CALIBRATE BEFORE YOU BRANCH: FEEDBACK-CALIBRATED, COST-AWARE TREES FOR BLOCK-DIFFUSION SPECULATIVE DECODING

Anonymous authors
Paper under double-blind review

ABSTRACT

Speculative decoding (SD) accelerates large language model inference by using a lightweight drafter to propose future tokens that are verified in parallel by the target model. Block-diffusion speculative decoding further reduces drafting latency by predicting multiple future positions in a single forward pass. To exploit alternative candidates within such parallel drafts, recent methods organize them into draft trees for joint verification. However, existing tree construction primarily ranks candidate paths using factorized draft probabilities, which may not reflect the target model's actual acceptance preferences. We propose FACT, a cost-aware draft-tree allocation framework that jointly addresses candidate prioritization and budget selection. Specifically, FACT uses historical target-verification outcomes to construct a frozen correction table that calibrates candidate priorities without additional inference-time model calls. It further combines draft-coverage estimates with profiled full-round costs to adaptively determine whether expanding the tree is worthwhile, while an incremental builder reuses the search frontier across budget checkpoints and materializes only the selected tree. Experiments on six benchmarks with Qwen3-8B and Qwen3-4B show that FACT consistently increases tokens per verification round over DDTree-B128. On Qwen3-8B, FACT achieves 4.17–6.80$\times$ speedup over autoregressive decoding and reduces generation time by 0.27–2.73% relative to DDTree-B128. On Qwen3-4B, it reduces generation time by 1.24–3.05%. Code is available at `https://anonymous.4open.science/r/fact-anonymous-review-B853/`.

1 INTRODUCTION

Large language models support conversational assistants, code generation, and increasingly long reasoning tasks, but generating their responses still requires a sequence of expensive model evaluations. Faster decoding matters both for interactive latency and for workloads that generate many responses, such as high-concurrency batched serving and teacher-output collection for model distillation. Batching processes different requests together, yet the tokens within each autoregressive response remain sequential. Reducing this sequential work motivates speculative decoding (Stern et al., 2018; Leviathan et al., 2023; Chen et al., 2023).

Speculative decoding uses a lightweight draft model to propose several future tokens and a large target model to verify them in parallel. When verification accepts several tokens, one target pass advances the response by more than one position. The benefit depends on both draft quality and drafting cost. Accurate proposals reduce target calls, but producing those proposals must remain inexpensive. Autoregressive drafters still predict successive draft positions sequentially, leaving a drafting bottleneck even when target verification is parallel (Chen et al., 2026).

Recent block-parallel drafting methods address this bottleneck by predicting several future positions together. DFlash uses a lightweight block-diffusion drafter conditioned on target-model context features to produce an entire draft block in one forward pass (Chen et al., 2026). This reduces sequential drafting work, but vanilla DFlash submits only one continuation for verification. An early mismatch ends acceptance, leaving other plausible tokens in the draft unexplored.

DDTree exploits these alternatives by building a branching draft tree from DFlash's per-position distributions (Ringel & Romano, 2026). Its best-first construction selects nodes under a fixed budget,

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.

retaining each selected node's ancestors. A single target pass then verifies the tree. DDTree thus expands coverage without additional draft passes. However, its node-selection objective is defined by products of draft probabilities. These position-wise predictions do not condition on the particular preceding tokens within each candidate branch, unlike the target model. Optimal allocation under this factorized draft surrogate therefore need not maximize actual target acceptance. Under a limited budget, over-prioritizing one branch can exclude a target-accepted continuation.

The missing signal is not another draft probability, but evidence of which candidates the target actually accepts. Although DFlash already uses target features, DDTree's construction objective does not explicitly calibrate node priorities with historical verification outcomes. Those outcomes provide feedback about the draft surrogate's errors. The challenge is to use this feedback when building future trees without evaluating every candidate with the target in advance. Such evaluation would undermine the purpose of speculation. Moreover, a better tree must justify its construction and verification costs, since more accepted tokens alone do not guarantee lower latency, as cost-aware tree methods also emphasize (Zhang et al., 2026; Oh et al., 2026).

We address this challenge with feedback-calibrated trees (FACT). Offline calibration pools verification outcomes into a table indexed by draft depth, token rank, and draft confidence. At inference time, the frozen table corrects candidate priorities through inexpensive lookups, without another drafter or target call. We retain lazy child-and-sibling expansion, choosing budgets with a raw-draft coverage proxy and a full-round cost profile. The builder reuses its frontier between budget checkpoints and materializes only the selected tree. C++ reduces host-side construction overhead.

Our contributions are summarized below.

- We propose FACT, a framework for block-diffusion speculative decoding that combines feedback-calibrated candidate priorities with cost-aware tree budgets to improve target acceptance and decoding efficiency, without additional inference-time drafter or verifier calls.
- To handle incomplete verification feedback, we fit a frozen correction table only from candidates whose parents lie on the observed accepted path. To control expansion cost, we combine raw draft-coverage bounds with full-round cost profiles and an incremental C++ builder that reuses the frontier across budgets and materializes only the selected tree.
- On six datasets with Qwen3-8B and Qwen3-4B, FACT increases tokens per verification round over original DDTree-B128 on every dataset and reduces generation time by 0.27–2.73% and 1.24–3.05%, respectively. Gains persist with matched C++ builders.

Section 4 presents the allocation rule. Section 5 evaluates acceptance, latency, and implementation.

2 Preliminary and Related Work

2.1 Preliminary

Drafting and verification. Let $\mathcal{V}$ be the vocabulary, x a token prefix, and $p(v \mid x)$ and $q(v \mid x)$ the target and draft next-token distributions. The target assigns a continuation $y_{1:m}$ probability $p(y_{1:m} \mid x) = \prod_{j=1}^{m} p(y_j \mid x, y_{<j})$. Speculative decoding proposes multiple tokens before evaluating their prefixes in one target pass. In the stochastic scheme of Leviathan et al. (2023), a proposal $v \sim q(\cdot \mid x)$ is accepted with probability

$$a(v \mid x) = \min\left\{1, \frac{p(v \mid x)}{q(v \mid x)}\right\}. \tag{1}$$

At the first rejection, the remaining draft suffix is discarded and a replacement is sampled from the normalized residual $[p(\cdot \mid x) - q(\cdot \mid x)]_+$, where $[z]_+ = \max(z, 0)$. If every proposal is accepted, an additional token is sampled from the target. We use the greedy tree verifier defined below.

Block drafts and candidate trees. At round t, DDTree (Ringel & Romano, 2026) carries a target-selected token z_t as the verifier root. Let $\bar{x}_t = (x_t, z_t)$ denote the committed context followed by this root. One DFlash (Chen et al., 2026) pass predicts H position-wise distributions $q_d(\cdot \mid \bar{x}_t)$ in parallel. Let $v_{d,r}$ be the token of original draft rank r at position d, with $r = 0$ the highest rank, and write $q_{d,r} = q_d(v_{d,r} \mid \bar{x}_t)$. Retain the top K candidates at each position without renormalization. A node is a rank path $u = (r_1, \ldots, r_{|u|})$, with $0 \le r_j < K$, representing the token sequence

$v(u) = (v_{1,r_1}, \ldots, v_{|u|,r_{|u|}})$. A legal tree T contains at most B such nonempty paths of depth at most H and is ancestor-closed, meaning that $u \in T \Rightarrow u_{1:j} \in T$ for every $1 \le j < |u|$. The root is excluded from B. Suppressing the round index, DDTree orders candidates by the factorized log-score

$$L_q(u) = \sum_{j=1}^{|u|} \log q_{j,r_j}, \qquad \exp(L_q(u)) = \prod_{j=1}^{|u|} q_{j,r_j}. \tag{2}$$

Unlike $p(y_{1:m} \mid x)$, this product ignores preceding tokens within each candidate.

Accepted tokens per round. Tree attention lets each node attend to the cached context, root, itself, and its ancestors, but not sibling branches. Let $p_t(v \mid u)$ be the next-token distribution returned by the target tree verifier at node u, with $u = \varnothing$ denoting the root. Greedy verification chooses $\widehat{v}_t(u) = \arg\max_{v \in \mathcal{V}} p_t(v \mid u)$ with fixed tie breaking. The number of accepted draft tokens is

$$A_t(T_t) = \max\big(\{0\} \cup \big\{|u| : u \in T_t,\ v_{j,r_j} = \widehat{v}_t(u_{<j})\ \forall\, 1 \le j \le |u|\big\}\big), \tag{3}$$

where $u_{<j} = (r_1, \ldots, r_{j-1})$ is the parent path, empty for $j = 1$. The round advances by $1 + A_t(T_t)$ tokens, counting the root once. Its first unmatched target token becomes the next root. The final round is truncated at the output limit.

Acceptance–cost trade-off. For N emitted tokens, R verification rounds, and total request wall time W, define tokens per round τ and effective round cost $\bar{c}$ by

$$\tau = \frac{N}{R}, \qquad \bar{c} = \frac{W}{R}, \qquad \frac{N}{W} = \frac{\tau}{\bar{c}}. \tag{4}$$

Here W includes prefill, drafting, tree construction, verification, and cache updates. Increasing tree coverage helps throughput only if its acceptance gain offsets its added cost. The allocation problem is therefore to select useful paths and their budget, rather than maximize draft scores alone.

2.2 Related Work

Speculative decoding. Blockwise parallel decoding predicts several future tokens and accepts the longest prefix matching a greedy scorer (Stern et al., 2018). Speculative sampling extends parallel verification to stochastic generation through rejection sampling that preserves the target distribution (Leviathan et al., 2023; Chen et al., 2023). BiLD instead coordinates a small and a large decoder through fallback and rollback policies (Kim et al., 2023). Other approaches change the source of proposals. REST retrieves candidate continuations (He et al., 2024), while Lookahead Decoding generates and verifies candidates without an auxiliary draft model (Fu et al., 2024).

Learned drafters exploit different prediction structures. Medusa uses parallel prediction heads (Cai et al., 2024), and Hydra introduces sequential dependencies among those heads (Ankner et al., 2024). EAGLE drafts in feature space (Li et al., 2024a), whereas EAGLE-3 combines multi-layer target features with direct token prediction and a training-time test (Li et al., 2025). FR-Spec reduces large-vocabulary drafting overhead using a frequency-ranked vocabulary subset (Zhao et al., 2025). These designs balance proposal accuracy against drafting cost to reduce the overall cost of generation.

Block-diffusion speculative decoding. Masked diffusion language models generate tokens through parallel denoising (Sahoo et al., 2024), and block diffusion combines within-block denoising with autoregressive generation across blocks (Arriola et al., 2025). For speculative drafting, DiffuSpec produces multi-token proposals in one forward pass, uses causal-consistency path search, and adapts draft length with acceptance feedback (Li et al., 2026). DFlash trains a lightweight block-diffusion drafter conditioned on target-model features to predict future positions in parallel (Chen et al., 2026). Vanilla DFlash verifies only a single trajectory, leaving alternative tokens unexplored after an early rejection. DDTree extracts branching candidates from the same parallel draft (Ringel & Romano, 2026). However, one-pass position-wise probabilities do not provide target-conditioned probabilities for every candidate prefix, so greater draft coverage need not yield longer acceptance.

Tree attention and tree-based speculative decoding. Tree attention evaluates branching continuations while restricting each node to the shared context and its ancestral path. SpecInfer organizes speculative proposals into token trees for parallel verification (Miao et al., 2024). SpecTr formulates multi-candidate verification through optimal transport (Sun et al., 2023). Several methods adapt which branches are retained. EAGLE-2 constructs context-dependent trees using draft confidence (Li et al., 2024b), Sequoia optimizes tree structure through dynamic programming (Chen

et al., 2024), and OPT-Tree searches for high estimated acceptance length under a node budget (Wang et al., 2025). SpecExec builds a cache of likely continuations for massively parallel speculative execution, including offloaded inference (Svirschevski et al., 2024). Closest to our setting, DDTree constructs an ancestor-closed tree from one-pass position-wise distributions without additional drafter calls (Ringel & Romano, 2026). Its best-first rule maximizes factorized draft coverage under a fixed node budget. This optimality concerns the draft surrogate rather than actual target acceptance, and does not determine when further expansion is worth its cost.

Cost-aware draft-tree construction. Within tree-based block-diffusion decoding, CaDDTree (Zhang et al., 2026) models drafting and verification latency and chooses tree size through a per-round throughput objective. Under convex verification cost, its analysis yields a greedy stopping rule. BASTION by Oh et al. (2026) combines a path-confidence acceptance surrogate with a hardware-calibrated latency model and an online budget controller, expanding a tree while its estimated marginal benefit justifies the added cost. These methods explicitly address cost-aware stopping. Their decisions, however, depend on how well draft-side acceptance surrogates and measured cost profiles predict the actual benefit and latency of expansion.

Our work couples cost-aware construction with a frozen correction derived from past target-verification outcomes. Classical confidence calibration aligns predicted probabilities with empirical correctness (Guo et al., 2017). Here, the correction adjusts unnormalized node-selection priorities rather than producing calibrated target probabilities. This lets historical verification feedback guide which candidates receive the available tree budget.

3 Recovering a Missed Continuation in a Case Study

Can feedback improve acceptance at a fixed budget? An archived Qwen3-8B/GSM8K round isolates correction by keeping the prefix, draft probabilities, and 128-node budget fixed. The calibrated tree uses FACT's frozen table at strength 0.25. Target labels evaluate coverage only after construction.

A low-ranked but useful continuation. After "So far,", the observed continuation is `we have accounted` (three space-prefixed tokens). DDTree-B128 retains only the first two. The omitted token has depth-three draft probability 0.02437 (rank nine) and raw path log-score -6.0413.

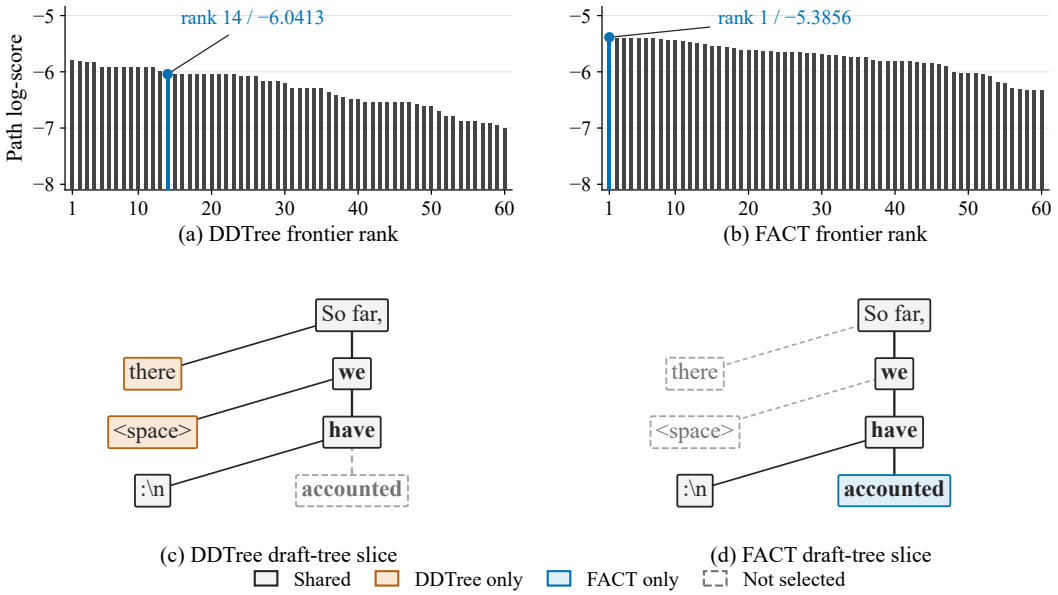


Figure 1: GSM8K replay ($B = 128$, $\alpha = 0.25$) with top-60 frontier priorities (top) and draft-tree slices (bottom). Blue bars mark `we have accounted`, and dashed nodes are absent.

Recovering the path through feedback. In this case, feedback raises the path log-score to -5.3856, and the calibrated builder selects it at position 116 with both ancestors retained. Immediately before selection, it ranks 14th in DDTree's frontier and first in FACT's (Figure 1). These frontiers contain different candidates. Holding the raw frontier's top 20 candidates fixed instead pro-

motes the continuation from rank 14 to rank 9 (Table 9). Other non-matching paths still rank higher, but the full-tree reconstruction recovers the useful continuation within the same B128 budget. The aligned slices show that `accounted` is absent from DDTree but retained by FACT.

The raw verifier accepted two drafted tokens and supplied `accounted` as its next token. The calibrated tree covers all three recorded labels, giving root-inclusive round counts $\tau_t^{\text{raw}} = 3$ and $\tau_t^{\text{cal}} \geq 4$ in same-prefix replay. The latter is a lower bound because the suffix beyond `accounted` is unobserved. This case motivates using feedback to recover branches undervalued by draft confidence. Section 4 develops the method, and Appendix C provides additional candidate-level analysis.

4 Feedback-calibrated, cost-aware draft trees

Overview. Figure 2 organizes FACT into offline preparation and an online decoding loop. Before decoding, historical target-verification outcomes are used to fit a correction table that captures discrepancies between draft probabilities and observed acceptance (Section 4.1). Separately, profiling measures the full-round cost of different node budgets at different context lengths (Section 4.3). Both the correction table and cost profile remain fixed during decoding. Together, they guide which candidate paths enter the tree and how far tree expansion proceeds.

In each decoding round, a single block-diffusion draft pass supplies the candidate tokens and probabilities. Table lookups adjust their priorities (Section 4.2), and an incremental builder expands the tree in corrected priority order. At each budget checkpoint, the controller compares current draft coverage and a bound on further coverage against the profiled costs to decide whether to stop or extend the tree (Section 4.4). Continuing expansion reuses the selected nodes and search frontier. Once the budget is chosen, the builder materializes the selected tree for parallel target verification and token commitment. The drafter, ancestor-closure constraint, and verification rule remain unchanged, with no additional model calls for correction or budget selection. Section 4.5 describes the C++ implementation of tree construction.

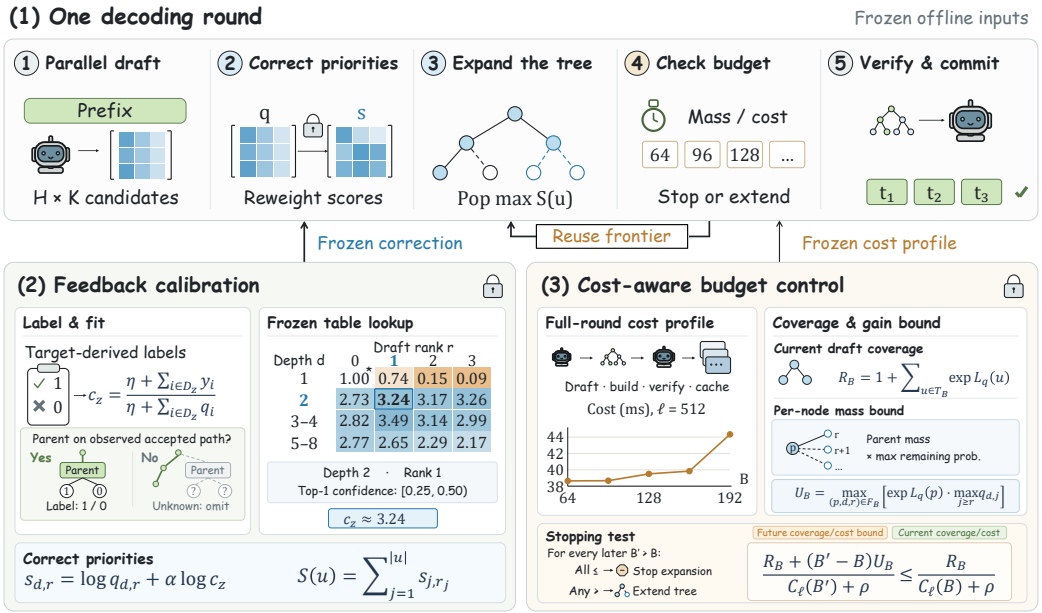


Figure 2: One round of FACT (top), with offline feedback calibration (bottom left) and cost-aware budget control (bottom right). Filled/open nodes denote selected/exposed candidates.

4.1 Constructing the feedback correction table

We collect feedback from candidates exposed by the lazy tree builder. After target verification, a candidate is labeled only when its parent lies on the observed accepted path. The target continuation supplies a binary label y_i, which is one if the candidate token matches that continuation and zero otherwise. Candidates below an unaccepted or unvisited parent have no target label and are omitted

rather than treated as failures. The verifier's next-token output labels the child of the final accepted parent.

Each labeled occurrence belongs to a cell $z = b(d, r, h)$ defined by depth d, original draft rank r, and a confidence bin h of the position's full-vocabulary top-1 draft probability. Grouping candidates by these features pools verification evidence across prefixes and datasets.

For the labeled occurrences $\mathcal{D}_z$ in cell z, we fit the smoothed mass ratio

$$c_z = \frac{\eta + \sum_{i \in \mathcal{D}_z} y_i}{\eta + \sum_{i \in \mathcal{D}_z} q_i}, \qquad \eta > 0, \tag{5}$$

where q_i is the candidate's full-vocabulary draft probability. Adding the same positive constant to numerator and denominator shrinks low-support cells toward the neutral value one. Empty cells are neutral. The fitted values form a lookup table that remains fixed during decoding. Appendix A.1 specifies the binning scheme and calibration settings.

4.2 Correcting candidate priorities

For a new draft block, we retain K candidates at each of H positions, using the notation of Section 2.1. Their probabilities $q_{d,r}$ remain normalized over the full vocabulary. Each candidate uses its depth, original rank, and top-1 confidence bin to look up $c_{b(d,r,h)}$ in the frozen table. Its local priority and the score of a rank path are

$$s_{d,r} = \log q_{d,r} + \alpha \log c_{b(d,r,h)}, \qquad S(u) = \sum_{j=1}^{|u|} s_{j,r_j}, \qquad \alpha \geq 0. \tag{6}$$

The strength α controls the correction and differs from the smoothing constant η. A cell with $c_z > 1$ raises candidate priority, while $c_z < 1$ lowers it. For example, $c_z = 2$ changes the multiplicative local score by 2^α. Choosing $0 < \alpha < 1$ tempers the observed mass-ratio correction. The strength stays fixed. Both $\exp(s_{d,r})$ and $\exp(S(u))$ are unnormalized priority scores.

The builder preserves DDTree's original lazy child-and-sibling exposure. Popping an exposed node inserts its next-ranked sibling and, when another draft position exists, its rank-zero child. A heap orders exposed paths by $S(u)$, with deterministic rank-path tie breaking, until exactly B candidate nodes have been selected. Ancestor closure follows from exposing a child only after its parent has entered the tree. Corrected scores can vary non-monotonically with original rank, while sibling exposure follows the original rank order. Selection is restricted to the exposed frontier.

Algorithm 1 Calibrated prefix allocation at a supplied budget

Require: Draft probabilities q, candidate tokens, budget B, frozen table c
1: Compute local corrected scores s using Eq. (6)
2: $T \leftarrow \varnothing$, $F \leftarrow \{(\varnothing, 1, 0)\}$
3: **while** $|T| < B$ **do**
4: Remove the exposed entry (p, d, r) with maximum corrected path score
5: Append $u = p \,\|\, r$ to T
6: Expose $(p, d, r + 1)$ if $r + 1 < K$
7: Expose $(u, d + 1, 0)$ if $d < H$
8: **end while**
9: Materialize token IDs, positions, parent indices, and visibility for T
10: **return** T to the existing target verifier

4.3 Coverage and cost estimation

Full-round cost. Before decoding, we time full decoding rounds at a finite set $\mathcal{G}$ of node-budget checkpoints, grouping the measurements by context length. For each context bucket ℓ and budget B, the profile stores a full-round wall cost $C_\ell(B) > 0$ covering drafting, tree construction, target verification, and cache updates. Within each bucket and budget, we average round times per calibration prompt, then average these prompt-level costs across prompts and profiling passes. During decoding, ℓ is the smallest bucket boundary at least as large as the current context length, and costs are retrieved from this fixed profile. We add a fixed allowance $\rho \geq 0$ for controller overhead, in the same units as $C_\ell(B)$, giving the estimate $C_\ell(B) + \rho$. Appendix A.2 details profiling and lookup.

Current draft coverage. After the calibrated heap has selected B nodes, let T_B be the current tree, excluding the verifier root. Starting from the root contribution one, the builder adds the raw path

mass $\exp(L_q(u))$ whenever it selects a node u, where $L_q(u)$ is defined in Eq. (2). This maintains the draft-coverage proxy

$$R_B = 1 + \sum_{u \in T_B} \exp(L_q(u)) \,. \tag{7}$$

The leading one represents the verifier root. Under the factorized draft model, reaching node u has probability $\exp(L_q(u))$. Summing the reach indicators over an ancestor-closed tree gives its expected covered prefix length. Thus R_B is a coverage proxy under the factorized draft model. Corrected scores select T_B, but Eq. (7) uses raw mass because the correction is not a probability.

Remaining node-mass bound. For each draft position, we compute the suffix maxima $M_{d,r} = \max_{j \geq r} \log q_{d,j}$ by scanning the candidate list from the last rank to the first. At budget B, a frontier entry $(p, d, r) \in F_B$ represents parent path p and its next exposed candidate at position d and rank r. Combining the parent's raw path mass with the suffix maximum gives the bound

$$U_B = \exp\left(\max_{(p,d,r) \in F_B} [L_q(p) + M_{d,r}] \right), \qquad U_B = 0 \text{ if } F_B = \varnothing. \tag{8}$$

The suffix maximum covers both the exposed candidate and its later siblings. Descendants multiply these path masses by probabilities no greater than one, so U_B bounds the raw mass of any node that can subsequently be selected. The suffix maxima are computed once per draft block, while R_B and the frontier used to evaluate U_B change as the tree grows.

4.4 Budget allocation and stopping

We examine the checkpoints in $\mathcal{G}$ in increasing order, after removing any checkpoint whose profiled cost is no lower than that of a larger checkpoint. At checkpoint B, the current coverage-to-cost ratio is $R_B/(C_\ell(B) + \rho)$. Reaching a later checkpoint $B' > B$ adds at most $B' - B$ nodes, each with raw mass bounded by U_B. Its coverage is therefore at most $R_B + (B' - B)U_B$. The builder stops at B when

$$\frac{R_B + (B' - B)U_B}{C_\ell(B') + \rho} \leq \frac{R_B}{C_\ell(B) + \rho} \quad \text{for every remaining } B' \in \mathcal{G}. \tag{9}$$

If the inequality holds for every later checkpoint, none can improve the current coverage-to-cost ratio under the fixed profile. Otherwise, the builder continues from the existing heap to the next checkpoint and recomputes the test. It returns the first checkpoint satisfying Eq. (9), or the largest retained checkpoint if no earlier one satisfies the condition. Appendix A.2 gives the full algorithm.

Computational complexity. Each local priority correction takes constant time. With maximum budget $B_{\max} = \max \mathcal{G}$, heap construction costs $O(B_{\max} \log B_{\max})$ time and $O(B_{\max})$ auxiliary state. The finite checkpoint tests add only frontier bookkeeping. Materializing dense visibility remains $O(B^2)$ for the selected budget.

4.5 C++ implementation of tree construction

We implement CPU heap expansion, budget-checkpoint tests, and tree materialization in C++ to reduce host-side construction overhead. The compiled builder takes the same raw and corrected score arrays as the Python reference and preserves its candidate exposure, rank-path tie breaking, and stopping rule. It retains selected nodes, path scores, and the frontier across checkpoints, then materializes token IDs, positions, parent indices, and visibility once for the selected tree.

Float32 priority correction, GPU top-K extraction, draft inference, target verification, and cache handling remain outside this replacement. We also provide a C++ implementation of the uncalibrated fixed-B128 builder for comparisons using the same construction backend.

5 Experiments

5.1 Setup

Models and Workloads. We conduct experiments with Qwen3-8B and Qwen3-4B (Qwen Team, 2025) as target models, each paired with its corresponding DFlash-b16 drafter (Chen et al., 2026). Our evaluation set contains 182 prompts drawn from six benchmarks covering code generation, mathematical reasoning, and general instruction following. Specifically, we use 32 prompts each from HumanEval (Chen et al., 2021), MBPP (Austin et al., 2021), GSM8K (Cobbe et al., 2021), MATH-500 (Hendrycks et al., 2021; Lightman et al., 2024), and Alpaca (Taori et al., 2023), and 22

prompts from AIME25 (Dekoninck et al., 2026). All methods use greedy decoding and generate 512 output tokens per prompt. Runtime is measured over two runs for each configuration, with each job executed on a single NVIDIA GeForce RTX 4090 GPU.

Methods. We compare our proposed FACT with autoregressive decoding (AR), DFlash, and the original 128-node DDTree (RAW128) under the same target model and input requests, with matching drafter settings for the draft-based methods. In addition, FACT uses a C++ tree builder, adaptively selects the tree budget from 64 to 192 nodes, and sets the correction strength to 0.25 for Qwen3-8B and 0.15 for Qwen3-4B. Furthermore, we evaluate builder and correction-strength variants to study their individual effects. EAGLE-3 (Li et al., 2025) provides a vLLM-based reference (Kwon et al., 2023), with speedup relative to its vLLM AR baseline.

Metric. We use end-to-end generation time as the primary metric of decoding efficiency. For each dataset, we aggregate the generated tokens N_m, verification rounds R_m, and generation time W_m to compute the average number of tokens generated per verification round τ_m, the speedup over autoregressive decoding vsAR_m, and the relative generation-time reduction $\mathrm{save}(m, b)$ with respect to the comparison method b as follows.

$$\tau_m = \frac{N_m}{R_m}, \qquad \mathrm{vsAR}_m = \frac{W_{\mathrm{AR}}}{W_m}, \qquad \mathrm{save}(m, b) = 100\left(1 - \frac{W_m}{W_b}\right). \tag{10}$$

where W_{AR} is the corresponding AR generation time. Furthermore, Appendix A provides the evaluation settings, and Appendix B reports additional results and ablations.

5.2 MAIN RESULTS

Table 1: Tokens per verification round τ and speedup over AR ($\times$). FACT uses $\alpha = 0.25$ for Qwen3-8B and 0.15 for Qwen3-4B. Raw128 denotes DDTree-B128.

Dataset	EAGLE-3		DFlash		Raw128		FACT	
	τ	vsAR	τ	vsAR	τ	vsAR	τ	vsAR
Qwen3-8B ($\alpha = 0.25$)								
HumanEval	2.9226	2.3243	7.4237	5.3188	9.7582	6.6100	10.0208	6.7956
GSM8K	2.7270	2.1591	6.8040	4.8703	8.9140	6.0146	9.0972	6.0904
MBPP	2.7854	2.2143	6.1226	4.3875	8.5601	5.7854	8.8086	5.9243
AIME25	2.3570	1.8733	6.0788	4.3441	8.2460	5.5404	8.4819	5.5992
MATH-500	2.5496	2.0210	7.1891	5.1526	9.4596	6.3969	9.6889	6.5080
Alpaca	2.7239	2.1808	4.1073	2.9482	6.1134	4.1590	6.1525	4.1701
Qwen3-4B ($\alpha = 0.15$)								
HumanEval	2.2376	1.5967	4.8075	3.5403	6.7815	4.7368	7.0439	4.8597
GSM8K	2.3615	1.6920	4.8202	3.5411	6.7285	4.6847	6.9898	4.7797
MBPP	2.2010	1.5841	4.4293	3.2721	6.4708	4.5245	6.7758	4.6667
AIME25	2.1961	1.5612	6.2266	4.5719	8.3252	5.7953	8.6713	5.9735
MATH-500	2.2611	1.6261	6.3332	4.6445	8.4323	5.8485	8.6232	5.9220
Alpaca	2.1686	1.5676	2.9091	2.1390	4.7026	3.2682	4.9025	3.3643

To evaluate the decoding efficiency of FACT, we compare it with AR, DFlash, and RAW128 on six datasets using Qwen3-8B and Qwen3-4B. As shown in Table 1, we observe that FACT achieves higher τ than RAW128 across all six datasets and both model sizes, indicating that fewer verification rounds are required to generate the same number of output tokens. On Qwen3-8B, FACT achieves a 4.17–6.80$\times$ speedup over AR and reduces generation time by 0.27–2.73% relative to RAW128 and 20.03–29.30% relative to DFlash. On Qwen3-4B, FACT achieves a 3.36–5.97$\times$ speedup over AR and reduces generation time by 1.24–3.05% relative to RAW128. These gains are consistently observed across all datasets and both timing repeats with a fixed correction strength for each model size. For additional reference, EAGLE-3 achieves a 1.87–2.32$\times$ speedup on Qwen3-8B and 1.56–1.69$\times$ on Qwen3-4B relative to its corresponding vLLM AR baseline (Appendix Table 7).

5.3 SEPARATING ALLOCATION AND IMPLEMENTATION

To separate allocation gains from builder-backend effects, we compare FACT and RAW128 with matched C++ and Python builders on both model sizes. As shown in Appendix Tables 3 and 8, we find that FACT reduces generation time relative to RAW128-C++ by 0.36–2.53% on Qwen3-8B and 0.97–4.01% on Qwen3-4B when both methods use C++ builders.

With Python builders, the Qwen3-8B policy at strength 0.25 is faster than RAW128 on four datasets, nearly tied on GSM8K, and 0.52% slower on Alpaca. The GSM8K difference changes sign between repeats. On Qwen3-4B, the strength-0.15 policy reduces generation time by 0.26–2.03%.

5.4 CORRECTION-STRENGTH SENSITIVITY

To assess sensitivity to correction strength, we compare settings of 0.15 and 0.25 across the six datasets for both model sizes using the C++ builder. Appendix Table 4 shows that increasing the strength from 0.15 to 0.25 on Qwen3-8B raises τ on five datasets but leaves it unchanged on AIME25. Yet 0.15 is faster on AIME25, reflecting tree construction and verification costs.

On Qwen3-4B, strength 0.15 is faster on four datasets, while 0.25 is faster on HumanEval and MATH-500 (Appendix Table 8). Based on this two-strength comparison, we report 0.15 for Qwen3-4B and 0.25 for Qwen3-8B, with one shared setting per model across all datasets.

5.5 WHEN ACCEPTANCE GAINS TRANSLATE INTO SPEED

To examine how acceptance gains translate into speed, we compare FACT with RAW128-C++ on Qwen3-8B/Alpaca using matched C++ builders. For equal output work, let $\bar{c}_m = W_m/R_m$ be the effective time per verification round. Then the identity

$$\frac{W_b}{W_m} = \frac{\tau_m}{\tau_b}\frac{\bar{c}_b}{\bar{c}_m} \tag{11}$$

separates round reduction from effective round cost. As shown in Appendix Table 5, we observe that τ rises by about 0.64%, but time falls by only 0.36% relative to RAW128-C++, as higher effective round cost absorbs part of the acceptance gain. Faster decoding requires the reduction in verification rounds to outweigh the increase in cost per round.

6 DISCUSSION AND LIMITATIONS

Feedback granularity. The pooled table captures recurring discrepancies between draft candidate mass and observed target continuations. Its labels cover observed parents, while coarse depth/rank/confidence cells pool heterogeneous prefixes. This aggregation limits prefix-specific calibration and contributes to variation in the benefit of correction across workloads. Prefix-conditioned feedback is a promising direction for more precise candidate allocation.

Cost estimation. The controller uses raw factorized draft mass and a frozen full-round cost profile. Its budget decisions depend on how closely these quantities track target acceptance and realized latency. Profile drift and prefix-dependent acceptance introduce estimation error. Adaptive profiling and richer acceptance models could improve these decisions at additional computational cost.

Qwen3-8B replay coverage. Across 984 recorded rounds per dataset, correction recovers the first omitted target token while preserving earlier accepted tokens in 69 GSM8K rounds and 100 MBPP rounds. Observed-prefix coverage decreases in 56 and 47 rounds, respectively. These counts describe observed-prefix coverage. Complete-system latency and mean acceptance are reported in Section 5.

Experimental scope. The evaluation covers fixed development subsets with two Qwen3 target/drafter pairs on RTX 4090 GPUs. It measures fixed-length generation efficiency. Python and C++ agree on outputs, acceptance counts, and budget trajectories for measured requests. Generated token sequences sometimes differ from AR in the Qwen3-8B experiments and EAGLE-3 comparisons. Evaluation on independent datasets, additional model families, and natural-length outputs, together with task-quality measurements, would broaden the assessment.

7 CONCLUSION

We introduced feedback-calibrated trees for block-diffusion speculative decoding. Historical verification outcomes guide candidate priorities, while a cost-profiled stopping rule controls tree size within an incremental builder. On Qwen3-8B and Qwen3-4B, the method increases tokens per round and reduces generation time relative to DDTree-B128 across all six datasets. Future work will explore prefix-conditioned calibration and more adaptive cost estimation.

AI USE STATEMENT

Generative AI tools, including OpenAI Codex, assisted with method discussion, mathematical formulation, experiment implementation and troubleshooting, interpretation of recorded results, literature retrieval, manuscript drafting, and LaTeX preparation and revision. Model-assisted checks also supported reviews of code, result summaries, and manuscript consistency. Reported numerical results derive from experiment records and measured counters, not AI-generated estimates. The authors retain responsibility for validating the implementation, references, scientific claims, and final submission.

ETHICS STATEMENT

This work studies inference efficiency using existing model checkpoints and benchmark datasets, without a new human-subject study. The fixed-length runtime evaluation does not establish factual accuracy, task quality, fairness, or safety. Faster generation may also accelerate harmful uses of language models, so deployment must retain appropriate safety controls. The anonymous method artifact contains aggregate correction and cost tables, not historical conversation traces or model weights, and retains third-party code license notices. Use and redistribution of underlying models and data must respect their licenses, access conditions, and privacy restrictions.

REPRODUCIBILITY STATEMENT

Sections 4 and 5 specify the allocation algorithm, compared methods, workloads, and pooled metrics. Appendix A documents correction-table fitting, cost-profile construction, model settings, and measurement boundaries. Appendix B provides complete results and timing repeats, and Appendix C provides candidate-level replay analysis. The anonymous artifact linked in the abstract contains Python and C++ builders, frozen model-specific tables, configurations, and CPU correctness tests. Its README specifies dependencies, commands, and reproduction scope. The artifact supports inspection and testing of the allocation method, but does not include the full end-to-end benchmark runtime, evaluation prompt splits, or historical calibration traces. It is not a complete reproduction package for the reported speedups.

REFERENCES

Zachary Ankner, Rishab Parthasarathy, Aniruddha Nrusimha, Christopher Rinard, Jonathan Ragan-Kelley, and William Brandon. Hydra: Sequentially-dependent draft heads for Medusa decoding. In *First Conference on Language Modeling*, 2024. URL https://openreview.net/forum?id=FbhjirzvJG.

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 *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=tyEyYT267x.

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. doi: 10.48550/arXiv.2108.07732. URL https://arxiv.org/abs/2108.07732.

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*, volume 235 of *Proceedings of Machine Learning Research*, pp. 5209–5235. PMLR, 2024. URL https://proceedings.mlr.press/v235/cai24b.html.

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. doi: 10.48550/arXiv.2302.01318. URL https://arxiv.org/abs/2302.01318.

Jian Chen, Yesheng Liang, and Zhijian Liu. DFlash: Block diffusion for flash speculative decoding. *arXiv preprint arXiv:2602.06036*, 2026. doi: 10.48550/arXiv.2602.06036. URL https://arxiv.org/abs/2602.06036.

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. doi: 10.48550/arXiv.2107.03374. URL https://arxiv.org/abs/2107.03374.

Zhuoming Chen, Avner May, Ruslan Svirschevski, Yuhsun Huang, Max Ryabinin, Zhihao Jia, and Beidi Chen. Sequoia: Scalable and robust speculative decoding. In *Advances in Neural Information Processing Systems*, volume 37, 2024. URL https://proceedings.neurips.cc/paper_files/paper/2024/hash/ea1f5f0878d43ff4fb8bf64ef4a2326c-Abstract-Conference.html.

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. doi: 10.48550/arXiv.2110.14168. URL https://arxiv.org/abs/2110.14168.

Jasper Dekoninck, Nikola Jovanović, Tim Gehrunger, Kári Rögnvaldsson, Ivo Petrov, Chenhao Sun, and Martin Vechev. Beyond benchmarks: MathArena as an evaluation platform for mathematics with LLMs. *arXiv preprint arXiv:2605.00674*, 2026. doi: 10.48550/arXiv.2605.00674. URL https://arxiv.org/abs/2605.00674.

Yichao Fu, Peter Bailis, Ion Stoica, and Hao Zhang. Break the sequential dependency of LLM inference using lookahead decoding. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 14060–14079. PMLR, 2024. URL https://proceedings.mlr.press/v235/fu24a.html.

Chuan Guo, Geoff Pleiss, Yu Sun, and Kilian Q. Weinberger. On calibration of modern neural networks. In *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pp. 1321–1330. PMLR, 2017. URL https://proceedings.mlr.press/v70/guo17a.html.

Zhenyu He, Zexuan Zhong, Tianle Cai, Jason 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. doi: 10.18653/v1/2024.naacl-long.88. URL https://aclanthology.org/2024.naacl-long.88/.

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. URL https://datasets-benchmarks-proceedings.neurips.cc/paper/2021/hash/be83ab3ecd0db773eb2dc1b0a17836a1-Abstract-round2.html.

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.

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. URL https://arxiv.org/abs/2309.06180.

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. URL https://proceedings.mlr.press/v202/leviathan23a.html.

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. Association for Computational Linguistics, 2026. doi: 10.18653/v1/2026.findings-acl.1048. URL https://aclanthology.org/2026.findings-acl.1048/.

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*, volume 235 of *Proceedings of Machine Learning Research*, pp. 28935–28948. PMLR, 2024a. URL https://proceedings.mlr.press/v235/li24bt.html.

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. doi: 10.18653/v1/2024.emnlp-main.422. URL https://aclanthology.org/2024.emnlp-main.422/.

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*, volume 38, 2025. doi: 10.52202/085713-4562. URL https://doi.org/10.52202/085713-4562.

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 *The Twelfth International Conference on Learning Representations*, 2024. URL https://proceedings.iclr.cc/paper_files/paper/2024/hash/aca97732e30bcf1303bc22ac3924fd16-Abstract-Conference.html.

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. Association for Computing Machinery, 2024. doi: 10.1145/3620666.3651335. URL https://doi.org/10.1145/3620666.3651335.

Soowon Oh, Nam Cao, Yujin Kim, Hojung Jung, Huzama Ahmad, Sangmin Bae, and Se-Young Yun. Bastion: Budget-aware speculative decoding with tree-structured block diffusion drafting. *arXiv preprint arXiv:2605.29727*, 2026. doi: 10.48550/arXiv.2605.29727. URL https://arxiv.org/abs/2605.29727.

Qwen Team. Qwen3: Think deeper, act faster. Qwen blog, 2025. URL https://qwenlm.github.io/blog/qwen3/.

Liran Ringel and Yaniv Romano. Accelerating speculative decoding with block diffusion draft trees. *arXiv preprint arXiv:2604.12989*, 2026. doi: 10.48550/arXiv.2604.12989. URL https://arxiv.org/abs/2604.12989.

Subham Sekhar Sahoo, Marianne Arriola, Yair Schiff, Aaron Gokaslan, Edgar Marroquin, Justin T. Chiu, Alexander Rush, and Volodymyr Kuleshov. Simple and effective masked diffusion language models. In *Advances in Neural Information Processing Systems*, volume 37, 2024. URL https://proceedings.neurips.cc/paper_files/paper/2024/hash/eb0b13cc515724ab8015bc978fdde0ad-Abstract-Conference.html.

Mitchell Stern, Noam Shazeer, and Jakob Uszkoreit. Blockwise parallel decoding for deep autoregressive models. In *Advances in Neural Information Processing Systems*, volume 31, 2018. URL https://proceedings.neurips.cc/paper_files/paper/2018/hash/c4127b9194fe8562c64dc0f5bf2c93bc-Abstract.html.

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*, volume 36, 2023. URL https://proceedings.neurips.cc/paper_files/paper/2023/file/6034a661584af6c28fd97a6f23e56c0a-Paper-Conference.pdf.

Ruslan Svirschevski, Avner May, Zhuoming Chen, Beidi Chen, Zhihao Jia, and Max Ryabinin. SpecExec: Massively parallel speculative decoding for interactive LLM inference on consumer devices. In *Advances in Neural Information Processing Systems*, volume 37, 2024. URL https://proceedings.neurips.cc/paper_files/paper/2024/hash/1d91d5689e251d27993a3c2182dddcf7-Abstract-Conference.html.

Rohan Taori, Ishaan Gulrajani, Tianyi Zhang, Yann Dubois, Xuechen Li, Carlos Guestrin, Percy Liang, and Tatsunori B. Hashimoto. Alpaca: A strong, replicable instruction-following model. Stanford Center for Research on Foundation Models, 2023. URL https://crfm.stanford.edu/2023/03/13/alpaca.html.

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. doi: 10.1162/tacl_a_00735. URL https://aclanthology.org/2025.tacl-1.8/.

Shuai Zhang, Huachuan Qiu, Hongliang He, and Yong Dai. Cost-aware diffusion draft trees for speculative decoding. *arXiv preprint arXiv:2606.01813*, 2026. doi: 10.48550/arXiv.2606.01813. URL https://arxiv.org/abs/2606.01813.

Weilin Zhao, Tengyu Pan, Xu Han, Yudi Zhang, Ao Sun, Yuxiang Huang, Kaihuo Zhang, Weilun Zhao, Yuxuan Li, Jie Zhou, Hao Zhou, Jianyong Wang, Zhiyuan Liu, and Maosong Sun. FR-Spec: Accelerating large-vocabulary language models via frequency-ranked speculative sampling. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 3909–3921. Association for Computational Linguistics, 2025. doi: 10.18653/v1/2025.acl-long.198. URL https://aclanthology.org/2025.acl-long.198/.

APPENDIX

A IMPLEMENTATION DETAILS

A.1 CALIBRATION REPRESENTATION

The pooled correction contains $5 \times 16 \times 4 = 320$ cells. The collector retains candidate occurrences whose parent has an observed target continuation, including the child of the final accepted parent. It retains decision groups with positive and negative observed labels. Each labeled occurrence contributes to its calibration cell. The fitter recovers candidate probability from its logged value after a 10^{-12} floor and accumulates probability mass and binary match labels.

Depth groups are $\{1\}, \{2\}, \{3, 4\}, \{5, 6, 7, 8\}, \{9, \ldots, 15\}$. Confidence is the full-vocabulary top-1 probability, binned at 0.25, 0.5, and 0.75. Ranks above 15 are neutral. All datasets share the pooled table for their model pair. Candidate lists retain original draft-rank order. Calibration changes heap priorities, not sibling exposure order. Strongly corrected later siblings are still exposed lazily. The smoothed mass ratio in Eq. (5) uses $\eta = 24$.

The 8B table is fitted from four recorded transition sources, namely ShareGPT-derived, MathInstruct-derived, mixed, and CodeAlpaca-derived collections. Its fit comprises 904,194 transition rows and 5,426,051 labeled node occurrences. A separate table is fitted for the 4B target/drafter pair. Both tables remain fixed during evaluation. Figure 2 uses schematic upper matrices and a measured Qwen3-8B correction-table slice below. Here, $*$ denotes a neutral empty cell.

A.2 COST-PROFILE CONSTRUCTION AND LOOKUP

Each model pair has a frozen full-round cost profile for node budgets 64, 96, 128, 160, and 192. The 8B profile averages two calibration passes at strength 0.25, with equal weighting across prompts. The cost whiskers in Figure 2 span these two passes.

Table 2: Qwen3-8B full-round calibration costs (ms), excluding the 0.30 ms reserve. Context buckets denote token lengths and B64–B192 denote node budgets.

Context bucket	B64	B96	B128	B160	B192
128	38.731	38.672	39.577	39.757	42.672
256	38.773	38.848	39.688	39.744	43.384
512	38.642	38.669	39.504	39.837	44.338
1024	38.520	38.570	39.382	40.204	44.779

The runtime bucket boundaries are $\{128, 256, 512, 1024, 2048, 4096\}$. A context maps to the smallest boundary at least as large as its length. The 8B profile covers 128, 256, 512, and 1024. Missing profiles select a calibrated fixed-B128 tree.

Each budget uses $C_\ell(B) + 0.30$ ms. Python and C++ share the frozen profile and budget rule.

Algorithm 2 Complete incremental allocation for one draft block

Require: Candidate probabilities q, token IDs, context length n, table c, profile C
Compute raw and corrected local scores using Eq. (6)
$\ell \leftarrow$ smallest context bucket at least n
if a complete positive C_ℓ is unavailable **then**
    return calibrated fixed-B128 tree from Algorithm 1
end if
Remove budgets dominated by a larger budget with no higher cost
Initialize tree T, corrected-priority frontier F, and $R \leftarrow 1$
for retained checkpoints B in increasing order **do**
    while $|T| < B$ **do**
        Pop highest-priority exposed node u and append it to T
        $R \leftarrow R + \exp(L_q(u))$
        Expose next sibling and rank-zero child, if they exist
    end while
    Compute U_B by Eq. (8)
    if no later checkpoint improves the proxy by Eq. (9) **then**
        break
    end if
end for
Materialize the selected tree once and return it to the target verifier

Why the frontier bound holds. A future node is an exposed sibling, a later sibling represented by an entry, or a descendant reached through those candidates. The suffix maximum bounds the raw local probability of any such sibling. Additional descendant probabilities cannot exceed one. Multiplying by the raw probability of the represented parent therefore bounds each reachable future node's mass. At most $B' - B$ nodes are added before checkpoint B', yielding the numerator in Eq. (9). The bound concerns raw draft mass, not target acceptance. The algorithm stops at the first certified checkpoint, without rollback to a previously visited tree.

A.3 MODEL AND WORKLOAD CONFIGURATION

The 8B checkpoints are `Qwen/Qwen3-8B` and `z-lab/Qwen3-8B-DFlash-b16`. The draft block has 15 future positions plus the root. Candidate extraction keeps the top 128 logits and subtracts the full-vocabulary log-normalizer, without renormalizing inside the candidate pool. Evaluation uses greedy decoding with BF16 weights and thinking mode disabled. Each request generates 512 tokens with EOS stopping disabled.

Compared configurations. At each model size, we compare eight configurations of AR, DFlash, RAW128 with Python and C++ builders, and FACT at strengths 0.15 and 0.25 with each builder. The primary setting uses strength 0.25 for 8B and 0.15 for 4B, shared across all six datasets. Chang-

ing strength affects only α. The frozen cost profile and adaptive-budget rule remain unchanged. Adaptive checkpoints are $\{64, 96, 128, 160, 192\}$, with capacity 192. Both RAW128 implementations retain capacity 128. The correction table and cost profile remain fixed.

Workload construction. The suite contains 32 single-turn prompts each from HumanEval, GSM8K, MBPP, MATH-500, and Alpaca, plus 22 from AIME25. Two timing repeats of these 182 distinct prompts yield 364 request instances per method. Alpaca prompts derive from the Stanford Alpaca data. An instruction and any nonempty input form a single user turn.

A.4 MEASUREMENT BOUNDARY

Generation time includes prefill, drafting, tree construction, target verification, and cache updates. Model loading, warmup, and one-time library compilation are excluded. Visible output is clipped to 512 tokens consistently, and the recorded round count includes the final round. Thus τ is visible tokens per target verification round, including the root contribution. Equation (10) pools counts and wall time across both repeats within each dataset.

B COMPLETE RESULTS AND TIMING VARIATION

B.1 QWEN3-8B

Table 3: Qwen3-8B matched-backend comparisons. FACT(.25) time savings use Raw128-C++ for C++ and original Raw128 for Python. vsAR is speedup ($\times$).

Dataset	Raw128-C++ vsAR	FACT vsAR	C++ time saved	Python time saved
HumanEval	6.6235	6.7956	+2.53%	+2.07%
GSM8K	6.0265	6.0904	+1.05%	+0.04%
MBPP	5.8031	5.9243	+2.05%	+1.40%
AIME25	5.5577	5.5992	+0.74%	+0.60%
MATH-500	6.4137	6.5080	+1.45%	+0.90%
Alpaca	4.1549	4.1701	+0.36%	-0.52%

Table 4: Qwen3-8B correction-strength sensitivity with the C++ builder and adaptive budgets. τ denotes tokens per verification round and vsAR denotes speedup ($\times$).

Dataset	$\alpha = .15, \tau$	vsAR	$\alpha = .25, \tau$	vsAR
HumanEval	9.7234	6.6016	10.0208	6.7956
GSM8K	9.0320	6.0722	9.0972	6.0904
MBPP	8.7662	5.9165	8.8086	5.9243
AIME25	8.4819	5.6328	8.4819	5.5992
MATH-500	9.5422	6.4281	9.6889	6.5080
Alpaca	6.1340	4.1673	6.1525	4.1701

Table 5: Qwen3-8B non-AR arms. τ denotes tokens per verification round and vsAR denotes speedup ($\times$), using the same paired AR total. Py denotes Python. The primary setting is .25 C++.

Dataset	Metric	DFlash	Raw128	.25 Py	.15 Py	Raw C++	.25 C++	.15 C++
HumanEval	τ	7.4237	9.7582	10.0208	9.7234	9.7582	10.0208	9.7234
GSM8K	τ	6.8040	8.9140	9.0972	9.0320	8.9140	9.0972	9.0320
MBPP	τ	6.1226	8.5601	8.8086	8.7662	8.5601	8.8086	8.7662
AIME25	τ	6.0788	8.2460	8.4819	8.4819	8.2460	8.4819	8.4819
MATH-500	τ	7.1891	9.4596	9.6889	9.5422	9.4596	9.6889	9.5422
Alpaca	τ	4.1073	6.1134	6.1525	6.1340	6.1134	6.1525	6.1340
HumanEval	vsAR	5.3188	6.6100	6.7497	6.5569	6.6235	6.7956	6.6016
GSM8K	vsAR	4.8703	6.0146	6.0171	6.0214	6.0265	6.0904	6.0722
MBPP	vsAR	4.3875	5.7854	5.8673	5.8697	5.8031	5.9243	5.9165
AIME25	vsAR	4.3441	5.5404	5.5739	5.5922	5.5577	5.5992	5.6328
MATH-500	vsAR	5.1526	6.3969	6.4553	6.3774	6.4137	6.5080	6.4281
Alpaca	vsAR	2.9482	4.1590	4.1377	4.1360	4.1549	4.1701	4.1673

Table 5 reports the Qwen3-8B results, with strength 0.25 for the primary configuration. AR has $\tau = 1$ and $\mathrm{vsAR} = 1$ throughout.

Table 6: Per-repeat Qwen3-8B time savings of FACT(.25)-C++ over Raw128 and Raw128-C++.

Dataset	vs Raw128, repeat 0	repeat 1	vs Raw128-C++, repeat 0	repeat 1
HumanEval	+2.702%	+2.758%	+2.476%	+2.587%
GSM8K	+0.872%	+1.611%	+0.575%	+1.512%
MBPP	+2.718%	+1.957%	+2.308%	+1.773%
AIME25	+0.565%	+1.536%	+0.306%	+1.178%
MATH-500	+1.708%	+1.707%	+1.262%	+1.635%
Alpaca	+0.318%	+0.213%	+0.567%	+0.159%

Table 6 separates the two timing repeats. The complete system reduces generation time relative to both original Raw128 and Raw128-C++ in both repeats on all six datasets.

B.2 EAGLE-3 REFERENCE

Table 7: EAGLE-3 results with RedHatAI (8B) and AngelSlim (4B) drafters, using three speculative tokens. Times sum both repeats. Speedups ($\times$) are relative to vLLM AR.

Dataset	AR time (s)	EAGLE-3 time (s)	vsAR (0)	vsAR (1)	vsAR (pooled)
Qwen3-8B					
HumanEval	586.564	252.357	2.3169	2.3318	2.3243
GSM8K	588.575	272.605	2.1566	2.1615	2.1591
MBPP	589.689	266.312	2.2084	2.2202	2.2143
AIME25	404.158	215.742	1.8740	1.8726	1.8733
MATH-500	589.857	291.860	2.0188	2.0233	2.0210
Alpaca	588.403	269.811	2.1854	2.1762	2.1808
Qwen3-4B					
HumanEval	340.634	213.342	1.5954	1.5979	1.5967
GSM8K	339.616	200.721	1.6885	1.6955	1.6920
MBPP	339.367	214.230	1.5814	1.5868	1.5841
AIME25	234.484	150.194	1.5634	1.5590	1.5612
MATH-500	340.815	209.590	1.6268	1.6254	1.6261
Alpaca	340.259	217.060	1.5688	1.5664	1.5676

EAGLE-3 uses three speculative tokens per step in vLLM, with RedHatAI (Qwen3-8B) and AngelSlim (Qwen3-4B) drafters. Table 7 reports speedups over vLLM AR. Qwen3-4B τ uses per-request round counts from output-matched replays. Qwen3-8B τ is measured for all six datasets.

B.3 QWEN3-4B

Table 8: Qwen3-4B implementation/strength comparisons. Both strengths share frozen correction/cost tables and adaptive budgets. The primary setting is .15 C++. Py denotes Python, τ denotes tokens per verification round, and vsAR denotes speedup ($\times$).

Dataset	Raw C++ vsAR	.15 Py vsAR	.15 C++ vsAR	.25 C++ τ	vsAR
HumanEval	4.7512	4.8137	4.8597	7.1546	4.9340
GSM8K	4.7002	4.7478	4.7797	6.8783	4.7171
MBPP	4.4795	4.6077	4.6667	6.6683	4.5764
AIME25	5.8238	5.9114	5.9735	8.5463	5.8933
MATH-500	5.8648	5.8638	5.9220	8.6459	5.9363
Alpaca	3.2780	3.3360	3.3643	4.8047	3.3053

Table 8 retains the strength-0.25 alternative to the 4B main-table setting, together with the Python and matched C++ controls. The main setting uses strength 0.15 on all six datasets and reduces generation time relative to original Raw128 in both repeats on every dataset.

C ADDITIONAL CASE-STUDY ANALYSIS

The case in Section 3 uses Qwen3-8B with its DFlash-b16 drafter on GSM8K. DDTree accepts two draft tokens in the observed round, giving a root-inclusive round length of three.

Both trees are reconstructed from the same top-128 draft candidates at each of 15 positions. Only construction priorities change, using the same frozen correction table as FACT. Both trees contain 128 draft nodes, excluding the common verifier root. Correction replaces 23 rank paths.

Figure 1 shows the top 60 of 116 entries in each builder's separate frontier after 115 selections. Blue marks the recorded continuation. Bar endpoints are unnormalized path log-scores with a common lower display bound. The highlighted path ranks 14th and first within the respective frontiers.

Table 9 fixes the raw frontier's top 20 candidates after 115 selections, ranked by raw scores and DDTree's tie-breaking rule without target labels. Rescoring these paths with the frozen table, without expansion, moves the supported continuation from rank 14 to rank 9 within this fixed set.

The recorded target choices support `we have accounted`. The other 19 displayed paths differ within these three known positions. The labels indicate agreement with the observed target prefix.

Labels are applied only after construction. The raw verifier accepts `we have` and supplies `accounted` as its next token. The calibrated tree covers all three. The reconstruction therefore gives a lower bound of one additional covered target token, with later suffixes unobserved.

The case was selected retrospectively for its short recovered path.

Table 9: The raw frontier's top 20 candidates after 115 selections, ordered by raw score. Scores and FACT ranks use the same 20 paths. Leading spaces are omitted, and `\n` denotes a newline.

Raw rank	Candidate path	Raw log-score	FACT log-score	FACT rank
1	`** days:\n`	-5.7913	-5.6606	14
2	`so`	-5.8082	-6.0550	20
3	`good had`	-5.8268	-5.6387	13
4	`total had`	-5.8268	-6.0429	19
5	`we days have`	-5.9163	-5.2029	3
6	`we already of`	-5.9163	-5.1952	2
7	`good have:`	-5.9163	-5.3445	6
8	`good number:\n`	-5.9163	-5.2709	4
9	`total have:`	-5.9163	-5.7487	16
10	`total number:\n`	-5.9163	-5.6752	15
11	`** have have`	-5.9163	-5.8508	18
12	`the have:\n`	-5.9163	-5.8053	17
13	`we have:\n-`	-5.9989	-4.8678	1
14	**`we have accounted`**	-6.0413	-5.3856	9
15	`we good:`	-6.0413	-5.3502	7
16	`we've:`	-6.0413	-5.3559	8
17	`we **:\n`	-6.0413	-5.3306	5
18	`good good have`	-6.0413	-5.4057	10
19	`good've have`	-6.0413	-5.4115	12
20	`good already:\n`	-6.0413	-5.4088	11

Distributions and draft-tree changes. Figures 3 and 4 pair full frontier distributions with aligned neighborhoods of the completed draft trees ($B = 128$, excluding the root, and $\alpha = 0.25$). Each pair uses the same draft snapshot but independently expanded candidate sets. The unnormalized scores use a common lower display bound within each pair. Roots show the committed context suffix, and bold tokens follow the observed target prefix. Other branches are omitted, and dashed boxes mark nodes absent from that tree. Leading token spaces are omitted. `<space>` and `\n` denote whitespace. The selected examples illustrate both a GSM8K recovery and an MBPP regression. In the MBPP case, the focal score increases but its relative priority falls, and the calibrated tree loses a previously covered token. The additional recovery and regression snapshots follow 124 and 81 selections, replacing 16 and 13 rank paths, respectively. As in Figure 1, these are recorded-prefix replays, not new measurements of τ.

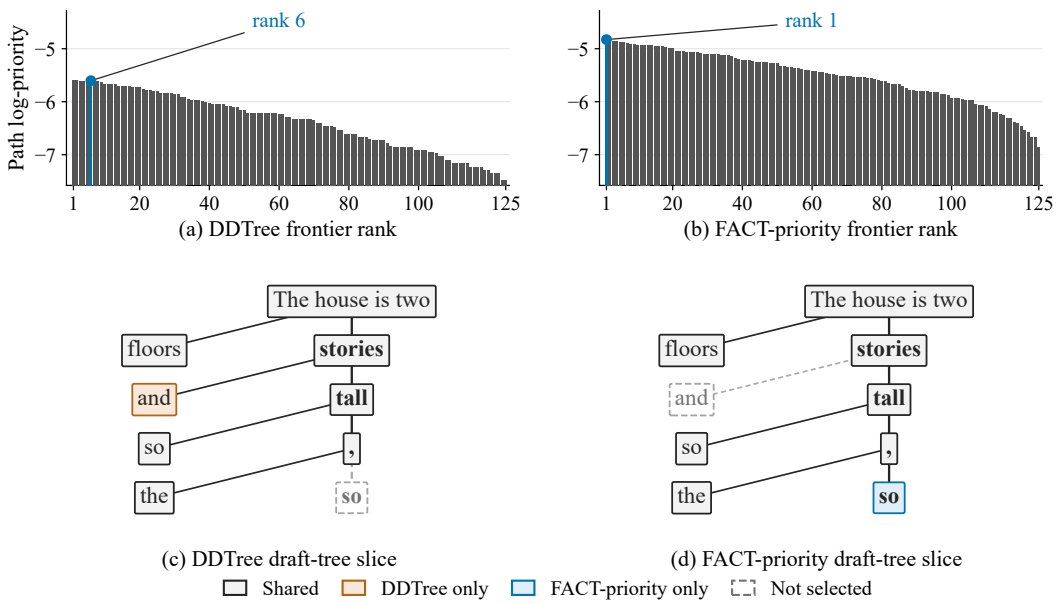


Figure 3: GSM8K recovery on `stories tall, so`. Native frontier priorities (top) and draft-tree slices (bottom). Recorded-prefix coverage changes from 3 to 4 tokens at $B = 128$.

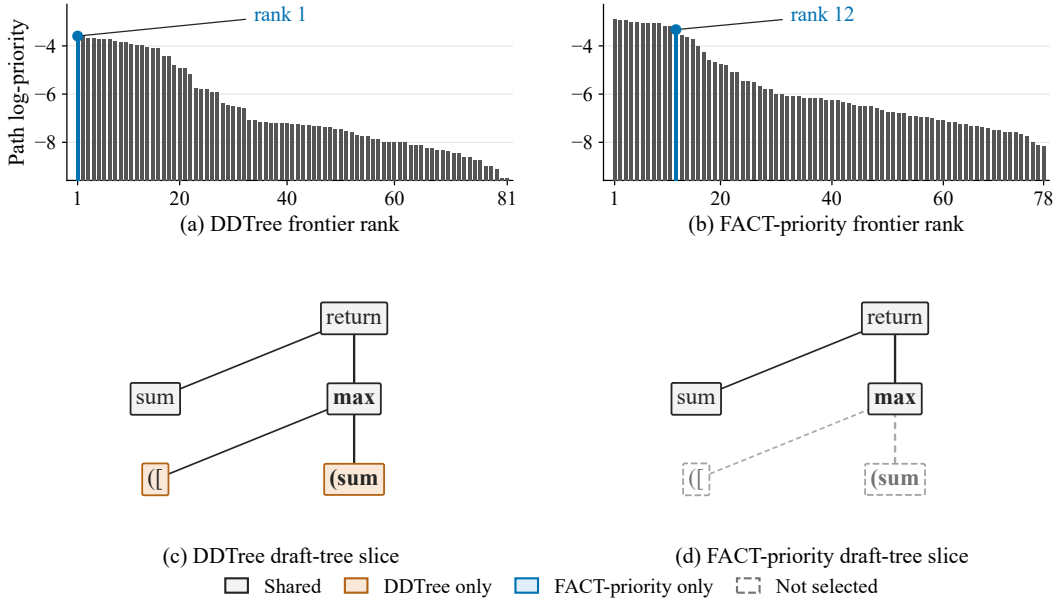


Figure 4: MBPP regression on `max(sum`. Native frontier priorities (top) and draft-tree slices (bottom). Recorded-prefix coverage changes from 2 to 1 tokens at $B = 128$.