{(i)}$ of target model given context $[\mathbf{x}, \tilde{\mathbf{x}}_{1:k-1}^{(i)}]$. Here, $L_i = \sum_{j=1}^{|\tilde{\mathbf{x}}^{(i)}|} P\left(\tilde{\mathbf{x}}^{(i)} \mid \mathbf{x}, \tilde{\mathbf{x}}_{1:j-1}^{(i)}\right)$ denotes



(a) Draft cost vs. draft tokens x. **(b)** Verification cost vs. draft tokens x.

Fig. 3: Measured latencies (dots) and fitted cost models (lines) for drafting and verification on RTX Pro 6000. Draft latency corresponds to the total latency of generating the full draft tree, where x is the total number of tokens in the tree. Verification latency corresponds to the forward-pass latency of the target model with x input tokens.

the estimated acceptance length of the i-th draft path $\tilde{\mathbf{x}}^{(i)}$ in $\mathcal{T}$. This follows from the fact that the expected number of consecutively accepted tokens equals the sum of the probabilities that each prefix is accepted during verification [18]. We approximate the acceptance probability $p\big(\tilde{\mathbf{x}}_k^{(i)} \mid \mathbf{x}, \tilde{\mathbf{x}}_{1:k-1}^{(i)}\big)$ using the draft model's predicted probability because 1) draft model is trained to align with the target model; and 2) EAGLE-2 [18] shows there is a strong positive correlation of prediction behaviors between target model and its corresponding draft model.

Cost Modeling. As shown in Fig. 3, we profile device-specific draft and verification latencies as a function of the total number of drafted tokens $|\mathcal{T}|$ in the tree $\mathcal{T}$. Drafting scales roughly linearly with $|\mathcal{T}|$ because draft models are small and typically memory-bound, yielding near-constant per-token cost, as demonstrated in Fig. 3a. In practice, draft latency follows a layer-wise staircase: tokens at the same depth are generated in parallel, so cost increases only when a new depth level is added; the continuous form serves as a smooth surrogate for the marginal analysis. Verification grows much faster because the target model is large and, in high-batch compute-saturated regimes, verification becomes compute-bound, causing empirically convex latency growth as the number of draft tokens increases, as shown in Fig. 3b.

Accordingly, we use linear model to model the draft cost in Fig. 3 (a):

$$C_{\text{draft}}(\mathcal{T}) = \lambda|\mathcal{T}| + \beta, \tag{4}$$

and adopt a power-exponential model to approximate the verification latency:

$$C_{\text{verify}}(\mathcal{T}) = \gamma(\exp(\delta\,|\mathcal{T}|^{\rho}) - 1) + \eta. \tag{5}$$

Here λ, γ, δ, and ρ are fitted per device, with the bias terms (β and η) fixed to 0 to ensure both models pass through the origin. This profiling is lightweight,

making the estimation of the hyperparameters in Eqs. (4) and (5) inexpensive. In practice, each fit requires only five forward passes. For example, profiling LLaMA-3.1-Instruct-8B [9] on MT-Bench [36] takes about 10 seconds on an RTX Pro 6000 GPU, which is negligible and accounts for only $\approx 1.67\%$ of the MT-Bench test set inference time. This enables fast fitting of the two cost models, $C_{\text{draft}}(\mathcal{T})$ and $C_{\text{verify}}(\mathcal{T})$, which in turn supports efficient tree expansion in the next section.

3.2 Speedup-Maximizing Draft Tree

Draft-tree construction is a trade-off: expanding more nodes can increase the expected acceptance length, but it also raises drafting and verification cost, which may reduce end-to-end speedup. Therefore, we must carefully choose which tree nodes to expand so as to improve expected acceptance length while incurring minimal additional drafting and verification cost. We formulate tree growth as a sequential decision process that, at each layer, selects which candidates to keep by comparing their marginal benefit in expected acceptance length against their marginal system cost, thereby maximizing end-to-end speedup.

Sequential Decision Formulation. As shown in Fig. 2c, we construct the draft tree sequentially over layers $\ell = 1, \ldots, d$. Let S_ℓ denote the set of all selected nodes after completing layer ℓ, with $S_0 = \{\text{root}\}$. We maintain a layer-wise active set A_ℓ (with $A_0 = \{\text{root}\}$), whose elements are the nodes retained at layer ℓ and expanded at layer $\ell{+}1$.

At layer ℓ, we expand every node in $A_{\ell-1}$ by drawing its top-k candidates from the draft-model distribution, producing the candidate set $\mathcal{U}_\ell(A_{\ell-1})$ of size $k|A_{\ell-1}|$. For each candidate token $u \in \mathcal{U}_\ell(A_{\ell-1})$, a selection operator $\mathcal{E}_\ell$ assigns a binary label $e_\ell(u) \in \{0, 1\}$, where $e_\ell(u) = 1$ indicates that u is retained and $e_\ell(u) = 0$ that it is pruned. The operator thus maps the full candidate set to the subset of survivors:

$$A_\ell \;=\; \mathcal{E}_\ell\big(\mathcal{U}_\ell(A_{\ell-1})\big) \;=\; \big\{\, u \in \mathcal{U}_\ell(A_{\ell-1}) \;\big|\; e_\ell(u) = 1 \,\big\}. \tag{6}$$

The tree evolves as

$$S_\ell \;=\; S_{\ell-1} \cup A_\ell. \tag{7}$$

Let $L \le d$ be the terminal layer: either $L = d$, or L is the first layer ℓ such that $A_\ell = \emptyset$ or $|S_\ell|$ reaches a predetermined budget B. The induced tree is $\mathcal{T} = S_L$. Our objective is to choose the selection operators $\{\mathcal{E}_\ell\}_{\ell=1}^{d}$ to maximize the final-tree reward:

$$\begin{aligned} \max_{\{\mathcal{E}_\ell\}_{\ell=1}^{d}} \quad & \frac{c_T\, L^{\text{tree}}(S_L)}{C_{\text{draft}}(S_L) + C_{\text{verify}}(S_L)} \\ \text{s.t.} \quad & |S_L| \;\le\; B, \end{aligned} \tag{8}$$

where $|S_L| \le B$ constrains the tree size. Because verification latency grows super-linearly with the number of draft tokens (Fig. 3b), we impose a total verification

budget B_{verify} to keep verification in the memory-bound (near-flat) region of the cost curve, and split it evenly across the batch, giving a per-sequence budget $B = B_{\text{verify}}/b$ with b the batch size.

Optimizing Eqn. (8) seeks selection operators that maximize the expected speedup of speculative decoding. The objective is the ratio of the cost of vanilla autoregressive decoding by the target model, $c_T\, L^{\text{tree}}(S_L)$, to the cost of speculative decoding, $C_{\text{draft}}(S_L) + C_{\text{verify}}(S_L)$, where S_L is the final draft tree. Since the tree is built layer by layer, this optimization reduces to a sequence of layer-wise decisions $\{\mathcal{E}_\ell\}_{\ell=1}^{d}$: at each layer, retain only the candidates that maximize the tree's expected speedup. Next, we describe an efficient strategy to solve this sequential decision problem.

Optimizing the Sequential Decision Objective Computing the optimal action sequence in Eqn. (8) requires evaluating all valid subtrees of the full k-ary expansion tree. Since the total number of candidate nodes across d layers is $\sum_{\ell=1}^{d} k^\ell = \mathcal{O}(k^d)$, the number of possible configurations grows as $\mathcal{O}(2^{k^d})$, making exhaustive search intractable. Instead, we propose a greedy policy that makes locally optimal decisions at each layer.

Specifically, we include a candidate node u (i.e., set $e_\ell(u) = 1$) only if it increases the reward, i.e., $\Delta\mathcal{R}(u) > 0$. The reward is defined as

$$\mathcal{R}(\mathcal{T}) \;=\; \frac{C_{\text{target}}}{C_{\text{spec}}}, \quad C_{\text{target}} = c_T \cdot L^{\text{tree}}(\mathcal{T}), \quad C_{\text{spec}} = C_{\text{draft}}(\mathcal{T}) + C_{\text{verify}}(\mathcal{T}), \tag{9}$$

and the marginal increments upon adding u are

$$\Delta C_{\text{target}}(u) \;=\; c_T{\cdot}\Delta L^{\text{tree}}(u), \qquad \Delta C_{\text{spec}}(u) \;=\; \Delta C_{\text{draft}}(u){+}\Delta C_{\text{verify}}(u). \tag{10}$$

Working with the log-reward $J = \log \mathcal{R}(\mathcal{T}) = \log C_{\text{target}} - \log C_{\text{spec}}$, the change upon adding u is

$$\Delta J(u) = \log\!\left(1 + \frac{\Delta C_{\text{target}}(u)}{C_{\text{target}}}\right) - \log\!\left(1 + \frac{\Delta C_{\text{spec}}(u)}{C_{\text{spec}}}\right) \approx \frac{\Delta C_{\text{target}}(u)}{C_{\text{target}}} - \frac{\Delta C_{\text{spec}}(u)}{C_{\text{spec}}}, \tag{11}$$

using $\log(1 + x) \approx x$ for small x. We include u if and only if $\Delta J(u) > 0$, i.e.,

$$\Delta J(u) = \alpha \cdot \frac{\Delta C_{\text{target}}(u)}{\Delta C_{\text{spec}}(u)} - \frac{C_{\text{target}}}{C_{\text{spec}}} > 0, \qquad \alpha \in (0, 1], \tag{12}$$

which recovers the standard condition when $\alpha = 1$. Here, $\alpha \in (0, 1]$ accounts for optimistic draft-based acceptance estimates under draft–target mismatch. The global terms C_{target} and C_{spec} are computed on the current tree using Eqn. (9). Next, we will compute the marginal terms in Eqn. (10) in 2 steps.

Step 1: Marginal benefit $\Delta C_{\text{target}}(u)$. To estimate $\Delta L^{\text{tree}}(u)$ efficiently, recall that L^{tree} averages the expected acceptance length over all root-to-leaf paths (Eqn. (2)). Accordingly, the marginal benefit of expanding u is diluted by $|\mathcal{P}|$:

$$\Delta L^{\text{tree}}(u) \;\approx\; \frac{1}{|\mathcal{P}|}\,\Delta L(u), \qquad \Delta L(u) \;=\; P(\tilde{\mathbf{x}}_u \mid \text{anc}(u)), \tag{13}$$

where $P(\tilde{\mathbf{x}}_u \mid \text{anc}(u))$ is the cumulative acceptance probability defined in Eqn. (3).
Step 2: Marginal cost $\Delta C_{\text{spec}}(u)$. We approximate the cost of adding one node ($\Delta n = 1$) by differentiating the fitted cost models with respect to the current tree size $|\mathcal{T}|$:

$$\Delta C_{\text{spec}}(u) \approx C'_{\text{draft}}(|\mathcal{T}|) + C'_{\text{verify}}(|\mathcal{T}|). \tag{14}$$

With $C_{\text{draft}}(\mathcal{T}) = \lambda|\mathcal{T}| + \beta$ and $C_{\text{verify}}(\mathcal{T}) = \gamma(\exp(\delta|\mathcal{T}|^{\rho}) - 1) + \eta$, this yields

$$\Delta C_{\text{spec}}(u) \approx \lambda + \gamma\delta\rho|\mathcal{T}|^{\rho-1}\exp(\delta|\mathcal{T}|^{\rho})\,. \tag{15}$$

Since we have calculated the marginal and global terms, we can decide whether to keep (expand) a candidate node u by checking $\Delta J(u) > 0$:

$$e_\ell(u) \;=\; \begin{cases} 1, & \text{if } \Delta J(u) \;=\; \alpha \cdot \dfrac{\Delta C_{\text{target}}(u)}{\Delta C_{\text{spec}}(u)} - \dfrac{C_{\text{target}}}{C_{\text{spec}}} \;>\; 0, \\ 0, & \text{otherwise.} \end{cases} \tag{16}$$

Now we present the full SMART algorithm. At each layer ℓ, we (i) generate top-k candidates per parent, (ii) compute each candidate's marginal benefit (Eqn. (13)) and marginal cost (Eqn. (15)), (iii) evaluate the current tree-level reward (Eqn. (9)), and (iv) apply the decision rule (Eqn. (16)) to determine which nodes to expand. The process continues until reaching the maximum depth d, exhausting the budget B, or when no candidate remains (i.e., $A_\ell = \emptyset$).

The greedy policy runs in $\mathcal{O}(kB)$ time: at each layer ℓ, we evaluate $\Delta J(u)$ in $\mathcal{O}(1)$ for each of the $k|A_{\ell-1}|$ candidates, and the total number of candidates across all layers is bounded by kB. This reduces the complexity from $\mathcal{O}(2^{k^d})$ for exhaustive search to linear in the verification budget B. While not globally optimal, the greedy policy produces high-quality trees by ensuring each expansion locally improves the reward. Algorithm 1 in the appendix summarizes our greedy construction procedure.

4 Experiments

Models & Datasets. We evaluate SMART across a diverse range of models to demonstrate its generalizability. For **MLLMs**, we evaluate LLaVA-1.5 (7B/13B) [22] and Qwen2VL-7B-Instruct [33] on widely used multimodal benchmarks: VQAv2 [2], AI2D [15], ScienceQA [23], ChartQA [25], TextVQA [27], and HallusionBench [10]. For **LLMs**, we report results on LLaMA-3.1-Instruct-8B [9], Vicuna-1.3-13B [7], DeepSeek-R1-Distill-LLaMA-8B [11], and LLaMA-3.3-70B [9] across three standard benchmarks covering chat, coding, and reasoning: MT-Bench [36], HumanEval [6], and GSM8K [8].
Baselines. For **MLLMs**, we integrate SMART into frameworks that utilize likelihood-maximizing tree construction (e.g., MSD [21]) and compare against Medusa [3], EAGLE [19], EAGLE-2 [18], and MSD. For **LLMs**, we integrate SMART into Eagle-3 [20] and compare against a broad suite of representative

Table 1: Comparison of speedup ratio SR and acceptance rate β on standard MLLM benchmarks with temperature $T \in \{0, 1\}$. The subscripts denote the relative improvement compared to the corresponding baseline. For example, at $T = 0$, MSD+SMART on LLaVA-1.5 7B achieves the average SR of 1.53 with an additional +29.7% gain over the MSD value of 1.18.

	Model	Method	VQAv2		AI2D		SQA Image		ChartQA		TextVQA		Hallusion		Average	
			$SR\uparrow$	$\beta\uparrow$	$SR\uparrow$	$\beta\uparrow$	$SR\uparrow$	$\beta\uparrow$	$SR\uparrow$	$\beta\uparrow$	$SR\uparrow$	$\beta\uparrow$	$SR\uparrow$	$\beta\uparrow$	$SR\uparrow$	$\beta\uparrow$
Temperature = 0	LLaVA-1.5 7B	Medusa	0.88	0.48	0.82	0.44	0.86	0.46	0.91	0.51	0.95	0.54	1.01	0.58	0.91	0.50
		EAGLE-1	0.95	0.52	0.88	0.48	0.92	0.50	0.98	0.55	1.02	0.58	1.08	0.62	0.97	0.54
		EAGLE-2	1.08	0.59	0.98	0.53	1.02	0.54	1.06	0.62	1.15	0.64	1.18	0.68	1.08	0.60
		MSD	1.23	0.67	1.09	0.58	1.09	0.56	1.14	0.68	1.26	0.68	1.26	0.71	1.18	0.65
		MSD+SMART	**1.55**	**0.81**	**1.45**	**0.74**	**1.44**	**0.72**	**1.59**	**0.82**	**1.55**	**0.79**	**1.62**	**0.83**	**1.53**$_{+29.7\%}$	**0.79**$_{+21.5\%}$
	LLaVA-1.5 13B	Medusa	0.95	0.44	0.88	0.41	1.01	0.46	1.10	0.54	0.91	0.42	0.88	0.44	0.96	0.45
		EAGLE-1	1.02	0.48	0.95	0.45	1.08	0.50	1.18	0.58	0.98	0.46	0.95	0.48	1.03	0.49
		EAGLE-2	1.15	0.54	1.08	0.50	1.18	0.54	1.32	0.62	1.12	0.51	1.08	0.51	1.16	0.54
		MSD	1.30	0.59	1.17	0.53	1.28	0.57	1.45	0.66	1.22	0.54	1.17	0.54	1.26	0.57
		MSD+SMART	**1.56**	**0.76**	**1.42**	**0.71**	**1.53**	**0.76**	**1.72**	**0.84**	**1.51**	**0.73**	**1.42**	**0.72**	**1.53**$_{+21.4\%}$	**0.75**$_{+31.6\%}$
	Qwen2VL 7B Instruct	Medusa	0.85	0.54	0.79	0.48	0.82	0.47	0.98	0.54	0.88	0.50	0.91	0.54	0.87	0.51
		EAGLE-1	0.92	0.58	0.85	0.52	0.88	0.51	1.05	0.58	0.95	0.54	0.98	0.58	0.94	0.55
		EAGLE-2	1.05	0.65	0.96	0.56	0.98	0.55	1.15	0.62	1.08	0.58	1.08	0.61	1.05	0.60
		MSD	1.18	0.71	1.05	0.58	1.06	0.57	1.24	0.65	1.16	0.60	1.15	0.63	1.14	0.62
		MSD+SMART	**1.22**	**0.86**	**1.24**	**0.77**	**1.26**	**0.74**	**1.34**	**0.84**	**1.24**	**0.57**	**1.22**	**0.82**	**1.25**$_{+9.6\%}$	**0.77**$_{+24.2\%}$
Temperature = 1	LLaVA-1.5 7B	Medusa	1.74	0.24	1.43	0.28	1.58	0.29	1.33	0.26	0.97	0.19	1.46	0.26	1.42	0.25
		EAGLE-1	1.85	0.26	1.52	0.30	1.68	0.31	1.42	0.28	1.05	0.21	1.55	0.28	1.51	0.27
		EAGLE-2	2.05	0.28	1.65	0.32	1.85	0.33	1.52	0.29	1.15	0.23	1.68	0.30	1.65	0.29
		MSD	2.21	0.30	1.77	0.34	2.00	0.35	1.63	0.31	1.22	0.24	1.77	0.31	1.77	0.31
		MSD+SMART	**2.84**	**0.40**	**2.41**	**0.45**	**2.56**	**0.46**	**2.24**	**0.43**	**1.46**	**0.33**	**2.19**	**0.41**	**2.28**$_{+28.8\%}$	**0.42**$_{+35.5\%}$
	LLaVA-1.5 13B	Medusa	1.67	0.27	1.39	0.30	1.91	0.33	1.36	0.26	1.07	0.20	1.23	0.24	1.44	0.27
		EAGLE-1	1.78	0.29	1.48	0.32	2.02	0.35	1.45	0.28	1.15	0.22	1.32	0.26	1.53	0.29
		EAGLE-2	1.95	0.31	1.58	0.33	2.25	0.36	1.58	0.29	1.25	0.24	1.42	0.27	1.67	0.30
		MSD	2.14	0.33	1.71	0.35	2.40	0.38	1.71	0.31	1.33	0.25	1.48	0.28	1.79	0.31
		MSD+SMART	**2.54**	**0.42**	**2.23**	**0.47**	**2.92**	**0.46**	**1.98**	**0.40**	**1.63**	**0.34**	**1.78**	**0.40**	**2.18**$_{+21.8\%}$	**0.41**$_{+32.3\%}$
	Qwen2VL 7B Instruct	Medusa	1.24	0.52	0.91	0.38	1.49	0.42	0.95	0.42	1.14	0.42	0.88	0.42	1.10	0.43
		EAGLE-1	1.32	0.55	0.98	0.41	1.58	0.45	1.02	0.45	1.22	0.45	0.95	0.45	1.18	0.46
		EAGLE-2	1.42	0.59	1.06	0.43	1.68	0.47	1.08	0.46	1.32	0.47	1.02	0.47	1.26	0.48
		MSD	1.50	0.62	1.12	0.44	1.79	0.48	1.15	0.48	1.38	0.48	1.07	0.48	1.33	0.50
		MSD+SMART	**1.69**	**0.65**	**1.25**	**0.53**	**2.04**	**0.55**	**1.16**	**0.58**	**1.45**	**0.56**	**1.10**	**0.50**	**1.45**$_{+9.0\%}$	**0.56**$_{+12.0\%}$

baselines, including SPS [16], EAGLE, EAGLE-2, GRIFFIN [13], and EAGLE-3. For a fair comparison, all methods are evaluated under the same hardware (RTX Pro 6000 GPUs) and decoding configurations. Due to limited space, we report results for integrating SMART into additional baselines in the appendix.

Metrics & Evaluation Protocol. Following prior work, we evaluate performance at decoding temperatures $T \in \{0, 1\}$. Since SMART is mathematically lossless, our evaluation focuses on efficiency via two key metrics. **(1) Speedup Ratio (SR):** The end-to-end wall-clock latency improvement relative to vanilla autoregressive decoding ($SR = 1.00\times$). **(2) Acceptance Rate (β):** The fraction of drafted tokens accepted during verification. We specifically report the acceptance *rate* rather than the acceptance *length*. Since SMART's pruning induces variable draft lengths across steps, raw per-step accepted token counts are no longer directly comparable; a normalized rate provides a more consistent measure of the draft model's efficiency relative to the chosen tree size.

Table 2: Comparison of speedup ratio (SR) and acceptance rate (β) on standard LLM benchmarks under temperature settings $T \in \{0, 1\}$. Subscripts denote the relative improvement over the corresponding baseline (EAGLE-3). For example, at $T = 0$ on LLaMA-3.1-Instruct-8B, EAGLE-3+SMART achieves an average SR of 1.59, representing a +16.9% improvement over the EAGLE-3 baseline value of 1.36.

Model	Method	Temperature = 0								Temperature = 1							
		MT-bench		HumanEval		GSM8K		Average		MT-bench		HumanEval		GSM8K		Average	
		$SR\uparrow$	$\beta\uparrow$	$SR\uparrow$	$\beta\uparrow$	$SR\uparrow$	$\beta\uparrow$	$SR\uparrow$	$\beta\uparrow$	$SR\uparrow$	$\beta\uparrow$	$SR\uparrow$	$\beta\uparrow$	$SR\uparrow$	$\beta\uparrow$	$SR\uparrow$	$\beta\uparrow$
LLaMA-3.1 Instruct 8B	SPS	0.49	0.24	0.54	0.27	0.43	0.21	0.49	0.24	0.86	0.17	0.92	0.18	0.75	0.17	0.84	0.17
	EAGLE	0.70	0.35	0.98	0.37	0.78	0.35	0.82	0.36	1.15	0.20	1.58	0.30	1.48	0.26	1.41	0.25
	EAGLE-2	1.01	0.47	1.34	0.54	1.08	0.48	1.14	0.50	1.43	0.27	2.12	0.42	1.82	0.33	1.79	0.34
	GRIFFIN	1.17	0.55	1.50	0.68	1.20	0.59	1.29	0.60	1.58	0.33	2.61	0.52	2.01	0.41	2.07	0.42
	EAGLE-3	1.35	0.67	1.44	0.74	1.28	0.68	1.36	0.70	1.65	0.43	2.45	0.51	2.22	0.48	2.11	0.47
	EAGLE-3+SMART	**1.56**	**0.67**	**1.71**	**0.74**	**1.51**	**0.68**	**1.59**$_{+16.9\%}$	**0.80**$_{+14.2\%}$	**1.84**	**0.47**	**2.73**	**0.57**	**2.56**	**0.52**	**2.38**$_{+12.8\%}$	**0.52**$_{+10.6\%}$
Vicuna-1.3 13B	SPS	0.48	0.25	0.53	0.28	0.42	0.21	0.48	0.25	0.40	0.16	0.43	0.17	0.35	0.15	0.39	0.16
	EAGLE	0.71	0.42	0.82	0.46	0.68	0.40	0.74	0.43	0.55	0.27	0.63	0.30	0.56	0.30	0.58	0.29
	EAGLE-2	0.95	0.54	1.19	0.60	0.95	0.52	1.03	0.55	0.88	0.38	0.97	0.42	0.74	0.38	0.81	0.40
	EAGLE-3	1.20	0.74	1.39	0.85	1.24	0.71	1.28	0.76	0.95	0.49	1.20	0.56	1.03	0.50	1.06	0.52
	EAGLE-3+SMART	**1.43**	**0.79**	**1.72**	**0.88**	**1.52**	**0.76**	**1.56**$_{+21.9\%}$	**0.81**$_{+6.6\%}$	**1.14**	**0.52**	**1.43**	**0.60**	**1.24**	**0.54**	**1.27**$_{+19.8\%}$	**0.55**$_{+5.8\%}$
DeepSeek R1 8B	SPS	0.52	0.26	0.58	0.29	0.46	0.22	0.52	0.26	0.46	0.18	0.49	0.19	0.40	0.17	0.45	0.18
	GRIFFIN	1.06	0.50	1.32	0.63	1.42	0.68	1.27	0.61	0.92	0.39	1.14	0.46	1.36	0.55	1.14	0.46
	EAGLE-3	1.24	0.61	1.49	0.71	1.61	0.76	1.46	0.70	1.06	0.48	1.22	0.51	1.56	0.58	1.28	0.52
	EAGLE-3+SMART	**1.45**	**0.69**	**1.65**	**0.77**	**1.87**	**0.82**	**1.68**$_{+15.1\%}$	**0.76**$_{+8.6\%}$	**1.21**	**0.51**	**1.34**	**0.55**	**1.62**	**0.59**	**1.39**$_{+8.6\%}$	**0.55**$_{+5.8\%}$
LLaMA 3.3 70B	SPS	0.98	0.25	1.09	0.28	0.86	0.21	0.98	0.25	0.81	0.17	0.86	0.18	0.70	0.16	0.79	0.17
	EAGLE-3	2.46	0.63	2.92	0.72	2.67	0.66	2.69	0.67	2.07	0.43	2.80	0.65	2.55	0.55	2.48	0.54
	EAGLE-3+SMART	**2.97**	**0.65**	**3.72**	**0.77**	**3.32**	**0.70**	**3.35**$_{+24.5\%}$	**0.70**$_{+4.5\%}$	**2.56**	**0.51**	**3.36**	**0.68**	**3.02**	**0.60**	**2.99**$_{+20.2\%}$	**0.60**$_{+11.1\%}$

4.1 Main results

Results on MLLMs. Table 1 shows that on multimodal benchmarks, SMART can substantially improve both the speedup ratio and the acceptance rate. For example, MSD+SMART consistently outperforms MSD, with average SR gains of +20.2% at T=0 and +19.9% at T=1 across three MLLMs. The largest improvements appear on LLaVA-1.5: at T=0, the average SR increases from 1.18× to 1.53× on LLaVA-1.5-7B and from 1.26× to 1.53× on LLaVA-1.5-13B. These gains are accompanied by higher acceptance rates: SMART preferentially expands nodes with higher expected acceptance, prunes low-benefit branches, and terminates early when no promising tokens remain. In contrast, MSD always selects a fixed number of draft tokens, even when additional tokens are unlikely to be accepted. SMART allocates the computation budget to more promising tokens and thereby improves verification efficiency.

Results on LLMs. Table 2 demonstrates that across four LLMs, SMART consistently improves both the speedup ratio and the acceptance rate. For instance, EAGLE-3+SMART achieves substantial speedups over EAGLE-3: +19.6% at T=0 and +15.4% at T=1 on average across LLMs. Improvements are consistent across chat (MT-Bench), coding (HumanEval), and math reasoning (GSM8K). On LLaMA-3.1-Instruct-8B, EAGLE-3+SMART increases the average SR from 1.36× to 1.59× at T=0 and from 2.11× to 2.38× at T=1. On the larger LLaMA-3.3-70B, SMART further raises the average SR from 2.69× to 3.35× at T=0 and from 2.48× to 2.99× at T=1. Overall, SMART delivers robust gains across diverse model scales and task families. We also observe acceptance-rate

Table 3: Speedup comparison between MSD and SMART across different GPUs and batch sizes. SMART maintains consistent speedup while MSD degrades at large batches.

GPU	Batch Size	MSD				MSD + SMART (Ours)			
		ChartQA	TextVQA	Hallusion	**Avg**	ChartQA	TextVQA	Hallusion	**Avg**
RTX Pro 6000	1	2.27×	2.09×	2.23×	2.20×	2.18×	2.10×	2.22×	2.17×
	8	1.88×	1.80×	1.83×	1.84×	1.98×	1.85×	1.96×	1.93×
	16	1.14×	1.26×	1.26×	1.22×	1.59×	1.55×	1.61×	1.58×
	24	0.98×	0.95×	0.96×	0.96×	1.51×	1.41×	1.44×	1.45×
	32	0.86×	0.79×	0.81×	0.82×	1.40×	1.38×	1.39×	1.39×
L40S	1	1.85×	1.79×	1.82×	1.82×	1.78×	1.76×	1.77×	1.77×
	4	1.65×	1.58×	1.67×	1.63×	1.67×	1.59×	1.68×	1.65×
	8	1.25×	1.20×	1.21×	1.22×	1.52×	1.44×	1.53×	1.50×
	12	0.91×	0.89×	0.90×	0.90×	1.40×	1.37×	1.42×	1.40×

Table 4: Speedup across different token budgets on RTX Pro 6000 with batch size of 16. Budget of 200 achieves optimal performance, balancing tree size and verification cost. Lower budgets (100) under-utilize parallelism while higher budgets (300-400) incur excessive verification overhead.

Token Budget	T = 0				T = 1			
	ChartQA	**TextVQA**	**Hallusion**	**Avg**	**ChartQA**	**TextVQA**	**Hallusion**	**Avg**
100	1.39×	1.42×	1.47×	1.43×	1.95×	1.33×	2.05×	2.06×
200	1.60×	1.56×	1.57×	**1.58**×	2.24×	1.46×	2.19×	**2.28**×
300	1.33×	1.24×	1.27×	1.28×	1.86×	1.16×	1.77×	1.85×
400	1.32×	1.24×	1.26×	1.27×	1.85×	1.16×	1.76×	1.83×

improvements across all four LLMs, suggesting SMART reduces wasted draft expansions.

4.2 Ablation Study

Scaling with batch size on different hardwares. Table 3 shows that SMART provides limited gains at small batch sizes, where decoding is largely memory-bound and likelihood-maximizing trees already contain most of the "useful" draft tokens. In this regime, SMART mainly prunes low-utility tokens but does not create additional high-utility candidates beyond those already present, leading to near ties with MSD (e.g., RTX Pro 6000 at batch 1: 2.17× vs. 2.20×). As batch size increases, execution becomes increasingly compute-bound and verification overhead grows super-linearly, making every selected draft token expensive. In this setting, MSD degrades sharply (RTX Pro 6000: 2.20× → 0.82× from batch 1 to 32; L40S: 1.82× → 0.90× from batch 1 to 12), whereas SMART remains robust by allocating the budget to truly beneficial tokens and avoiding wasteful verification. Consequently, SMART maintains $> 1\times$ speedup throughout, retain-

Table 5: Speedup ablation over discount factor α on RTX Pro 6000 at batch size 16. $\alpha \in [0.7, 0.9]$ achieves optimal performance, balancing conservative pruning with sufficient tree expansion.

α	**Temperature = 0**				**Temperature = 1**			
	ChartQA	TextVQA	Hallusion	Avg	ChartQA	TextVQA	Hallusion	Avg
1.0	1.57×	1.46×	1.51×	1.51×	1.45×	1.38×	1.42×	1.42×
0.9	1.58×	1.51×	1.57×	1.55×	1.52×	1.44×	1.48×	1.48×
0.8	1.58×	1.52×	1.57×	**1.56**×	1.53×	1.45×	1.49×	**1.49**×
0.7	1.58×	1.50×	1.57×	1.55×	1.51×	1.43×	1.47×	1.47×
0.6	1.57×	1.50×	1.56×	1.54×	1.49×	1.41×	1.45×	1.45×
0.5	1.57×	1.50×	1.55×	1.54×	1.46×	1.38×	1.43×	1.42×

ing 1.58× at batch 16 and 1.39× at batch 32 on RTX Pro 6000, and 1.50× at batch 8 and 1.40× at batch 12 on L40S.

Verification token budget. Table 4 ablates the verification token budget on RTX Pro 6000 with batch size of 16. Performance exhibits a clear optimum at budget 200, which yields the best average speedup (1.58× at T=0 and 2.28× at T=1). Lower budget (100) reduces speedup (1.43× at T=0), indicating underutilization of available parallelism due to overly aggressive pruning. Larger budgets (300–400) significantly hurt speedup ($\approx$ 1.27–1.28× at T=0), as additional drafted tokens increase verification work and dominate end-to-end latency. Overall, these results support allocating a moderate per-sequence verification budget that balances tree expansion against verification overhead.

Discount factor α. Table 5 ablates the discount factor α used to conservatively down-weight predicted marginal benefit under draft–target mismatch. Across both temperatures, performance remains stable over a wide range of moderate discounts: $\alpha \in [0.7, 0.9]$ yields the best or near-best average speedup, peaking at $\alpha = 0.8$ (1.56× at T=0 and 1.49× at T=1). Setting α too large (e.g., α = 1.0) is overly permissive, leading to aggressive expansion and higher verification overhead, which reduces speedup (1.51× at T=0 and 1.42× at T=1). Conversely, smaller α values prune more aggressively and may discard useful draft tokens. Overall, SMART is insensitive within a wide operating range, and we use $\alpha = 0.8$ by default.

5 Conclusion

We presented **SMART**, a training-free, system-aware framework for speedup-maximizing draft tree construction in speculative decoding. Motivated by the efficiency paradox that large draft trees can incur super-linear drafting and verification overhead, SMART casts tree growth as a sequential decision problem. By expanding nodes only when their marginal benefit–cost ratio exceeds the tree-level speedup under a per-sequence budget, SMART reduces wasteful drafting and verification and maintains robust wall-clock gains across compute-bound

batching regimes and diverse GPU architectures. Across LLMs and MLLMs, SMART consistently improves strong tree-based backbones while preserving the lossless guarantee, delivering average additional speedups of **20.0%** (MLLMs) and **15.4%** (LLMs) without performance degradation.

Limitation Discussion. Due to limited compute resources, we only evaluate SMART on RTX Pro 6000 and L40S GPUs, and do not include other data-center accelerators such as A100, H100, or H200. Nonetheless, SMART is system-aware and hardware-agnostic by design, and we expect similar speedup trends on these architectures.

6 Acknowledgement

This work was supported by the Singapore Ministry of Education (MOE) Academic Research Fund (AcRF) Tier 1 grant (Proposal ID: 25-SIS-SMU-003).

References

1. Achiam, J., Adler, S., Agarwal, S., Ahmad, L., Akkaya, I., Aleman, F.L., Almeida, D., Altenschmidt, J., Altman, S., Anadkat, S., et al.: Gpt-4 technical report. arXiv preprint arXiv:2303.08774 (2023)
2. Antol, S., Agrawal, A., Lu, J., Mitchell, M., Batra, D., Zitnick, C.L., Parikh, D.: Vqa: Visual question answering. In: Proceedings of the IEEE international conference on computer vision. pp. 2425–2433 (2015)
3. Cai, T., Li, Y., Geng, Z., Peng, H., Lee, J.D., Chen, D., Dao, T.: Medusa: Simple llm inference acceleration framework with multiple decoding heads. arXiv preprint arXiv:2401.10774 (2024)
4. Chen, B., Xu, Z., Kirmani, S., Ichter, B., Sadigh, D., Guibas, L., Xia, F.: Spatialvlm: Endowing vision-language models with spatial reasoning capabilities. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. pp. 14455–14465 (2024)
5. Chen, C., Borgeaud, S., Irving, G., Lespiau, J.B., Sifre, L., Jumper, J.: Accelerating large language model decoding with speculative sampling. arXiv preprint arXiv:2302.01318 (2023)
6. Chen, M., Tworek, J., Jun, H., Yuan, Q., Pinto, H.P.D.O., Kaplan, J., Edwards, H., Burda, Y., Joseph, N., Brockman, G., et al.: Evaluating large language models trained on code. arXiv preprint arXiv:2107.03374 (2021)
7. Chiang, W.L., Li, Z., Lin, Z., Sheng, Y., Wu, Z., Zhang, H., Zheng, L., Zhuang, S., Zhuang, Y., Gonzalez, J.E., et al.: Vicuna: An open-source chatbot impressing gpt-4 with 90%* chatgpt quality. See https://vicuna. lmsys. org (accessed 14 April 2023) **2**(3), 6 (2023)
8. Cobbe, K., Kosaraju, V., Bavarian, M., Chen, M., Jun, H., Kaiser, L., Plappert, M., Tworek, J., Hilton, J., Nakano, R., et al.: Training verifiers to solve math word problems. arXiv preprint arXiv:2110.14168 (2021)
9. Grattafiori, A., Dubey, A., Jauhri, A., Pandey, A., Kadian, A., Al-Dahle, A., Letman, A., Mathur, A., Schelten, A., Vaughan, A., et al.: The llama 3 herd of models. arXiv preprint arXiv:2407.21783 (2024)

10. Guan, T., Liu, F., Wu, X., Xian, R., Li, Z., Liu, X., Wang, X., Chen, L., Huang, F., Yacoob, Y., et al.: Hallusionbench: an advanced diagnostic suite for entangled language hallucination and visual illusion in large vision-language models. In: Proceedings of the IEEE/CVF conference on computer vision and pattern recognition. pp. 14375–14385 (2024)
11. Guo, D., Yang, D., Zhang, H., Song, J., Wang, P., Zhu, Q., Xu, R., Zhang, R., Ma, S., Bi, X., et al.: Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. arXiv preprint arXiv:2501.12948 (2025)
12. Hu, S., Li, J., Lu, Z., Zhou, P.: Bridging draft policy misalignment: Group tree optimization for speculative decoding. arXiv preprint arXiv:2509.22134 (2025)
13. Hu, S., Li, J., Xie, X., Lu, Z., Toh, K.C., Zhou, P.: Griffin: Effective token alignment for faster speculative decoding. arXiv preprint arXiv:2502.11018 (2025)
14. Huang, K., Guo, X., Wang, M.: Specdec++: Boosting speculative decoding via adaptive candidate lengths. arXiv preprint arXiv:2405.19715 (2024)
15. Kembhavi, A., Salvato, M., Kolve, E., Seo, M., Hajishirzi, H., Farhadi, A.: A diagram is worth a dozen images. In: European conference on computer vision. pp. 235–251. Springer (2016)
16. Leviathan, Y., Kalman, M., Matias, Y.: Fast inference from transformers via speculative decoding. In: International Conference on Machine Learning. pp. 19274–19286. PMLR (2023)
17. Li, F., Zhang, R., Zhang, H., Zhang, Y., Li, B., Li, W., Ma, Z., Li, C.: Llava-next-interleave: Tackling multi-image, video, and 3d in large multimodal models. arXiv preprint arXiv:2407.07895 (2024)
18. Li, Y., Wei, F., Zhang, C., Zhang, H.: Eagle-2: Faster inference of language models with dynamic draft trees. In: Proceedings of the 2024 conference on empirical methods in natural language processing. pp. 7421–7432 (2024)
19. Li, Y., Wei, F., Zhang, C., Zhang, H.: Eagle: Speculative sampling requires rethinking feature uncertainty. arXiv preprint arXiv:2401.15077 (2024)
20. Li, Y., Wei, F., Zhang, C., Zhang, H.: Eagle-3: Scaling up inference acceleration of large language models via training-time test. arXiv preprint arXiv:2503.01840 (2025)
21. Lin, L., Lin, Z., Zeng, Z., Ji, R.: Speculative decoding reimagined for multimodal large language models. arXiv preprint arXiv:2505.14260 (2025)
22. Liu, H., Li, C., Wu, Q., Lee, Y.J.: Visual instruction tuning. Advances in neural information processing systems **36**, 34892–34916 (2023)
23. Lu, P., Mishra, S., Xia, T., Qiu, L., Chang, K.W., Zhu, S.C., Tafjord, O., Clark, P., Kalyan, A.: Learn to explain: Multimodal reasoning via thought chains for science question answering. Advances in neural information processing systems **35**, 2507–2521 (2022)
24. Mamou, J., Pereg, O., Korat, D., Berchansky, M., Timor, N., Wasserblat, M., Schwartz, R.: Dynamic speculation lookahead accelerates speculative decoding of large language models. arXiv preprint arXiv:2405.04304 (2024)
25. Masry, A., Do, X.L., Tan, J.Q., Joty, S., Hoque, E.: Chartqa: A benchmark for question answering about charts with visual and logical reasoning. In: Findings of the association for computational linguistics: ACL 2022. pp. 2263–2279 (2022)
26. Miao, X., Oliaro, G., Zhang, Z., Cheng, X., Wang, Z., Zhang, Z., Wong, R.Y.Y., Zhu, A., Yang, L., Shi, X., et al.: 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. pp. 932–949 (2024)

27. Singh, A., Natarajan, V., Shah, M., Jiang, Y., Chen, X., Batra, D., Parikh, D., Rohrbach, M.: Towards vqa models that can read. In: Proceedings of the IEEE/CVF conference on computer vision and pattern recognition. pp. 8317–8326 (2019)
28. Sridhar, A., Sinnadurai, N., Lie, S., Thangarasa, V.: Tapout: A bandit-based approach to dynamic speculative decoding. arXiv preprint arXiv:2511.02017 (2025)
29. Sun, P., Jiang, Y., Chen, S., Zhang, S., Peng, B., Luo, P., Yuan, Z.: Autoregressive model beats diffusion: Llama for scalable image generation. arXiv preprint arXiv:2406.06525 (2024)
30. Sun, Z., Suresh, A.T., Ro, J.H., Beirami, A., Jain, H., Yu, F.: Spectr: Fast speculative decoding via optimal transport. Advances in Neural Information Processing Systems **36**, 30222–30242 (2023)
31. Touvron, H., Lavril, T., Izacard, G., Martinet, X., Lachaux, M.A., Lacroix, T., Rozière, B., Goyal, N., Hambro, E., Azhar, F., et al.: Llama: Open and efficient foundation language models. arXiv preprint arXiv:2302.13971 (2023)
32. Touvron, H., Martin, L., Stone, K., Albert, P., Almahairi, A., Babaei, Y., Bashlykov, N., Batra, S., Bhargava, P., Bhosale, S., et al.: Llama 2: Open foundation and fine-tuned chat models. arXiv preprint arXiv:2307.09288 (2023)
33. Wang, P., Bai, S., Tan, S., Wang, S., Fan, Z., Bai, J., Chen, K., Liu, X., Wang, J., Ge, W., Fan, Y., Dang, K., Du, M., Ren, X., Men, R., Liu, D., Zhou, C., Zhou, J., Lin, J.: Qwen2-vl: Enhancing vision-language model's perception of the world at any resolution. arXiv preprint arXiv:2409.12191 (2024)
34. Zhang, L., Wang, X., Huang, Y., Xu, R.: Learning harmonized representations for speculative sampling. arXiv preprint arXiv:2408.15766 (2024)
35. Zhang, Z., Xu, J., Liang, T., Chen, X., He, Z., Wang, R., Tu, Z.: Draft model knows when to stop: Self-verification speculative decoding for long-form generation. In: Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing. pp. 16696–16708 (2025)
36. Zheng, L., Chiang, W.L., Sheng, Y., Zhuang, S., Wu, Z., Zhuang, Y., Lin, Z., Li, Z., Li, D., Xing, E., et al.: Judging llm-as-a-judge with mt-bench and chatbot arena. Advances in neural information processing systems **36**, 46595–46623 (2023)
37. Zhu, D., Chen, J., Shen, X., Li, X., Elhoseiny, M.: Minigpt-4: Enhancing vision-language understanding with advanced large language models. arXiv preprint arXiv:2304.10592 (2023)