UBTREE: PARALLEL TREE DRAFTING VIA UNIGRAM AND BIGRAM MODELS FOR SPECULATIVE DECODING

Anonymous authors
Paper under double-blind review

ABSTRACT

Speculative decoding accelerates language model inference by verifying multiple draft tokens predicted by lightweight draft models in a single target-model forward pass. Recent parallel drafters have achieved breakthrough performance in frontier production models, yet their gains diminish as the target distribution becomes more entropic, particularly for later positions within draft blocks and under long-context settings, where limited draft diversity increasingly constrains acceptance. To overcome this bottleneck without sacrificing parallelism, we introduce **UBTree**, a parallel drafter that couples a **U**nigram proposer with a **B**igram selector to construct drafting **Tree**s. The unigram proposer is trained with the standard cross-entropy objective to generate candidate tokens independently for each position, while a lightweight bigram selector predicts transition scores between adjacent candidate pairs. Unlike the proposer, the selector is trained with a renormalized KL objective on high-temperature data. This *tree-native* training broadens the supervision beyond the greedy path, encouraging plausible alternative branches that improve the chance of accepting additional tokens during tree verification. Across seven standardized benchmarks with Qwen3-4B and Qwen3-8B, UBTree achieves an average speedup of 5.84–$6.94\times$ over autoregressive decoding and outperforms DARTree in all 28 comparisons. Production-scale evaluation further demonstrates UBTree's advantage over frontier baselines such as DSpark.

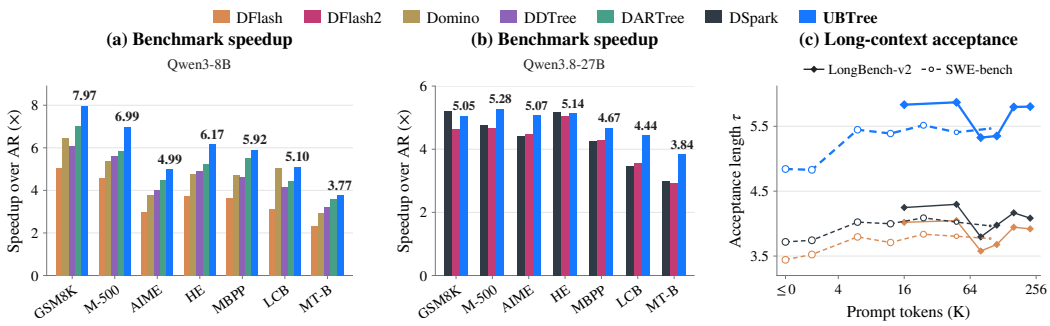

Figure 1: **(a) Speedups on Qwen3-8B:** Speedup over autoregressive decoding across seven benchmarks on Qwen3-8B instruct at $T_{\text{infer}} = 1$. **(b) Speedups on Qwen3.8-27B:** Serving speedup in SGLang on Qwen3.8-27B at $T_{\text{infer}} = 1$. **(c) Long-context acceptance:** Acceptance length τ on Ling3-Flash-124B for LongBench-v2 and SWE-bench over different context lengths. UBTree outperforms competing methods across most benchmarks and model scales.

1 INTRODUCTION

Speculative decoding (Leviathan et al., 2023; Chen et al., 2023) offers a lossless approach to accelerating large language models (LLMs): a lightweight drafter proposes multiple future tokens, and the target model verifies them together while preserving its output distribution. The benefit of speculative decoding depends not only on the acceptance length of the draft by the target model but also on the computational cost. Longer accepted drafts do not necessarily yield greater speedup if producing them incurs substantial latency.

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

Recent advances in speculative decoding have been driven by breakthroughs in parallel drafting (Liu et al., 2026; Chen et al., 2026). While their autoregressive counterparts (Li et al., 2024a; 2026d) condition each draft token sequentially on preceding choices, parallel drafters predict all future positions in a drafting block with a single proposer-backbone forward pass. Parallelism reduces the sequential overhead of using larger proposer backbones which increases the acceptance length. However, parallel drafters lack draft diversity. Consequently, their performance degrades when high-entropy target distributions inherently produce different paths. Moreover, the missing token-wise dependency may worsen the degradation by mixing these different paths. Existing approaches try to complement this drawback by restoring token-wise dependencies to improve precisions of the single-path draft through lightweight autoregressive correction heads (Huang et al., 2026; Cheng et al., 2026) or intra-backbone designs (Inco AI, 2026). However, these strategies do not fundamentally solve the diversity bottleneck because there is still only one single draft path covered. This particularly limits the further improvement of parallel drafters and their applications under scenarios with high entropy target distributions (Devic et al., 2025; Yang et al., 2026). **Draft diversity thus becomes a central bottleneck for current parallel speculative decoding.**

A natural solution to this bottleneck is tree drafting (Miao et al., 2024; Cai et al., 2024; Li et al., 2024b), which retains multiple draft paths as tree branches, increasing the chance that target verification matches one of them. By providing diverse yet plausible candidate drafts, tree drafting is more robust in high-entropy scenarios. Combining tree drafting and parallel drafters by constructing budgeted trees from pretrained block-parallel drafters achieves considerable speedups (Ringel & Romano, 2026; Li et al., 2026c). Nevertheless, these methods exhibit two limitations. First, the tension between token-wise dependency and full parallelism constrains the tradeoff between draft quality and drafting latency. Existing tree-based methods use autoregressive heads to capture token-wise dependencies, which precludes full parallelization. Conversely, omitting token-wise dependency limits the acceptance length and, consequently, the end-to-end speedup. Second, tree-based parallel methods suffer from a *train–inference mismatch*: they directly apply parallel drafters trained for single-path drafting to multi-path tree verification. Such drafters are optimized to produce the top draft for single-path verification, but are suboptimal for proposing multiple plausible paths for tree verification. Consequently, although lower-ranked alternatives in the draft tree can populate additional branches, they are not sufficiently optimized to align with the target model's alternatives or worth allocating verification budget to.

We address these limitations with **UBTree**, a tree-based drafter with a parallel bigram selector to model token-wise dependencies and tree-native training to close the train–inference gap. We incorporate two designs in UBTree:

- **Architecture.** UBTree separates drafting into three complementary stages and keeps all neural computation parallel. First, we use DFlash (Chen et al., 2026) as a *unigram proposer* to independently generate candidate tokens for every position in a drafting block. Second, a redesigned *bigram selector* scores adjacent candidate-token pairs conditioned on the proposer hidden states. It derives predecessor and successor representations from frozen target embeddings through separate MLPs and learns depth-dependent weights for combining proposer logits and selector scores. Third, we use the resulting transition scores to construct the verification tree. By simplifying the token-wise dependency into bigram correlation, we overcome the challenge of parallelizing token-wise dependency in tree drafting.
- **Training.** Unlike other drafters trained on target-regenerated rollouts at the inference temperature, we use a renormalized KL training objective on high-temperature regenerated target trajectories to diversify the positive training labels. This *tree-native* training strategy allocates more supervision signals to plausible alternative paths in the target distribution and distinguishes them better from low-quality alternative paths in tree construction, thereby relieving the train–inference mismatch in tree drafting.

We extensively evaluate UBTree on both academic and production-scale benchmarks. UBTree outperforms all existing methods on both academic-scale and production benchmarks. It achieves an average decoding speedup of 5.84–$6.94\times$ over autoregressive decoding on Qwen3-4B and Qwen3-8B with increased acceptance lengths. It also rivals frontier baselines such as DSpark (Cheng et al., 2026) in the acceleration of production LLMs like Ling3-Flash-124B. Additionally, UBTree shows outstanding durability on long-context tasks compared to baselines. In summary, UBTree provides a frontier solution for accelerating LLMs in both academic settings and production environments.

2 BACKGROUND

Speculative decoding. Given a verified prefix $x_{\leq t}$, a lightweight draft model proposes γ future tokens, which the target model verifies in one forward pass (Leviathan et al., 2023; Chen et al., 2023). Each round accepts consecutive draft tokens from the start of the proposed block until the first rejection. Let $\tau \in [1, \gamma + 1]$ denote the average of accepted tokens per round or *acceptance length*, including a token x_0 prefilled by the last verification. With T_{draft} covering proposal construction and T_{verify} covering verification and commit work, the average per-token *latency* and *speedup* are (Chen et al., 2026; Huang et al., 2026)

$$L_{\text{spec}} = \frac{T_{\text{draft}} + T_{\text{verify}}}{\tau}, \qquad \eta = \frac{L_{\text{target}}}{L_{\text{spec}}}, \tag{1}$$

where L_{target} is the per-token latency of target-only autoregressive decoding.

DFlash & DFlash 2. DFlash (Chen et al., 2026) proposes a γ-token block in parallel, conditioned on target hidden features through key-value context of one target prefilling forward. We denote the prefilled token x_0 by *anchor token*. For each future draft position $i = 1, ..., \gamma$, one proposer forward pass produces final hidden state h_i, which is subsequently mapped to proposer logit u_i through the frozen target LM head. The draft tokens of DFlash are predicted in parallel from masks where little token-wise dependencies are involved. DFlash 2 (Inco AI, 2026) fixes this by adding convolutions to the DFlash backbone and a low-rank bigram selector. For predecessor a and successor b at position i, the selector score $\delta_i(a, b)$ is added to the proposer logit $u_i(b)$ to obtain a combined score:

$$\delta_i(a, b) = \big(P(h_i) \odot \phi(a)\big)^\top \psi(b), \qquad u'_i(a, b) = u_i(b) + \delta_i(a, b), \tag{2}$$

where $\odot$ stands for Hadamard product. Here, the linear projector P maps h_i to a low-dimensional space, and ϕ and ψ are low-rank embeddings for predecessor and successor tokens, respectively. The selector scores all adjacent pairs from the position-wise top-K candidate sets in parallel; token selection then follows a single chain using the combined scores.

Tree drafting. DFlash and DFlash 2 are *single-path* drafting methods, which propose a single γ-token draft path to match a γ-token target path for the target model to verify. *Tree drafting*, by contrast, constructs a draft tree storing multiple candidate draft paths. Taking candidate tokens as nodes, different paths share nodes across continuations with common prefixes. We then use tree attention to verify the whole draft tree in one target-model pass and picks the path with the longest accepted token continuations by the target model in the tree (Miao et al., 2024; Cai et al., 2024). This verification procedure maintains the lossless property of speculative decoding, and accepted path need not be the proposer's top-ranked.

DDTree (Ringel & Romano, 2026) and DARTree (Li et al., 2026c) construct budgeted trees from block-parallel proposals. DDTree uses best-first search, ranking paths by accumulated position-wise token log-probabilities. DARTree corrects token log-probabilities by a lightweight autoregressive head and ranks the cumulative log-probabilities with a depth decay and a W node budget at each depth. These retained nodes accumulate into a candidate supertree of maximum depth γ, from which the B highest-scoring non-root nodes are selected for verification. Correction at the next depth still depends on the tokens selected at the preceding depth, leading to extra overheads.

3 METHOD

Two key designs of UBTree are 1) parallelizing token-wise dependency modeling in tree drafting and 2) training the tree drafter towards optimal tree drafting. Section 3.1 first explains how we incorporate token-dependencies with fully parallelized neural compuation in tree drafting. We use two parallel neural forwards to propose candidate tokens at each block position (Section 3.1.1) and select transition paths among these candidates (Section 3.1.2). The verification tree is then built from paths selected by the selector (Section 3.1.3). Section 3.2 presents our *tree-native* training, which regenerates the target rollouts for training under high temperatures and trains the drafter with a renormalized KL objective. We summarize these two designs of UBTree in Figure 2.

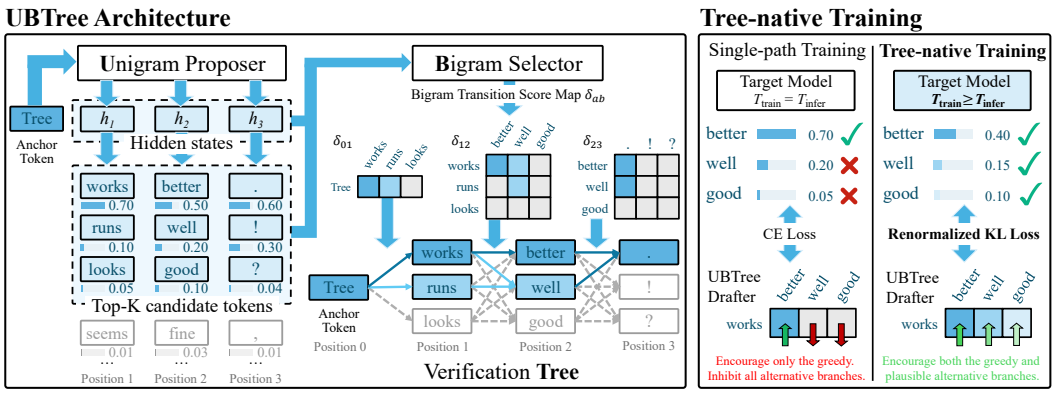

Figure 2: Overview of UBTree. **Left: UBTree architecture.** The unigram proposer generates top-K candidates at each draft position in parallel. The bigram selector scores adjacent candidate transitions in parallel, and tree construction uses these precomputed scores to build a verification tree. **Right: tree-native training.** Our training combines high-temperature target traces with renormalized KL supervision to encourage plausible alternative branches.

3.1 UBTree Architecture

3.1.1 Unigram Proposer

The unigram proposer is designed to propose candidate tokens at each position without considering cross-token dependencies. Let c be the key-value context from verified prefixes and x_0 be the prefilled anchor token. The unigram proposer can be formalized by

$$q_i(x_i \mid x_0, c) = \text{softmax}(u_i(x_0, c)), \quad i = 1, 2, ..., \gamma, \tag{3}$$

where i denotes the position in the block. Since the proposal at each position does not depend on each other, its neural forward could be easily parallelized. In practice, we deploy a pretrained DFlash backbone (Chen et al., 2026) as our unigram proposer, where its output logits are $u_i(x_0, c)$ and probabilities are $q_i(x_i \mid x_0, c)$. We also denote its predicted hidden states by h_i. We do not use DFlash2 backbone (Inco AI, 2026) because it differs from DFlash backbone by strengthening token-wise connection, which is not the task of our unigram proposal.

Candidate sets. We pick the tokens with top-K probabilities in q_i at each position as candidate tokens of this position $\mathcal{C}_i$, with $\mathcal{C}_0 = \{x_t\}$. The Cartesian product $\mathcal{C}_1 \times \cdots \times \mathcal{C}_\gamma$ forms K^γ candidate paths. Enumerating these paths is infeasible, so UBTree scores their adjacent transitions and searches a bounded subset by the bigram selector in the next section.

3.1.2 Bigram Selector

Inspired by DFlash 2 (Inco AI, 2026), we incorporate a lightweight bigram selector scores adjacent predecessor–successor token pairs from candidates, conditioned on the proposer hidden state h_i at the successor position. This trainable selector models token-wise dependencies in a parallel manner. To improve the selector, we redesign its architecture with deep codebooks and depth calibration.

Deep codebooks. For predecessor token a and successor candidate b at position i, DFlash 2 learns two vocabulary-wide codebooks ϕ and ψ by two embedding layers (See equation 2). UBTree instead derives the codebooks from the frozen target token embedding $e_v \in \mathbb{R}^{d_e}$ and two bias-free MLPs:

$$\phi(v) = W_{\phi,2}\ \text{SiLU}(W_{\phi,1} e_v), \qquad \psi(v) = W_{\psi,2}\ \text{SiLU}(W_{\psi,1} e_v). \tag{4}$$

At inference, we precompute both MLPs once and store them as a vocabulary-wide lookup table. While keeping inference efficiency, our deep codebook introduces nonlinearity and makes use of features in the target embedding. Hence, it improves the selector performance.

Depth calibration. DFlash 2 combines proposer logits and selector scores with fixed unit weights. However, their reliability can vary across draft depths. UBTree therefore learns two positive, depth-dependent scales:

$$s_i(a, b) = \alpha_i u_i(b) + \lambda_i \delta_i(a, b), \qquad \alpha_i = \exp(\rho_i), \quad \lambda_i = \exp(\kappa_i). \tag{5}$$

The ratio λ_i/α_i controls the weight of selector scores relative to proposer logits, while their common magnitude controls the concentration of the normalized successor distribution. These scales depend only on depth and preserve parallel edge scoring.

Parallel computation. Once the proposer provides the candidate sets and hidden states, we compute transition scores for all $K + (\gamma - 1)K^2$ predecessor-successor token pairs in parallel. At depth i, let $\mathbf{\Phi}_{i-1}$ have rows $\phi(a)^\top$ for predecessor candidates $a \in \mathcal{C}_{i-1}$, and let $\mathbf{\Psi}_i$ have rows $\psi(b)^\top$ for successor candidates $b \in \mathcal{C}_i$. With proposer's hidden state h_i at the successor position, we obtain

$$\mathbf{\Delta}_i = \mathbf{\Phi}_{i-1} \operatorname{diag}\big(P(h_i)\big) \mathbf{\Psi}_i^\top \in \mathbb{R}^{|\mathcal{C}_{i-1}| \times K}. \tag{6}$$

Here, $\operatorname{diag}(\cdot)$ places the input vector on the diagonal. The entry at the row for a and column for b is the selector score $\delta_i(a, b)$. After combining $\delta_i(a, b)$ with proposer logits $u_i(b)$ via equation 5, tree construction can proceed depth by depth sequentially without any neural computation.

3.1.3 Tree Construction and Verification

With unique scores for every transition $s_i(a, b)$ in $C_{i-1} \times C_i$, we are able to score all K^γ candidate paths consisting of these transitions without neural computation. Specifically, we can normalize $s_i(a, b)$ into log-probabilities over successor candidates, then rank paths by cumulative log-probabilities. However, to save the computation cost, we use DARTree's depth-wise progressive tree construction and global pruning procedure (Li et al., 2026c) instead of traversing all possible paths. After tree construction, target model will verify the whole tree in one-forward pass via tree attention, and keep the longest accepted continuation for lossless acceleration. Appendix A specifies the normalization, path scoring, search and verification procedure.

3.2 Tree-native Training

Tree drafting recommends multiple paths for target verification, rather than only proposing the greedy path. Hence, we need to diversify positive training signals to help the selector better recognize alternative plausible paths other than the greedy one. To this end, we use higher temperature T_{train} to regenerate the training data than the inference temperature T_{infer} and supervise the training by a renormalized KL objective.

Training with $T_{\text{train}} \geq T_{\text{infer}}$ for different T_{infer}. The training temperature T_{train} controls both the continuation distribution seen by the proposer and selector and the target distribution used for KL supervision. Temperature-matched training would set $T_{\text{train}} = T_{\text{infer}}$. As mentioned above, the selector needs to improve the plausibility of multiple paths in the tree, rather than focusing only on the top path. We can therefore make $T_{\text{train}} \geq T_{\text{infer}}$ to make those high-probability alternative paths more distinguishable. T_{train} then becomes an independent hyper-parameter to be tuned empirically. The best T_{train} amplifies the probability gap between plausible and other choices. Interestingly, T_{train} is disentangled with T_{infer} to some extent, that one UBTree drafter trained on T_{train} can be used in different T_{infer}, which simplifies the drafter adaptation. **We therefore use drafters trained at $T_{\text{train}} = 1$ for both $T_{\text{infer}} = 0$ and $T_{\text{infer}} = 1$ in our experiments.**

Renormalized KL loss. To train the selector to better discriminate between alternates other than the top choice, we use the forward KL divergence between the renormalized probabilities (Zhu et al., 2026) of the target and that of the selector over the proposer's top-K candidate tokens instead of the classical cross-entropy (CE) loss. For each valid future position, let $z_i(v)$ be the target logit and $s_i(y^*_{i-1}, v)$ be the selector logit given the predecessor token y^*_{i-1}. For $T_{\text{train}} > 0$, we define the renormalized target probabilities $\widetilde{p}_i(v)$ at training temperature T_{train} and the selector probabilities $\widetilde{q}_i(v)$ on the top-K token support $\mathcal{C}_i$:

$$\widetilde{p}_i(v) = \frac{\exp(z_i(v)/T_{\text{train}})}{\sum_{b \in \mathcal{C}_i} \exp(z_i(b)/T_{\text{train}})}, \qquad \widetilde{q}_i(v) = \frac{\exp(s_i(y^*_{i-1}, v))}{\sum_{b \in \mathcal{C}_i} \exp(s_i(y^*_{i-1}, b))}. \tag{7}$$

For $T_{\text{train}} = 0$, the target distribution is understood in the limit $T_{\text{train}} \to 0^+$. The forward KL divergence between $\widetilde{p}_i(v)$ and $\widetilde{q}_i(v)$ constitutes the main term of our training objective. An auxiliary cross-entropy term, also used in the first-stage proposer training, is applied solely to the proposer q_i to maintain the candidate distribution. Using weight β to balance two terms, the joint objective is

$$\mathcal{L} = \mathrm{KL}(\widetilde{p}_i | \widetilde{q}_i) - \beta \log q_i(y^*_i \mid x_{\leq t}), \qquad \beta = 0.1. \tag{8}$$

4 EXPERIMENTS

We evaluate UBTree on benchmarks and report decoding speedup η over autoregressive baseline and average acceptance length τ. Avg. means average results across all listed benchmarks.

4.1 ACADEMIC-SCALE EXPERIMENT

Benchmarks. We evaluate UBTree on Qwen3-4B and Qwen3-8B (Yang et al., 2025a) with thinking disabled (instruct models), and on Qwen3-4B with thinking enabled (reasoning models), under $T_{\text{infer}} = \{0, 1\}$. The benchmarks cover math (GSM8K (Cobbe et al., 2021), MATH-500 (Lightman et al., 2024), and AIME), code (HumanEval (Chen et al., 2021), MBPP (Austin et al., 2021), and LiveCodeBench (Jain et al., 2025)), and dialogue (MT-Bench (Zheng et al., 2023)).

Implementation. We train UBTree on OpenPerfectBlend (Xu et al., 2024) with regenerated target responses under $T_{\text{train}} = 1$. We retrain DFlash (Chen et al., 2026) proposer with block size 16 with SpecForge (Li et al., 2026b) for 6 epochs till convergence, and then jointly train the proposer and selector for 2 epochs, with selector and backbone learning rates of 6×10^{-4} and 2×10^{-5}, respectively. Unlike baselines that need different checkpoints for different decoding temperatures T_{infer}, we use the same checkpoint of UBTree trained under $T_{\text{train}} = 1$ is used at $T_{\text{infer}} = \{0, 1\}$. Our evaluations use $1\times$NVIDIA H200 with batch size 1 and the *transformers* backend. For tree construction, we follow DARTree (Li et al., 2026c) with $B = 64$ non-root nodes, a search width of $W = 12$, and $K = 64$ candidate tokens per position, but use a different depth bonus of $\zeta = 0$. Training and evaluation protocols are detailed in Appendix B.

Baselines. We compare UBTree with single-path methods DFlash (Chen et al., 2026) and Domino (Huang et al., 2026), and tree-based EAGLE-3 (Li et al., 2026d), DDTree (Ringel & Romano, 2026), and DARTree (Li et al., 2026c), with budgets in Table 1. We retrain EAGLE-3 and DFlash on the same OpenPerfectBlend dataset for 6 epochs at corresponding T_{infer}, with both models converging within this training schedule, and report Domino results (with block size 17) from its original paper, trained on OpenPerfectBlend as well. We re-evaluate DDTree and DARTree using retrained DFlash and released Domino checkpoints, respectively, under their original setups. Due to lack of released checkpoints, Domino and DARTree are compared only for instruct models. For fair comparison, we cap the maximum draft depth of DARTree at 16 to match DDTree and UBTree, since the underlying Domino checkpoint uses a block size of 17 and would otherwise permit a larger maximum acceptance length.

Results. Table 1 shows that UBTree achieves average speedups of $6.88\times$ and $6.94\times$ on Qwen3-4B and Qwen3-8B under greedy decoding, and $6.00\times$ and $5.84\times$ under temperature-1 sampling. It outperforms DARTree over all 28 benchmarks in speedups, improving the average by 7.4–13.4%. With thinking enabled on Qwen3-4B, UBTree outperforms DDTree by 25.3% and 29.9% in average speedup at $T_{\text{infer}} = 0$ and 1, respectively. Our superiority especially stands out under $T_{\text{infer}} = 1$, validating the effectiveness of our high entropy adaptions.

Concurrent serving. Table 2 compares UBTree and baselines in SGLang (Zheng et al., 2024) at client concurrency $C \in \{1, 8, 16, 32\}$. All methods use SGLang in BF16 on one NVIDIA H200 with CUDA Graphs. Results are aggregated from the same benchmarks as in Section 4.1. Following DARTree (Li et al., 2026c), we employ an *adaptive-tree* strategy that selects different tree budget B and width W according to the serving load. We tune B/W separately for each method and concurrency using the same candidate configuration set and selection criterion. At $C \in \{1, 8, 16, 32\}$, UBTree uses $B/W = 64/12, 64/12, 64/12, 16/3$ for Qwen3-4B and $B/W = 64/12, 64/12, 16/12, 16/3$ for Qwen3-8B and DARTree uses $B/W = 64/12, 64/12, 32/8, 16/3$ on both models. UBTree achieves the highest throughput in all settings, outperforming the next-best method by 7.0–16.3%. At $C = 32$, UBTree reaches 6560.4 and 5851.4 tokens/s on Qwen3-4B and Qwen3-8B, respectively, exceeding the strongest single-path baseline by 7.3% and 7.7%.

4.2 PRODUCTION-SCALE EXPERIMENT

Evaluation. We evaluate UBTree on two production models with $T_{\text{infer}} = 1$: Ling3-Flash (124B-A5.1B MoE) and Qwen3.8-27B (Qwen Team, 2026), on the same benchmarks as in Section 4.1. We compare UBTree with DSpark (Cheng et al., 2026) and DFlash (Chen et al., 2026) on the Ling3-Flash, and with DSpark and DFlash 2 (Inco AI, 2026) on Qwen3.8-27B. We construct target-specific training sets by regenerating responses with each target under thinking-disabled decoding.

Table 1: Speedup and average acceptance length (τ) on instruct and reasoning models. All τ values include the target-produced bonus token. M-500, HE, LCB, and MT-B denote MATH-500, HumanEval, LiveCodeBench, and MT-Bench. † values are paper-reported.

Instruct Models

Model	Method	MATH						CODE						CHAT			
		GSM8K		M-500		AIME		HE		MBPP		LCB		MT-B		Avg.	
		Speedup	τ	Speedup	τ	Speedup	τ	Speedup	τ	Speedup	τ	Speedup	τ	Speedup	τ	Speedup	τ
Qwen3-4B $T_{\text{infer}}=0$	*Single-path*																
	DFlash (16)	5.95	8.57	5.76	8.34	4.45	6.34	4.28	6.08	4.03	5.82	3.83	5.39	2.69	4.60	4.43	6.45
	Domino† (17)	8.02	10.12	7.14	9.01	5.97	7.49	5.66	7.02	5.59	7.11	5.33	6.86	3.28	5.12	5.86	7.53
	Tree-based																
	EAGLE-3 (16)	4.04	7.20	3.97	6.96	3.45	6.05	3.46	6.07	3.42	6.05	3.19	5.53	2.40	4.67	3.42	6.08
	EAGLE-3 (64)	4.42	7.97	4.45	7.85	4.02	7.11	4.00	7.07	3.95	7.11	3.75	6.58	2.80	5.46	3.91	7.02
	DDTree (64)	6.69	10.22	6.62	10.10	5.51	8.49	5.18	7.89	5.10	7.80	4.78	7.23	3.51	6.14	5.34	8.27
	DARTree (64)	7.89	12.50	7.63	11.90	6.55	**10.37**	6.09	9.78	6.33	**10.07**	5.75	9.04	3.93	7.23	6.31	10.13
	UBTree (64)	**8.54**	**12.58**	**8.18**	**12.15**	**6.92**	10.13	**7.07**	**10.22**	**6.66**	9.98	**6.37**	**9.24**	**4.39**	**7.32**	**6.88**	**10.23**
Qwen3-4B $T_{\text{infer}}=1$	*Single-path*																
	DFlash (16)	5.28	7.67	4.56	6.89	3.10	4.64	4.09	5.86	3.84	5.52	3.25	4.62	2.52	4.26	3.81	5.64
	Domino† (17)	6.79	8.61	5.67	7.35	3.75	4.83	5.02	6.30	4.96	6.36	5.04	6.45	3.02	4.56	4.89	6.35
	Tree-based																
	EAGLE-3 (16)	3.66	6.61	3.51	6.36	2.72	4.88	3.34	5.93	3.21	5.80	2.80	4.94	2.21	4.34	3.06	5.55
	EAGLE-3 (64)	4.13	7.66	3.96	7.29	3.20	6.00	3.75	6.84	3.71	6.92	3.27	5.96	2.54	5.12	3.51	6.54
	DDTree (64)	6.19	9.49	5.62	8.76	4.14	6.31	5.17	7.81	4.90	7.50	4.19	6.37	3.37	5.66	4.80	7.41
	DARTree (64)	6.94	11.01	6.05	9.73	4.10	6.99	5.84	9.10	5.76	9.28	4.71	7.33	3.64	6.53	5.29	8.57
	UBTree (64)	**7.95**	**11.79**	**7.13**	**10.54**	**5.10**	**7.63**	**6.42**	**9.43**	**6.15**	**9.30**	**5.27**	**7.72**	**4.02**	**6.68**	**6.00**	**9.01**
Qwen3-8B $T_{\text{infer}}=0$	*Single-path*																
	DFlash (16)	5.98	8.59	5.84	8.51	4.75	6.73	4.46	6.31	4.07	5.87	3.91	5.48	2.64	4.54	4.52	6.58
	Domino† (17)	7.92	10.03	7.38	9.43	5.85	7.41	5.89	7.39	5.53	7.04	5.27	7.04	3.29	5.18	5.88	7.65
	Tree-based																
	EAGLE-3 (16)	4.23	7.43	4.20	7.30	3.71	6.46	3.68	6.42	3.62	6.39	3.54	6.07	2.53	4.88	3.64	6.42
	EAGLE-3 (64)	4.57	8.20	4.66	8.11	4.34	7.51	4.23	7.39	4.13	7.39	4.11	7.10	2.99	5.77	4.15	7.35
	DDTree (64)	6.65	10.19	6.60	10.12	5.64	8.63	5.43	8.17	5.07	7.68	4.92	7.30	3.49	6.09	5.40	8.31
	DARTree (64)	8.26	12.49	7.61	12.13	6.71	**10.37**	6.41	10.09	6.32	**10.02**	5.88	9.19	4.01	**7.33**	6.46	**10.23**
	UBTree (64)	**8.65**	**12.58**	**8.46**	**12.30**	**7.05**	10.17	**7.12**	**10.12**	**6.64**	9.79	**6.43**	**9.24**	**4.25**	7.25	**6.94**	10.21
Qwen3-8B $T_{\text{infer}}=1$	*Single-path*																
	DFlash (16)	5.05	7.37	4.57	6.77	2.99	4.47	3.74	5.26	3.64	5.20	3.14	4.41	2.34	3.96	3.64	5.35
	Domino† (17)	6.47	8.34	5.40	7.20	3.78	4.92	4.75	5.98	4.73	6.02	5.06	6.72	2.94	4.61	4.73	6.26
	Tree-based																
	EAGLE-3 (16)	3.93	7.07	3.74	6.68	3.01	5.30	3.35	5.89	3.35	6.01	3.01	5.25	2.36	4.55	3.25	5.82
	EAGLE-3 (64)	4.39	8.05	4.23	7.65	3.51	6.34	3.88	7.00	3.88	7.14	3.53	6.32	2.73	5.45	3.73	6.85
	DDTree (64)	6.10	9.23	5.61	8.68	4.04	6.31	4.91	7.37	4.63	7.02	4.14	6.22	3.23	5.48	4.67	7.19
	DARTree (64)	7.04	11.01	5.86	9.59	4.48	7.21	5.26	8.36	5.53	**8.91**	4.45	7.06	3.59	**6.48**	5.17	8.37
	UBTree (64)	**7.97**	**11.55**	**6.99**	**10.44**	**4.99**	**7.43**	**6.17**	**8.85**	**5.92**	8.80	**5.10**	**7.47**	**3.77**	6.36	**5.84**	**8.70**

Reasoning Models

Model	Method	MATH						CODE						CHAT			
		GSM8K		M-500		AIME		HE		MBPP		LCB		MT-B		Avg.	
		Speedup	τ	Speedup	τ	Speedup	τ	Speedup	τ	Speedup	τ	Speedup	τ	Speedup	τ	Speedup	τ
Qwen3-4B $T_{\text{infer}}=0$	*Single-path*																
	DFlash (16)	5.18	7.28	4.53	6.45	3.67	5.26	3.91	5.46	3.72	5.17	3.37	4.68	2.99	4.64	3.91	5.56
	Tree-based																
	EAGLE-3 (16)	3.49	6.01	3.04	5.23	2.65	4.62	2.82	4.83	2.80	4.78	2.43	4.16	2.25	4.11	2.78	4.82
	EAGLE-3 (64)	4.19	7.24	3.75	6.48	3.29	5.77	3.52	6.05	3.51	6.04	3.04	5.28	2.74	5.07	3.44	5.99
	DDTree (64)	6.09	9.19	5.54	8.42	4.77	7.20	4.85	7.29	4.68	7.01	4.31	6.45	3.82	6.15	4.87	7.39
	UBTree (64)	**7.64**	**10.95**	**6.99**	**10.01**	**5.73**	**8.53**	**6.21**	**8.85**	**5.98**	**8.54**	**5.45**	**7.75**	**4.71**	**7.22**	**6.10**	**8.84**
Qwen3-4B $T_{\text{infer}}=1$	*Single-path*																
	DFlash (16)	4.80	6.77	4.36	6.18	3.57	5.15	3.76	5.23	3.55	4.93	3.21	4.47	2.89	4.40	3.73	5.30
	Tree-based																
	EAGLE-3 (16)	3.01	5.28	2.78	4.86	2.41	4.22	2.54	4.42	2.47	4.28	2.24	3.92	2.00	3.69	2.49	4.38
	EAGLE-3 (64)	3.64	6.57	3.34	6.00	2.98	5.37	3.17	5.66	3.06	5.48	2.77	4.99	2.35	4.49	3.04	5.51
	DDTree (64)	4.87	7.35	4.69	7.11	4.01	6.10	4.17	6.22	3.97	5.93	3.57	5.37	3.28	5.11	4.08	6.17
	UBTree (64)	**6.56**	**9.50**	**6.12**	**8.78**	**5.12**	**7.47**	**5.44**	**7.74**	**5.18**	**7.41**	**4.57**	**6.61**	**4.10**	**6.17**	**5.30**	**7.67**

For baseline and UBTree training, Qwen3.8-27B uses OpenPerfectBlend (Xu et al., 2024) prompts, whereas Ling3-Flash uses supervised-finetuning corpus with 2.5M samples and 15B tokens under 64K context length. For Qwen3.8-27B, we load the officially released Qwen3.8-27B DFlash 2 checkpoint and fine-tune both DFlash 2 selector and UBTree selector for 2 epochs with a learning rate of 3×10^{-4} on our regenerated Qwen3.8-27B rollouts. For Ling3-Flash, we train DFlash with block size 8 for 3 epochs with the learning rate 2×10^{-5}. UBTree is then trained based on DFlash for 2 epochs with a learning rate of 3×10^{-4}. Our tree verification uses the setup in Section 4.1.

SGLang Integration. We implement UBTree in SGLang (Zheng et al., 2024). DFlash and DSpark use SGLang's official implementation, while the Qwen3.8-27B implementations follow the official DFlash 2 codebase. We use TP degree 4 on four H200 GPUs, with the same degree for the target and

Table 2: Concurrent-serving results on Qwen3-4B and Qwen3-8B under $T_{\text{infer}} = 1$. Each cell reports aggregate output tokens/s / speedup over the AR baseline row at the same concurrency.

Model	Method	$C = 1$	$C = 8$	$C = 16$	$C = 32$
Qwen3-4B	Autoregressive baseline	253.5 / 1.00×	1404.9 / 1.00×	2208.3 / 1.00×	3307.4 / 1.00×
	DFlash	796.4 / 3.14×	2287.2 / 1.63×	3664.4 / 1.66×	6071.9 / 1.84×
	Domino	816.3 / 3.22×	2258.9 / 1.61×	3636.5 / 1.65×	6115.5 / 1.85×
	DDTree	844.6 / 3.33×	2390.8 / 1.70×	3669.8 / 1.66×	4810.4 / 1.45×
	DARTree (adaptive B/W)	978.5 / 3.86×	2405.1 / 1.71×	3770.3 / 1.71×	5619.9 / 1.70×
	UBTree (adaptive B/W)	**1089.1** / 4.30×	**2797.4** / 1.99×	**4271.9** / 1.93×	**6560.4** / 1.98×
Qwen3-8B	Autoregressive baseline	180.9 / 1.00×	1126.2 / 1.00×	1865.9 / 1.00×	2770.9 / 1.00×
	DFlash	634.7 / 3.51×	2181.7 / 1.94×	3596.0 / 1.93×	5265.4 / 1.90×
	Domino	676.2 / 3.74×	2137.9 / 1.90×	3582.7 / 1.92×	5435.2 / 1.96×
	DDTree	686.8 / 3.80×	2013.9 / 1.79×	3303.5 / 1.77×	3603.6 / 1.30×
	DARTree (adaptive B/W)	824.4 / 4.56×	2280.1 / 2.02×	3514.5 / 1.88×	4844.9 / 1.75×
	UBTree (adaptive B/W)	**881.7** / 4.87×	**2644.4** / 2.35×	**4069.9** / 2.18×	**5851.4** / 2.11×

Table 3: Speedup and acceptance length (τ) on production-scale models. Each benchmark reports speedup and average acceptance length, and the Avg. columns are means over all benchmarks. M-500, HE, LCB, and MT-B denote MATH-500, HumanEval, LiveCodeBench, and MT-Bench.

Model	Method	MATH						CODE						CHAT			
		GSM8K		M-500		AIME		HE		MBPP		LCB		MT-B		Avg.	
		Speedup	τ	Speedup	τ	Speedup	τ	Speedup	τ	Speedup	τ	Speedup	τ	Speedup	τ	Speedup	τ
Ling3-Flash	DSpark	5.14	6.09	4.93	5.81	4.22	4.96	5.37	6.32	4.50	5.93	4.18	4.98	3.19	3.74	4.50	5.40
	DFlash	4.85	5.73	4.83	5.58	4.17	4.80	5.11	6.00	4.39	5.55	3.97	4.63	3.11	3.58	4.35	5.12
	UBTree	**5.83**	**7.37**	**5.87**	**7.18**	**5.43**	**6.60**	**5.94**	**7.42**	**5.00**	**6.85**	**5.22**	**6.54**	**4.24**	**5.27**	**5.36**	**6.75**
Qwen3.8-27B	DSpark	**5.20**	6.74	4.76	6.07	4.41	5.65	**5.18**	6.85	4.26	5.79	3.47	4.53	3.00	3.82	4.32	5.64
	DFlash 2	4.65	6.29	4.67	6.29	4.48	6.00	5.06	6.98	4.30	6.11	3.55	4.82	2.92	3.95	4.24	5.78
	UBTree	5.05	**7.59**	**5.28**	**7.50**	**5.07**	**7.33**	5.14	**7.66**	**4.67**	**7.32**	**4.44**	**6.54**	**3.84**	**5.61**	**4.78**	**7.08**

draft models. All experiments use batch size 1 and greedy drafter decoding with thinking disabled. Ling3-Flash and Qwen3.8-27B use hybrid attention with Kimi Delta Attention (Kimi Team et al., 2025) and Gated DeltaNet (Yang et al., 2025b) layers, respectively. Inspired by Bole (Wang et al., 2026a), each node inherits recurrent and convolution states from its parent in tree verification, and only the states and KV entries on the accepted path are committed.

Results. Table 3 shows that UBTree achieves the highest acceptance length in all 14 model–benchmark comparisons and the highest speedup in 12 of them. Averaged over the seven benchmarks, UBTree reaches $5.36\times$ and $4.78\times$ speedup on Ling3-Flash and Qwen3.8-27B, respectively.

4.3 LONG-CONTEXT TASKS

Setup. Based on the correspondence between speedup and acceptance length proven in Section 4.2, we evaluate acceptance length on Ling3-Flash using LongBench-v2 (Bai et al., 2025) for long inputs and SWE-bench (Jimenez et al., 2024) for long-continuation agentic tasks. For SWE-bench, all methods replay identical records. Evaluation protocols are detailed in Appendix B.3.

Results. Figure 1 (c) shows that UBTree achieves the highest acceptance length in every context-length bin, exceeding the strongest baseline by 34.6–42.1% on LongBench-v2 and 29.0–37.8% on SWE-bench. These results show that UBTree retains its advantage under both long initial prompts and growing agent histories. Quantitative results are available in Appendix C.2.

4.4 ABLATION STUDIES

Architecture & high-temperature training. We first ablate our two designs, selector architecture and high-temperature training temperature, on Qwen3-4B under $T_{\text{infer}} = 0$ and setups in Section 4.1. As shown in Table 4 (Top), our selector architecture, combining deep codebooks and depth calibration, improves both speedup and acceptance length across all benchmarks, increasing their averages from $5.95\times$ to $6.57\times$ and from 8.82 to 9.86. A higher training temperature further helps reach an average speedup of $6.88\times$ and acceptance length of 10.23, validating our tree-native training.

Table 4: Selector, training-temperature, and training-objective ablations on Qwen3-4B with thinking disabled. **Top:** speedup and acceptance length (τ) at $T_{\text{infer}} = 0$. **Middle:** acceptance length across training and inference temperatures, with the same checkpoints evaluated in both panels. **Bottom:** training-objective ablation with the fixed architecture and training temperature.

	MATH						CODE						CHAT			
	GSM8K		M-500		AIME		HE		MBPP		LCB		MT-B		Avg.	
	Speedup	τ	Speedup	τ	Speedup	τ	Speedup	τ	Speedup	τ	Speedup	τ	Speedup	τ	Speedup	τ
DFlash 2 selector+Tree	7.58	11.08	7.42	10.87	6.13	8.98	5.88	8.48	5.64	8.44	5.40	7.77	3.60	6.15	5.95	8.82
UBTree selector+Tree	8.16	12.11	7.83	11.67	6.79	10.06	6.73	9.61	6.21	9.54	6.04	8.78	4.26	7.23	6.57	9.86
$+T_{\text{train}} = 1$	**8.54**	**12.58**	**8.18**	**12.15**	**6.92**	**10.13**	**7.07**	**10.22**	**6.66**	**9.98**	**6.37**	**9.24**	**4.39**	**7.32**	**6.88**	**10.23**

T_{train}	$T_{\text{infer}} = 0$: Acceptance length (τ)								$T_{\text{infer}} = 1$: Acceptance length (τ)							
	MATH			CODE			CHAT		MATH			CODE			CHAT	
	GSM8K	M-500	AIME	HE	MBPP	LCB	MT-B	Avg.	GSM8K	M-500	AIME	HE	MBPP	LCB	MT-B	Avg.
0.8	12.58	12.12	**10.57**	10.00	9.82	9.15	7.26	10.21	11.72	10.47	7.52	9.39	9.17	7.63	6.58	8.93
1.0	12.58	**12.15**	10.13	**10.22**	**9.98**	**9.24**	**7.32**	**10.23**	**11.79**	10.54	**7.63**	9.43	9.30	7.72	6.68	**9.01**
1.1	**12.64**	12.04	10.08	10.09	9.77	9.02	7.15	10.11	11.71	10.52	7.61	**9.48**	9.12	7.71	6.69	8.98
1.2	12.59	11.94	9.90	10.10	9.87	9.01	7.22	10.09	11.72	**10.61**	7.47	9.31	**9.40**	**7.80**	**6.72**	9.00

T_{infer}	Selector loss	β	Speedup	τ	T_{infer}	Selector loss	β	Speedup	τ
0	Hard-label CE	0.1	6.84	10.08	1	Hard-label CE	0.1	5.99	8.93
0	Forward KL	0.1	6.88	10.23	1	Forward KL	0.1	6.00	9.01
0	Forward KL	0	6.87	10.14	1	Forward KL	0	6.01	9.01

Training temperature sweeping. We then examine the optimal training temperature T_{train} of UBTree on Qwen3-4B by varying $T_{\text{train}} \in \{0.8, 1.0, 1.1, 1.2\}$ and evaluating each checkpoint at $T_{\text{infer}} \in \{0, 1\}$ in Table 4 (Middle). Among the tested temperatures, $T_{\text{train}} = 1$ achieves the highest average acceptance length at both inference temperatures.

Training objectives. We also compare different choices of training objectives by the average speedup and acceptance length over seven benchmarks in Table 4 (Bottom). Both using the cross-entropy (CE) loss for the selector and disabling the CE loss for the proposer slightly degrade the acceptance length at lower inference temperatures. Additional details are provided in Appendix B.4.

5 Related Work

Parallel drafting and token-wise dependencies. DFlash generates tokens in a draft block in parallel (Chen et al., 2026). Domino and DSpark restore token-wise dependencies through sequential correction (Huang et al., 2026; Cheng et al., 2026), whereas DFlash 2 uses a low-rank selector to score adjacent candidate pairs in parallel for single-path drafting (Inco AI, 2026). UBTree retains this scoring formulation but redesigns the token representations and learns depth-dependent weights for combining proposer logits and selector scores in tree verification.

Tree-based speculative decoding. SpecInfer and Medusa retain multiple draft paths for tree verification (Miao et al., 2024; Cai et al., 2024), while EAGLE-2 adapts tree structure to proposal confidence (Li et al., 2024b). DART applies n-gram-guided pruning to parallel predictions (Liu et al., 2026). DDTree and DARTree construct trees from block-parallel proposals using position-wise probabilities and path-conditioned correction, respectively (Ringel & Romano, 2026; Li et al., 2026c). Compared to existing tree drafting methods, UBTree exploits *tree-native* training to supervise the selector with target probabilities over alternatives rather than individual sampled tokens.

6 Conclusion

UBTree offers a new perspective on speculative decoding: tree verification calls for designs that explicitly account for multiple plausible generation paths. Its strong performance across benchmarks demonstrates the effectiveness of this principle. We also discuss limitations of our work in Appendix D. Future work could further advance inference-time tree size selection, adapting the verification budget to balance candidate coverage and verification cost across different hardware, serving loads, and latency requirements.

AI USE STATEMENT

We used generative AI tools for language editing to improve the clarity, grammar, and readability of the manuscript. We have not used generative AI tools for retrieval and discovery, research ideation or execution, or drafting sections of the paper. Generating synthetic datasets and proving mathematical claims are not applicable to this work. We have reviewed all AI-assisted revisions and take responsibility for the final content of this work, including its text, claims, and results.

ETHICS STATEMENT

This work focuses on improving the computational efficiency of language model inference. We do not identify additional ethical concerns specific to the proposed acceleration method beyond those associated with the underlying models and their applications. Deployment should follow the safety requirements and usage policies of these models.

REPRODUCIBILITY STATEMENT

Experimental settings are described in Section 4. Additional training and inference configurations, evaluation protocols, and ablation details are provided in Appendices B and C. We will publicly release our training code upon acceptance.

REFERENCES

Zihao An, Huajun Bai, Ziqiong Liu, Dong Li, and Emad Barsoum. Pard: Accelerating llm inference with low-cost parallel draft model adaptation. In *International Conference on Learning Representations*, volume 2026, pp. 144562–144579, 2026.

Marianne Arriola, Aaron Gokaslan, Justin T. Chiu, Zhihan Yang, Zhixuan Qi, Jiaqi Han, Subham Sekhar Sahoo, and Volodymyr Kuleshov. Block diffusion: Interpolating between autoregressive and diffusion language models. In *International Conference on Learning Representations*, 2025. URL https://arxiv.org/abs/2503.09573.

Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie Cai, Michael Terry, Quoc Le, et al. Program synthesis with large language models. *arXiv preprint arXiv:2108.07732*, 2021.

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. In *International Conference on Learning Representations*, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/hash/656d6174fd5ffb01c843f269649ab5cb-Abstract-Conference.html.

Yushi Bai, Shangqing Tu, Jiajie Zhang, Hao Peng, Xiaozhi Wang, Xin Lv, Shulin Cao, Jiazheng Xu, Lei Hou, Yuxiao Dong, et al. Longbench v2: Towards deeper understanding and reasoning on realistic long-context multitasks. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 3639–3664, 2025.

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 *Forty-first International Conference on Machine Learning*, 2024. URL https://openreview.net/forum?id=PEpbUobfJv.

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

Jian Chen, Yesheng Liang, and Zhijian Liu. DFlash: Block diffusion for flash speculative decoding. In *Forty-third International Conference on Machine Learning*, 2026. URL https://openreview.net/forum?id=Oz335dV48X.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde De Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, et al. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*, 2021.

Xin Cheng, Xingkai Yu, Chenze Shao, Jiashi Li, Yunfan Xiong, Yi Qian, Jiaqi Zhu, Shirong Ma, Xiaokang Zhang, Jiasheng Ye, et al. Dspark: Confidence-scheduled speculative decoding with semi-autoregressive generation. *arXiv preprint arXiv:2607.05147*, 2026.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, et al. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*, 2021.

Siddartha Devic, Charlotte Peale, Arwen Bradley, Sinead Williamson, Preetum Nakkiran, and Aravind Gollakota. Trace length is a simple uncertainty signal in reasoning models. *arXiv preprint arXiv:2510.10409*, 2025.

DiffusionGemma Team, Adrien Ali Taïga, James Assiene, Daniele Calandriello, Rahma Chaabouni, João Gante, et al. DiffusionGemma technical report. *arXiv preprint arXiv:2608.00146*, 2026. URL https://arxiv.org/abs/2608.00146.

Roman Garipov, Fedor Velikonivtsev, Ivan Ermakov, Ruslan Svirschevski, Vage Egiazarian, and Max Ryabinin. Autojudge: Judge decoding without manual annotation. *Advances in Neural Information Processing Systems*, 38:94605–94642, 2026.

Fabian Gloeckle, Badr Youbi Idrissi, Baptiste Roziere, David Lopez-Paz, and Gabriel Synnaeve. Better & faster large language models via multi-token prediction. In *Proceedings of the 41st International Conference on Machine Learning*, pp. 15706–15734, 2024. URL https://proceedings.mlr.press/v235/gloeckle24a.html.

Maximilian Holsman, Yukun Huang, and Bhuwan Dhingra. Fuzzy speculative decoding for a tunable accuracy-runtime tradeoff. In *Findings of the Association for Computational Linguistics: ACL 2025*, pp. 26257–26273, 2025. doi: 10.18653/v1/2025.findings-acl.1346. URL https://aclanthology.org/2025.findings-acl.1346/.

Jianuo Huang, Yaojie Zhang, Qituan Zhang, Hao Lin, Hanlin Xu, and Linfeng Zhang. Domino: Decoupling causal modeling from autoregressive drafting in speculative decoding. *arXiv preprint arXiv:2605.29707*, 2026.

Inception Labs, Samar Khanna, Siddhant Kharbanda, Shufan Li, Harshit Varma, Eric Wang, Sawyer Birnbaum, Ziyang Luo, Yanis Miraoui, Akash Palrecha, Stefano Ermon, Aditya Grover, and Volodymyr Kuleshov. Mercury: Ultra-fast language models based on diffusion. *arXiv preprint arXiv:2506.17298*, 2025. URL https://arxiv.org/abs/2506.17298.

Inco AI. DFlash 2: Keep drafting parallel, August 2026. URL https://inco.ai/blog/dflash2/.

Naman Jain, Alex Gu, Wen-Ding Li, Fanjia Yan, Tianjun Zhang, Sida Wang, Armando Solar-Lezama, Koushik Sen, and Ion Stoica. Livecodebench: Holistic and contamination free evaluation of large language models for code. In *International Conference on Learning Representations*, volume 2025, pp. 58791–58831, 2025.

Carlos E. Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik Narasimhan. SWE-bench: Can language models resolve real-world GitHub issues? In *International Conference on Learning Representations*, 2024.

Sehoon Kim, Karttikeya Mangalam, Suhong Moon, Jitendra Malik, Michael W. Mahoney, Amir Gholami, and Kurt Keutzer. Speculative decoding with big little decoder. In *Advances in Neural Information Processing Systems*, volume 36, 2023. URL https://proceedings.neurips.cc/paper_files/paper/2023/hash/7b97adeafa1c51cf65263459ca9d0d7c-Abstract-Conference.html.

Kimi Team, Yu Zhang, Zongyu Lin, Xingcheng Yao, Jiaxi Hu, Fanqing Meng, et al. Kimi Linear: An expressive, efficient attention architecture. *arXiv preprint arXiv:2510.26692*, 2025.

Yaniv Leviathan, Matan Kalman, and Yossi Matias. Fast inference from transformers via speculative decoding. In *International conference on machine learning*, pp. 19274–19286. PMLR, 2023.

Guanghao Li, Zhihui Fu, Min Fang, Qibin Zhao, Ming Tang, Chun Yuan, and Jun Wang. Diffuspec: Unlocking diffusion language models for speculative decoding. In *Findings of the Association for Computational Linguistics: ACL 2026*, pp. 20896–20910, 2026a.

Shenggui Li, Chao Wang, Yikai Zhu, Yubo Wang, Fan Yin, Shuai Shi, Yefei Chen, Xiaomin Dong, Qiaoling Chen, Jin Pan, et al. SpecForge: A flexible and efficient open-source training framework for speculative decoding. *arXiv preprint arXiv:2603.18567*, 2026b.

Tianyi Li, Yaxin Luo, Xinyi Shang, and Zhiqiang Shen. Dartree: Speculative diffusion decoding with autoregressive draft trees. *arXiv preprint arXiv:2608.13524*, 2026c.

Yuhui Li, Fangyun Wei, Chao Zhang, and Hongyang Zhang. EAGLE: Speculative sampling requires rethinking feature uncertainty. In *Forty-first International Conference on Machine Learning*, 2024a. URL https://openreview.net/forum?id=1NdN7eXyb4.

Yuhui Li, Fangyun Wei, Chao Zhang, and Hongyang Zhang. 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, 2024b.

Yuhui Li, Fangyun Wei, Chao Zhang, and Hongyang Zhang. Eagle-3: Scaling up inference acceleration of large language models via training-time test. *Advances in Neural Information Processing Systems*, 38:136737–136756, 2026d.

Hunter Lightman, Vineet Kosaraju, Yuri Burda, Harrison Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. Let's verify step by step. In *International Conference on Learning Representations*, volume 2024, pp. 39578–39601, 2024.

Aiwei Liu, Minghua He, Shaoxun Zeng, Sijun Zhang, Linhao Zhang, Chuhan Wu, Wei Jia, Yuan Liu, Xiao Zhou, and Jie Zhou. WeDLM: Reconciling diffusion language models with standard causal attention for fast inference. *arXiv preprint arXiv:2512.22737*, 2025. URL https://arxiv.org/abs/2512.22737.

Fuliang Liu, Xue Li, Ketai Zhao, Yinxi Gao, Ziyan Zhou, Zhonghui Zhang, Zhibin Wang, Wanchun Dou, Sheng Zhong, and Chen Tian. Dart: Diffusion-inspired speculative decoding for fast llm inference. *arXiv preprint arXiv:2601.19278*, 2026.

Xupeng Miao, Gabriele Oliaro, Zhihao Zhang, Xinhao Cheng, Zeyu Wang, Zhengxin Zhang, Rae Ying Yee Wong, Alan Zhu, Lijie Yang, Xiaoxiang Shi, 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.

Harikrishna Narasimhan, Wittawat Jitkrittum, Ankit Singh Rawat, Seungyeon Kim, Neha Gupta, Aditya Krishna Menon, and Sanjiv Kumar. Faster cascades via speculative decoding. In *International Conference on Learning Representations*, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/hash/6f43166f50f26e8d8f3edc5545b0749f-Abstract-Conference.html.

Shen Nie, Fengqi Zhu, Zebin You, Xiaolu Zhang, Jingyang Ou, Jun Hu, Jun Zhou, Yankai Lin, Ji-Rong Wen, and Chongxuan Li. Large language diffusion models. *Advances in Neural Information Processing Systems*, 38:50608–50646, 2026.

Qwen Team. Qwen3.8-Max: A new bar for coding and cowork, August 2026. URL https://qwen.ai/blog?id=qwen3.8.

Liran Ringel and Yaniv Romano. Accelerating speculative decoding with block diffusion draft trees. *arXiv preprint arXiv:2604.12989*, 2026.

Li Wang, Yi Su, Xiabao Wu, Chiran You, Yongchao Liu, Zhan Qiu, Juelu Zhang, Jiajun Zheng, Fangxin Liu, Jie Zhang, et al. Bole: Efficient tree speculation for hybrid-attention language models. *arXiv preprint arXiv:2608.01651*, 2026a.

Ziyi Wang, Siva Rajesh Kasa, Ankith M S, Santhosh Kumar Kasa, Jiaru Zou, Sumit Negi, Ruqi Zhang, Nan Jiang, and Qifan Song. DIVERSED: Relaxed speculative decoding via dynamic ensemble verification. In *Proceedings of the 29th International Conference on Artificial Intelligence and Statistics*, volume 300, pp. 2341–2349, 2026b. URL https://proceedings.mlr.press/v300/wang26d.html.

Zhihui Xie, Jiacheng Ye, Lin Zheng, Jiahui Gao, Jingwei Dong, Zirui Wu, Xueliang Zhao, Shansan Gong, Xin Jiang, Zhenguo Li, et al. Dream-coder 7b: An open diffusion language model for code. *arXiv preprint arXiv:2509.01142*, 2025.

Tengyu Xu, Eryk Helenowski, Karthik Abinav Sankararaman, Di Jin, Kaiyan Peng, Eric Han, Shaoliang Nie, Chen Zhu, Hejia Zhang, Wenxuan Zhou, et al. The perfect blend: Redefining rlhf with mixture of judges. *arXiv preprint arXiv:2409.20370*, 2024.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, et al. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025a.

Penghui Yang, Cunxiao Du, Fengzhuo Zhang, Haonan Wang, Tianyu Pang, Chao Du, and Bo An. Longspec: Long-context lossless speculative decoding with efficient drafting and verification. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 1826–1844, 2026.

Songlin Yang, Jan Kautz, and Ali Hatamizadeh. Gated delta networks: Improving mamba2 with delta rule. In *International Conference on Learning Representations*, volume 2025, pp. 29687–29707, 2025b.

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric Xing, et al. Judging llm-as-a-judge with mt-bench and chatbot arena. *Advances in neural information processing systems*, 36:46595–46623, 2023.

Lianmin Zheng, Liangsheng Yin, Zhiqiang Xie, Chuyue Sun, Jeff Huang, Cody H Yu, Shiyi Cao, Christos Kozyrakis, Ion Stoica, Joseph E Gonzalez, et al. Sglang: Efficient execution of structured language model programs. *Advances in neural information processing systems*, 37:62557–62583, 2024.

Yongchao Zhou, Kaifeng Lyu, Ankit Singh Rawat, Aditya Krishna Menon, Afshin Rostamizadeh, Sanjiv Kumar, Jean-François Kagy, and Rishabh Agarwal. DistillSpec: Improving speculative decoding via knowledge distillation. In *International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=rsY6J3ZaTF.

Siqi Zhu, Xuyan Ye, Hongyu Lu, Weiye Shi, and Ge Liu. The many faces of on-policy distillation: Pitfalls, mechanisms, and fixes. *arXiv preprint arXiv:2605.11182*, 2026.

A TREE CONSTRUCTION DETAILS

UBTree follows the search schedule of DARTree's pruned variant (Algorithm 1 in Li et al., 2026c), reviewed in Section 2. DARTree evaluates path-conditioned correction probabilities and updates correction states between depths; UBTree instead supplies precomputed proposer and selector scores, so these correction-state updates are not needed.

Path scores. For a fixed verified context $x_{\le t}$, we use the candidate sets from Section 3.1.1 and combined scores from equation 5. These scores incorporate both proposer logits and selector scores conditioned on the proposer's hidden states. For each predecessor $a \in \mathcal{C}_{i-1}$, normalization over successor candidates defines a conditional distribution, whose log-probabilities accumulate along a candidate path:

$$\begin{aligned} q_i^{\mathrm{bi}}(b \mid a, x_{\le t}) &= \frac{\exp s_i(a,b)}{\sum_{v\in\mathcal{C}_i} \exp s_i(a,v)}, \qquad b \in \mathcal{C}_i, \\ \ell(y_{1:d}) &= \sum_{i=1}^{d} \log q_i^{\mathrm{bi}}(y_i \mid y_{i-1}, x_{\le t}). \end{aligned} \tag{9}$$

Here, $y_0 = x_t$ and $1 \le d \le \gamma$ is the path depth. The root represents the empty draft prefix ϵ, with $\ell(\epsilon) = 0$. Each non-root node is identified by its complete draft prefix $y_{1:d}$; identical tokens reached through different parents remain distinct nodes.

Depth bonus. Following DARTree's scoring form, the search permits a nonpositive depth bonus when ranking tree nodes. The reported UBTree evaluations use $\zeta = 0$:

$$\ell_\zeta(y_{1:d}) = \ell(y_{1:d}) + \zeta d, \qquad \zeta = 0. \tag{10}$$

We set $\ell_\zeta(\epsilon) = 0$ and use ζ to distinguish this depth bonus from the training-loss weight β. The bonus is applied to path scores after probability normalization. It leaves rankings within each depth unchanged and penalizes deeper nodes during global pruning only when $\zeta < 0$.

Depth-wise expansion. Let $\mathcal{F}_d$ be the retained frontier at depth d, initialized by $\mathcal{F}_0 = \{\epsilon\}$. At each depth, we extend every retained path with the candidates at that position and select the highest-scoring extensions:

$$\begin{aligned} \mathcal{E}_d &= \{y_{1:d} : y_{1:d-1} \in \mathcal{F}_{d-1},\ y_d \in \mathcal{C}_d\}, \\ \mathcal{F}_d &= \mathrm{Top}_W(\mathcal{E}_d; \ell_\zeta), \qquad d = 1, \dots, \gamma. \end{aligned} \tag{11}$$

We interpret $y_{1:0} = \epsilon$. The operator $\mathrm{Top}_W(\mathcal{E}_d; \ell_\zeta)$ retains up to W nodes with the largest depth-penalized scores across all extensions at depth d, not W children per parent. Scoring an extension adds its conditional log-probability from equation 9 and ζ to its parent's score using the precomputed combined scores, without another selector evaluation.

Global pruning. The retained frontiers form a candidate supertree. We select at most B non-root nodes from this supertree by the same depth-penalized score:

$$\mathcal{S} = \bigcup_{d=1}^{\gamma} \mathcal{F}_d, \qquad \mathcal{T} = \{\epsilon\} \cup \mathrm{Top}_B(\mathcal{S}; \ell_\zeta). \tag{12}$$

The root is kept separately and does not count toward B. Since every normalized probability is at most one and $\zeta \le 0$, $\ell_\zeta(y_{1:d}) \le \ell_\zeta(y_{1:d-1})$. Preferring ancestors when scores tie therefore ensures that every selected node retains all its ancestors, following the prefix-monotonicity argument in Lemma 1 of Li et al. (2026c). This selection is restricted to the materialized supertree $\mathcal{S}$; finite-width expansion can discard paths before global pruning.

Construction summary. Algorithm 1 summarizes tree construction from precomputed proposer and selector scores, without further selector evaluation. We use $y_{1:0} = \epsilon$ and $y_0 = x_t$. The operator Top_m retains up to m highest-scoring nodes, preferring ancestors when scores tie.

Verification via tree-attention. After tree construction, we verify the prefix-closed draft tree with at most $B = 64$ non-root nodes in a single target-model forward pass using tree attention. Each node is evaluated under its corresponding autoregressive prefix, and verification follows the child matching the target-model sample at each step until the first mismatch. The first unmatched sample is retained as the bonus token. After verification, the KV cache and recurrent states along the accepted path are

Algorithm 1 UBTree Construction from Precomputed Scores

Input: Verified context $x_{\le t}$, candidate sets $\{\mathcal{C}_i\}_{i=1}^{\gamma}$, precomputed combined scores $\{s_i\}_{i=1}^{\gamma}$, search width W, and non-root node budget B.
Output: A prefix-closed tree $\mathcal{T}$ with at most B non-root nodes.

$\zeta \leftarrow 0$
Precompute all candidate-pair log-probabilities $\log q_i^{\mathrm{bi}}$ using equation 9.
$\mathcal{F}_0 \leftarrow \{\epsilon\}$, $\ell_\zeta(\epsilon) \leftarrow 0$, $\mathcal{S} \leftarrow \varnothing$
for $d = 1, \dots, \gamma$ **do**
  $\mathcal{E}_d \leftarrow \{y_{1:d} : y_{1:d-1} \in \mathcal{F}_{d-1},\ y_d \in \mathcal{C}_d\}$
  for all $y_{1:d} \in \mathcal{E}_d$ **do**
    $\ell_\zeta(y_{1:d}) \leftarrow \ell_\zeta(y_{1:d-1}) + \log q_d^{\mathrm{bi}}(y_d \mid y_{d-1}, x_{\le t}) + \zeta$
  end for
  $\mathcal{F}_d \leftarrow \mathrm{Top}_W(\mathcal{E}_d; \ell_\zeta)$
  $\mathcal{S} \leftarrow \mathcal{S} \cup \mathcal{F}_d$
end for
$\mathcal{T} \leftarrow \{\epsilon\} \cup \mathrm{Top}_B(\mathcal{S}; \ell_\zeta)$
return $\mathcal{T}$

carried forward to the next decoding round, while those of all other branches are discarded. Since every committed token is sampled from the target model under exactly the same prefix as in standard autoregressive decoding, the procedure is lossless.

B Experimental Setup Details

We provide additional training and evaluation settings for the academic-scale experiments, long-context evaluations, and ablations in Section 4.

B.1 Training and Inference Configuration

Table 5 summarizes the configuration for the academic-scale experiments. We initialize the depth-calibration parameters to $\rho_i = \kappa_i = 0$, so the proposer logits and selector scores initially have unit weights. Academic-scale inference loads the target and drafter weights in BF16 and captures draft post-processing in a single CUDA Graph.

B.2 Evaluation and Timing Protocol

For our academic-scale evaluations, we use fixed seed-0 subsets with the sample counts listed in Table 6. Each configuration is evaluated once on the selected subset. Generation stops at EOS or after 2,048 new tokens. MT-Bench contains 80 two-turn dialogues, yielding 160 evaluated turns.

For each question or dialogue turn, time per output token (TPOT) is computed from the elapsed decoding time and the output-token count within the same timing window. The same decoding-time measurement protocol is used for all locally evaluated methods and the autoregressive baseline. Within each benchmark, speedup is the ratio of the mean autoregressive TPOT to the mean speculative-decoding TPOT. Acceptance length is averaged over questions or dialogue turns.

B.3 Long-context Evaluation Protocol

Shared settings. We compare UBTree with DSpark and DFlash on Ling3-Flash using the serving configuration in Section 4.2: tensor parallelism of degree 4 on four H200 GPUs, greedy decoding, and thinking disabled. Both long-context evaluations use concurrency 1 with CUDA Graphs disabled. We measure root-inclusive acceptance length τ as the number of generated tokens per target verification call, including the target-produced token. All methods use a maximum draft depth of 7, giving a maximum acceptance length of 8. DSpark and DFlash verify a single chain, while UBTree uses a budget of 64 non-root tree nodes per round.

Table 5: UBTree configuration for the academic-scale experiments, unless otherwise specified. The verification budget excludes the anchor.

Setting	Value
Proposer training	6 epochs
Joint proposer–selector training	2 epochs
Selector rank	256
Codebook MLP hidden width	1024
Training candidate support size	256
Draft depth	15 future positions plus an anchor
Candidate tokens per position K	64
Search width W	12
Verification budget B	64 non-root nodes
Depth bonus ζ	0

Table 6: Evaluation subset sizes. MT-Bench contains two-turn dialogues; the other counts refer to individual examples.

Benchmark	Examples / dialogues
GSM8K	128
MATH-500	128
AIME	30
HumanEval	128
MBPP	128
LiveCodeBench	128
MT-Bench	80

LongBench-v2. We evaluate long-input generation on LongBench-v2 (Bai et al., 2025). From the 503 source examples, we retain the same 401 requests with at most 260K prompt tokens for all methods. Generation stops at EOS or after 2,048 new tokens. We group requests by their prompt length before generation into six context-length bins with upper boundaries of 32K, 64K, 96K, 128K, 192K, and 260K tokens. Each bin contains the naturally occurring benchmark inputs in that length range. Table 8 reports acceptance length and request counts for these bins, together with results over the full evaluation subset.

SWE-bench. We evaluate acceptance as agent histories grow on SWE-bench (Jimenez et al., 2024). We first run target-only inference on 30 cases and freeze the resulting 1,482 complete requests. Each method then replays the identical recorded messages, tool calls, and tool outputs with a fixed generation limit of 256 tokens. This shared replay protocol aligns the agent context across methods at every request. To match the context-length axis in Figure 1(c), we group requests by the increase in prompt tokens relative to the first request of the corresponding case, using bins ≤ 0, 0–4K, 4–8K, 8–16K, 16–32K, 32–64K, and 64–131K. The ≤ 0 bin contains requests whose prompt length is no greater than the initial prompt, including context truncation or reconstruction events. Table 9 reports acceptance length and the number of cases represented in each bin; its Overall column reports request-pooled acceptance length over all 1,482 requests.

As a complementary analysis, we summarize the same replay requests by trajectory progress. We divide each case chronologically into four stages containing approximately equal numbers of requests: 0–25%, 25–50%, 50–75%, and 75–100%. Within each case and stage, we compute acceptance length as total generated tokens divided by total target verification calls, then average equally across the 30 cases. These stages measure relative progress through an agent trajectory. Since the amount of context added per request varies across cases and requests, the stage boundaries do not correspond to fixed context lengths.

B.4 Ablation Protocol

For the training-temperature ablation in Table 4, we regenerate target responses at each T_{train} and use them for both proposer training and joint proposer–selector training. Each joint run starts from the proposer trained at the corresponding temperature. The target distribution for KL supervision uses the same T_{train}, following Eq. equation 7. Across temperatures, we retain the same selector architecture, training configuration, and tree-search configuration. Each resulting checkpoint is evaluated at both $T_{\text{infer}} = 0$ and $T_{\text{infer}} = 1$. For the training-objective ablation, all variants start from the same proposer checkpoint and use identical training responses. Hard-label selector CE excludes positions whose sampled target token falls outside the training candidate support, whereas forward KL supervises all valid positions.

C Additional Experiments

C.1 Drafting Latency Analysis

We profile Qwen3-4B using Hugging Face Transformers-based implementations on 128 GSM8K questions at $T_{\text{infer}} = 0$, using the search budgets in Section 4.1. For each question, all methods run on the same H200 GPU in randomized order, and we average measurements over ten complete passes. Timing excludes prefill and the first complete speculative round. Stage latencies are measured with CUDA events, while total round latency is measured independently using synchronized wall-clock time. For UBTree, draft post-processing includes LM-head projection, candidate selection, selector scoring, and tree construction, which are timed jointly. The ms/token column reports TPOT averaged over questions.

Table 7 shows that UBTree reduces draft post-processing latency from DARTree's 3.36 ms to 1.38 ms per round, a reduction of 58.8%, while target verification takes approximately 20.9 ms for both methods. The principal latency reduction therefore occurs on the parallel selector side. UBTree achieves a higher acceptance length than DARTree (12.58 versus 12.50) while reducing decoding latency from 2.41 to 2.21 ms per token.

Table 7: Drafting latency profiling on Qwen3-4B. Stage and total latencies are in ms per verification round.

Method	Proposer stage	Draft post-processing	Verification	Other	Total	τ	ms/token
DARTree	3.51	3.36	20.94	1.64	29.47	12.50	2.41
UBTree	3.41	1.38	20.88	1.61	27.30	12.58	2.21

C.2 Long-context Tasks

Tables 8 and 9 report the quantitative results underlying Figure 1(c), together with the overall acceptance length on each benchmark. All results follow the long-context evaluation protocol in Appendix B.3.

Results. UBTree achieves the highest acceptance length in every context-length bin on both benchmarks. On LongBench-v2, UBTree reaches $\tau = 5.326$–5.870, improving over the strongest baseline, DSpark, by 34.6–42.1%. At 192–260K prompt tokens, UBTree attains $\tau = 5.804$, compared with 4.084 for DSpark and 3.919 for DFlash, retaining nearly the same acceptance length as in the shortest bin (5.833). Overall, UBTree achieves $\tau = 5.693$, a 37.6% improvement over DSpark. On SWE-bench, UBTree reaches $\tau = 4.829$–5.516 across accumulated-context bins, exceeding DSpark by 29.0–37.8%. Its overall request-pooled acceptance length is 5.341, compared with 3.981 for DSpark and 3.737 for DFlash, improving over the strongest baseline by 34.2%. These results demonstrate UBTree's sustained acceptance advantage across both long initial prompts and growing agent histories.

Trajectory-stage results. Under the complementary trajectory-stage aggregation defined in Appendix B.3, UBTree achieves acceptance lengths of 5.043, 5.364, 5.348, and 5.476 across the four

Table 8: Acceptance length (τ) on LongBench-v2 with Ling3-Flash, grouped by request-start prompt length. All methods use the same 401 requests. The Overall column reports results over the full evaluation subset. Bold marks the best acceptance length per column.

Prompt tokens	$\leq$ 32K	32–64K	64–96K	96–128K	128–192K	192–260K	Overall
Requests	109	71	47	69	73	32	401
DSpark	4.248	4.297	3.795	3.976	4.166	4.084	4.136
DFlash	4.020	4.046	3.578	3.678	3.944	3.919	3.898
UBTree	**5.833**	**5.870**	**5.326**	**5.351**	**5.798**	**5.804**	**5.693**

Table 9: Acceptance length (τ) on SWE-bench with Ling3-Flash, grouped by accumulated prompt tokens relative to the first request in each case. Covered cases gives the number of cases represented in each bin. The Overall column reports request-pooled acceptance length over all 1,482 requests from 30 cases. Bold marks the best acceptance length per column.

Accumulated tokens	≤ 0	0–4K	4–8K	8–16K	16–32K	32–64K	64–131K	Overall
Covered cases	30	30	28	27	23	13	2	30
DSpark	3.718	3.743	4.023	3.998	4.089	4.023	3.965	3.981
DFlash	3.442	3.524	3.795	3.709	3.836	3.805	3.770	3.737
UBTree	**4.842**	**4.829**	**5.447**	**5.389**	**5.516**	**5.408**	**5.465**	**5.341**

successive SWE-bench stages, compared with 3.835, 3.989, 3.942, and 3.999 for the strongest baseline, DSpark. The corresponding gains range from 31.5% to 36.9%. This summary captures UBTree's advantage throughout agent trajectories, while the 29.0–37.8% gains reported in the main text correspond to the accumulated-context results in Table 9. The two ranges summarize the same replay workload under different groupings.

D LIMITATIONS

Despite UBTree's strong empirical performance, two limitations remain.

Tree drafting is sensitive to very high concurrency. Speculative decoding is motivated by the spare compute available when autoregressive inference is limited by memory bandwidth, as is common in LLM decoding. Verifying multiple draft tokens together uses this capacity to amortize memory-access costs. As concurrency increases and inference becomes compute-bound, the spare capacity available for speculation shrinks, potentially eliminating speedups for both single-path and tree-based methods. Sensitivity to this transition is shared by all tree-based drafters and is not specific to UBTree. Tree verification spends additional compute on alternative continuations and therefore favors the part of the compute–bandwidth spectrum with more spare compute. Adaptive tree drafting can mitigate this limitation by varying the verification budget with available compute: branching allows the number of verified candidates to vary beyond the token count of a fixed single-path block, although the proposer block size still bounds draft depth. Our load-dependent tree budgets already retain speedups at the tested concurrency levels (Table 2), and more flexible runtime adaptation could further improve resource use. UBTree can also serve low-concurrency workloads where spare compute is abundant. We therefore view this limitation as a constraint on the favorable deployment regime, rather than a fundamental obstacle to UBTree's practical value.

Adaptive tree strategies rely on searched configurations. Our current adaptive strategy uses tree budgets B and search widths W selected through configuration search for each model and concurrency level. These settings are fixed for a given serving load; they do not determine the budget from the transition-score distribution of each decoding round. Although the tree's contents depend on the predicted scores, the budget-selection policy remains empirically tuned and is not guaranteed to be optimal as prediction uncertainty and available compute change. A more principled strategy would select tree sizes at each decoding round using estimates of acceptance benefit derived from transition

scores, together with a cost model reflecting hardware and serving load. We leave developing and evaluating such a policy to future work.

E Additional Related Work

Speculative decoding. In addition to autoregressive (Leviathan et al., 2023; Chen et al., 2023; Li et al., 2024a), parallel (Chen et al., 2026; Liu et al., 2026; Inco AI, 2026), semi-autoregressive (Huang et al., 2026; Cheng et al., 2026), and tree-based (Cai et al., 2024; Miao et al., 2024; Li et al., 2024b; 2026d; Ringel & Romano, 2026; Li et al., 2026c) drafting methods mentioned, existing research also explored mixing methods for speculative decoding. PARD adapts autoregressive models for parallel token prediction (An et al., 2026), while DiffuSpec uses pretrained diffusion language models as drafters (Li et al., 2026a).

Lossy acceleration for LLMs. Lossy acceleration reduces inference cost by allowing deviations from the target model's output distribution. BiLD (Kim et al., 2023) coordinates small and large models through confidence-based fallback and discrepancy-based rollback. The lossy extension of DistillSpec (Zhou et al., 2024) relaxes token acceptance using a lenience factor, while Fuzzy Speculative Decoding (Holsman et al., 2025) bases acceptance on divergence between draft and target distributions. Speculative cascades (Narasimhan et al., 2025) implement model-deferral rules through speculative execution, selecting between draft and target distributions; DIVERSED (Wang et al., 2026b) instead learns context-dependent mixtures of these distributions for verification. Judge Decoding (Bachmann et al., 2025; Garipov et al., 2026) accepts otherwise-rejected tokens using a lightweight correctness classifier on target hidden states. In contrast to these relaxed verification methods, UBTree improves candidate coverage and path quality through joint proposer–selector training while retaining target-distribution-preserving verification.

Multi-token language generation. Another line of work incorporates multi-token prediction into the language model itself. Gloeckle et al. (2024) train multiple prediction heads on a shared backbone to predict several future tokens simultaneously. Diffusion language models, including LLaDA (Nie et al., 2026) and Dream (Xie et al., 2025), learn to recover masked tokens and generate text through iterative parallel denoising. LLaDA is trained from scratch, whereas Dream adapts pretrained autoregressive weights. Block diffusion (Arriola et al., 2025) combines autoregressive generation across blocks with parallel denoising within each block, enabling flexible-length generation and prefix KV caching. Large-scale block diffusion with different recipes enriches the explored space of taming diffusion for multi-token fast generation, including WeDLM (Liu et al., 2025), Mercury (Inception Labs et al., 2025), and DiffusionGemma (DiffusionGemma Team et al., 2026). These approaches modify the generative model or its training to enable multi-token generation, while having the risk of degrading the sample quality.