RETREE: BUDGET-EFFICIENT TREE SPECULATIVE DECODING VIA PATH GUIDANCE AND SIBLING RECOVERY

Anonymous authors
Paper under double-blind review

ABSTRACT

Tree speculative decoding spends a fixed target-model budget on multiple candidate continuations, but exact verification can stop while scored siblings remain unused. We present ReTree, an approximate decoding policy that recovers an existing sibling using historical divergence counts, current target probability support, and stop-token constraints. Recovery reuses the same tree forward pass and introduces no learned correction head. A request-local path prior additionally changes which candidates enter the tree. On Qwen3-4B and Qwen3-8B at a 16-node budget, the measured speed gains over DDTree are 4.2–5.1% at temperature 0 and 8.8–9.1% at temperature 1. Recovery accounts for most of the gain; path guidance increases committed length but does not consistently improve speed over recovery alone. Additional CSD-style and MARS-style comparisons show that other relaxed verifiers can exceed ReTree's speed at some operating points. These comparisons do not establish quality-matched superiority. We characterize the contribution as tree-local reuse of already scored candidates, with an explicit speed–quality trade-off rather than preservation of the target distribution.

1 INTRODUCTION

Speculative decoding reduces autoregressive inference cost by drafting future tokens cheaply and verifying them in parallel with the target model (Stern et al., 2018; Leviathan et al., 2023; Chen et al., 2023). Its speed depends on proposal cost, proposal accuracy, and verification width. Autoregressive drafters and feature heads model token dependencies but incur sequential or structural overhead (Cai et al., 2024; Li et al., 2024a; 2025). DFlash predicts a block in one pass, reducing proposal cost while leaving its position-wise distributions independent of earlier sampled block tokens (Chen et al., 2026). Domino, DSpark, DominoTree, and JetSpec add causal, semi-autoregressive, or branch-correction components to address this limitation, with additional training and construction costs (Huang et al., 2026; Cheng et al., 2026; Hu et al., 2026; Lin & Jang, 2026).

Tree speculative decoding tackles a second limitation: a single draft path can miss many plausible continuations. By organizing candidates into a tree and verifying all selected nodes with tree attention, the target can follow whichever branch matches its own posterior (Miao et al., 2024; Cai et al., 2024; Li et al., 2024b; Chen et al., 2024). DDTree extends this idea to block drafters by building a tree from parallel per-position distributions under a node budget (Ringel & Romano, 2026). Yet the budget is not a minor implementation detail. In a serving system, speculative nodes consume target-side sequence length, memory bandwidth, and batching capacity (Kwon et al., 2023; Zheng et al., 2024). Increasing the tree size often raises accepted length, but after a point the extra verification work can reduce realized speedup. This makes tree decoding a cost-utilization problem: the useful metric is end-to-end time per committed token together with task quality. Pre-verification work per token is a diagnostic of construction cost.

Under a fixed node budget, exact tree following stops when the target-selected token is absent from the current node's children. Other scored children can still have target support. Figure A1(b) illustrates lexical and formatting alternatives, but probability support alone does not establish semantic equivalence. Reusing these alternatives therefore requires an explicit relaxation of the commit rule and a corresponding quality evaluation.

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.

We propose ReTree, a tree-local sibling recovery policy. When exact following fails, it may select an existing child that passes a historical frequency gate, a current target probability-ratio gate, and stop-token constraints. The recovered child's descendants have already been scored in the same target pass. Request-local n-gram guidance is a separate construction component that can change the available siblings. Our ablation identifies recovery as the main source of speed gains; it does not establish that path guidance consistently improves speed or that the two components have a universal synergistic effect.

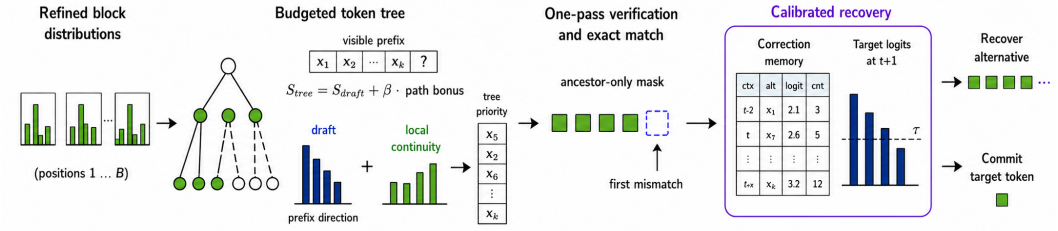


Figure 1: Overview of ReTree. Path-Guided Tree Construction allocates a fixed node budget with a request-local continuity prior. Target-Gated Sibling Recovery reuses already scored siblings when exact tree following stops, then updates its correction memory for later mismatches.

Figure 1 summarizes the proposal, one-pass verification, and recovery stages. The commit rule is deliberately approximate: an existing sibling can be emitted even when it differs from the target-selected token.

Experiments on Qwen3-4B and Qwen3-8B show consistent gains over a same-budget tree verifier across greedy and stochastic decoding. The budget sweep further shows that ReTree keeps pre-verification cost per accepted token nearly flat, whereas correction-head tree construction becomes increasingly expensive as the node budget grows.

The paper makes three contributions:

- We formulate reuse of target-scored siblings as a fixed-budget, approximate commit policy, explicitly distinguishing extra committed tokens from exact agreement with the target decoder.
- We adapt frequency and probability gating to existing tree children, allowing recovery and continuation within one ancestor-masked target pass. Request-local path guidance provides an independently ablated allocation rule.
- We separate construction, verification, and commit costs, and compare component ablations and relaxed-verification operating points. The results delimit the budget-dependent benefits without claiming quality-matched dominance over other approximate methods.

2 Related Work

Speculative Decoding and Drafter Design Speculative decoding drafts future tokens with a cheaper mechanism and verifies them with the target model in parallel (Stern et al., 2018; Leviathan et al., 2023; Chen et al., 2023). Existing systems use autoregressive or distilled drafters (Sun et al., 2023; Zhou et al., 2024; Liu et al., 2023; He et al., 2024), attach multi-token heads (Cai et al., 2024; Ankner et al., 2024), reuse target hidden states (Li et al., 2024a; 2025), or exploit early exits and shallow subnetworks (Elhoushi et al., 2024; Liu et al., 2024). These methods mainly improve proposal quality or proposal latency. ReTree instead asks how much of an already verified tree can be converted into committed tokens.

Parallel Block Drafting and Path Dependence Parallel block drafters reduce proposal latency by predicting multiple positions in one pass. DFlash produces a full block of logits from target features (Chen et al., 2026), following a broader line of discrete and block diffusion language modeling (Lou et al., 2024; Nie et al., 2025; Arriola et al., 2025). DART shows that parallel-logit trees benefit from local continuity constraints (Liu et al., 2026). Domino, DSpark, JetSpec, and DominoTree address

weak intra-block or branch dependency with causal heads, semi-autoregressive modules, confidence scheduling, or branch correction (Huang et al., 2026; Cheng et al., 2026; Hu et al., 2026; Lin & Jang, 2026). ReTree uses a request-local prior for candidate allocation. This component does not repair the block drafter's conditional distributions, and its effect on committed length is distinct from the speed gain due to relaxed verification.

Tree Verification and Node Budgets Tree verification allocates speculative capacity to multiple continuations. SpecInfer organizes candidates as a tree for serving (Miao et al., 2024); Medusa and EAGLE variants verify head combinations or dynamic trees (Cai et al., 2024; Li et al., 2024b); Sequoia studies hardware-aware tree structures (Chen et al., 2024). OPT-Tree studies objective-driven draft-tree allocation under a budget (Wang et al., 2025), while traversal verification reduces premature stops by changing the verification order for tree drafts (Weng et al., 2025). DDTree builds a prefix-closed tree directly from block-draft marginals and verifies it with an ancestor-only mask (Ringel & Romano, 2026). Cost-aware and block-iterative variants emphasize that tree size is a serving resource (Zhang et al., 2026; Shi et al., 2026). ReTree keeps the one-pass ancestor-masked verifier, but improves the useful work obtained from a fixed node budget rather than simply increasing the budget or adding a branch scorer.

Memory-Based Construction and Relaxed Verification Token Recycling reuses target predictions to construct subsequent candidate trees (Luo et al., 2025). Goose combines context matching and transition statistics in a spine tree, while retaining exact greedy token matching during verification (Jin et al., 2026). Graft combines pruning and retrieval to refill candidate trees without relaxing verification (Shen et al., 2026). These methods use memory or retrieval primarily to change the proposal.

CSD combines correction counts with probability-ratio gating to rescue rejected draft tokens (Zhou et al., 2026). MARS relaxes verification for a top-two candidate using a target-logit ratio (Song et al., 2026). ReTree follows CSD's frequency-and-support principle, but restricts its eligible set to children exposed by the active tree at a mismatch. Both linear rescue of an existing draft token and tree-local rescue can reuse precomputed target scores; the absence of an extra target pass is therefore not unique to ReTree. The empirical question is whether selecting among scored siblings provides a favorable speed–quality trade-off. We evaluate CSD-style recovery on DFlash and MARS-style verification on DDTree as adapted controls, rather than treating exact decoders as the only relevant comparison.

3 Method Overview

3.1 Problem Setup

Let M be the target language model, $x_{1:t}$ the cached committed prefix, and b an already selected next target token. The first root comes from prompt prefill; later roots are the terminal target tokens carried from the preceding round. A stop root terminates generation before another round is launched. Selecting b does not require an additional target invocation per round. A block drafter then provides $L = B - 1$ speculative token distributions after the root:

$$q_i(\cdot) = D_i(\cdot \mid x_{1:t}, b), \quad i = 1, \ldots, L. \tag{1}$$

Linear block speculation samples or selects one trajectory from these distributions and verifies that trajectory. ReTree instead constructs a prefix-closed tree $\mathcal{T}$ with at most N non-root speculative nodes. The root is verified with the tree but is not counted in N. The target therefore evaluates at most $N + 1$ current tokens in the round.

The goal is to increase committed tokens under this fixed node budget while bounding the relaxation with target-side evidence. Standard tree verification commits the longest exact path. ReTree keeps exact following as the first choice, but adds a guarded approximate step when it stops. Recovery is allowed only among children already present in $\mathcal{T}$ at the mismatch node.

3.2 PATH-GUIDED TREE CONSTRUCTION

For a partial continuation $u_{1:d}$ with depth $d \leq L$, the base tree score is the factorized block-draft score

$$S_{\text{draft}}(u_{1:d}) = \sum_{i=1}^{d} \log q_i(u_i). \tag{2}$$

This score is not an autoregressive target probability; it is a proposal score used to allocate verification nodes. The weakness of this score is that it comes from position-wise block marginals: a locally likely token at one depth may be a poor continuation of the specific branch selected at earlier depths. ReTree therefore adds a lightweight path-consistency prior computed from the current request. Let $C(p, a)$ be the number of times token a follows prefix p inside the visible context window of the same prompt and generated prefix. For a candidate token u_i, we define a bounded path bonus

$$\begin{aligned} B_i(u_i) &= \mathbf{1}\{r_i(u_i) \leq R\} \min(B_{\max}, A_i(u_i)), \\ A_i(u_i) &= \textstyle\sum_{n=2}^{n_{\max}} \eta^{n-2} \log(1 + C(p_{i,n}, u_i)). \end{aligned} \tag{3}$$

where $r_i(u_i)$ is the draft rank (starting at 1), $p_{i,n}$ is the length-$(n-1)$ suffix before u_i, and R is a rank cap. The final tree construction score is

$$S_{\text{tree}}(u_{1:d}) = S_{\text{draft}}(u_{1:d}) + \beta \sum_{i=1}^{d} B_i(u_i). \tag{4}$$

This signal is request-local and non-anticipating: it is built from the current prompt and generated prefix only, not from an offline corpus or future target tokens. The rank cap makes the guidance conservative. It can reorder plausible draft candidates, but it cannot promote a very low-rank token solely because it matches a surface pattern. In this sense, the path prior is a cheap inference-time proxy for local branch consistency, not a replacement for a trained causal head. The prior is inspired by DART's observation that trees constructed from parallel logits benefit from local continuity constraints, but it is used here as a rank-gated budget-allocation term inside a target-verified tree rather than as part of a separate diffusion-style drafter or pruning pipeline. Two candidate branches may receive comparable draft marginal scores, but the branch that continues a local pattern in the current request obtains a larger path bonus and is more likely to occupy the fixed tree budget. The guidance therefore changes node allocation, not the target verification rule. Figure A1(a) illustrates this bounded re-ranking effect.

A best-first heap prioritizes exposed prefixes by Eq. (4). Candidates retain their original draft-probability order at each depth. Selecting a prefix adds its final token under its selected parent, then exposes the next draft-ranked sibling and the highest-draft-probability child, when available. Selection stops at N nodes or an empty heap, preserving prefix closure. The branch-specific score ranks only exposed candidates; this heuristic does not enumerate the globally highest-scoring N prefixes. Appendix A.2 gives the candidate pool and expansion rules.

The budget N controls speculative allocation per request: larger trees explore more alternatives at higher target-side cost. ReTree aims to obtain more target-supported committed tokens from this allocation without making construction a bottleneck.

3.3 ONE-PASS TREE VERIFICATION

ReTree flattens the root and selected tree nodes into one target input. A node at depth d receives the position corresponding to the d-th speculative step after the root. We build a visibility matrix

$$V_{ij} = 1 \quad \text{iff node } j \text{ is node } i \text{ or an ancestor of } i, \tag{5}$$

and convert it into an attention mask. Each tree node can attend to the verified prefix and to its own ancestors, but not to other branches. The target model therefore scores all selected branches in one forward pass.

After the target pass, the verifier follows the target posterior through the tree. At temperature 0.0, the posterior token at a node is the target argmax. At temperature 1.0, we sample this posterior token from the target logits with the same decoding temperature. The exact walk and the recovery gate are both conditioned on this target posterior token. If it exists among the node's children, the

corresponding child is accepted and the walk continues. If it does not exist, exact matching stops and recovery is considered.

Let π_i contain the root and the path through current node i. Its logits ℓ_i represent $M(\cdot \mid x_{1:t}, \pi_i)$ and are used to choose a child. After accepting or recovering child u, traversal moves to u and uses ℓ_u, already conditioned on the path including u. Multiple recoveries may occur in one round. At a leaf or failed recovery, the final target token is selected from the current node's logits and carried to the next round; a stop token terminates generation. Cache commit retains only the chosen path. Correct conditioning after recovery does not make the recovery choice itself distribution preserving.

3.4 Recovery-Memory Initialization and Online Evolution

The recovery rule needs a running estimate of which tree-local divergences recur. Following the two-stage protocol of CSD, we initialize this estimate once from tree-verification traces collected on calibration data disjoint from evaluation, then continue updating it causally during inference. This initialization is a forward-only collection pass: it optimizes no parameters and uses no benchmark labels. At a node where the target selects token v, we rank the node's children by target logit and record the top non-stop alternatives u that are not equal to v. The count $\mathcal{M}(u, v)$ is incremented for each recorded pair. If the target token or child token is a stop token, the pair is skipped.

The update records only divergences exposed inside the tree verifier, rather than arbitrary draft errors. Ranking by target logit further concentrates the memory on children that the target considers plausible at the mismatch context.

Unlike linear correction, a tree mismatch exposes several siblings whose membership depends on node allocation. ReTree therefore records only events between the target posterior and siblings that consumed budget and were scored at an actual tree stop. Recovery then selects one existing child, preserving a single prefix-closed commit path for cache compaction.

When path guidance is enabled, the tree distribution changes slightly: some locally coherent children become more likely to enter the fixed node budget. We therefore initialize and update the recovery memory under the same active tree construction used at inference. This keeps its statistics matched to the candidate distribution that produced the tree children.

All reported ReTree runs use prior initialization followed by online evolution. The frequency gate uses a snapshot of $\mathcal{M}$ taken before the current mismatch is recorded. The current event therefore cannot make itself immediately recoverable; it can only provide evidence for later mismatches. This causal ordering allows the memory to adapt to the request stream without using the outcome of the current recovery decision.

3.5 Target-Gated Sibling Recovery

At inference time, sibling recovery is invoked only after exact tree following fails. Let v be the target posterior token at the mismatch node, and let $\ell(\cdot)$ be the target logit at that node. A child u is recoverable only if

$$\mathcal{M}(u, v) \geq \lambda, \qquad \ell(u) - \ell(v) \geq \log \rho, \qquad u, v \notin \mathcal{S}_{\text{stop}}, \tag{6}$$

where λ is the frequency threshold, ρ is the target-support ratio, and $\mathcal{S}_{\text{stop}}$ is the stop-token set. Since both logits share the same softmax denominator, the second condition is equivalent to $p_M(u)/p_M(v) \geq \rho$ at the current node. It measures relative target support, not semantic equivalence. Candidate children are considered in target-logit order, and the first child that satisfies all gates is accepted. No new node is added to the tree, and no additional target pass is requested; recovery only changes whether a target-scored sibling at the current node can be committed.

The rule is deliberately bounded. The frequency condition requires the same child-target divergence to recur across previously observed contexts. The logit condition requires the current target model to support the child at the specific mismatch node. Thus recovery is checked by the target model rather than accepted from draft-side confidence alone. This constraint limits unsupported substitutions, although it does not restore distributional equivalence. The stop-token condition blocks both directions of a stop-token substitution, so recovery cannot convert a target stop decision into a non-stop continuation or vice versa. Frequency is checked before the online update, preventing the current

mismatch from making itself immediately recoverable. Nevertheless, accepting $u \neq v$ changes the target output and does not preserve greedy equivalence or the target sampling distribution.

3.6 Algorithm and Complexity

Appendix Algorithm 1 summarizes one decoding round. The target verifies at most $N + 1$ current tokens, where N is the non-root tree budget. Path guidance changes only the best-first tree score. Recovery does not request another target forward pass and does not add new tree nodes. Its overhead is a small lookup in $\mathcal{M}$ and a logit-order scan over the current node's children. Thus, the method changes tree allocation and the acceptance rule under a fixed verification budget rather than increasing the budget.

4 Experiments

4.1 Setup

Models and evaluations. We evaluate on Qwen3-4B and Qwen3-8B (Yang et al., 2025) with thinking mode disabled. Our experiments are measured on NVIDIA RTX 5090 GPUs with block size 16 and a maximum generation length of 2048 tokens. We report results at temperatures 0.0 and 1.0. The main table focuses on the low-budget 16-node setting, while the budget sweep covers tree budgets $\{16, 32, 64, 128, 256\}$. Tasks span Math: GSM8K (Cobbe et al., 2021), MATH-500 (Hendrycks et al., 2021; Lightman et al., 2023), and AIME25 (Mathematical Association of America, 2025); Code: HumanEval (Chen et al., 2021), MBPP (Austin et al., 2021), and LiveCodeBench (LCB) (Jain et al., 2025); and Chat: MT-Bench (Zheng et al., 2023). We report end-to-end decoding speedup over an autoregressive baseline and mean committed length τ per verification round. Tables call this accepted length; it includes recovered tokens and does not measure exact agreement with the target decoder. For math tasks we report conditional exact-match scores among successfully extracted answers, with the denominator defined in Appendix B.3.

Calibration and implementation. DDTree and ReTree use the native DFlash draft model as the proposal network; the target model and drafter are otherwise shared with the DFlash baseline. Following CSD, we use a one-time, forward-only calibration pass on data disjoint from evaluation, followed by causal online updates (Zhou et al., 2026). No evaluation prompt, answer, or generated output enters the prior. ReTree records only sibling divergences exposed by the active tree policy. Calibration is outside timed decoding and uses no gradients; online lookup and updates are included in post-verification latency. Appendix B gives the sources and sample counts. Unless otherwise specified, the frequency threshold is six, the target-support ratio is 0.01, and both record and recovery candidate sets use top-8 filtering. The path-guided tree uses $n_{\max} = 4$, rank cap $R = 8$, coefficient $\beta = 0.15$, longer-match weight $\eta = 1.5$, bonus cap $B_{\max} = 6$, and a 2048-token context window. Online updates are enabled in all reported ReTree runs and are applied after the current frequency decision, without benchmark labels. The same initialized table is reloaded for each dataset run and then evolves in a fixed request order within that run. The same reference backend is used for DDTree and ReTree unless otherwise noted. We use $(\rho, \lambda) = (0.01, 6)$ throughout the reported ReTree runs. These shared settings alone do not establish optimality or threshold robustness. Path-guidance lookup, heap construction, and tree compilation are included in the pre-verification latency measurements. Recovery is evaluated only after the target logits are available; its lookup and commit logic are therefore included in post-verification latency and in end-to-end speedup. The recovery memory is a sparse lookup table rather than a learned module and introduces neither learned parameters nor additional target-model evaluations.

Latency accounting. We use $L_{\text{round}} = L_{\text{pre}} + L_{\text{target}} + L_{\text{post}}$. The three terms cover proposal and tree preparation, the target forward pass, and tree walking, recovery, and cache commit, respectively. End-to-end speedup includes all three. The diagnostic L_{pre}/τ reports proposal-side work per committed token; it is not a substitute for total latency.

Baselines. The autoregressive baseline is block-size-1 target decoding in the same measurement harness. Table 1 compares EAGLE-3, DFlash, DDTree, and ReTree on Qwen3-4B and Qwen3-8B. EAGLE-3 uses 16 candidates, while DDTree and ReTree use 16 tree nodes. Figure 2 separately sweeps the budget against DominoTree; Table 2 gives internal variants. Averages are arithmetic

Table 1: Speedup over autoregressive (AR) decoding and mean accepted length τ (reported as speedup/τ, with the $\times$ symbol omitted) on the seven-task evaluation suite for Qwen3-4B and Qwen3-8B. Experimental settings and averaging rules are described in the text.

Qwen3-4B, temperature $= 0.0$								
Method	GSM8K	MATH-500	AIME25	HumanEval	MBPP	LCB	MT-Bench	Avg.
EAGLE-3	2.24/3.32	2.10/3.11	2.08/3.10	2.09/3.09	2.02/2.99	1.95/2.93	1.95/2.94	2.06/3.07
DFlash	4.80/6.53	5.70/7.84	5.31/7.27	4.86/6.64	4.45/6.09	5.05/7.09	2.61/4.35	4.68/6.54
DDTree	5.02/7.45	5.68/8.55	5.11/7.65	5.08/7.54	4.66/6.96	5.16/7.91	2.98/5.13	4.81/7.31
ReTree	**5.29/7.97**	**6.05/9.27**	**5.54/8.56**	**5.41/8.19**	**4.81/7.29**	**5.29/8.38**	**3.04/5.37**	**5.06/7.86**
Qwen3-8B, temperature $= 0.0$								
Method	GSM8K	MATH-500	AIME25	HumanEval	MBPP	LCB	MT-Bench	Avg.
EAGLE-3	2.21/3.27	2.09/3.10	2.07/3.07	2.17/3.21	1.93/2.86	1.80/2.95	1.82/2.75	2.01/3.03
DFlash	4.80/6.54	5.69/7.87	**5.25**/7.08	4.80/6.50	4.33/5.95	**5.15**/7.27	2.52/4.24	4.65/6.49
DDTree	4.87/7.36	5.53/8.62	4.84/7.86	4.99/7.53	4.43/6.74	4.94/7.87	2.84/5.09	4.63/7.30
ReTree	**5.14/7.89**	**5.82/9.18**	5.11/**8.61**	**5.12/7.91**	**4.64/7.15**	5.00/**8.24**	**2.92/5.38**	**4.82/7.77**
Qwen3-4B, temperature $= 1.0$								
Method	GSM8K	MATH-500	AIME25	HumanEval	MBPP	LCB	MT-Bench	Avg.
EAGLE-3	2.18/3.26	2.00/3.00	1.86/2.80	2.02/3.04	1.97/2.95	1.87/2.83	1.90/2.91	1.97/2.97
DFlash	4.39/6.00	4.75/6.67	3.45/4.92	4.41/6.04	4.11/5.66	4.57/6.50	2.44/4.07	4.02/5.69
DDTree	4.67/6.95	4.98/7.55	3.90/5.94	4.76/7.03	4.37/6.53	4.77/7.35	2.83/4.86	4.33/6.60
ReTree	**5.04/7.59**	**5.55/8.60**	**4.66/7.36**	**5.18/7.80**	**4.71/7.18**	**5.01/7.92**	**2.92/5.20**	**4.72/7.38**
Qwen3-8B, temperature $= 1.0$								
Method	GSM8K	MATH-500	AIME25	HumanEval	MBPP	LCB	MT-Bench	Avg.
EAGLE-3	2.18/3.26	1.96/3.00	1.86/2.80	2.05/3.09	1.97/2.95	1.90/2.93	1.91/2.91	1.98/2.99
DFlash	4.35/5.98	4.51/6.40	3.30/4.73	4.01/5.52	3.75/5.21	4.60/6.69	2.25/3.80	3.82/5.48
DDTree	4.49/6.83	4.62/7.32	3.50/5.79	4.36/6.58	4.06/6.17	**4.81**/7.68	2.54/4.54	4.05/6.42
ReTree	**4.90/7.55**	**5.31/8.47**	**4.18/7.19**	**4.70/7.23**	**4.34/6.70**	**4.81/7.89**	**2.63/4.86**	**4.41/7.13**

means of the seven task-level values, with equal task weights; boldface marks the best speedup and τ separately.

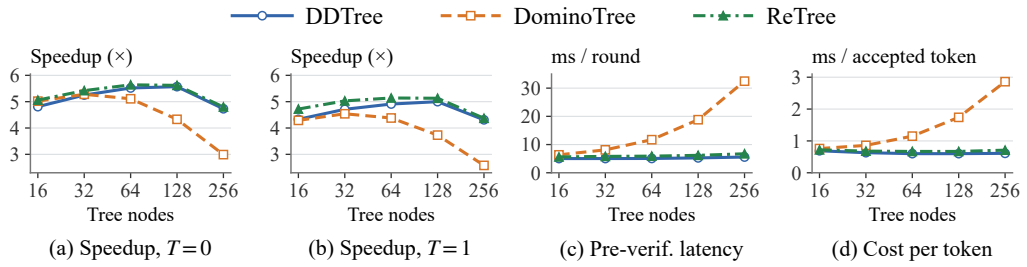
4.2 Main Results

At temperature 0.0, ReTree reaches 5.06×/7.86 on Qwen3-4B versus DDTree's 4.81×/7.31, and 4.82×/7.77 on Qwen3-8B versus 4.63×/7.30. At temperature 1.0 it reaches 4.72×/7.38 and 4.41×/7.13, again improving both metrics. The measured relative gains are 5.1%/7.4% and 4.2%/6.3% (speed/committed length) at temperature 0 for 4B and 8B, and 9.1%/11.7% and 8.8%/11.2% at temperature 1, respectively. Accepted length increases on all seven Qwen3-4B tasks at temperature 0.0, with the largest gains on MATH-500 and AIME25; code and chat show the same speed–acceptance direction.

The task-level gains span different output structures. Math reasoning contains short lexical and formatting divergences, whereas code adds identifiers, punctuation, and indentation. Both benefit from retaining several bounded alternatives. Under stochastic decoding, where exact-token agreement is harder, ReTree still improves the same-budget tree on both model sizes.

Figure 2 separates end-to-end throughput from pre-verification overhead when comparing ReTree with DDTree and DominoTree; Appendix Table A1 gives the values. DominoTree (Lin & Jang, 2026) obtains the longest paths through learned conditional branch scoring; JetSpec follows the related correction-head direction (Hu et al., 2026). ReTree instead changes a DDTree-style verifier without a new head, drafter retraining, or additional target candidates.

The latency and cost panels explain why accepted length alone does not determine throughput. At temperature 0.0, DominoTree's accepted-length advantage over ReTree grows from 6.1% at 16 nodes to 20.3% at 256 nodes, but its pre-verification latency grows from 13.1% to 382.4% higher than ReTree. At $T = 1$, only the 16-node path-length ordering reverses. Measured as latency per accepted token, ReTree stays nearly flat (0.67–0.72 ms/token at temperature 0.0 and 0.74–0.83



(a) Speedup, $T=0$ (b) Speedup, $T=1$ (c) Pre-verif. latency (d) Cost per token

Figure 2: Budget sweep on Qwen3-4B (RTX 5090): (a,b) end-to-end speedup; (c,d) pre-verification latency and cost at $T = 0$. The $T = 1$ latency trend is similar.

Table 2: Factorial Qwen3-4B ablation at temperature 0.0 in our RTX 5090 setup. Each cell reports speedup/τ, averaged over GSM8K, MATH-500, AIME25, HumanEval, MBPP, LCB, and MT-Bench. Recovery uses the heap tree without path guidance; ReTree enables both modules. Boldface marks the best speedup and accepted length at each budget.

TB	Base	Recovery	Path	ReTree
16	4.81/7.31	5.04/7.68	4.81/7.46	**5.06/7.86**
32	5.25/7.99	5.38/8.22	5.19/8.09	**5.42/8.48**
64	5.52/8.41	**5.64**/8.61	5.45/8.53	**5.64/8.86**
128	5.57/8.79	**5.69**/8.99	5.44/8.89	5.62/**9.16**
256	4.73/9.06	**4.93**/9.29	4.67/9.14	4.80/**9.43**

ms/token at temperature 1.0), whereas DominoTree rises from 0.76 to 2.86 and from 0.90 to 3.31 ms/token. Thus DominoTree obtains longer paths, but their benefit does not amortize the growing construction cost; ReTree keeps this cost stable. Applying recovery to DominoTree in Appendix Table A2 shows that it is complementary to learned branch scoring.

Appendix Table A3 includes all round-level costs. Target-forward latency is similar for ReTree and DominoTree; recovery takes 0.12–0.17 ms per round within the post-verification stage. Under greedy decoding, complete round latency differs by only 1.2% at 16 nodes and is 2.6%, 15.3%, 30.4%, and 42.9% lower for ReTree at 32–256 nodes. The growing difference is primarily construction cost, rather than excluded recovery or commit work. Stochastic decoding follows the same trend.

4.3 Relaxed Verification and Task Quality

Table 3 presents measured speedups at additional relaxed-verification operating points on Qwen3-4B at temperature 0. DFlash uses block size 16; DDTree and ReTree use a 16-node tree. The row labels are measured increases in committed length relative to each control's own exact baseline, not shared gate thresholds or matched quality levels (Appendix B.2). At the 5% operating point, ReTree's 5.06× exceeds 4.86× and 5.01×; at the 10% and 15% points both controls are faster. These timings demonstrate that relaxation can recover speed in both chain and tree settings. Without paired quality scores, they do not establish a speed–quality ranking.

Recovery is a gated approximate rule. Its probability-ratio test does not bound task error: $\rho = 0.01$ permits a candidate with one hundredth of the reference token's probability before the other gates. Table 4 reports conditional exact-match scores: records without extracted predictions or references are excluded. MATH-500 starts from 128 examples, but effective denominators may differ by method. Appendix B.3 relates extraction coverage to full-set accuracy and discusses the unresolved greedy-baseline discrepancy. These scores do not establish paired quality gains, code/chat quality preservation, or a quality-matched ranking of the relaxed controls.

Table 3: Measured relaxed-verification speedup over AR on Qwen3-4B, $T = 0$. $\Delta\tau$ is relative to the corresponding base decoder: DFlash (4.68×) or DDTree (4.81×). ReTree reaches 5.06×. The operating points are not matched by task quality.

$\Delta\tau$	DFlash + CSD-style recovery	DDTree + MARS-style verification
5%	4.86×	5.01×
10%	5.11×	5.23×
15%	5.32×	5.47×

Table 4: Conditional exact-match accuracy on Qwen3-4B at the 16-node budget. Percentages use each method's records with extracted predictions and references; they are not full-set accuracy. MATH-500 uses up to 128 examples.

Temp.	Method	GSM8K	MATH-500	AIME25
0.0	DFlash	91.4%	70.3%	13.3%
	DDTree	91.4%	68.0%	10.0%
	ReTree	91.4%	69.0%	13.3%
1.0	DFlash	92.2%	72.7%	16.7%
	DDTree	90.6%	74.2%	16.7%
	ReTree	91.4%	74.2%	16.7%

4.4 Ablation Study

Table 2 separates the two components. At 16 nodes, Recovery raises speedup from 4.81× to 5.04×, accounting for 0.23 of the full method's 0.25 increase. Path alone adds 0.08–0.15 committed tokens across budgets, but does not improve speed over Base. The complete method yields the largest committed length; it is slightly faster than Recovery at 16 and 32 nodes, tied at 64, and slower at 128 and 256. Thus candidate allocation affects path length, while its added cost can erase the speed benefit. The small low-budget differences are not claimed to be statistically significant.

4.5 Recovery and Budget Behavior

In the measured budget sweep, the committed-length gain over Base decreases from 0.55 accepted tokens at 16 nodes to 0.37 at 256, and recovered-token counts fall from 5418 to 3188 on the logged five-task subset. Counts also depend on generated lengths, so this decrease alone does not establish a lower per-token recovery probability. Still, ReTree improves speed at every budget without enlarging the target candidate set. This matters when tree nodes compete for batching capacity (Kwon et al., 2023; Zheng et al., 2024). At 256 nodes DominoTree accepts 11.34 tokens versus 9.43 for ReTree, but its construction cost yields only 2.99× versus 4.80× speedup; useful tokens per unit of work, rather than accepted length alone, determine throughput.

5 Conclusion

ReTree reuses target-scored siblings through a gated approximate commit rule. Recovery drives the speed gains; path guidance increases committed length but has a budget-dependent speed effect. Low-budget gains over DDTree are 4.2–5.1% at temperature 0 and 8.8–9.1% at temperature 1. Relaxed controls can be faster. The contribution is tree-local recovery with explicit accounting of construction, verification, and commit costs.

Limitations. Recovery does not preserve the target distribution. The available quality measurements are limited to math tasks and do not establish quality-matched superiority. Threshold transfer, frozen-memory performance, request-order sensitivity, and uncertainty in small speed differences remain unresolved. Evaluation uses two models from one family and one hardware platform; it does not establish gains under concurrent serving. Memory must be refreshed when the drafter, tokenizer, stop policy, or tree allocator changes.

AI Assistance Statement.

Generative AI tools were used to assist with manuscript writing and polishing, literature retrieval and discovery, and research ideation. All AI-assisted content was reviewed, verified, and revised by the authors, who take full responsibility for the final manuscript and its contents.

Reproducibility statement

An anonymous implementation is available at https://anonymous.4open.science/r/ReTree-7CD6. Section 4 specifies the models, hardware, evaluation tasks, and latency accounting. Appendix A gives the algorithmic details, Appendix B describes calibration and implementation settings, and Appendix C reports the complete budget sweep and round-level latency breakdown.

References

Zachary Ankner, Rishab Parthasarathy, Aniruddha Nrusimha, Christopher Rinard, Jonathan Ragan-Kelley, and William Brandon. Hydra: Sequentially-dependent draft heads for medusa decoding. *arXiv preprint arXiv:2402.05109*, 2024.

Marianne Arriola, Aaron Kerem 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.

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

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

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. *arXiv preprint arXiv:2602.06036*, 2026.

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

Zhuoming Chen, Avner May, Ruslan Svirschevski, Yuhsun Huang, Max Ryabinin, Zhihao Jia, and Beidi Chen. Sequoia: Scalable, robust, and hardware-aware speculative decoding. *arXiv preprint arXiv:2402.12374*, 2024.

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, Christopher Hesse, and John

Schulman. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*, 2021.

Mostafa Elhoushi, Akshat Shrivastava, Diana Liskovich, Basil Hosmer, Bram Wasti, Liangzhen Lai, Anas Mahmoud, Bilge Acun, Saurabh Agarwal, Ahmed Roman, Ahmed A. Aly, Beidi Chen, and Carole-Jean Wu. LayerSkip: Enabling early exit inference and self-speculative decoding. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 12622–12642. Association for Computational Linguistics, 2024.

Zhenyu He, Zexuan Zhong, Tianle Cai, Jason D. Lee, and Di He. REST: Retrieval-based speculative decoding. In *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pp. 1582–1595. Association for Computational Linguistics, 2024.

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. Measuring mathematical problem solving with the MATH dataset. In *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks*, volume 1, 2021.

Karl Moritz Hermann, Tomáš Kočiský, Edward Grefenstette, Lasse Espeholt, Will Kay, Mustafa Suleyman, and Phil Blunsom. Teaching machines to read and comprehend. In *Advances in Neural Information Processing Systems*, 2015.

Lanxiang Hu, Zhaoxiang Feng, Yulun Wu, Haoran Yuan, Yujie Zhao, Yu-Yang Qian, Bojun Wang, Peng Zhao, Daxin Jiang, Yibo Zhu, Tajana Rosing, and Hao Zhang. JetSpec: Breaking the scaling ceiling of speculative decoding with parallel tree drafting. *arXiv preprint arXiv:2606.18394*, 2026.

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.

Naman Jain, King Han, Alex Gu, Wen-Ding Li, Fanjia Yan, Tianjun Zhang, Sida I. 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*, 2025.

Tao Jin, Phuong Minh Nguyen, and Naoya Inoue. Goose: Anisotropic speculation trees for training-free speculative decoding. *arXiv preprint arXiv:2604.02047*, 2026. URL https://arxiv.org/abs/2604.02047.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with PagedAttention. In *Proceedings of the ACM SIGOPS 29th Symposium on Operating Systems Principles*, 2023.

Yaniv Leviathan, Matan Kalman, and Yossi Matias. Fast inference from transformers via speculative decoding. In *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pp. 19274–19286. PMLR, 2023.

Yuhui Li, Fangyun Wei, Chao Zhang, and Hongyang Zhang. EAGLE: Speculative sampling requires rethinking feature uncertainty. In *Proceedings of the 41st International Conference on Machine Learning*, pp. 28935–28948, 2024a.

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. Association for Computational Linguistics, 2024b.

Yuhui Li, Fangyun Wei, Chao Zhang, and Hongyang Zhang. EAGLE-3: Scaling up inference acceleration of large language models via training-time test. In *Advances in Neural Information Processing Systems*, 2025.

Hunter Lightman, Vineet Kosaraju, Yura Burda, Harri Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. Let's verify step by step. *arXiv preprint arXiv:2305.20050*, 2023.

Saw S. Lin and Jyh-Shing Roger Jang. DominoTree: Conditional tree-structured drafting with Domino for speculative decoding. *arXiv preprint arXiv:2607.08642*, 2026.

Fangcheng Liu, Yehui Tang, Zhenhua Liu, Yunsheng Ni, Duyu Tang, Kai Han, and Yunhe Wang. Kangaroo: Lossless self-speculative decoding for accelerating LLMs via double early exiting. In *Advances in Neural Information Processing Systems*, 2024.

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.

Xiaoxuan Liu, Lanxiang Hu, Peter Bailis, Alvin Cheung, Zhijie Deng, Ion Stoica, and Hao Zhang. Online speculative decoding. *arXiv preprint arXiv:2310.07177*, 2023.

Aaron Lou, Chenlin Meng, and Stefano Ermon. Discrete diffusion modeling by estimating the ratios of the data distribution. In *Proceedings of the 41st International Conference on Machine Learning*, 2024.

Xianzhen Luo, Yixuan Wang, Qingfu Zhu, Zhiming Zhang, Xuanyu Zhang, Qing Yang, and Dongliang Xu. Turning trash into treasure: Accelerating inference of large language models with token recycling. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, 2025. URL https://aclanthology.org/2025.acl-long.338/.

Mathematical Association of America. American invitational mathematics examination (AIME), 2025. https://maa.org/maa-invitational-competitions/, 2025.

Xupeng Miao, Gabriele Oliaro, Zhihao Zhang, Xinhao Cheng, Zeyu Wang, Zhengxin Zhang, Rae Ying Yee Wong, Alan Zhu, Lijie Yang, Xiaoxiang Shi, Chunan Shi, Zhuoming Chen, Daiyaan Arfeen, Reyna Abhyankar, and Zhihao Jia. SpecInfer: Accelerating large language model serving with tree-based speculative inference and verification. In *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 3*, pp. 932–949. ACM, 2024. doi: 10.1145/3620666.3651335.

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. *arXiv preprint arXiv:2502.09992*, 2025.

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

Yuhao Shen, Tianyu Liu, Xinyi Hu, Quan Kong, Baolin Zhang, Jun Dai, Jun Zhang, Shuang Ge, Lei Chen, Yue Li, Mingcheng Wan, and Cong Wang. Draft less, retrieve more: Hybrid tree construction for speculative decoding. *arXiv preprint arXiv:2605.20104*, 2026. URL https://arxiv.org/abs/2605.20104.

Weijie Shi, Qiang Xu, Fan Deng, Yaguang Wu, Jiarun Liu, Yehong Xu, Hao Chen, Jia Zhu, Jiajie Xu, Xiangjun Huang, Jian Yang, and Xiaofang Zhou. SpecBlock: Block-iterative speculative decoding with dynamic tree drafting. *arXiv preprint arXiv:2605.07243*, 2026.

Jingwei Song, Xinyu Wang, Hanbin Wang, Xiaoxuan Lei, Bill Shi, Shixin Han, Eric Yang, Xiao-Wen Chang, and Lynn Ai. MARS: Unleashing the power of speculative decoding via margin-aware verification. *arXiv preprint arXiv:2601.15498*, 2026. URL https://arxiv.org/abs/2601.15498.

Mitchell Stern, Noam Shazeer, and Jakob Uszkoreit. Blockwise parallel decoding for deep autoregressive models. In *Advances in Neural Information Processing Systems*, 2018.

Ziteng Sun, Ananda Theertha Suresh, Jae Hun Ro, Ahmad Beirami, Himanshu Jain, and Felix Yu. SpecTr: Fast speculative decoding via optimal transport. In *Advances in Neural Information Processing Systems*, 2023.

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

Yepeng Weng, Qiao Hu, Xujie Chen, Li Liu, Dianwen Mei, Huishi Qiu, Jiang Tian, and Zhongchao Shi. Traversal verification for speculative tree decoding. *arXiv preprint arXiv:2505.12398*, 2025.

An Yang et al. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025.

Shuai Zhang, Huachuan Qiu, Hongliang He, and Yong Dai. Cost-aware diffusion draft trees for speculative decoding. *arXiv preprint arXiv:2606.01813*, 2026.

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric P. Xing, Hao Zhang, Joseph E. Gonzalez, and Ion Stoica. Judging LLM-as-a-judge with MT-Bench and chatbot arena. In *Advances in Neural Information Processing Systems*, 2023.

Lianmin Zheng, Liangsheng Yin, Zhiqiang Xie, Chuyue Sun, Jeff Huang, Cody Hao Yu, Shiyi Cao, Christos Kozyrakis, Ion Stoica, Joseph E. Gonzalez, Clark Barrett, and Ying Sheng. SGLang: Efficient execution of structured language model programs. In *Advances in Neural Information Processing Systems*, 2024.

Xuwen Zhou, Fangxin Liu, Chao Wang, Xiao Zheng, Hao Zheng, Min He, Li Jiang, and Haibing Guan. Calibrated speculative decoding: Frequency-guided candidate selection for efficient inference. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 29677–29688. Association for Computational Linguistics, 2026. doi: 10.18653/v1/2026.acl-long.1369. URL https://aclanthology.org/2026.acl-long.1369/.

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.

SUPPLEMENTARY MATERIAL

This supplement provides the algorithmic details, implementation settings, task-level accuracy checks, complete budget sweep, compatibility study, and latency decomposition supporting the main paper. Sections are labeled A–C; supplementary tables and figures use the A prefix.

A ALGORITHMIC DETAILS

A.1 DECODING-ROUND PSEUDOCODE

Algorithm 1 ReTree Decoding Round

Input: committed prefix x, selected root b, target M, drafter D, budget N, memory $\mathcal{M}$
Output: newly committed segment and next root, or termination

Compute block-draft distributions and request-local path counts.
Build $\mathcal{T}$ by draft-rank expansion and scored heap selection (Appendix A.2).
Run one ancestor-masked target pass over b and the selected nodes.
Set current node i to the root and committed segment $P \leftarrow [b]$.
loop
  Select v from ℓ_i: argmax at $T = 0$, target sampling at $T = 1$.
  if $v \in \mathcal{S}_{\text{stop}}$ **then**
    return $\text{emit}(P) \,\|\, [v]$ and terminate generation.
  end if
  if a child of i has token v **then**
    Let u be this exact child.
  else
    Set u to the result of Algorithm 2 at i if it has children; otherwise set u to failure.
    if recovery fails **then**
      Retain the KV cache of P; **return** $\text{emit}(P) \,\|\, [v]$ with v as the next root.
    end if
  end if
  Append token u to P and set $i \leftarrow u$; the next step uses ℓ_u.
end loop

The function $\text{emit}(P)$ omits the root if the preceding round already output it; that root is not counted twice. Only the first round newly outputs its input root. A leaf has no children, so its target-selected token terminates the round through the failure branch. Multiple successful recoveries can precede that termination. The application also truncates at its configured generation-length limit.

Let K_r and K_u be the recovery and update limits, and d_i the number of children at a mismatch. After child ranking, one attempt needs $O(K_r + K_u)$ lookups and logit comparisons. A full comparison sort additionally costs $O(d_i \log d_i)$; per-round recovery work sums over all visited mismatch nodes. The target scores the same preselected nodes throughout this traversal.

A.2 PATH-GUIDED TREE ALLOCATION

Path guidance changes how draft candidates consume the fixed tree budget without altering target verification or introducing a learned head. Its request-local, rank-gated bonus uses counts from the visible prompt and committed prefix. The heap priority is

$$S_{\text{tree}}(u_{1:d}) = \sum_{i=1}^{d} \log q_i(u_i) + \beta \sum_{i=1}^{d} B_i(u_i), \tag{7}$$

where the cap on B_i bounds this allocation bonus; it is not an acceptance criterion.

To make the parent dependence explicit, let $p = u_{1:i-1}$ be a candidate parent prefix after the root. The incremental score of child token a is

$$g_i(a \mid p) = \log q_i(a) + \beta B_i(a; p), \qquad S_{\text{tree}}(pa) = S_{\text{tree}}(p) + g_i(a \mid p). \tag{8}$$

Here $B_i(a; p)$ is Eq. (3) evaluated using the suffix before a in $(x_{1:t}, b, p)$. The count table C is still formed only from the visible prompt and committed prefix; speculative prefixes query it without being counted as observations. Exposed siblings share the same parent score, so their relative priorities depend on g_i. Across parents the heap compares full prefix scores. The block distribution q_i is shared at depth i, whereas the path bonus depends on p.

Candidate order and expansion. At each depth i, retain $K = \min(N, |\mathcal{V}|)$ tokens $a_{i,1}, \ldots, a_{i,K}$ in descending draft probability q_i, where $\mathcal{V}$ is the vocabulary. This ordering is shared across parents and is not recomputed after adding the path bonus. Initialize the heap with the one-token prefix $a_{1,1}$ after root b. When prefix $pa_{i,r}$ is popped, add its final token as a node under the already selected parent p. If $r < K$, expose sibling prefix $pa_{i,r+1}$; if $i < L$, expose child prefix $pa_{i,r}a_{i+1,1}$. Each new entry receives its own full prefix score from Eq. (8), with the bonus evaluated on that branch. Repeat until N non-root nodes are selected or the heap is empty. Equal heap priorities are resolved by parent insertion index, then depth and draft rank; equal draft probabilities follow the backend top-K ordering.

Best-first selection refers only to the exposed heap frontier. A later draft-ranked sibling may have a larger augmented score but remain unexposed, and a positive bonus may make a child's score exceed its parent's. The procedure is therefore a prefix-closed allocation heuristic, without a global top-N or optimality guarantee. Verification remains conditioned on the selected tree path.

A.3 Recovery Rule and Online Memory

This appendix expands the recovery rule defined in the main text. Recovery is invoked only after exact tree following fails at a tree node. Let v be the target posterior token at that node, and let $C(i)$ be the children already present under node i. Prior initialization and online updates build a table $\mathcal{M}$ over tree-local sibling divergences:

$$\mathcal{M}(u, v) = \sum_{r \in \mathcal{E}_{<\mathrm{now}}} \mathbf{1}\left[\begin{array}{c} v_r = v, \quad u \in \mathrm{TopK}_{K_u}(C_r^{\mathrm{eligible}}), \\ u \neq v_r, \quad u, v_r \notin \mathcal{S}_{\mathrm{stop}} \end{array} \right], \tag{9}$$

where $\mathcal{E}_{<\mathrm{now}}$ contains recorded mismatch events from prior initialization and completed online updates, excluding the current event. At event r, v_r is the target posterior token and C_r^{eligible} contains the exposed children after removing exact matches and stop tokens; it is empty if v_r is a stop token. The top-K_u children are selected by the target logits at that event's node, after this filtering. Thus each recorded child increments only its observed pair (u, v_r), rather than counts for other target tokens.

At inference, a child $u \in C(i)$ is recoverable only if

$$\mathcal{M}(u, v) \geq \lambda, \qquad \ell_i(u) - \ell_i(v) \geq \log \rho, \qquad u, v \notin \mathcal{S}_{\mathrm{stop}}, \tag{10}$$

where $\ell_i(\cdot)$ is the target logit at node i, λ is the frequency threshold, and ρ is the target-support ratio. The frequency gate is evaluated before the online update. Thus, the current mismatch cannot make itself immediately recoverable. All reported ReTree runs use this causal online-update order. For a fixed tree, logit vector, and memory state, increasing either ρ or λ can only remove eligible siblings. These thresholds therefore define a conservative-to-aggressive recovery frontier. The main ReTree measurements use $(\rho, \lambda) = (0.01, 6)$. The additional CSD-style and MARS-style measurements in Table 3 are indexed by observed committed-length increases; these labels are not gate values. Neither the default thresholds nor a throughput comparison alone establishes a quality-optimal operating point.

A.4 Budget-Sweep Metrics

The main experiments report speedup, accepted length, and pre-verification overhead. For the budget sweep we additionally use the amortized construction cost

$$\mathrm{Cost}(N) = \frac{\text{pre-verification latency}(N)}{\tau(N)}, \tag{11}$$

where N is the tree budget and $\tau(N)$ is the mean accepted length. This metric separates two effects that are often conflated: a method may accept a longer path, but still lose throughput if each accepted

Algorithm 2 Target-Gated Sibling Recovery

Input: node i, children $C(i)$, target token v, logits ℓ_i, memory $\mathcal{M}$
Output: recovered child index or failure

Rank children $u \in C(i)$ by $\ell_i(u)$.
For the first K_r ranked children, store $f(u) = \mathbf{1}[\mathcal{M}(u, v) \geq \lambda]$ before updates.
if online update is enabled **then**
    Record up to K_u eligible pairs using Eq. (9).
end if
for each child u among the first K_r ranked children **do**
    if u or v is a stop token **then**
        continue
    end if
    if $f(u)$ and $\ell_i(u) - \ell_i(v) \geq \log \rho$ **then**
        return the index of child u
    end if
end for
return failure

token requires much more branch-construction work. This diagnostic excludes the target verification pass; end-to-end speedup remains the complete latency measure. Lower cost is better. Figure 2 plots this trade-off, and Table A1 gives the complete numeric values.

B Experimental Settings and Accuracy

B.1 Implementation Notes

All reported ReTree runs use the same DFlash drafter as the corresponding DFlash baseline; ReTree changes only the inference-time tree allocation and verification policy. The block size is 16, and the budget sweep uses tree budgets $N \in \{16, 32, 64, 128, 256\}$ at temperatures 0.0 and 1.0. Benchmark decoding uses a maximum of 2048 generated tokens, while prior initialization uses a shorter maximum generation length to collect divergence statistics efficiently. The released scripts fix the dataset order and random seeds used for shuffling. Before initialization, normalized-prompt hashes are compared with all evaluation prompts and any collision is removed. Each reported configuration is then evaluated with one benchmark pass over the specified split.

For the path-guided heap, we use the rank-gated n-gram variant. The default settings are maximum n-gram order 4, context window 2048, rank cap 8, bonus weight 0.15, logarithmic count scaling, longer-match weight 1.5, and maximum per-token bonus 6.0. For sibling recovery, the memory uses frequency threshold $\lambda = 6$, target-support ratio $\rho = 0.01$, update record top-k 8, and rescue top-k 8. The main experiments initialize the memory from a prior table and then update it online. Each update follows the pre-update frequency check in Algorithm 2. These settings keep the method gradient-free and avoid additional target-model passes during decoding.

Following the task-specific protocol used by CSD, initialization is a one-time forward-only collection pass at temperature 0.6, with 2,000 samples for most domains and all 7,500 examples from the seven original-MATH training configurations for competition mathematics; no repeated sampling is used. The GSM8K training split supplies the math-word-problem source, the original MATH training set supplies the competition-math source, the MBPP training split supplies a code-domain proxy, and the CNN/DailyMail training split (Hermann et al., 2015) supplies a chat-domain proxy. The HumanEval, MATH-500, AIME25, LiveCodeBench, and MT-Bench evaluation prompts are never used to build the prior. Exact normalized-prompt deduplication provides a final overlap check before collection. The resulting task-domain priors are merged before evaluation, after which the same counts evolve online. Only integer counts for token pairs (u, v) exposed as non-stop siblings are retained. Lookup cost is constant in the vocabulary size and linear only in the small ranked sibling set scanned at a mismatch. During online evolution, completed earlier verification events may affect later decisions, but the pre-update snapshot in Algorithm 2 prevents an event from rescuing itself. Neither initialization nor online evolution uses benchmark labels.

B.2 Additional Relaxed-Verification Controls

Table 3 contains Qwen3-4B measurements at temperature 0. The DFlash control uses a block of size 16 and CSD-style recovery; the DDTree control uses 16 non-root nodes and MARS-style verification. Each is compared with its own exact base decoder. Writing τ_m and $\tau_{\mathrm{base}(m)}$ for their mean committed lengths, the row label is

$$\Delta\tau_m = 100\left(\frac{\tau_m}{\tau_{\mathrm{base}(m)}} - 1\right)\%. \tag{12}$$

The 5%, 10%, and 15% rows therefore index observed length gains, not identical thresholds, absolute lengths, or quality levels across the two methods. The ReTree setting $(\rho, \lambda) = (0.01, 6)$ does not specify either control's gate settings. Per-point gate settings and paired quality scores are not reported for these operating points. The table supports a limited throughput comparison: relaxing verification can accelerate both base decoders, but these observations do not identify a quality-matched frontier.

B.3 Task Accuracy and Extraction Coverage

Table 4 covers Qwen3-4B at the 16-node budget. MATH-500 is capped at 128 examples after shuffling with seed 0. The current evaluator excludes records without extracted predictions or references, so the table reports a conditional score rather than accuracy on a fixed full set.

For a common evaluation set E with valid reference answers, let R_m contain the examples for which method m yields a valid extracted answer, and let $C_m \subseteq R_m$ contain its correct answers. Then

$$A_m^{\mathrm{ext}} = 100\frac{|C_m|}{|R_m|}, \qquad c_m = \frac{|R_m|}{|E|}, \qquad A_m^{\mathrm{all}} = 100\frac{|C_m|}{|E|} = c_m A_m^{\mathrm{ext}}, \tag{13}$$

where A_m^{ext} and A_m^{all} are percentages and c_m is extraction coverage. The conditional score requires $|R_m| > 0$; if R_m is empty it is undefined, while full-set accuracy is zero. Full-set accuracy treats an unparseable prediction as incorrect. Reference-invalid examples must be excluded identically for every method before defining E. Thus conditional scores can only be converted into full-set accuracy when the common set and method-specific coverage are known. These coverage counts are not reported in Table 4, whose rounded percentages alone do not establish extraction coverage or paired quality differences.

At temperature 0, the exact baselines differ on MATH-500 and AIME25. Attributing this difference to the verifier requires matched example IDs, prompts, target checkpoints, numerical and decoding settings, stop rules, and scoring. The current measurements do not resolve its cause. No numerical code/chat quality results or paired quality measurements for Table 3 are reported here. Consequently, the speed results do not establish task-quality preservation for those settings.

C Additional Budget and Recovery Results

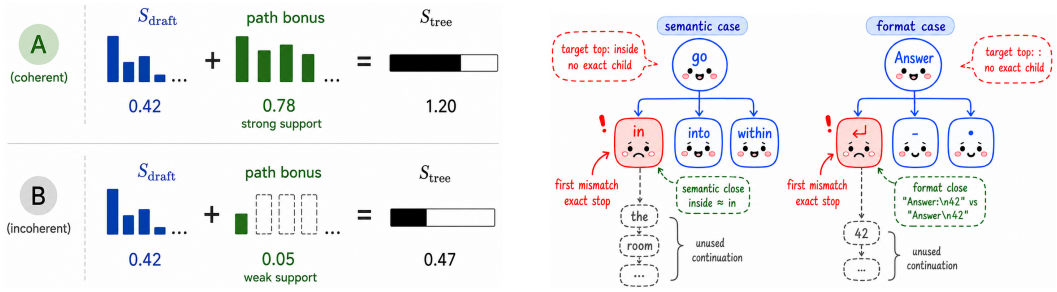
This appendix gives the numeric budget sweep and recovered-token counts used in the main-paper analysis. Table A1 makes the curves in Figure 2 reproducible, and Table A3 reports the complete per-round latency breakdown. In addition, Figure A3 reports where sibling recovery contributes additional accepted tokens.

C.1 Illustrative Construction and Recovery Cases

Figure A1 illustrates the two components; these examples do not establish semantic equivalence or measured task quality.

C.2 Low-Budget Benchmark Overview

Figure A2 visualizes the task-level speed and committed-length comparison from Table 1.



(a) Path guidance **(b) Tree-local near misses**

Figure A1: (a) Request-local path guidance. The rank-gated n-gram prior breaks ties among plausible block-draft candidates by preferring continuations that match local patterns in the current prompt and generated prefix. (b) Illustrative lexical and formatting near misses at an exact tree stop; target support does not imply semantic equivalence.

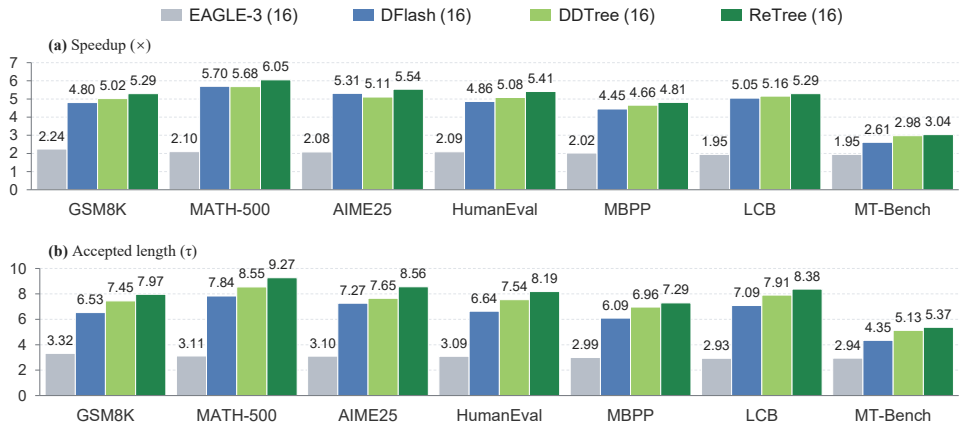


Figure A2: Low-budget benchmark on Qwen3-4B at temperature 0.0. All tree methods use a 16-node budget, and DFlash uses block size 16. ReTree improves both end-to-end speedup and accepted length over the same-budget tree verifier across math, code, and chat tasks.

C.3 Reading the Budget Sweep

Table A1 gives the values plotted in Figure 2: speedup, accepted length, pre-verification latency, and latency per accepted token. These columns separate path quality from construction cost. A method can accept more tokens per verification round but still lose end-to-end throughput if the extra branch construction is too expensive.

C.4 Compatibility with Learned Tree Proposals

Target-Gated Sibling Recovery is a verifier-side decision rule, so it can also be attached to a learned tree proposal. Table A2 applies the same recovery logic to DominoTree.

The result is not used as the main comparison because it combines ReTree's recovery with an additional learned branch scorer. It is instead a compatibility study: recovery remains useful even when branch quality is improved by a correction-head tree.

C.5 Reading the Recovery Counts

Figure A3 reports raw recovered-token counts for Qwen3-4B at temperature 0 on the five-task subset. Counts decline with budget in these runs. This is consistent with increased coverage by exact tree following, but counts also depend on the number of requests, output lengths, and the generated

Table A1: Full Qwen3-4B budget sweep measured in our RTX 5090 setup. Values are arithmetic means over GSM8K, MATH-500, AIME25, HumanEval, MBPP, LCB, and MT-Bench. Higher speedup/τ is better; lower Lat./Cost is better.

Budget	Method	Speedup ↑	τ ↑	Lat. (ms) ↓	Cost (ms/token) ↓
		Temperature = 0.0			
16	DDTree	4.81×	7.31	**5.03**	**0.69**
	DominoTree	5.02×	**8.34**	6.36	0.76
	ReTree	**5.06**×	7.86	5.62	0.72
32	DDTree	5.25×	7.99	**5.04**	**0.63**
	DominoTree	5.28×	**9.39**	8.12	0.86
	ReTree	**5.42**×	8.48	5.81	0.68
64	DDTree	5.52×	8.41	**5.07**	**0.60**
	DominoTree	5.11×	**10.24**	11.73	1.15
	ReTree	**5.64**×	8.86	5.90	0.67
128	DDTree	5.57×	8.79	**5.27**	**0.60**
	DominoTree	4.33×	**10.81**	18.81	1.74
	ReTree	**5.62**×	9.16	6.18	0.67
256	DDTree	4.73×	9.06	**5.57**	**0.61**
	DominoTree	2.99×	**11.34**	32.47	2.86
	ReTree	**4.80**×	9.43	6.73	0.71
		Temperature = 1.0			
16	DDTree	4.33×	6.60	**5.26**	**0.80**
	DominoTree	4.29×	7.14	6.40	0.90
	ReTree	**4.72**×	**7.38**	5.89	0.80
32	DDTree	4.71×	7.22	**5.19**	**0.72**
	DominoTree	4.54×	**8.08**	8.20	1.01
	ReTree	**5.03**×	7.87	5.88	0.75
64	DDTree	4.91×	7.55	**5.27**	**0.70**
	DominoTree	4.38×	**8.82**	11.72	1.33
	ReTree	**5.14**×	8.10	6.01	0.74
128	DDTree	5.00×	7.91	**5.37**	**0.68**
	DominoTree	3.73×	**9.41**	18.83	2.00
	ReTree	**5.13**×	8.42	6.38	0.76
256	DDTree	4.31×	8.24	**5.83**	**0.71**
	DominoTree	2.58×	**9.92**	32.79	3.31
	ReTree	**4.37**×	8.66	7.19	0.83

Table A2: Compatibility study on Qwen3-4B. Each cell reports speedup/τ for DominoTree and DominoTree with Target-Gated Sibling Recovery; higher is better.

Temp.	Method	16	32	64	128	256
0.0	DominoTree	5.02/8.34	5.28/9.39	5.11/10.24	4.33/10.81	2.99/11.34
	DT + Rec.	**5.60/9.23**	**5.74/10.17**	**5.51/10.98**	**4.61/11.47**	**3.19/11.92**
1.0	DominoTree	4.29/7.14	4.54/8.08	4.38/8.82	3.73/9.41	2.58/9.92
	DT + Rec.	**5.06/8.41**	**5.31/9.40**	**5.01/10.04**	**4.15/10.58**	**2.90/11.00**

trajectories. A recovery-rate claim requires normalization by generated tokens or eligible mismatch events.

Tasks are plotted separately because their output structures and lengths differ. These counts describe where recovery was used; they do not quantify semantic equivalence, task accuracy, or the fraction of requests whose outputs changed.

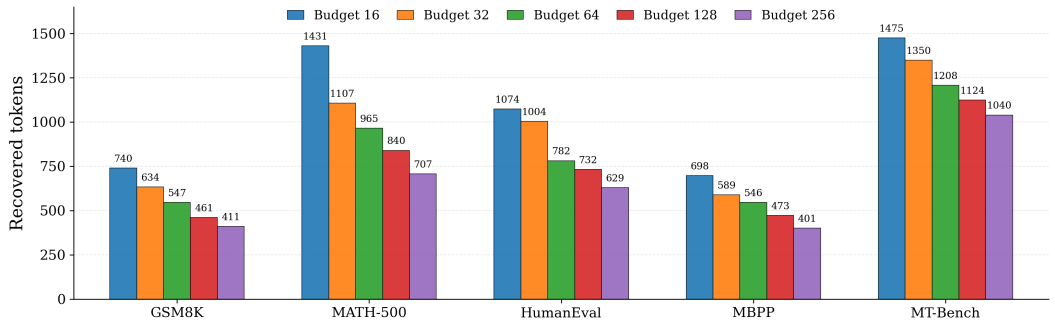


Figure A3: Recovered tokens by tree budget on the logged Qwen3-4B temperature-0.0 five-task subset. Bars show additional tokens accepted by Target-Gated Sibling Recovery.

C.6 Complete Round Latency

The pre-verification latency in Table A1 isolates proposal and construction work. Table A3 additionally reports the target pass and post-verification processing, so that

$$L_{\text{round}} = L_{\text{pre}} + L_{\text{target}} + L_{\text{post}}. \tag{14}$$

Table A3: Complete Qwen3-4B latency decomposition in milliseconds per decoding round, averaged over the seven evaluation tasks on RTX 5090. Recovery is a subset of ReTree's Post stage and appears in brackets; it is not added again to Total.

	ReTree				DominoTree			
TB	Pre	Target	Post [Rec.]	Total	Pre	Target	Post	Total
			Temperature = 0.0					
16	5.62	19.56	1.14 [0.12]	26.32	6.36	18.84	0.82	26.02
32	5.81	20.06	1.14 [0.12]	27.00	8.12	18.77	0.83	27.72
64	5.90	19.80	1.17 [0.13]	26.88	11.73	19.16	0.85	31.75
128	6.18	20.62	1.22 [0.15]	28.01	18.81	20.55	0.87	40.24
256	6.73	25.48	1.28 [0.16]	33.49	32.47	25.28	0.92	58.67
			Temperature = 1.0					
16	5.89	19.84	1.29 [0.13]	27.02	6.40	19.00	0.94	26.34
32	5.88	19.57	1.30 [0.13]	26.75	8.20	19.03	0.96	28.19
64	6.01	19.55	1.29 [0.14]	26.85	11.72	18.92	0.97	31.61
128	6.38	20.72	1.37 [0.16]	28.47	18.83	20.30	1.00	40.13
256	7.19	25.38	1.59 [0.17]	34.16	32.79	25.20	1.25	59.24

For ReTree, recovery is timed inside L_{post}. It reuses logits from L_{target} and therefore adds control-flow latency but no second target-model forward. The 0.12–0.17 ms Recovery column is already included in Post and Total.

C.7 Reporting Scope

The Lat. column in Table A1 covers draft proposal, tree construction, and pre-verification bookkeeping. Cost divides this latency by committed length. Both exclude verification and post-processing and do not measure total decoding cost.

The reported aggregates weight the seven tasks equally. Mean speedup is not reconstructed from mean committed length and mean round latency: an average of task-level ratios generally differs from a ratio of averages.

All rows use the main experiments' task set, generation limit, online memory updates, and stop-token exclusions. Together, Tables A1 and A3 show ReTree's modest pre-verification overhead over DDTree and lower construction cost per committed token than DominoTree.