# CAST: COST-AWARE SPECULATIVE TREES FROM ONE-PASS BLOCK DRAFTERS

**Anonymous authors**
Paper under double-blind review

## ABSTRACT

Speculative decoding accelerates large language model inference by drafting future tokens cheaply and verifying them with the target model in parallel. Block drafters score a whole block of future tokens in one forward pass, yet standard decoding verifies only the top-scoring chain and discards the other candidates. Because these candidates are already scored, verifying more of them adds target computation but no extra drafting. We introduce CAST (Cost-Aware Speculative Trees), which packs these candidates into a tree and verifies it in a single target pass, leaving the target model, drafter weights, and decoding rule untouched. To decide how wide the tree should be, CAST adds candidates while the expected gain from the next one outweighs the verification time it adds. The width therefore adapts to each deployment from a latency measurement, without sweeping over widths. We evaluate CAST across five domains on three GPU generations and two model families. At its predicted width, CAST is faster than the standard chain in all eight settings, by up to $43\%$. We also find that the best width depends strongly on the deployment. Where verification cost jumps at a kernel boundary, a 128-token tree is only $2\%$ faster than the standard chain, whereas the tree at the predicted width is $20\%$ faster. Furthermore, we prove that CAST leaves the target output distribution unchanged under both greedy and sampled decoding. We will release our code.

## 1 INTRODUCTION

Large language models (LLMs) generate text one token at a time, with a full forward pass of the model for each new token (Brown et al., 2020; Grattafiori et al., 2024; Yang et al., 2025). Speculative decoding (SD) reduces this sequential cost without changing the output distribution (Leviathan et al., 2023; Chen et al., 2023; Xia et al., 2024; Hu et al., 2025a). In each round, a lightweight drafter proposes several future tokens that the target model then verifies in a single forward pass. Block drafters such as DFlash (Chen et al., 2026a) make this drafting step particularly cheap. DFlash conditions a small block-diffusion head (Nie et al., 2025; Arriola et al., 2025) on the hidden features of the target model and, in one forward pass, returns a probability distribution over the vocabulary for every position of a future block.

In standard decoding, however, most of this output is discarded. The decoder keeps the most likely token at each position, verifies the resulting chain, and drops every other candidate that the drafter has already scored (Stern et al., 2018; Christopher et al., 2024; An et al., 2025; Zhang et al., 2026b). As Figure 1 shows, if the target rejects an early token of the chain, the round commits only a few tokens, even when the token the target wanted was among the dropped candidates. A candidate tree can retain such alternatives, and tree verification checks the entire tree in one target pass by letting each token attend only to its own ancestors (Miao et al., 2024; Cai et al., 2024).

To build such a tree, a decoder must first choose its width. How many candidates should the tree hold? Existing answers target autoregressive drafters such as the EAGLE family (Li et al., 2024a;b; 2025), which produce candidates through dependent draft steps. Adaptive and hardware-aware tree construction accounts for this drafting cost (Li et al., 2024b; Wang et al., 2025; Chen et al., 2024). These choices do not transfer directly to a block drafter, because one pass has already scored every candidate and a wider tree adds verification work but no drafting work.



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.

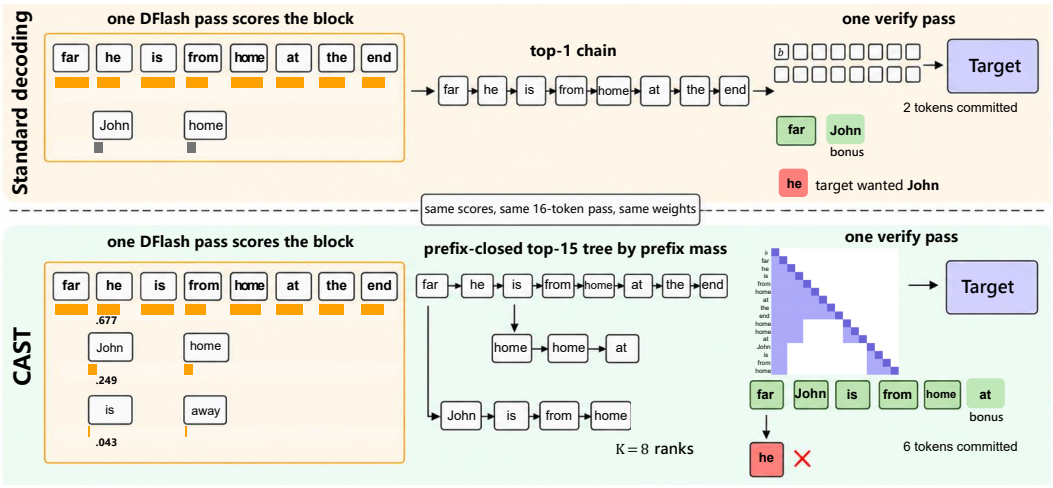


Figure 1: One round of standard DFlash decoding (top) and CAST (bottom) on a GSM8K prompt, using the same drafter scores (amber bars) and one verification pass of 16 packed tokens. Standard decoding verifies only the top-1 chain and discards the other scored candidates (gray bars), whereas CAST verifies a prefix-closed tree of the 15 highest-scoring nonroot candidates and the pending root $b$ under an ancestor mask. Green and red mark accepted and rejected tokens.

The price of this extra verification, in turn, depends on where the model runs. It differs across GPUs, grows with the number of concurrent requests, and on some GPUs jumps once the tree crosses a kernel boundary. The best width therefore changes across deployments. Finding it by sweeping widths on every deployment is costly. A concurrent study also weighs the benefit of tree nodes against their cost (Wang & Zhou, 2026), while another line retrains or extends block drafters to capture dependencies between positions (Hu et al., 2026a; Oda et al., 2026). We instead keep the drafter fixed and ask how its tree width should follow the deployment.

We introduce CAST (Cost-Aware Speculative Trees), a decoding policy that turns the discarded candidates of a block drafter into a verified tree. In each round, CAST runs the unchanged drafter once, scores candidate sequences by the product of the drafter's probabilities along each sequence, and keeps the highest-scoring sequences as a tree for the target to verify in one pass. To choose the width, CAST applies a cost-aware stopping rule. The rule adds the next candidate as long as its chance of acceptance exceeds the number of tokens the decoder would produce, at its current speed, during the verification time that the candidate adds. Because this test needs only the drafter's scores and a short latency measurement, the width adapts to each deployment without a sweep. Under serving load, the same scores divide one shared verification budget among concurrent requests, a problem studied for autoregressive drafters (Hu et al., 2026b). We evaluate CAST with DFlash as the block drafter.

We summarize our contributions and findings as follows.

- In §3, we show that keeping the highest-scoring candidates maximizes the expected number of accepted tokens under the drafter's scores for any width. We then derive the cost-aware stopping rule, extend it to a budget shared by requests, and prove that CAST preserves the target output distribution under greedy and sampled decoding.
- Experiments in §4 show that CAST is faster than the standard DFlash chain in all eight hardware and model settings, with average gains of 20–36% and up to 43% on a single domain. Its predicted width matches the best swept width in six settings. In offline replay, the tree also lengthens accepted rounds for a second one-pass drafter (Liu et al., 2026a).
- We observe that the best width depends strongly on the deployment. Where verification cost jumps at a kernel boundary, a 128-token tree is only 2.4% faster than the standard chain, whereas the tree at the predicted width is 19.8% faster. Under serving load, the best static width shrinks from a wide tree to the chain as concurrency grows, and best-first allocation of a shared budget lifts goodput over the best static width by up to 13.7%.

## 2 Background and Setup

**Speculative decoding.** Let $p$ denote the target model over vocabulary $\Sigma$, and let $\circ$ denote concatenation. Speculative decoding runs in rounds, and each round commits the accepted draft tokens together with one token selected by the target. If $A$ draft tokens are accepted, the round commits $R = A + 1$ tokens, matching the acceptance length $\tau$ reported by DFlash. Decoding throughput is the expected committed length $\mathbb{E}[R]$ divided by the round time (Leviathan et al., 2023; Sadhukhan et al., 2024). A decoder therefore gets faster by committing more tokens in each round or by making rounds cheaper.

**Round state.** A round starts from two pieces of state. The *cache-backed prefix* $x$ holds the tokens whose target key-value (KV) cache and DFlash feature cache are already computed. The *pending root* $b$ is the token committed by the previous round but not yet processed by the target. It serves as the first input of the next draft block and as the root of the next candidate tree.

**One-pass block drafters.** We use only the public DFlash interface (Chen et al., 2026a). Conditioned on the cached features for $x$, the DFlash head consumes a block $[b, \mathrm{mask}, \ldots, \mathrm{mask}]$ of $L{+}1{=}16$ tokens and, in one forward pass, returns a distribution $q_j(\cdot \mid x, b)$ for each future position $j \le L$. The drafter is not called again within the block. Standard DFlash decoding keeps the most likely token at each position and verifies the resulting chain. CAST instead uses these distributions as candidate scores and ranks a candidate sequence $y_{1:d}$ of length $d \le L$ by the *plug-in* score

$$\hat{\pi}(y_{1:d} \mid x, b) = \prod_{j \le d} q_j(y_j \mid x, b). \tag{1}$$

This product serves only to rank candidates. It does not assume that the target distribution factorizes over positions, and §4 measures how well it ranks them.

**Candidate trees and verification.** A depth-$d$ node is a candidate sequence $y_{1:d}$ with $1 \le d \le L$. A candidate tree is *prefix-closed*, meaning that it contains every ancestor of each of its nodes, and its width or budget $N$ counts its *nonroot* nodes. Verifying a budget-$N$ tree therefore packs $N{+}1$ tokens, the pending root included, into one target pass. Following SpecInfer and Medusa (Miao et al., 2024; Cai et al., 2024), this pass uses an attention mask under which each node sees only the prefix and its own ancestors, and Lemma 1 shows that it reproduces exactly what $N{+}1$ separate autoregressive forward passes would compute. Under greedy decoding, the round commits the longest tree path that agrees with the target and appends the target's next token, which becomes the new pending root.

**Sampled decoding.** Our optimality analysis in §3 uses greedy verification. For temperature $T{>}0$, CAST runs a *proposal-aware stochastic tree verifier*, which applies recursive speculative sampling without replacement to the retained siblings at each node. By Theorem 3, its committed tokens follow the distribution of autoregressive sampling from the target at temperature $T$.

## 3 The CAST Width Policy

CAST changes the verifier input while keeping the target, drafter weights, feature-cache contract, and decoding rule fixed.

### 3.1 The cost of width

**The width trade-off.** Adding a candidate node can raise the committed round length $R$ but adds selection, heap, mask, packing, target-forward, and KV-gather work. A wider tree pays off only when the gain in $R$ outweighs that work. The component probe of Table 5 isolates the target-forward part of that cost, and Theorem 2 turns it into a width $N^*$.

**Draft side.** One drafter pass already scores all $L$ future positions. Widening a tree within that block adds timed host-side work but no drafter call. Depth beyond $L$ needs another draft step.

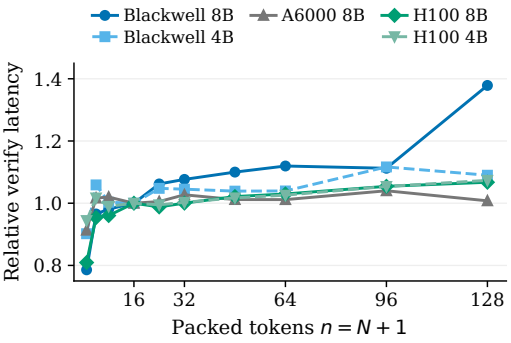


Figure 2: Batch-one target-forward latency relative to 16 packed tokens across GPUs and targets, with packed count $n = N + 1$.

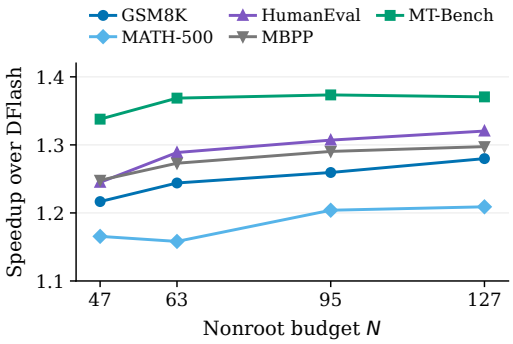


Figure 3: Qwen3-8B width sweeps on H100 SXM across five domains. Speedup normalizes to standard DFlash using decode-only latency.

**Verify side.** The target-forward latency $\ell_{\text{tgt}}(n)$ in Figure 2 and Table 5 grows only locally with the packed-token count $n = N+1$. From $n$=16 to $n$=64 it rises only $+12\%$ on Blackwell-8B and a few percent on every other GPU and target pair, then jumps at the Blackwell-8B $n$=128 kernel/tile boundary alone. Over $n \in [16, 128]$ a linear fit $\ell_{\text{tgt}}(n) \approx c_0 + c_1^{\text{fw}} n$ with intercept $c_0$ and forward slope $c_1^{\text{fw}}$ approximates the measurements, with probe details in Appendix D.

### 3.2 Budget-optimal candidate trees

The results here are round-local and greedy. Using the convention of §2, a *candidate tree* is a prefix-closed set $S$ of nonempty strings after the root $b$, with depth at most $L$ and budget $N = |S|$. Under greedy verification, let $Y$ be the target's greedy continuation after $b$. The accepted-prefix length, the committed round length, and the target prefix law of a node $v$ are

$$A(S) = \max\{k : Y_{1:k} \in S\}, \qquad R(S) = A(S) + 1, \qquad \pi^\star(v) = \Pr\left[Y_{1:|v|} = v \mid x, b\right].$$

The fixed-belief results replace $\pi^\star$ by any prefix belief $\rho(v)$, nonincreasing along prefixes, and write $\mathbb{E}_\rho$ for expectation under a law with these prefix masses. By Lemma 2, expected accepted length then decomposes into $\sum_{v \in S} \rho(v)$, which turns tree selection into a top-$N$ choice.

**Theorem 1** (Top-prefix optimality under a fixed belief)**.** *Let $\mathcal{U}$ be a fixed finite prefix-closed universe of feasible nonroot strings ($N \le |\mathcal{U}|$) and let $\rho$ be prefix masses, nonincreasing from ancestors to descendants. Among prefix-closed $S \subseteq \mathcal{U}$ with $|S| = N$, $\mathbb{E}_\rho[A(S)]$ is maximized by the $N$ strings in $\mathcal{U}$ with largest $\rho$ (ties toward prefixes), and this set is itself prefix-closed.*

By Corollary 1, setting $\rho = \hat{\pi}$ from (1) gives the exact top-$N$ tree for the plug-in objective inside the retained universe, under the one-pass marginal ranking interface of Assumption 1, which describes the drafter and not the target law. The public decoder keeps the top $K$=8 tokens at each future position and expands the tree best-first in decreasing $\hat{\pi}$ order, a rank cap that matches the full-rank tree's accepted length to within $1.1\%$ at the evaluated budgets, as measured in Appendix G.1.

Proposition 1 bounds the gap between the retained-universe oracle tree and the plug-in tree by a sum of prefix-belief estimation errors over the symmetric difference of the two trees. The drafter is well calibrated at the first tree level, as reported in Appendix G.

### 3.3 Cost-aware width

Theorem 1 says which nodes to include once $N$ is fixed. The cost slope determines how large $N$ should be. Write $g(N) = \mathbb{E}_\rho[A(S_N)]$ for the expected accepted length of the optimal budget-$N$ tree $S_N$ over a fixed finite prefix-closed candidate universe, and let the round time be

$$\ell(N) \;=\; c_{\text{draft}} + h(N) + \ell_{\text{tgt}}(n_0 + N),$$

where $c_{\text{draft}}$ is the drafter forward cost, $h(N)$ is packing, mask, heap, and KV-gather overhead, $n_0 = 1$ counts the always-verified root, and $\ell_{\text{tgt}}$ is the target-forward latency of §3.1.

**Theorem 2** (Marginal stopping rule). *The function $g$ is concave in $N$ with increments $g(N)-g(N-1) = \rho_{(N)}$, where $\rho_{(N)}$ is the $N$-th largest prefix mass in the universe (an order statistic). Assume $h(N) + \ell_{\text{tgt}}(n_0 + N)$ is locally affine with nonnegative total slope $c_1$ over the allowed range. Then throughput $(g(N)+1)/\ell(N)$ is unimodal, and from budget $N$, adding node $N+1$ does not reduce it if and only if*

$$\underbrace{\rho_{(N+1)}}_{\text{prefix mass of next node}} \geq \underbrace{\frac{g(N)+1}{\ell(N)}}_{\text{throughput (tok/ms)}} \cdot \underbrace{c_1}_{\text{cost/node (ms)}} .$$

*A maximizer is found by adding nodes until this condition first fails, with boundary handling at the smallest and largest allowed budgets.*

The rule keeps adding candidates while the next node's prefix mass outweighs the system's token-time exchange rate. In our experiments we set $c_1$ to the forward-only slope $c_1^{\text{fw}}$ that §3.1 fits to the probe measurements. Because the probe omits the overhead $h$, whose slope is nonnegative, this choice under-estimates $c_1$, and the sweeps of §4 bound the resulting error.

**Scope.** The affine assumption is local. For example, hybrid linear-attention targets pay a fixed chunk-kernel entry cost that the chain and the tree share, after which their cost is again locally affine, as Appendix D.1 shows.

**Empirical validation.** On Blackwell/Qwen3-8B the threshold is about $0.015$, crossed between $N$=47 and 63, so we deploy $N^*$=63. The same rule selects wider trees on flatter cost curves. Appendix D gives the cost slopes and sweep boundaries.

**Width under serving load.** Under load, $B$ concurrent requests share one verifier budget $M$, the sum of their widths. One drafter pass serves the batch at any width, while verification grows with the total packed count. Because expected accepted length is additive across requests, global best-first expansion maximizes it over the disjoint candidate forests, as Proposition 2 proves. The batched stopping rule of Proposition 3 adds the next global node while its mass reaches the batch token-time exchange rate. Both results reduce to the batch-one rule at $B$=1 and permit unequal widths.

### 3.4 The CAST decoder

CAST is a round-level decoder for the unmodified one-pass DFlash interface, whose DFlash-specific part is the pending-root round contract of §2. Given the cache-backed prefix $x$, pending root $b$, feature cache, and budget $N$, each round runs the unchanged DFlash drafter once on the block $[b, \text{mask}, \ldots, \text{mask}]$, keeps the top $K$ tokens at each future position and enumerates the top-$N$ prefix scores of Theorem 1 inside that universe. It verifies the packed tree in one ancestor-masked target pass, then walks the tree, emits the accepted path and the next pending root, and rebuilds the caches, with the full algorithm in Appendix C.

**Exactness and sampling.** At temperature 0 the walk commits the target greedy continuation by construction. In fp32, CAST and standard DFlash both reproduce autoregressive decoding token for token on Qwen3-8B and Qwen3-4B. In bf16, both depart from autoregressive decoding only at near-ties of the top two target logits, where shape-dependent rounding can flip the greedy token.

For $T$>0 the proposal-aware stochastic verifier of §2 instantiates SpecInfer/SpecTr-style recursive speculative sampling (Miao et al., 2024; Sun et al., 2023; 2024; Weng et al., 2025; Zhou et al., 2026) over the prefix-closed tree, a rule that Theorem 3 proves distribution-preserving. The total variation of its committed tokens from autoregressive sampling is below the Monte-Carlo floor in fp64 and about $0.004$ in bf16, as Appendix D.2 details.

## 4 Experiments

**Setup.** Our main experiments use the public, unmodified z-lab DFlash-b16 heads for non-thinking Qwen3-4B and Qwen3-8B (Yang et al., 2025), bf16, batch 1, and greedy decoding except at $T$=1.

The rule thresholds drafter-score order statistics from 1,500 offline GSM8K rounds, using one curve across deployments and changing only the measured cost slope. Width selection uses scores and costs, not timing-sweep outcomes. Appendix D.2 describes the paired prompt-level bootstrap.

We time autoregressive (AR) decoding, DFlash, and CAST in one shared decoding loop on an H100 SXM across five math, code, and chat domains, using 40 prompts in each domain and the same token budget for every method. Runs on Blackwell (RTX PRO 6000 Server Edition) and A6000 cover GSM8K, MT-Bench, and HumanEval. We report decode-only steady-state time $t$ in ms/token, speedup $t_{\mathrm{AR}}/t$, and gain $t_{\mathrm{DFlash}}/t_{\mathrm{CAST}} - 1$ over DFlash, averaging domain ratios for each setting. Appendices D and E give timer boundaries, datasets, and budgets.

**Speedup at the predicted width.** In Table 1, CAST improves on DFlash across every H100 domain for Qwen3-8B and Qwen3-4B. Both targets average 5.62× speedup over AR, with longer committed rounds throughout.

EAGLE-3 uses its official code and public Qwen3 heads at the reported settings, and it shares the machine, prompts, decode-only timer, and AR baseline of the DFlash and CAST rows. The same cost rule selects $N^*$=127 for the block-10 head of LLaMA-3.1-8B (Grattafiori et al., 2024), extending the benefit to a second target family and block size. Appendix H.1 gives that sweep.

The nearly flat H100 SXM cost curve keeps the stopping threshold below candidate mass through the largest evaluated budget, and the sweeps in Figure 3 peak at or near that budget in every domain.

Table 1: Qwen3 decoding on H100 SXM. Cells report speedup over AR and committed round length $R$. Parentheses give baseline packed tokens or the CAST nonroot budget.

| Model | Method | MATH | | | | CODE | | | | CHAT | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | GSM8K | | MATH-500 | | HumanEval | | MBPP | | MT-Bench | | *Avg.* | |
| Temperature = 0 | | Speedup | $R$ | Speedup | $R$ | Speedup | $R$ | Speedup | $R$ | Speedup | $R$ | Speedup | $R$ |
| Q3-8B | EAGLE-3 (16) | 2.58× | 4.45 | 2.37× | 4.11 | 2.23× | 3.97 | 2.37× | 4.11 | 2.07× | 3.74 | 2.32× | 4.08 |
| | EAGLE-3 (60) | 3.08× | 5.34 | 2.79× | 4.88 | 2.69× | 4.76 | 2.80× | 4.93 | 2.41× | 4.49 | 2.75× | 4.88 |
| | DFlash (16) | 4.54× | 6.23 | 5.57× | 7.81 | 4.67× | 6.52 | 4.82× | 6.56 | 2.30× | 4.06 | 4.38× | 6.24 |
| | CAST ($N^*$=127) | **5.81×** | **8.46** | **6.74×** | **9.87** | **6.16×** | **9.04** | **6.25×** | **9.09** | **3.16×** | **5.52** | **5.62×** | **8.40** |
| Q3-4B | EAGLE-3 (16) | 1.81× | 3.29 | 1.79× | 3.11 | 1.62× | 3.00 | 1.71× | 3.03 | 1.59× | 2.84 | 1.70× | 3.05 |
| | EAGLE-3 (60) | 2.08× | 3.81 | 2.03× | 3.61 | 1.86× | 3.41 | 1.94× | 3.46 | 1.87× | 3.34 | 1.96× | 3.53 |
| | DFlash (16) | 4.40× | 6.13 | 5.34× | 7.49 | 4.61× | 6.49 | 4.96× | 6.72 | 2.30× | 3.95 | 4.32× | 6.15 |
| | CAST ($N^*$=127) | **5.79×** | **8.47** | **6.62×** | **9.76** | **6.04×** | **9.02** | **6.34×** | **9.11** | **3.29×** | **5.78** | **5.62×** | **8.43** |
| Temperature = 1 | | Speedup | $R$ | Speedup | $R$ | Speedup | $R$ | Speedup | $R$ | Speedup | $R$ | Speedup | $R$ |
| Q3-8B | EAGLE-3 (16) | 2.22× | 4.30 | 2.32× | 3.91 | 2.19× | 3.90 | 2.22× | 3.89 | 1.82× | 3.52 | 2.15× | 3.90 |
| | EAGLE-3 (60) | 2.63× | 5.17 | 2.65× | 4.62 | 2.52× | 4.57 | 2.58× | 4.66 | 2.16× | 4.29 | 2.51× | 4.66 |
| | DFlash (16) | 4.11× | 5.69 | 4.39× | 6.36 | 4.08× | 5.77 | 3.83× | 5.31 | 2.15× | 3.63 | 3.71× | 5.35 |
| | CAST ($N^*$=127) | **5.16×** | **7.93** | **5.39×** | **8.46** | **5.35×** | **8.27** | **4.97×** | **7.56** | **2.94×** | **5.30** | **4.76×** | **7.50** |
| Q3-4B | EAGLE-3 (16) | 1.60× | 3.21 | 1.68× | 3.00 | 1.67× | 3.00 | 1.64× | 2.98 | 1.38× | 2.80 | 1.59× | 3.00 |
| | EAGLE-3 (60) | 1.83× | 3.75 | 1.89× | 3.48 | 1.83× | 3.41 | 1.81× | 3.42 | 1.53× | 3.22 | 1.78× | 3.46 |
| | DFlash (16) | 4.16× | 5.80 | 4.54× | 6.55 | 4.27× | 5.96 | 4.29× | 5.96 | 2.21× | 3.79 | 3.90× | 5.61 |
| | CAST ($N^*$=127) | **5.14×** | **7.88** | **5.56×** | **8.69** | **5.54×** | **8.59** | **5.41×** | **8.27** | **3.02×** | **5.44** | **4.93×** | **7.77** |

**Width across hardware.** The Blackwell-8B cost cliff erodes most of the gain at the largest width. Gains plateau at $N$=63–95 and fall at 127. Flatter A6000 and Blackwell-4B curves support wider trees. Appendix D gives their full sweeps.

Across all eight settings of Table 2, the predicted gain is within 1.7 percentage points of the sweep oracle. The settings cover three GPUs, two model families, two block sizes, and a 30B sparse mixture-of-experts target.

**Candidate scores.** Figure 4 uses the same 40-prompt offline replay slices as the main rows. In this replay at a fixed budget of 47 nonroot nodes, neither a fixed branching schedule nor uniform binary branching commits longer rounds than the 15-node chain in any domain, whereas the top-$N$ tree does in every domain. On GSM8K its committed round length reaches 7.7 against 6.3 for the chain,

Table 2: Predicted and sweep-oracle widths across the eight settings. Gain over DFlash is averaged over domains. Gap is measured in percentage points. Sweeps use $N \in \{15, 31, 47, 63, 95, 127\}$. The H100 SXM Qwen3-8B and Qwen3-4B sweeps start at $N$=47, and the A6000 sweeps end at $N$=95. ‡ marks cells measured on a second A6000 host.

| Setting (drafter block) | $N^*$ | Gain at $N^*$ | Oracle width (gain) | Gap | Gain at $N$=127 |
|---|---|---|---|---|---|
| H100 SXM, Qwen3-8B (b16) | 127 | +29.5% | 127 | +29.5% | +0.0 | +29.5% |
| H100 SXM, Qwen3-4B (b16) | 127 | +31.5% | 127 | +31.5% | +0.0 | +31.5% |
| Blackwell, Qwen3-8B (b16) | 63 | +19.8% | 95 | +21.5% | -1.7 | +2.4% |
| Blackwell, Qwen3-4B (b16) | 95 | +27.3% | 127 | +29.0% | -1.7 | +29.0% |
| A6000, Qwen3-8B (b16) | 95 | +28.7% | 95 | +28.7% | +0.0 | +25.1%‡ |
| A6000, Qwen3-4B (b16) | 95 | +32.6% | 95 | +32.6% | +0.0 | +31.7%‡ |
| H100 SXM, LLaMA-3.1-8B (b10) | 127 | +27.4% | 127 | +27.4% | +0.0 | +27.4% |
| H100 SXM, Qwen3-Coder-30B-A3B (b16) | 127 | +35.6% | 127 | +35.6% | +0.0 | +35.6% |

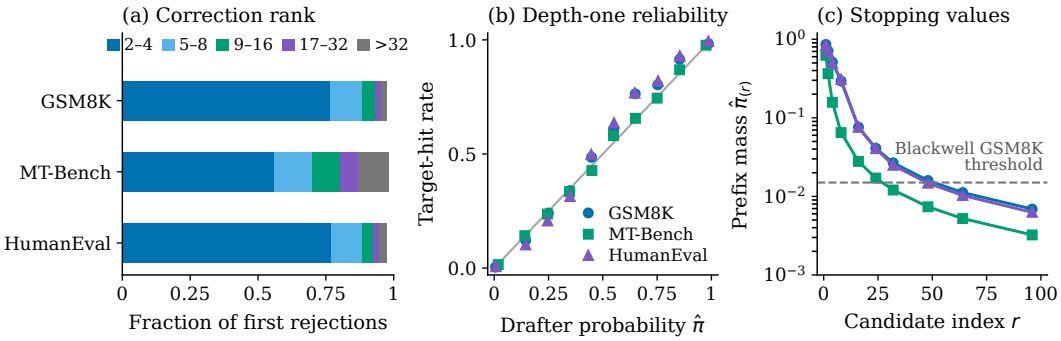


Figure 4: Qwen3-8B greedy replay diagnostics on 40-prompt slices. (a) Target-correction rank at the first DFlash rejection. (b) Depth-one reliability. (c) Sorted candidate prefix masses. The dashed line is the Blackwell GSM8K threshold.

and Appendix G gives the remaining cells. At the first chain rejection, the target correction lies at draft ranks 2–4 in 56–77% of events. Depth-one reliability and the falling prefix masses explain the ranking and stopping rule.

**Shape at a matched budget.** At 16 packed tokens, CAST keeps $N$=15 nonroot nodes from the same block-16 drafter as DFlash, so the two decoders differ only in tree shape. This shape alone makes CAST 5.9 to 29.2% faster than DFlash in all twelve Blackwell and A6000 cells of Figure 5. Table 9 gives the full round lengths and timings.

**Sampled decoding.** The $T$=1 blocks of Table 1 keep the same rule-predicted width and replace greedy verification by the proposal-aware stochastic verifier of Theorem 3. Sampling shortens the committed rounds of DFlash and CAST, yet the gain of CAST over DFlash stays between +22% and +37% in every domain on both targets. On Blackwell with Qwen3-8B, the gain exceeds 15% under each of three sampling seeds, as Table 3 shows.

**Robustness.** On Blackwell, a paired bootstrap over expanded prompt sets places every 95% confidence interval for the mean prompt-level gain

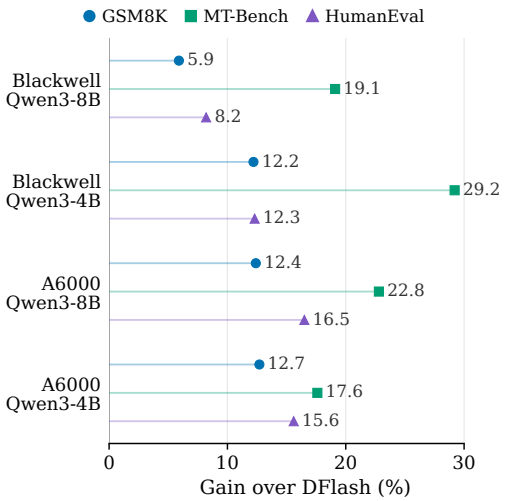


Figure 5: Gain over DFlash at 16 packed tokens.

Table 4: LLaMA-3.1-8B greedy decoding with the block-10 DFlash head on H100 SXM. Cells report speedup over AR. Parentheses give baseline packed tokens or the CAST nonroot budget.

| Method | | MATH | | CODE | | CHAT | |
|---|---|---|---|---|---|---|---|
| | | GSM8K | MATH-500 | HumanEval | MBPP | MT-Bench | *Avg.* |
| EAGLE-3 | (10) | 2.05× | 1.75× | 2.28× | 2.25× | 1.73× | 2.01× |
| EAGLE-3 | (60) | 2.67× | 2.31× | 2.80× | 2.81× | 2.34× | 2.59× |
| DFlash | (10) | 3.00× | 2.76× | 3.60× | 3.56× | 2.32× | 3.05× |
| CAST ($N^*$=127) | | **3.82**× | **3.59**× | **4.38**× | **4.49**× | **3.05**× | **3.87**× |

over DFlash above +14%, as Table 3 shows. The gain also holds for longer generations and prefixes on H100 SXM. On MATH-500 with Qwen3-8B, CAST is 21% faster than DFlash at 512 new tokens and 20% faster at 2048. On GovReport summarization prompts cut to 8k tokens, it is 28% faster on Qwen3-8B and 27% faster on Qwen3-4B, within 3 percentage points of the sweep oracle. Appendix E reports these cells.

**Comparison with EAGLE-3 on LLaMA.** In Table 4, CAST at the predicted width averages 3.87× over AR on LLaMA-3.1-8B, against 2.59× for the official EAGLE-3 with a 60-token tree. It is faster than that tree by 30 to 60% in every domain. On MT-Bench, DFlash alone trails EAGLE-3 at 2.32× against 2.34×, while CAST reaches 3.05×. On GSM8K, CAST and the 60-token EAGLE-3 tree commit the same $R$=6.08, yet EAGLE-3 runs eight draft passes in each round to build its tree and CAST runs one. Appendix H.1 reports every width.

Table 3: Mean gain over DFlash on Blackwell at the predicted width $N^*$, which is 63 for Qwen3-8B and 95 for Qwen3-4B. Brackets give 95% bootstrap intervals for greedy decoding and ranges over three seeds for sampling.

| GSM8K | MT-Bench | HumanEval |
|---|---|---|
| *Qwen3-8B, greedy decoding* | | |
| +18.1% | +24.6% | +16.9% |
| [16.4, 19.8] | [21.6, 27.7] | [14.8, 19.0] |
| *Qwen3-4B, greedy decoding* | | |
| +25.7% | +33.2% | +23.2% |
| [24.0, 27.4] | [29.4, 37.0] | [21.1, 25.3] |
| *Qwen3-8B, sampling at $T$=1* | | |
| +16.7% | +30.1% | +22.3% |
| [16.0, 18.0] | [28.7, 32.3] | [20.3, 25.3] |

**Second one-pass drafter.** In offline replay over the one-pass logits of DART (Liu et al., 2026a), a best-first tree of 63 nodes lengthens the committed round over the eight-node top-1 chain by 40% on GSM8K and 34% on HumanEval. Timed on an A6000 with Qwen3-8B, DART's own tree decoder runs faster with 60 nodes than with 8 in both domains, reaching a 2.14× speedup over autoregressive decoding on GSM8K and 2.34× on HumanEval. Appendix G.2 details both measurements.

## 5 Serving under load

No single static width serves the load range. The drafter forward is width-independent, one pass serving all $B$ requests, while the verifier width tax in Figure 6a grows from +10% at $B$=1 to +237% at $B$=32. In the batched harness of Appendix F, the best static width is therefore a fixed wide tree up to $B$=8 and the plain chain from $B$=16. Global best-first allocation from Section 3.3 instead splits one shared budget across requests. In offline replay of 32-request batches whose budget averages four nodes for each request, it raises total accepted tokens over a uniform split by 16.8 to 22.5%. Measured on Blackwell with Qwen3-8B and a ragged verifier kernel, it lifts goodput over the best static width by a median 7.0% at $B$=16 and 13.7% at $B$=32 across four runs.

## 6 Related Work

SMART applies a marginal benefit–cost rule to existing tree decoders (Wang & Zhou, 2026). For autoregressive drafters, OPT-Tree maximizes a draft-probability proxy for expected acceptance and stops deepening once the gain falls below a threshold tied to the cost of a draft step (Wang et al., 2025). A one-pass drafter adds no draft step within its block, and our threshold instead prices mea-

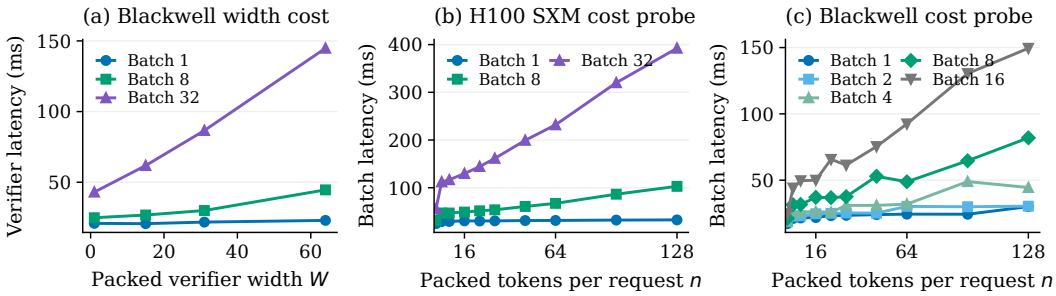


Figure 6: Qwen3-8B verification cost under load. (a) Blackwell verifier latency over the packed width $W$ of each request. (b,c) Batch latency over the packed tokens $n$ of each request on H100 SXM and Blackwell.

sured verification time. Our evaluation examines deployment-specific width prediction and shared-budget allocation across requests.

A related line is draft trees over one-pass parallel logits. DART prunes such logits with an external n-gram continuity signal (Liu et al., 2026a), answering which candidates to keep under a supplied construction rule. We derive the budget-optimal tree of Theorem 1 from the drafter's own product score and ask how wide it should be on the measured hardware.

JetSpec retrains the head into a causal-parallel tree drafter (Hu et al., 2026a), while Weaver adds an autoregressive adapter over the marginal logits (Oda et al., 2026). These approaches address conditional dependence. CAST instead retains the checkpoint and prices verification. ECHO shares depth and width across requests with AR drafters (Hu et al., 2026b). For a one-pass drafter, width adds no draft call, leaving verification as the shared cost.

D-cut (Liu et al., 2026b) prunes the verification depth of each request's chain under a profiled latency table, while our allocation splits prefix-closed tree width by best-first mass at measured goodput.

Beyond speculative decoding, draft-agreement routing allocates thinking budgets using agreement between inexpensive drafts (Lee et al., 2026a), and adaptive correct-only rewards reduce reasoning length through efficiency-aware training (Lee et al., 2026b). These approaches control the amount of reasoning, while CAST controls verification work without changing the target output distribution.

Appendix I reviews tree verification, adaptive AR trees, and drafter alignment. Our sampled verifier follows proposal-aware verification and traversal (Sun et al., 2024; Zhou et al., 2026; Weng et al., 2025; Sun et al., 2023; Hu et al., 2025b). Existing controllers adapt length, block size, or shape from predicted signals (Huang et al., 2024; Kim et al., 2026; Hou et al., 2025; Zhang et al., 2026a; Qian et al., 2026). Our width rule uses measured verification cost.

## Reproducibility Statement

All models, drafters, and datasets we use are public. The supplementary material contains anonymized code for the decoder, the cost probes, the width rule, the offline replay, and the batched harness. Appendix C specifies the decoder, including candidate construction, packed verification, and cache updates. Appendix D gives the prompt sets, the decode timing window, and the target-forward cost probes. Appendices E and F give the protocols of the H100 SXM and batched-harness measurements. Appendix B contains complete proofs of all formal results.

## Large Language Model Usage

A large language model assisted with LaTeX tables and language polishing. The authors reviewed and verified all outputs and take full responsibility for the paper.

### Ethics statement

CAST changes inference scheduling while preserving target weights and the greedy rule. We use public checkpoints and benchmark prompts and collect no new data. Deployments retain the base model's safety profile and should keep its safeguards, because faster inference can reduce harmful generation costs.

## References


Zihao An, Huajun Bai, Ziqiong Liu, Dong Li, and Emad Barsoum. PARD: Accelerating LLM Inference with Low-Cost PARallel Draft Model Adaptation, 2025. URL https://arxiv.org/abs/2504.18583v4.

Marianne Arriola, Aaron Gokaslan, Justin T. Chiu, Zhihan Yang, Zhixuan Qi, Jiaqi Han, Subham Sekhar Sahoo, and Volodymyr Kuleshov. Block Diffusion: Interpolating Between Autoregressive and Diffusion Language Models, 2025. URL https://arxiv.org/abs/2503.09573v3.

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. URL https://arxiv.org/abs/2108.07732.

Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, et al. Language models are few-shot learners. In *Advances in Neural Information Processing Systems*, 2020. URL https://arxiv.org/abs/2005.14165.

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. 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. URL https://arxiv.org/abs/2302.01318.

Jian Chen, Yesheng Liang, and Zhijian Liu. DFlash: Block Diffusion for Flash Speculative Decoding, 2026a. URL https://arxiv.org/abs/2602.06036v2.

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.

Wenyan Chen, Chengzhi Lu, Yanying Lin, and Dmitrii Ustiugov. FASER: Fine-Grained Phase Management for Speculative Decoding in Dynamic LLM Serving, 2026b. URL https://arxiv.org/abs/2604.20503.

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*, 2024. doi: 10.52202/079017-4116. URL https://doi.org/10.52202/079017-4116.

Jacob K Christopher, Brian R Bartoldson, Tal Ben-Nun, Michael Cardei, Bhavya Kailkhura, and Ferdinando Fioretto. Speculative Diffusion Decoding: Accelerating Language Generation through Diffusion, 2024. URL https://arxiv.org/abs/2408.05636v4.

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. URL https://arxiv.org/abs/2110.14168.

Raghavv Goel, Mukul Gagrani, Wonseok Jeon, Junyoung Park, Mingu Lee, and Christopher Lott. Direct alignment of draft model for speculative decoding with chat-fine-tuned llms. *arXiv preprint arXiv:2403.00858*, 2024. URL https://arxiv.org/abs/2403.00858.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, et al. The llama 3 herd of models, 2024. URL https://arxiv.org/abs/2407.21783.

Yunlong Hou, Fengzhuo Zhang, Cunxiao Du, Xuan Zhang, Jiachun Pan, Tianyu Pang, Chao Du, Vincent Y. F. Tan, and Zhuoran Yang. BanditSpec: Adaptive Speculative Decoding via Bandit Algorithms, 2025. URL https://arxiv.org/abs/2505.15141v2.

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, 2026a. URL https://arxiv.org/abs/2606.18394.

Xinyi Hu, Yuhao Shen, Baolin Zhang, Hengxin Zhang, Jun Dai, Shuang Ge, Lei Chen, Yue Li, and Mingcheng Wan. ECHO: Elastic Speculative Decoding with Sparse Gating for High-Concurrency Scenarios, 2026b. URL https://arxiv.org/abs/2604.09603.

Yunhai Hu, Zining Liu, Zhenyuan Dong, Tianfan Peng, Bradley McDanel, and Sai Qian Zhang. Speculative Decoding and Beyond: An In-Depth Survey of Techniques, 2025a. URL https://arxiv.org/abs/2502.19732v4.

Zhengmian Hu, Tong Zheng, Vignesh Viswanathan, Ziyi Chen, Ryan A. Rossi, Yihan Wu, Dinesh Manocha, and Heng Huang. Towards Optimal Multi-draft Speculative Decoding, 2025b. URL https://arxiv.org/abs/2502.18779v1.

Kaixuan Huang, Xudong Guo, and Mengdi Wang. SpecDec++: Boosting Speculative Decoding via Adaptive Candidate Lengths, 2024. URL https://arxiv.org/abs/2405.19715v3.

Zongle Huang, Lei Zhu, Zongyuan Zhan, Ting Hu, Weikai Mao, Xianzhi Yu, Yongpan Liu, and Tianyu Zhang. MoESD: Unveil Speculative Decoding's Potential for Accelerating Sparse MoE, 2025. URL https://arxiv.org/abs/2505.19645. NeurIPS 2025.

Taehyeon Kim, Hojung Jung, and Se-Young Yun. Multi-Drafter Speculative Decoding with Alignment Feedback, 2026. URL https://arxiv.org/abs/2604.05417v1.

Jungseob Lee, Seongtae Hong, Seungjun Lee, Jaehyung Seo, Junyoung Son, Sugyeong Eo, Chanjun Park, Hyeongju Park, Hyeonseok Moon, and Heuiseok Lim. DART: Draft-agreement routing for training-free adaptive thinking budgets in hybrid reasoning models, 2026a. URL https://arxiv.org/abs/2606.23181.

Jungseob Lee, Seungyoon Lee, Seongtae Hong, Minhyuk Kim, Chanjun Park, and Heuiseok Lim. Beyond penalizing mistakes: Stabilizing efficiency training in large reasoning models via adaptive correct-only rewards, 2026b. URL https://arxiv.org/abs/2606.22716.

Yaniv Leviathan, Matan Kalman, and Yossi Matias. Fast inference from transformers via speculative decoding. In *Proceedings of the 40th International Conference on Machine Learning*, pp. 19274–19286, 2023. URL https://proceedings.mlr.press/v202/leviathan23a.html.

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. 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*, 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*, 2025. doi: 10.52202/085713-4562. URL https://doi.org/10.52202/085713-4562. arXiv:2503.01840.

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. In *International Conference on Learning Representations*, 2024. URL https://arxiv.org/abs/2305.20050.

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, 2026a. URL https://arxiv.org/abs/2601.19278v1.

Tianyu Liu, Yuhao Shen, Rui Cen, Junhan Shi, Jiebin Zhang, Guangshuo Qin, Hong Liu, Song Liu, Guanghua Yu, and Jianchen Zhu. D-cut: Adaptive Verification Depth Pruning for Batched Speculative Decoding, 2026b. URL https://arxiv.org/abs/2607.14647.

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*, 2024. doi: 10.1145/3620666.3651335. URL https://doi.org/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, 2025. URL https://arxiv.org/abs/2502.09992v3.

Yuma Oda, Ryan Mathieu, Roman Knyazhitskiy, and Artur Chakhvadze. Trees from Marginals: Autoregressive Drafting with Factorized Priors, 2026. URL https://arxiv.org/abs/2607.06763.

Yu-Yang Qian, Hao-Cong Wu, Chen Chen, Jiacheng Sun, Zhenhua Dong, Peng Zhao, and Zhi-Hua Zhou. AdaFlash: Adaptive Speculative Decoding via On-Policy Distilled Diffusion Drafters, 2026. URL https://arxiv.org/abs/2607.19223.

Ranajoy Sadhukhan, Jian Chen, Zhuoming Chen, Vashisth Tiwari, Ruihang Lai, Jinyuan Shi, Ian En-Hsu Yen, Avner May, Tianqi Chen, and Beidi Chen. MagicDec: Breaking the Latency-Throughput Tradeoff for Long Context Generation with Speculative Decoding, 2024. URL https://arxiv.org/abs/2408.11049.

Anish Saxena, Po-An Tsai, Hritvik Taneja, Aamer Jaleel, and Moinuddin Qureshi. Utility-Driven Speculative Decoding for Mixture-of-Experts, 2025. URL https://arxiv.org/abs/2506.20675.

Aaryam Sharma. When Is a Draft Accepted? A Theory of Acceptance in Speculative Decoding, 2026. URL https://arxiv.org/abs/2606.30265.

Mitchell Stern, Noam Shazeer, and Jakob Uszkoreit. Blockwise parallel decoding for deep autoregressive models. In *Advances in Neural Information Processing Systems*, 2018. URL https://arxiv.org/abs/1811.03115.

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. doi: 10.52202/075280-1314. URL https://doi.org/10.52202/075280-1314.

Ziteng Sun, Uri Mendlovic, Yaniv Leviathan, Asaf Aharoni, Jae Hun Ro, Ahmad Beirami, and Ananda Theertha Suresh. Block verification accelerates speculative decoding. *arXiv preprint arXiv:2403.10444*, 2024. URL https://arxiv.org/abs/2403.10444.

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, 2025. URL https://doi.org/10.1162/tacl_a_00735.

Lifu Wang and Pan Zhou. Smart: When is it actually worth expanding a speculative tree?, 2026. URL https://arxiv.org/abs/2604.09731.

Zhibin Wang, Xuying Han, Zhaohua Yang, Fuliang Liu, Xue Li, Rong Gu, Sheng Zhong, and Chen Tian. SpecLA: Efficient Speculative Decoding for Linear-Attention Models, 2026. URL https://arxiv.org/abs/2607.16673.

Yepeng Weng, Qiao Hu, Xujie Chen, Li Liu, Dianwen Mei, Huishi Qiu, Jiang Tian, and Zhongchao Shi. Traversal verification for speculative tree decoding. In *Advances in Neural Information Processing Systems*, 2025. URL https://arxiv.org/abs/2505.12398. arXiv:2505.12398.

Zhaoxuan Wu, Zijian Zhou, Arun Verma, Alok Prakash, Daniela Rus, and Bryan Kian Hsiang Low. TETRIS: Optimal Draft Token Selection for Batch Speculative Decoding, 2025. URL https://arxiv.org/abs/2502.15197.

Heming Xia, Zhe Yang, Qingxiu Dong, Peiyi Wang, Yongqi Li, Tao Ge, Tianyu Liu, Wenjie Li, and Zhifang Sui. Unlocking Efficiency in Large Language Model Inference: A Comprehensive Survey of Speculative Decoding, 2024. URL https://arxiv.org/abs/2401.07851v3.

An Yang, Anfeng Li, Baosong Yang, et al. Qwen3 technical report, 2025. URL https://arxiv.org/abs/2505.09388.

Ming Yin, Minshuo Chen, Kaixuan Huang, and Mengdi Wang. A theoretical perspective for speculative decoding algorithm. *Advances in Neural Information Processing Systems*, 2024. doi: 10.52202/079017-4067. URL https://doi.org/10.52202/079017-4067. arXiv:2411.00841.

Hao Zhang, Yiming Hu, Yong Wang, Mingqiao Mo, Xin Xiao, and Xiangxiang Chu. BlockPilot: Instance-Adaptive Policy Learning for Diffusion-based Speculative Decoding, 2026a. URL https://arxiv.org/abs/2606.31315.

Jiebin Zhang, Zhenghan Yu, Song Liu, Eugene J. Yu, Zheng Li, Dawei Zhu, Jiangshan Duo, Weimin Xiong, Yifan Song, Guanghua Yu, Jianchen Zhu, and Sujian Li. DFlare: Scaling Up Draft Capacity for Block Diffusion Speculative Decoding, 2026b. URL https://arxiv.org/abs/2606.02091.

Lefan Zhang, Xiaodan Wang, Yanhua Huang, and Ruiwen Xu. Learning harmonized representations for speculative sampling. *International Conference on Learning Representations*, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/hash/575286a73f238b6516ce0467d67eadb2-Abstract-Conference.html. arXiv:2408.15766.

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 (NeurIPS) Datasets and Benchmarks Track*, 2023. doi: 10.52202/075280-2020. URL https://doi.org/10.52202/075280-2020.

Yongchao Zhou, Kaifeng Lyu, Ankit Singh Rawat, Aditya Krishna Menon, Afshin Rostamizadeh, Sanjiv Kumar, Jean-François Kagy, and Rishabh Agarwal. Distillspec: Improving speculative decoding via knowledge distillation. In *International Conference on Learning Representations*, 2024. URL https://proceedings.iclr.cc/paper_files/paper/2024/hash/8766fbc68e1ed1cdef712ce273e0a363-Abstract-Conference.html. arXiv:2310.08461.

Yuxuan Zhou, Fei Huang, Heng Li, Fengyi Wu, Tianyu Wang, Jianwei Zhang, Junyang Lin, and Zhi-Qi Cheng. Overcoming Joint Intractability with Lossless Hierarchical Speculative Decoding, 2026. URL https://arxiv.org/abs/2601.05724v2.

# A Conclusion and Limitations

## A.1 Conclusion

We present CAST, which recovers a prefix-closed candidate tree from one already-computed block and selects its width from measured verifier costs. Without sweeping over widths, the stopping rule predicts operating widths across three GPU generations. The predicted width ranges from 64 packed tokens for Qwen3-8B on Blackwell, where verification cost jumps at a kernel boundary, to 128 on H100 SXM, where it stays nearly flat. At the predicted width, the tree is faster than the standard chain in all eight settings. It also lengthens accepted rounds for a second one-pass drafter and extends to shared-budget allocation under load. Because the rule needs only offline drafter scores and a short latency probe, a deployment can recompute its width whenever the GPU or target changes.

## A.2 Limitations

**Evaluation scope.** The claim is a controlled comparison that changes only the decoder within each setting, not a cross-system ranking. Our decoding experiments use dense targets of up to 8B parameters and, in Appendix H.2, one 30B mixture-of-experts target with 3B active parameters.

**Serving and timing scope.** Our timing is the decode-only steady state defined in Appendix D and is not a serving throughput or cost claim.

**Tree and drafter scope.** The tree is optimal only inside the $K{=}8$ retained universe for the plug-in belief, and because the DFlash marginals are not path-conditioned, deeper branches stay exposed to calibration error. The free-width argument needs a one-pass drafter, since multi-step refinement would reintroduce a draft-side width cost. CAST inherits the DFlash assumptions of a trained head and the feature-cache contract and does not address DFlash training cost or block-size changes.

# B Proofs

This appendix proves the formal results of §2 and §3 in four steps.

- Appendix B.1 states the one-pass drafter interface and shows that one ancestor-masked pass reproduces separate autoregressive forwards, which gives exactness under greedy and sampled verification.
- Appendix B.2 writes expected accepted length as a sum of node masses and proves that the top-$N$ tree is optimal, together with its plug-in corollary and regret bound.
- Appendix B.3 proves the marginal stopping rule of Theorem 2.
- Appendix B.4 extends both results to a verifier budget shared by $B$ requests.

## B.1 Drafter interface and exact verification

**Assumption 1** (One-pass marginal ranking interface)**.** *Conditioned on the cached prefix $x$ and pending root token $b$, a one-pass block drafter returns one marginal proposal distribution $q_j(\cdot \mid x, b)$ for each future position $j = 1, \ldots, L$ in the same forward pass. These logits are computed before any within-block token value is sampled or committed. The tree builder therefore ranks candidate prefixes by the plug-in score of* (1)*.*

**Lemma 1** (Packed-logit equivalence)**.** *Pack the pending root $b$ and the nonroot nodes of the prefix-closed tree into one target input after the cache-backed prefix $x$, with the nodes in any ancestor-before-descendant order and each token placed at its depth-position. Run the target with an ancestor-only attention mask, under which the row of a node $v$ on path $y_{1:d}$ attends to exactly the*

*prefix $x \circ b$, the ancestor tokens $y_{1:d-1}$ on its root-to-$v$ path, and $v$ itself, each at its absolute position. Then, in exact arithmetic, the logits read off the row of $v$ are identical to those of a standalone autoregressive forward on $x \circ b \circ y_{1:d}$. One packed pass therefore reproduces what $N+1$ separate forwards over the root and nonroot nodes would compute.*

*Proof.* Under the ancestor-only mask the query at $v$'s row attends to exactly the rows of $x \circ b$, the ancestors $y_{1:d-1}$, and $v$, and each of these tokens sits at the same absolute position, hence the same positional (RoPE) phase, that it occupies in the standalone sequence $x \circ b \circ y_{1:d}$. Every attention and feed-forward sublayer therefore receives at $v$'s row the identical query and the identical attended key/value set as in the standalone forward and produces the identical hidden state layer by layer. Siblings and off-path nodes are masked and cannot influence $v$'s row. □

**Theorem 3** (Distribution preservation under sampling)**.** *Fix a temperature $T > 0$ and the baseline logit processors. At a verified node $v$ with prefix $x \circ b \circ y_{1:d}$, let $p_v$ be the target distribution at that prefix after temperature scaling and the fixed baseline logit processors, and let $q_v$ be any proposal positive on the retained siblings $\mathcal{C}_v$. The recursive speculative-sampling rule of §2 starts from the residual distribution $\mu = p_v$. At each step it draws a candidate $c$ from the available-renormalized proposal $\tilde{q}$ and accepts it with probability*

$$\min\left(1, \frac{\mu(c)}{\tilde{q}(c)}\right).$$

*On rejection it updates the residual to*

$$\mu \leftarrow \frac{(\mu - \tilde{q})_+}{\|(\mu - \tilde{q})_+\|_1}$$

*and continues, and if all retained siblings are rejected it commits a draw from the final residual. Then, in exact arithmetic, the committed token is distributed exactly as $p_v$. By the packed-logit equivalence of Lemma 1 and the tower property over visited nodes, the committed-token sequence has the same law as autoregressive sampling from the target at temperature $T$.*

*Proof.* This is recursive speculative sampling without replacement (Miao et al., 2024; Sun et al., 2023). Consider one step with residual $\mu$ and available-renormalized proposal $\tilde{q}$. The candidate $c$ is drawn with probability $\tilde{q}(c)$ and accepted with probability $\min(1, \mu(c)/\tilde{q}(c))$. The mass committed to $c$ at this step and the mass routed onward on rejection are therefore

$$\tilde{q}(c)\min\left(1, \frac{\mu(c)}{\tilde{q}(c)}\right) = \min\left(\mu(c), \tilde{q}(c)\right) \qquad \text{and} \qquad \mu - \min(\mu, \tilde{q}) = (\mu - \tilde{q})_+.$$

The normalized onward mass is exactly the conditional-on-rejection correction law. Summing the accept and reject branches telescopes the committed-token law to $\mu$, which equals $p_v$ at the first step. Induction over the without-replacement schedule preserves it, and the final-residual draw closes the recursion when all retained siblings are exhausted. By Lemma 1, the packed verify pass reproduces the logits that $N+1$ separate autoregressive forwards would compute, and $p_v$ is the true target law at each visited prefix. The tower property over committed nodes yields the sequence-level claim. □

### B.2 Optimal candidate trees

**Lemma 2** (Node-sum decomposition)**.** *For any prefix-closed $S$, under any continuation law with prefix masses $\rho$,*

$$\mathbb{E}_\rho[A(S)] = \sum_{v \in S} \rho(v).$$

*Proof.* Because $S$ is prefix-closed, $A(S) \geq k$ holds if and only if $Y_{1:k} \in S$. Summing tail probabilities then gives

$$\begin{aligned}
\mathbb{E}_\rho[A(S)] &= \sum_{k\geq 1} \Pr_\rho\left[A(S) \geq k\right] && \text{tail-sum formula} \\
&= \sum_{k\geq 1} \Pr_\rho\left[Y_{1:k} \in S\right] && \text{prefix closure of } S \\
&= \sum_{k\geq 1} \sum_{v\in S\cap\Sigma^k} \rho(v) && \text{definition of } \rho \\
&= \sum_{v\in S} \rho(v). && \square
\end{aligned}$$

*Proof of Theorem 1.* Theorem 1 states that the $N$ strings of largest mass form a prefix-closed tree and that this tree maximizes expected accepted length. The proof shows the two parts in turn.

*The top-$N$ set is prefix-closed.* For any strings $v, vw$, the event $\{Y_{1:|vw|} = vw\}$ is contained in $\{Y_{1:|v|} = v\}$, and hence $\rho(vw) \leq \rho(v)$. Order all strings in $\mathcal{U}$ by $\rho$ descending, breaking ties in favor of shorter strings and then lexicographically. If $v$ belongs to the first $N$ elements and $u$ is a prefix of $v$, then $u \in \mathcal{U}$ because $\mathcal{U}$ is prefix-closed. By monotonicity, $\rho(u) \geq \rho(v)$. If equality holds, the tie-break places $u$ before $v$. Thus every ancestor of $v$ also belongs to the first $N$ elements, and the top-$N$ set is prefix-closed.

*The top-$N$ set is optimal.* Write $\rho_{(i)}$ for the $i$-th largest mass in $\mathcal{U}$. By Lemma 2, every prefix-closed $N$-element subset $S \subseteq \mathcal{U}$ satisfies

$$\mathbb{E}_\rho[A(S)] = \sum_{v\in S} \rho(v) \;\leq\; \sum_{i=1}^{N} \rho_{(i)},$$

with equality for the first $N$ elements of the order above. The previous paragraph shows that these first $N$ elements are themselves prefix-closed, so they attain the bound and are optimal among prefix-closed candidate trees. If unrelated prefixes tie at the boundary, other optimal trees may also exist. The stated tie-break selects one prefix-closed optimum. $\square$

**Corollary 1** (DFlash plug-in tree)**.** *Under Assumption 1, setting $\rho = \hat{\pi}$ from (1) gives the exact top-$N$ tree for that plug-in objective inside the chosen feasible universe. Under a rank cap the guarantee is exact for the retained plug-in objective over the retained universe.*

*Proof.* Applying Theorem 1 with $\rho = \hat{\pi}$ gives the exact top-$N$ tree for that belief inside that universe. When the implementation first retains only top-$K$ tokens at each future position, the universe is the prefix-closed set of strings composed from those retained tokens. Omitted tokens are outside the feasible family, not covered by the exactness claim. $\square$

**Proposition 1** (Plug-in regret)**.** *Fix one round, one retained prefix universe $\mathcal{U}$, and one budget $N$. Let $\mathcal{F}_N(\mathcal{U})$ be the prefix-closed subsets of $\mathcal{U}$ with size $N$. Let $S^*$ maximize $\sum_{v\in S} \pi^\star(v)$ over $\mathcal{F}_N(\mathcal{U})$, and let $\hat{S}$ maximize $\sum_{v\in S} \hat{\pi}(v)$ over the same family. Then the gap between the retained-universe oracle tree and the plug-in tree, evaluated under the round's target prefix law, satisfies*

$$\sum_{v\in S^*} \pi^\star(v) - \sum_{v\in \hat{S}} \pi^\star(v) \;\leq\; \sum_{v\in S^* \,\triangle\, \hat{S}} \left|\pi^\star(v) - \hat{\pi}(v)\right|,$$

*a sum of prefix-belief estimation errors over the symmetric difference of the two trees.*

*Proof.* Write $\Delta(v) = \pi^\star(v) - \hat{\pi}(v)$ for the estimation error at node $v$. Adding and subtracting the plug-in scores gives

$$\begin{aligned}\sum_{v\in S^*} \pi^\star(v) - \sum_{v\in \hat{S}} \pi^\star(v) &= \underbrace{\Big(\sum_{v\in S^*} \hat{\pi}(v) - \sum_{v\in \hat{S}} \hat{\pi}(v)\Big)}_{\le 0 \text{ because } \hat{S} \text{ is } \hat{\pi}\text{-optimal}} + \sum_{v\in S^*} \Delta(v) - \sum_{v\in \hat{S}} \Delta(v) \\ &\le \sum_{v\in S^*\setminus \hat{S}} \Delta(v) - \sum_{v\in \hat{S}\setminus S^*} \Delta(v) \\ &\le \sum_{v\in S^* \triangle \hat{S}} \big|\Delta(v)\big|.\end{aligned}$$

The second line drops the nonpositive bracket and cancels the terms of $v \in S^* \cap \hat{S}$, which appear in both sums. The last line bounds each remaining term by its absolute value. □

### B.3 The marginal stopping rule

*Proof of Theorem 2.* Theorem 2 states that throughput is unimodal in the budget and that adding the next node does not reduce it if and only if the node's mass is at least throughput times the cost slope. The proof has three steps.

*Concavity of $g$.* By Theorem 1 and Lemma 2, the optimal budget-$N$ tree collects the $N$ largest masses, and therefore

$$g(N) = \sum_{i\le N} \rho_{(i)}, \qquad g(N) - g(N{-}1) = \rho_{(N)}.$$

These increments are nonincreasing, and $g$ is concave on integer budgets. Set $g(0) = 0$ and extend $g$ to $[0, \infty)$ by linear interpolation. The extension is concave and nondecreasing.

*Unimodality of throughput.* Let $\nu$ denote a continuous relaxation of the nonroot node budget. In the operating range, the theorem assumes that the non-drafter part of the round time, $h(\nu){+}\ell_{\mathrm{tgt}}(n_0{+}\nu)$, is locally affine with total slope $c_1 \ge 0$. Equivalently, after absorbing $c_{\mathrm{draft}}$, the root-token cost, and the local intercept into a constant $\ell_0 > 0$, write

$$\ell(\nu) = \ell_0 + c_1\nu.$$

Thus $\ell(\nu)$ is positive and nondecreasing, and $\ell(N{+}1) = \ell(N) + c_1$. Let $f(\nu) = (1 + g(\nu))/\ell(\nu)$ denote throughput. Because $\ell(\nu) > 0$, every level $\lambda$ satisfies

$$f(\nu) \ge \lambda \iff 1 + g(\nu) - \lambda\,\ell(\nu) \ge 0.$$

The right-hand side is a concave function of $\nu$, whose nonnegativity region is an interval. Every superlevel set of $f$ is therefore an interval, and $f$ is quasiconcave. A quasiconcave function restricted to integers is unimodal, because its superlevel sets restrict to integer intervals.

*Stopping condition.* Substituting $g(N{+}1) = g(N) + \rho_{(N+1)}$ and $\ell(N{+}1) = \ell(N) + c_1$ and multiplying both sides by the positive product $\ell(N)\,\ell(N{+}1)$ gives

$$\begin{aligned}f(N{+}1) \ge f(N) &\iff \big(1 + g(N) + \rho_{(N+1)}\big)\,\ell(N) \;\ge\; \big(1 + g(N)\big)\big(\ell(N) + c_1\big) \\ &\iff \rho_{(N+1)}\,\ell(N) \;\ge\; \big(1 + g(N)\big)\,c_1 \\ &\iff \rho_{(N+1)} \;\ge\; f(N)\,c_1.\end{aligned}$$

Adding node $N{+}1$ therefore does not reduce throughput if and only if this condition holds. By unimodality, the first budget at which it fails is a maximizer, with the smallest and largest allowed budgets as boundary cases. □

### B.4 Shared budget under load

**Proposition 2** (Shared-budget allocation is global best-first)**.** *Consider one batched round with requests $r = 1, \dots, B$, each carrying a fixed finite prefix-closed candidate universe $\mathcal{U}_r$ with prefix masses $\rho_r$ that are nonincreasing from ancestors to descendants (for the one-pass interface,*

*$\rho_r = \hat{\pi}_r$). Over prefix-closed selections $S_r \subseteq \mathcal{U}_r$ with shared budget $\sum_r |S_r| = M$, the expected total accepted length satisfies*

$$\mathbb{E}\Big[\sum_{r=1}^{B} A(S_r)\Big] = \sum_{r=1}^{B} \sum_{v \in S_r} \rho_r(v),$$

*and it is maximized by selecting the $M$ nodes of largest mass in the disjoint union $\mathcal{U} = \bigsqcup_r \mathcal{U}_r$ (ties toward prefixes). The maximizer is prefix-closed inside every request. Global best-first expansion across the batch in decreasing $\rho$ order is optimal for the shared budget.*

*Proof.* By Lemma 2 applied to request $r$, $\mathbb{E}[A(S_r)] = \sum_{v \in S_r} \rho_r(v)$ for any prefix-closed $S_r$, so by linearity $\mathbb{E}[\sum_r A(S_r)] = \sum_r \sum_{v \in S_r} \rho_r(v)$. Form the disjoint-union universe $\mathcal{U} = \bigsqcup_r \mathcal{U}_r$ with masses $\rho$ equal to $\rho_r$ on the copy of $\mathcal{U}_r$. Because the copies share no strings, a selection $S = \bigcup_r S_r$ is prefix-closed in $\mathcal{U}$ if and only if each $S_r$ is prefix-closed in $\mathcal{U}_r$, and $|S| = \sum_r |S_r| = M$. Maximizing $\sum_{v \in S} \rho(v)$ over prefix-closed $S \subseteq \mathcal{U}$ with $|S| = M$ is then exactly the setting of Theorem 1 on $\mathcal{U}$, whose masses are nonincreasing from ancestors to descendants within each component and which has no cross-component edges. Restricting the maximizer to each component yields the request-level selections. □

**Proposition 3** (Batched stopping rule under load)**.** *Let $G(M)$ be the sum of the $M$ largest masses in $\mathcal{U}$, which is concave in $M$ because its increments $\rho_{(M)}$ are nonincreasing, and let the round time be*

$$\ell(M, B) = c_{\mathrm{draft}}(B) + h(M, B) + \ell_{\mathrm{tgt}}(B,\, B + M),$$

*where $c_{\mathrm{draft}}(B)$ is the batched drafter cost, independent of $M$, $h(M, B)$ is the overhead at total budget $M$, $\ell_{\mathrm{tgt}}(B, n)$ is the target-forward latency of $B$ requests carrying $n$ packed tokens in total, and $h + \ell_{\mathrm{tgt}}$ is locally affine in $M$ with nonnegative slope $c_1(B)$. Then batch throughput $(G(M) + B)/\ell(M, B)$ is unimodal, and the global node $M{+}1$ is added while its mass satisfies*

$$\rho_{(M+1)} \;\geq\; \frac{G(M) + B}{\ell(M, B)}\, c_1(B).$$

*The statement recovers Theorem 2 at $B{=}1$.*

*Proof.* By Proposition 2 the optimal accepted mass at total budget $M$ is $G(M) = \sum_{i \leq M} \rho_{(i)}$, the sum of the $M$ largest masses of $\mathcal{U}$. Its increments $\rho_{(M)}$ are nonincreasing, so $G$ is concave. The committed batch length at budget $M$ is $G(M) + B$, since the $B$ roots are always committed. Writing $\ell(M, B) = \ell_0(B) + c_1(B)\, M$ after absorbing the width-independent drafter cost $c_{\mathrm{draft}}(B)$ and the local intercept into $\ell_0(B) > 0$, the quasiconcavity argument of Theorem 2 applies to $f(M) = (G(M) + B)/\ell(M, B)$. Replacing $1 + g(N)$ by $G(M) + B$ and $\ell(N)$ by $\ell(M, B)$ in the chain of equivalences of that proof gives

$$f(M{+}1) \geq f(M) \iff \rho_{(M+1)} \geq f(M)\, c_1(B),$$

which is the stated condition. □

## C Implementation details

This appendix details our single-device decoder.

**The round.** At round entry the emitted stream is $x \circ b$ while both caches still cover only $x$. One unchanged DFlash call on $[b, \mathrm{mask}, \ldots, \mathrm{mask}]$ returns the marginals $q_{1:L}$. The tree builder selects $N$ nonroot nodes $v_1, \ldots, v_N$, the target verifies the packed input $[b, z_1, \ldots, z_N]$, where $z_i$ is the last token of $v_i$, under an ancestor-only mask, and the walk starts at $b$ and descends to the child carrying the target-selected token while one exists. When no child matches, the target-selected token $c$ is emitted as the next pending root. After accepting the path $y_{1:A}$ the target KV cache is compacted to $x \circ b \circ y_{1:A}$, the DFlash feature cache is appended for the same committed positions, and $c$ stays outside both caches until the next round, which restores the entry state.

**Candidate construction.** The decoder retains the top $K{=}8$ tokens at each future position and runs best-first expansion in decreasing plug-in score $\hat{\pi}(v \mid x, b) = \prod_{j \le d} q_j(y_j \mid x, b)$ of each node $v = y_{1:d}$ over that retained universe. A min-heap keyed by negative cumulative log-probability is seeded with the root's top-$K$ children, and each pop emits one node (token, depth, parent, rank, path score) and pushes its top-$K$ children at the next depth while depth stays below $L$. Scores use the original $q_j$, not a renormalized top-$K$ distribution, and the children of a depth-$d$ prefix come from the shared marginal $q_{d+1}$. The parent changes only the cumulative product. Round preprocessing is one top-$K$ over the $L \times |\Sigma|$ logits moved once to host memory, and the heap stage performs at most $K{+}NK$ pushes and $N$ pops, all inside the timer.

**Packed verification.** Let $P = |x|$ be the number of cached prefix tokens. The packed input $[b, z_1, \ldots, z_N]$ has position id $P$ for $b$ and $P{+}\mathrm{depth}(v_i)$ for $z_i$. RoPE positions repeat across siblings and depth 1 aligns with $q_1$. An additive mask of shape $(1, 1, 1{+}N,\ P{+}1{+}N)$ grants every packed token visibility of the $P$ prefix entries and of its ancestor chain inside the pack, including its own row. The ancestor matrix is built by one vectorized pass over parent pointers on the host and copied to the device once each round, because both the tree topology and the prefix length change from round to round.

**KV gather and feature cache.** After the walk commits a path of length $A$, the target cache holds $P{+}1{+}N$ packed entries. The prefix plus the $A{+}1$ packed indices for the previous root and the accepted path are gathered in order for every layer. RoPE phases are already baked into stored keys at their depth-positions, which equal the tokens' final absolute positions. The DFlash feature cache is appended with the target features of exactly the committed positions, mirroring standard DFlash.

The timer window shared by AR, DFlash, and CAST is defined in Appendix D.

**Software.** AR, DFlash, and CAST run in bf16 and share one software environment within each setting.

**Stochastic verifier.** A simpler direct sampled walk, which replaces $\arg\max$ by a single sample from the verifier distribution at each visited node, is also distribution-preserving but accepts one candidate for each node. Its bf16 total variation from autoregressive sampling is $0.004$, the same as that of the proposal-aware verifier.

## D Measurement protocol and cross-hardware results

**Prompt sets.** GSM8K uses the first 40 prompts of the public test split and HumanEval the first 40 problems. MT-Bench uses first turns only from the first 40 prompts of the public prompt set, which cover four of the eight categories (writing, roleplay, reasoning, and math). The H100 SXM panels use 40 prompts by domain and add MATH-500 and MBPP.

**Timing window.** Batch-one decoding rows use matched decode-only steady-state timing for AR, standard DFlash, and CAST. For AR the timer opens after prefill, just before the first decoded token. For DFlash and CAST it opens after prefill and after the first draft construction, just before the first target verification, and it closes after output truncation, with CUDA synchronization before both reads. Tokenizer work and prefill are excluded for all systems, and the first draft construction is excluded for the two speculative decoders. Every later DFlash call, target verification, tree and mask construction, packing, KV gather, and feature-cache update is included. The denominator is the emitted new-token count after the end-of-sequence token or truncation. Prompt order is fixed within a run. In Tables 7, 8, and 9, each Blackwell value comes from the run with the median gain among three runs. The exceptions are Qwen3-8B at $N{=}127$, measured once, and Qwen3-4B at $N{=}31$, whose gain averages two runs. A6000 rows are single runs.

**Measurement isolation.** All headline rows are measured on otherwise idle GPUs. Co-tenant GPU load inflates the relative gain, because the memory-bound one-chain baseline degrades faster under shared-memory pressure than the compute-heavier wider-tree verify. A Blackwell-8B GSM8K run on a shared GPU measures about $+21\%$, against $+16\%$ on an idle GPU. Idle-GPU timing is therefore the conservative operating point.

**Verify-cost probe.** Table 5 is the component probe behind the cost law of §3.1. A transient puts the A100 and H100 NVL $n$=32 cells high, recovering by $n$=48. Blackwell-8B then jumps to 30.3 ms at $n$=128, a kernel or tile boundary that the 4B target does not hit in range. On the H100 SXM node the cost rises by $+6.7\%$ (8B) and $+7.4\%$ (4B) from $n$=16 to $n$=128 with no jump, a slope of 0.018 and 0.017 ms for each added packed token, which sends the rule to the grid's end.

The 4B target is far cheaper than 8B ($c_1^{\mathrm{fw}} \approx 0.015$ against 0.056 ms for each added packed token) with no cliff, so the rule predicts a wider tree for the smaller target on identical hardware, while the A6000 and the H100 SXM node (0.018 ms for 8B) bracket the flat extreme, where forward cost alone never caps width in range.

Table 5: Target-forward latency (ms) by packed-token count $n=N+1$, using bf16 and a 1k-token KV prefix. Each probe uses random packed tokens, eight warmups, and 30 timed iterations. The H100 SXM rows use scaled dot-product attention.

| GPU, target | $n$=1 | 16 | 24 | 32 | 48 | 64 | 96 | 128 |
|---|---|---|---|---|---|---|---|---|
| Blackwell, Qwen3-8B | 17.2 | 21.9 | 23.3 | 23.6 | 24.1 | 24.6 | 24.4 | 30.3 |
| A6000, Qwen3-8B | 117.0 | 128.1 | 128.8 | 131.5 | 129.6 | 129.6 | 133.2 | 129.1 |
| A100, Qwen3-8B | 30.2 | 34.4 | 33.2 | 39.4 | 34.9 | 35.3 | 35.9 | 35.6 |
| H100 NVL, Qwen3-8B | 14.7 | 17.9 | 17.8 | 20.8 | 18.2 | 18.4 | 19.2 | 19.2 |
| Blackwell, Qwen3-4B | 16.9 | 18.7 | 19.6 | 19.5 | 19.4 | 19.4 | 20.9 | 20.4 |
| H100 SXM, Qwen3-8B | 24.7 | 30.6 | 30.2 | 30.6 | 31.2 | 31.5 | 32.2 | 32.6 |
| H100 SXM, Qwen3-4B | 23.9 | 25.3 | 25.1 | 25.3 | 25.7 | 25.9 | 26.6 | 27.1 |

**Prefix length.** Table 6 gives the H100 SXM probe at 1k, 4k, 8k, and 16k tokens of KV prefix, measured in one session separate from Table 5, together with the LLaMA-3.1-8B cell of Appendix H.1. At the 1k prefix both sessions show a nearly flat cost, with a Qwen3-8B slope of $-0.016$ ms for each added packed token in Table 6 and 0.018 from Table 5. The slope is at most 0.029 ms at every prefix length, and the rule stays at $N^*$=127 throughout.

Table 6: H100 SXM target-forward latency (ms) across KV-prefix lengths at batch one in bf16. Slope is the latency increase in ms for each added packed token from $n$=16 to 128.

| Prefix | $n$=1 | 16 | 32 | 64 | 128 | Slope |
|---|---|---|---|---|---|---|
| *Qwen3-8B* | | | | | | |
| 1k | 23.6 | 26.6 | 25.1 | 25.2 | 24.8 | -0.016 |
| 4k | 22.7 | 25.5 | 25.4 | 26.1 | 26.6 | 0.010 |
| 8k | 22.9 | 35.9 | 36.2 | 37.5 | 37.9 | 0.018 |
| 16k | 23.7 | 56.7 | 58.0 | 59.9 | 60.0 | 0.029 |
| *Qwen3-4B* | | | | | | |
| 1k | 23.2 | 24.7 | 25.3 | 24.8 | 24.5 | -0.002 |
| 4k | 23.3 | 24.6 | 25.1 | 25.1 | 24.8 | 0.001 |
| 8k | 22.5 | 32.2 | 33.1 | 34.6 | 34.9 | 0.024 |
| 16k | 22.4 | 54.4 | 55.6 | 57.4 | 57.5 | 0.027 |
| *LLaMA-3.1-8B* | | | | | | |
| 1k | 16.7 | 17.8 | 18.0 | 18.4 | 17.9 | 0.001 |

**Cross-hardware cells.** Table 7 gives the rule-predicted cells, and Table 8 gives the full gain sweep. The A6000 $N$=127 cells (‡) are measured on a second A6000 host. On that host, the mean gain at $N$=127 is $+25.1\%$ for 8B and $+31.7\%$ for 4B. Both lie below the same host's $N$=95 means ($+26.6\%$ and $+32.6\%$), so the flat A6000 curve does not reward the last grid step and the predicted $N^*$=95 remains the A6000 oracle in Table 2.

**Same packed length.** Table 9 isolates tree shape from width. The $N$=15 tree sends exactly 16 packed tokens, the same count standard DFlash verifies, and the only change is shape. Across all twelve cells this lifts $R$ by $+0.69$ to $+1.17$ and runs 5.9 to 29.2% faster before any wider tree is allowed. These fixed-budget cells are measured directly, without sweep selection. Width beyond

Table 7: Decoding on Blackwell and A6000 for Qwen3-8B (left) and Qwen3-4B (right). Cells report decode-only ms/token, gain over DFlash at $N^*$, and committed round length $R$.

| | Decode time (ms/token) | | | | |
|---|---|---|---|---|---|
| Domain | AR | DFlash | CAST | Gain | $R$ |
| *Blackwell 8B, $N^*$=63* | | | | | |
| GSM8K | 18.0 | 4.25 | **3.65** | **+16.4%** | 7.97 |
| MT-Bench | 18.0 | 8.23 | **6.56** | **+25.4%** | 4.40 |
| HumanEval | 18.0 | 4.06 | **3.45** | **+17.6%** | 8.51 |
| *A6000 8B, $N^*$=95* | | | | | |
| GSM8K | 119.8 | 26.95 | **21.74** | **+24.0%** | 8.17 |
| MT-Bench | 122.2 | 53.38 | **39.82** | **+34.1%** | 4.55 |
| HumanEval | 123.1 | 26.46 | **20.67** | **+28.0%** | 8.67 |

| | Decode time (ms/token) | | | | |
|---|---|---|---|---|---|
| Domain | AR | DFlash | CAST | Gain | $R$ |
| *Blackwell 4B, $N^*$=95* | | | | | |
| GSM8K | 19.3 | 4.59 | **3.70** | **+24.3%** | 8.13 |
| MT-Bench | 17.9 | 8.17 | **6.10** | **+33.9%** | 4.66 |
| HumanEval | 16.7 | 3.85 | **3.11** | **+23.6%** | 8.48 |
| *A6000 4B, $N^*$=95* | | | | | |
| GSM8K | 120.1 | 27.46 | **21.39** | **+28.4%** | 8.12 |
| MT-Bench | 122.9 | 54.52 | **39.18** | **+39.1%** | 4.70 |
| HumanEval | 134.8 | 27.21 | **20.86** | **+30.4%** | 8.58 |

Table 8: Greedy width-sweep gains over DFlash across GPUs and targets. Bold marks each row's oracle. Underline marks the predicted width. ‡ marks cells measured on a second A6000 host.

| GPU | target | domain | $N$=15 | $N$=31 | $N$=47 | $N$=63 | $N$=95 | $N$=127 |
|---|---|---|---|---|---|---|---|---|
| Blackwell | Qwen3-8B | GSM8K | +5.9% | +13.1% | +15.5% | +16.4% | **+17.8%** | −0.8% |
| Blackwell | Qwen3-8B | MT-Bench | +19.1% | +21.8% | +24.8% | +25.4% | **+26.9%** | +7.5% |
| Blackwell | Qwen3-8B | HumanEval | +8.2% | +12.5% | +17.3% | +17.6% | **+19.7%** | +0.5% |
| Blackwell | Qwen3-4B | GSM8K | +12.2% | +17.9% | +22.8% | +22.7% | +24.3% | **+27.1%** |
| Blackwell | Qwen3-4B | MT-Bench | +29.2% | +31.6% | +34.2% | +34.5% | +33.9% | **+37.9%** |
| Blackwell | Qwen3-4B | HumanEval | +12.3% | +14.8% | +21.8% | +19.9% | **+23.6%** | +22.0% |
| A6000 | Qwen3-8B | GSM8K | +12.4% | +19.1% | +23.3% | **+24.1%** | +24.0% | +22.0%‡ |
| A6000 | Qwen3-8B | MT-Bench | +22.8% | +31.2% | +33.1% | **+34.7%** | +34.1% | +29.4%‡ |
| A6000 | Qwen3-8B | HumanEval | +16.5% | +23.9% | +26.3% | +26.5% | **+28.0%** | +23.8%‡ |
| A6000 | Qwen3-4B | GSM8K | +12.7% | +22.8% | +25.5% | **+28.5%** | +28.4% | +27.2%‡ |
| A6000 | Qwen3-4B | MT-Bench | +17.6% | +32.7% | +33.9% | **+39.2%** | +39.1% | +38.7%‡ |
| A6000 | Qwen3-4B | HumanEval | +15.6% | +6.3% | +28.5% | +28.0% | **+30.4%** | +29.1%‡ |

that, up to the rule-predicted $N^*$ of Table 7, adds a mean of 9.0 percentage points of gain, ranging from 4.7 to 12.1 points across the six Blackwell cells.

Table 9: Tree-shape ablation at 16 packed tokens, comparing standard DFlash with CAST at $N$=15. Cells report committed round length $R$, decode-only ms/token, and gain over DFlash on 40 prompts in each domain.

| GPU | target | domain | DFlash $R$ | $N$=15 $R$ | DFlash ms | $N$=15 ms | shape gain |
|---|---|---|---|---|---|---|---|
| Blackwell | Qwen3-8B | GSM8K | 6.02 | 6.71 | 4.40 | 4.16 | **+5.9%** |
| Blackwell | Qwen3-8B | MT-Bench | 3.13 | 3.90 | 8.40 | 7.05 | **+19.1%** |
| Blackwell | Qwen3-8B | HumanEval | 6.36 | 7.26 | 4.11 | 3.80 | **+8.2%** |
| Blackwell | Qwen3-4B | GSM8K | 5.93 | 6.85 | 4.25 | 3.79 | **+12.2%** |
| Blackwell | Qwen3-4B | MT-Bench | 3.11 | 4.00 | 8.77 | 6.79 | **+29.2%** |
| Blackwell | Qwen3-4B | HumanEval | 6.15 | 7.19 | 4.10 | 3.65 | **+12.3%** |
| A6000 | Qwen3-8B | GSM8K | 6.04 | 6.89 | 26.95 | 23.98 | **+12.4%** |
| A6000 | Qwen3-8B | MT-Bench | 3.12 | 3.91 | 53.38 | 43.45 | **+22.8%** |
| A6000 | Qwen3-8B | HumanEval | 6.28 | 7.45 | 26.46 | 22.72 | **+16.5%** |
| A6000 | Qwen3-4B | GSM8K | 5.94 | 6.74 | 27.46 | 24.36 | **+12.7%** |
| A6000 | Qwen3-4B | MT-Bench | 3.13 | 3.93 | 54.52 | 46.38 | **+17.6%** |
| A6000 | Qwen3-4B | HumanEval | 6.22 | 7.22 | 27.21 | 23.54 | **+15.6%** |

**Sampled decoding on Blackwell.** Table 10 gives the three-seed $T$=1 rows of the proposal-aware stochastic verifier on Blackwell-8B at $N^*$=63. The gain over the chain is +16.7% to +30.1% with seed ranges of a few points.

Table 10: Stochastic verification for Qwen3-8B on Blackwell at $T$=1 and $N^*$=63. Cells report committed round length $R$, decode-only ms/token, mean gain across seeds 0–2, and the seed-level gain range.

| domain | DFlash $R$ | CAST $R$ | DFlash ms/tok | CAST ms/tok | mean gain | seed range |
|---|---|---|---|---|---|---|
| GSM8K | 5.70 | 7.54 | 4.88 | 4.18 | **+16.7%** | +16.0 to +18.0% |
| MT-Bench | 3.69 | 5.10 | 9.43 | 7.25 | **+30.1%** | +28.7 to +32.3% |
| HumanEval | 5.66 | 7.72 | 5.11 | 4.18 | **+22.3%** | +20.3 to +25.3% |

Table 11: Verify-forward latency (ms) across packed-token counts for Qwen3.6 hybrid targets on Blackwell, bf16, batch one. Values are medians of 15 timed forwards. Each receives an untimed copy of the prefix cache.

| Target | Prefix | $n$=1 | 2 | 16 | 64 | 128 | 256 | 1024 |
|---|---|---|---|---|---|---|---|---|
| 27B dense hybrid | 1k | 46.9 | 67.9 | 69.1 | 68.5 | 68.6 | 74.6 | 194.7 |
| 27B dense hybrid | 8k | 47.3 | 68.5 | 70.0 | 78.2 | 80.2 | 93.9 | 237.2 |
| 35B MoE hybrid | 1k | 87.3 | 109.7 | 129.8 | 158.4 | 175.4 | 194.7 | 229.3 |
| 35B MoE hybrid | 8k | 88.2 | 111.1 | 133.9 | 163.7 | 179.8 | 198.4 | 246.2 |

### D.1 Hybrid linear-attention targets

We repeat the component probe on two public hybrid checkpoints, Qwen3.6-27B (dense, three linear-attention layers for every full-attention layer) and Qwen3.6-35B-A3B (the same interleaving with 256-expert sparse mixture-of-experts (MoE) blocks, 8 active), on one Blackwell card with fused gated-delta-rule kernels. Linear-attention layers mutate their recurrent state in place on every forward. Each timed forward therefore receives an untimed copy of the prefix cache, and we report the median of 15 timed forwards after 5 warmups.

**Two regimes and their consequence.** Table 11 shows a two-regime curve on both targets. A single-token pass rides the fused recurrent decode kernel and is context-flat. Any multi-token pass switches to the chunked kernel and pays a fixed entry of about +21 ms (45%) on the 27B target and +22 to 23 ms (26%) on the MoE target, after which the curve is locally affine again. On the dense hybrid at the 1k prefix the latency stays within 5% across $n \in [2, 128]$, with a slope of about zero over $n \in [16, 128]$, against about 0.1 ms for each added token at 8k. At the 1k prefix a 63-node tree therefore verifies at the cost of the 16-token chain to within noise on the dense hybrid, while the MoE hybrid keeps a prefix-independent slope of 0.39 to 0.40 ms for each added token and growth resumes by $n \approx 256$ on the dense hybrid. Concurrent engine work already provides tree verification for gated linear-attention layers (Oda et al., 2026; Wang et al., 2026).

### D.2 Statistics and output equivalence

**Bootstrap confidence intervals.** To bound prompt-selection variance, the Blackwell cells are measured again at the rule-predicted widths on expanded prompt sets of 200 GSM8K, 80 MT-Bench, and 164 HumanEval prompts. Each prompt is timed three times, and the prompt means are resampled with a paired prompt-level bootstrap (10,000 resamples). Table 3 reports the mean prompt-level gain over standard DFlash with its 95% confidence interval. Every interval lies well above zero.

**Run-to-run spread.** Across the three runs behind each Blackwell-8B cell at $N^*$=63, the gain varies by at most 2.3 points in every domain.

**Greedy equivalence.** In fp32, CAST and standard DFlash both reproduce AR decoding token for token in every evaluated domain and budget on Qwen3-8B and Qwen3-4B. In bf16, over 600 prompts across budgets, both decoders depart from AR only at near-ties of the top two target logits, where shape-dependent bf16 rounding can flip the greedy token.

**Sampling distribution.** For the $T{=}1$ rows, the tokens committed by the proposal-aware stochastic verifier are consistent with the target sampling law. In fp64 their total variation from autoregressive sampling is $0.003$, below the $0.013$ Monte-Carlo floor. In bf16 it is $0.004$, matching the direct sampled walk of Appendix C. Appendix F reports the same comparison under serving load.

## E H100 SXM PANELS IN FULL

**Evaluation protocol.** The H100 SXM panels use 40 prompts from each of GSM8K (Cobbe et al., 2021), MT-Bench (Zheng et al., 2023), HumanEval (Chen et al., 2021), MATH-500 (Lightman et al., 2024), and MBPP (Austin et al., 2021), with 256 new tokens (512 for MATH-500 and MBPP) and the decode-only timer of Appendix D.

**Width sweep.** Table 12 gives the full ms/token sweep behind Table 1. Two further repetitions of the whole panel reproduce every chain-to-CAST gain within 3.7 points, and no cell shows the cliff reversal of the Blackwell-8B sweep.

Table 12: Qwen3 greedy width sweeps on H100 SXM, using the public block-16 DFlash head of each target. Cells report mean decode-only ms/token. $R$ denotes the committed round length of DFlash and of the widest CAST tree.

| Domain | max new | AR | chain | $N{=}47$ | $N{=}63$ | $N{=}95$ | $N{=}127$ | $R$ chain | $R$ at 127 |
|---|---|---|---|---|---|---|---|---|---|
| *Qwen3-8B* | | | | | | | | | |
| GSM8K | 256 | 24.85 | 5.48 | 4.50 | 4.40 | 4.35 | **4.28** | 6.23 | 8.46 |
| MT-Bench | 256 | 24.79 | 10.75 | 8.04 | 7.86 | **7.83** | 7.85 | 4.06 | 5.52 |
| HumanEval | 256 | 24.77 | 5.31 | 4.26 | 4.12 | 4.06 | **4.02** | 6.52 | 9.04 |
| MATH-500 | 512 | 24.84 | 4.46 | 3.82 | 3.85 | 3.70 | **3.69** | 7.81 | 9.87 |
| MBPP | 512 | 25.39 | 5.27 | 4.22 | 4.14 | 4.08 | **4.06** | 6.56 | 9.09 |
| *Qwen3-4B* | | | | | | | | | |
| GSM8K | 256 | 23.87 | 5.42 | 4.33 | 4.26 | 4.15 | **4.13** | 6.13 | 8.47 |
| MT-Bench | 256 | 24.72 | 10.75 | 7.87 | 7.77 | 7.62 | **7.50** | 3.95 | 5.78 |
| HumanEval | 256 | 23.88 | 5.18 | 4.16 | 4.11 | 4.01 | **3.95** | 6.49 | 9.02 |
| MATH-500 | 512 | 25.21 | 4.72 | 4.02 | 3.93 | 3.91 | **3.81** | 7.49 | 9.76 |
| MBPP | 512 | 24.76 | 4.99 | 4.04 | 4.03 | 3.94 | **3.90** | 6.72 | 9.11 |

At $T{=}1$, the tree leads the chain by $+22\%$ to $+37\%$ across the ten cells of Table 1, and each seed's mean tree decode time is within $3\%$ of the three-seed cell mean.

**EAGLE-3 settings.** The EAGLE-3 rows of Table 1 run the official EAGLE code on the same node, prompts, and decode-only timer as the other rows, with the DFlash paper's tree settings (a draft tree of 16 tokens to match the block and of 60 as in the EAGLE-3 paper, seven draft steps, top-$k$ 10). Two public Qwen3-8B heads exist. The body rows use the stronger Tengyunw head, and Table 13 adds the AngelSlim head the DFlash paper reports against. The 4B rows use the AngelSlim Qwen3-4B head, the only public one.

Table 13: EAGLE-3 head comparison for Qwen3-8B greedy decoding on H100 SXM. Cells report decode-only ms/token and committed round length $R$ for 16- and 60-token draft trees. AR anchors are in ms/token.

| Head | Tree | GSM8K | | MATH-500 | | HumanEval | | MBPP | | MT-Bench | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | | ms/tok | $R$ | ms/tok | $R$ | ms/tok | $R$ | ms/tok | $R$ | ms/tok | $R$ |
| Tengyunw | 16 | 9.65 | 4.45 | 10.47 | 4.11 | 11.13 | 3.97 | 10.72 | 4.11 | 12.00 | 3.74 |
| Tengyunw | 60 | **8.07** | **5.34** | **8.91** | **4.88** | **9.21** | **4.76** | **9.06** | **4.93** | **10.28** | **4.49** |
| AngelSlim | 16 | 12.97 | 3.26 | 13.35 | 3.11 | 13.29 | 3.18 | 13.82 | 3.10 | 16.33 | 2.64 |
| AngelSlim | 60 | 11.23 | 3.78 | 11.65 | 3.59 | 11.59 | 3.65 | 11.99 | 3.58 | 14.22 | 3.05 |
| AR anchor | | 24.85 | | 24.84 | | 24.77 | | 25.39 | | 24.79 | |

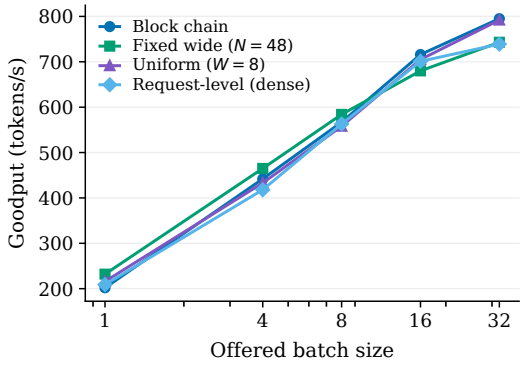


Figure 7: Qwen3-8B serving goodput on Blackwell, medians of four runs.

**Generation length and long prefix.** Table 14 reports MATH-500 at 512 and 2048 new tokens and GovReport summarization prompts cut to 8k tokens. The 8k row of Table 6 gives $N^*$=127, and on both targets the gain stays within 4 points from $N$=47 to 127, with the best width at 95. At 2048 new tokens the sweep covers $N$=63 and $N$=127.

Table 14: Generation length and KV prefix on H100 SXM at $T$=0 over 40 prompts, with CAST at $N^*$=127. The long-prefix rows generate 256 new tokens. Cells report speedup over AR, committed round length $R$, gain over DFlash, and the sweep oracle.

| Cell | AR ms/tok | DFlash ×AR ($R$) | CAST ×AR ($R$) | Gain | Oracle $N$ (gain) |
|---|---|---|---|---|---|
| *Generation length, Qwen3-8B, MATH-500* | | | | | |
| 512 new tokens | 24.8 | 5.57 (7.81) | **6.74** (9.87) | +21% | 127 (+21%) |
| 2048 new tokens | 24.0 | 5.82 (8.09) | **6.99** (10.20) | +20% | 127 (+20%) |
| *Long prefix, GovReport prompts cut to 8k tokens* | | | | | |
| Qwen3-8B | 25.1 | 1.39 (2.24) | **1.77** (3.43) | +28% | 95 (+30%) |
| Qwen3-4B | 24.6 | 1.43 (2.18) | **1.83** (3.37) | +27% | 95 (+30%) |

## F SERVING HARNESS

The batched harness extends the single-device decoder to $B$ concurrent requests. One batched DFlash pass returns every position marginal for all requests, an allocator assigns the shared verifier budget, one batched ancestor-masked target pass verifies all packs, and a ragged left-padded target cache is compacted each round. It is a dense prototype whose verify pack is right-padded to $1 + \max_r N_r$ tokens for request widths $N_r$, which charges every request the maximum allocation.

**Greedy equivalence under batching.** We compare every sequence committed under each allocation policy with AR greedy decoding. At $B$=4 over 18 prompts, every policy departs from it only at bf16 near-ties, where the top two target logits differ by at most 0.25, and the plain DFlash baseline itself departs at 11 such ties.

**Block shrink against width shrink.** Table 15 separates drafter length from verifier width at matched packed budgets, with a 256-new-token cap. At 8 packed tokens, a 7-node tree drawn from the block-16 drafter is the fastest configuration in every domain, ahead of the block-8 chain and of the same-size tree from a block-8 drafter. Shrinking the tree therefore costs less than shrinking the drafter block.

**Cost law under load.** The drafter forward is independent of width and grows only with batch, from 3.2 ms at $B$=1 to 19.3 ms at $B$=32. In Figure 6a, the verifier-forward cost at a width of $W$=64 for each request over $W$=1 grows from +10% at $B$=1 to +79% at $B$=8 and +237% at $B$=32.

Table 15: Block-length and tree-width ablation at matched packed budgets on H100 SXM, Qwen3-8B, $T=0$. Cells show decode-only ms/token with committed round length in parentheses.

| | 8 packed tokens | | | 16 packed tokens | |
|---|---|---|---|---|---|
| Domain | chain, block 8 | tree $N=7$, block 8 | tree $N=7$, block 16 | chain, block 16 | tree $N=15$, block 16 |
| GSM8K | 6.84 (4.86) | 6.51 (5.19) | **6.31 (5.37)** | 5.47 (6.23) | **4.96 (7.02)** |
| MT-Bench | 11.77 (3.28) | 10.29 (3.66) | **10.07 (3.80)** | 10.82 (4.06) | **8.87 (4.68)** |
| HumanEval | 6.74 (4.96) | 6.35 (5.32) | **6.33 (5.40)** | 5.37 (6.52) | **4.76 (7.48)** |

**Goodput and acceptance in the dense prototype.** Figure 7 gives goodput for four policies, the block chain, a fixed wide tree with $N=48$, a uniform width of $W=8$, and dense request-level allocation. Fixed wide leads at $B\leq 8$ and collapses at $B\geq 16$, where the plain chain is best. Request-level allocation lifts mean accepted round length by $5.3$ to $9.9\%$ over a uniform split, but the dense pad-to-max charges that width to every request, so its goodput sits at or below uniform. The padding tax, not the allocation, is what the varlen verifier below removes.

**Allocation oracle and the varlen verifier.** Under ragged packing the verify cost scales with the total node count. At a fixed shared budget goodput is proportional to total accepted tokens. An offline oracle over the measured request-level acceptance (900 requests by domain, 400 random batches) shows online global best-first raising total accepted tokens over a uniform split by $+16.8\%/+18.2\%/+22.5\%$ on GSM8K/MT-Bench/HumanEval at a scarce shared budget of four nodes for each of $B=32$ requests, within a few points of hindsight water-filling, and by only $1$ to $2\%$ once the budget is abundant. Bucketing requests by node count does not realize this, since each extra pass re-streams the 8B weights and goodput falls by up to $45\%$, and within one dense pass the padding rows still cost $9$ to $15\%$ of the verify forward.

We therefore implement a single-pass varlen verifier as a forward-only tree-attention kernel in Triton (cumulative-sequence-length ragged layout, query-level ancestor bitmask, grouped-query attention, bf16). Its logits agree with the dense verify to fp32 tolerance, and decoding with it also departs from AR greedy only at bf16 near-ties. It lifts committed-token goodput over uniform width by a median $+8.6\%$ at $B=16$ and $+13.9\%$ at $B=32$ (four runs, ranges $[+8.1, +14.6]\%$ and $[+13.2, +19.8]\%$), where the dense request-level verify reaches $-0.5\%$ and $-6.7\%$. Over the batch-wise best static policy, which is the chain at both batch sizes, it lifts goodput by a median $+7.0\%$ at $B=16$ and $+13.7\%$ at $B=32$, with a gain in every run. Best-first allocation, which Proposition 2 proves optimal for a shared budget, adds 9–$10\%$ committed acceptance at $B=8$–$16$.

**Continuous batching and sampling under load.** A minimal continuous-batching loop (Poisson arrivals, dynamic admission and retirement, both policies on the same varlen store and verify so only the allocator differs) separates the allocation from the padding-tax removal. At sustained occupancy (200 requests) request-level best-first beats uniform width by $0.9$ to $3.6\%$ across active batches of $8$ to $32$, and the loop's outputs again depart from AR greedy only at bf16 near-ties. Under the same scheduler at $T=1$ each request commits through the residual recursion of Theorem 3, and on an H100 the served committed tokens are consistent with autoregressive sampling (fp64 total variation $0.006$, below the $0.013$ Monte-Carlo floor, its maximum over served prefixes decaying as the inverse square root of the draw count).

## G Extended analysis

The first four analyses below replay the offline Qwen3-8B greedy logs from the same 40-prompt slices as the main rows, following the budget-sweep and acceptance-length analyses of EAGLE-2 and OPT-Tree (Li et al., 2024b; Wang et al., 2025).

**Adaptive against fixed-shape trees.** At fixed nonroot budget $N=47$, two fixed-shape trees do not beat the standard chain. The fixed schedule $(8, 4, 2, 2, 1, \ldots)$ reaches $R = 3.7/3.0/3.7$ and uniform binary branching $4.9/3.2/4.8$, against the chain's $6.3/3.2/6.3$ on GSM8K/MT-Bench/HumanEval. The adaptive top-$N$ path-product tree reaches $7.7/4.3/7.7$.

**Where rejections go.** For each first rejection of standard DFlash, Figure 4a ranks the target greedy correction in the DFlash marginal at the rejected offset. The correction has rank 2–4 in 56 to 77% of first-rejection events (GSM8K and HumanEval 77%, MT-Bench 56%), ranks 5–8 in another 11 to 14%, and lies outside the top 32 in only 2 to 11%.

**Calibration.** A hit means that the candidate node token equals the target greedy token at that node's prefix. At depth 1, the binned reliability in Figure 4b is close to the diagonal. In the $[0.9, 1.0]$ bin the empirical hit rate is 0.98 to 0.99, and mid-range bins are mildly underconfident ($[0.6, 0.7] \rightarrow$ 0.66 to 0.77).

**Stopping values.** For GSM8K on Blackwell, the threshold of the rule in §3.3 is about 0.015. The sorted candidate masses fall from 0.86 at the first candidate to 0.016 at the 48th and 0.011 at the 64th, so the rule predicts the optimum between $N$=47 and $N$=63, the flat region of Table 8.

### G.1 Rank-cap ablation

The public decoder of §3 retains the top $K$=8 tokens at each future position before the best-first expansion, while the fixed-budget problem of Theorem 1 admits the full budget-tied rank set. On the saved greedy round logs (1914/3090/2150 rounds on GSM8K/MT-Bench/HumanEval, Qwen3-8B target, public DFlash-b16 head) we replay both rules at matched node budget and score committed accepted length $R$ against the realized continuation. At budget 32 the $K$=8 tree comes within 0.5% of the full-rank tree's committed accepted length in every domain, because the realized target token sits beyond rank 8 only 9.5 to 15.2% of the time. At the deployed width $N^*$=63 on a dedicated top-64 log the gap stays within 1.1%. The cap is near-free at the operating point while scoring 8 candidates for each position instead of the full budget-tied rank set.

### G.2 Width transfer to DART

To test whether the width mechanism is specific to DFlash, we repeat the offline acceptance replay on DART (Liu et al., 2026a), an architecturally distinct one-pass drafter consisting of a single customized Transformer decoder layer that emits position-level logits for $L$ future positions in one forward. We use the public `fvliang/qwen8b-dart` head for the same Qwen3-8B target, log the top 32 tokens of its one-pass logits at each position over a greedy decode (mapped to the target vocabulary through its draft-vocabulary table), and score committed accepted length $R$ of a best-first tree with rank cap $K$=8 against the realized continuation. Over 1900 replayed rounds for each benchmark, one at each position of 20 greedy decodes, $R$ grows from 1.66 at budget 1 to 2.81 at budget 63 on GSM8K (+70%) and from 1.76 to 3.15 on HumanEval (+79%). The top-1 chain over the same eight positions reaches $R$ = 2.01 and 2.35, which the tree exceeds by 19.1% and 14.5% at the same eight nodes and by 39.9% and 34.2% at budget 63. DART's $L$=8 caps $R$ at 9, against 16 for the DFlash block, but the trend is the same. We also time DART's own tree decoder, which adds its small n-gram trie and runtime tree search, against autoregressive decoding of the same target on an A6000, with 30 prompts in each domain and 256 new tokens. At node budgets of 8 and 60, the tree decoder is 1.91× and 2.14× faster than autoregressive decoding on GSM8K and 2.02× and 2.34× faster on HumanEval. Across budgets of 8, 16, 32, and 60, its committed round length rises at every step, from 2.30 to 2.65 on GSM8K and from 2.49 to 2.92 on HumanEval.

## H Additional target families

### H.1 LLaMA-3.1-8B, block 10

Table 16 repeats the main protocol on LLaMA-3.1-8B-Instruct with the public UltraChat-trained block-10 DFlash head and official EAGLE-3 head, using H100 SXM, bf16, batch one, 40 prompts in each of five domains, and 256 new tokens (512 for MATH-500 and MBPP). The flat cost slope of 0.001 ms for each added packed token selects $N^*$=127. The tree beats the chain at every width and domain, with sweeps within 2.1% from 63 to 127. On GSM8K, EAGLE-3 at tree 60 matches the accepted length of CAST, but uses eight draft passes rather than one. In decode time, CAST at $N^*$=127 is 30 to 60% faster than the 60-token EAGLE-3 tree in every domain.

Table 16: LLaMA-3.1-8B greedy decoding with the block-10 DFlash head on H100 SXM. Cells report speedup over AR and committed round length $R$.

| Method (width) | GSM8K | | MATH-500 | | HumanEval | | MBPP | | MT-Bench | | Avg. | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | ×AR | $R$ | ×AR | $R$ | ×AR | $R$ | ×AR | $R$ | ×AR | $R$ | ×AR | $R$ |
| AR ms/token | 17.30 | | 19.67 | | 17.12 | | 19.89 | | 16.87 | | | |
| *Autoregressive drafter* | | | | | | | | | | | | |
| EAGLE-3 (10) | 2.05 | 4.60 | 1.75 | 4.05 | 2.28 | 5.15 | 2.25 | 5.11 | 1.73 | 4.14 | 2.01 | 4.61 |
| EAGLE-3 (60) | 2.67 | 6.08 | 2.31 | 5.28 | 2.80 | 6.56 | 2.81 | 6.40 | 2.34 | 5.59 | 2.59 | 5.98 |
| *One-pass block drafter* | | | | | | | | | | | | |
| DFlash (10) | 3.00 | 4.31 | 2.76 | 4.01 | 3.60 | 5.18 | 3.56 | 5.09 | 2.32 | 3.51 | 3.05 | 4.42 |
| CAST ($N$=15) | 3.41 | 5.11 | 3.23 | 4.83 | 4.01 | 6.02 | 4.00 | 5.90 | 2.74 | 4.23 | 3.48 | 5.22 |
| CAST ($N$=31) | 3.61 | 5.44 | 3.43 | 5.08 | 4.23 | 6.37 | 4.27 | 6.30 | 2.92 | 4.52 | 3.69 | 5.54 |
| CAST ($N$=47) | 3.75 | 5.68 | 3.45 | 5.16 | 4.32 | 6.55 | 4.36 | 6.47 | 3.02 | 4.76 | 3.78 | 5.72 |
| CAST ($N$=63) | 3.80 | 5.85 | 3.52 | 5.30 | 4.33 | 6.65 | 4.42 | 6.60 | 3.07 | 4.86 | 3.83 | 5.85 |
| CAST ($N$=95) | 3.84 | 6.02 | 3.55 | 5.45 | 4.38 | 6.83 | 4.48 | 6.80 | 3.05 | 4.94 | 3.86 | 6.01 |
| CAST ($N^*$=127) | 3.82 | 6.08 | 3.59 | 5.60 | 4.38 | 6.94 | 4.49 | 6.90 | 3.05 | 5.00 | 3.87 | 6.11 |

### H.2 QWEN3-CODER-30B-A3B, BLOCK 16

This appendix repeats the main protocol on Qwen3-Coder-30B-A3B-Instruct, a 30B sparse mixture-of-experts target with 3B active parameters, using the public block-16 DFlash head for that model on the same H100 SXM node, with greedy decoding, 40 prompts each on GSM8K, HumanEval, and MBPP, and 256 new tokens. AR, DFlash, and CAST all run the experts without a fused kernel.

At a 1k prefix, target-forward latency is 58.7 ms at $n$=16 and 57.4 ms at 128, a flat slope within run noise. In Table 17, the rule selects $N^*$=127, also the mean sweep oracle, with a 35.6% mean gain over DFlash. Gains rise through 127 on GSM8K and HumanEval, while MBPP peaks at 95 and gives back 1.0 point of gain at the final step. Mean gain improves only 0.7 points over that step, consistent with a broad plateau rather than a cost cliff.

Table 17: Qwen3-Coder-30B-A3B greedy decoding with the block-16 DFlash head on H100 SXM. Cells report speedup over AR and committed round length $R$.

| Method (width) | GSM8K | | HumanEval | | MBPP | | Avg. | |
|---|---|---|---|---|---|---|---|---|
| | ×AR | $R$ | ×AR | $R$ | ×AR | $R$ | ×AR | $R$ |
| AR ms/token | 62.3 | | 63.8 | | 62.6 | | | |
| DFlash (16) | 3.84 | 4.81 | 5.86 | 7.77 | 5.08 | 6.64 | 4.93 | 6.40 |
| CAST ($N$=15) | 4.52 | 5.65 | 6.53 | 8.61 | 5.87 | 7.52 | 5.64 | 7.26 |
| CAST ($N$=31) | 4.78 | 6.05 | 7.13 | 9.40 | 6.39 | 8.19 | 6.10 | 7.88 |
| CAST ($N$=47) | 5.05 | 6.37 | 7.29 | 9.58 | 6.50 | 8.40 | 6.28 | 8.12 |
| CAST ($N$=63) | 5.20 | 6.58 | 7.56 | 9.99 | 6.55 | 8.53 | 6.43 | 8.37 |
| CAST ($N$=95) | 5.26 | 6.69 | 7.72 | 10.26 | **6.91** | **8.98** | 6.63 | 8.64 |
| CAST ($N^*$=127) | **5.34** | **6.86** | **7.78** | **10.34** | 6.86 | 8.89 | **6.66** | **8.70** |

## I EXTENDED RELATED WORK

**One-pass draft trees.** DART also builds trees over one-pass parallel logits. Its drafter, described in Appendix G.2, is a single Transformer layer, and it prunes candidates with an external n-gram continuity signal and a trie at runtime (Liu et al., 2026a), answering which candidates to keep under a supplied construction rule or node budget. CAST asks how many nonroot nodes the loop should pack on the measured hardware.

JetSpec trains a causal-parallel draft head under a tree-causal mask, where every tree node is predicted in one forward pass yet conditions only on its ancestor path, and pairs it with best-first expansion and a paged tree-attention kernel (Hu et al., 2026a). Weaver keeps the pretrained DFlash drafter

and trains a lightweight autoregressive adapter over its top-$K$ marginals to restore the conditional dependencies a factorized drafter drops, together with rollback-free tree verification for gated linear-attention layers (Oda et al., 2026). Both target the acceptance ceiling of branch-agnostic marginals.

**Tree verification systems.** SpecInfer (Miao et al., 2024) builds token trees from small speculative models and verifies valid prefixes with topology-aware tree attention, Medusa (Cai et al., 2024) adds self-drafting heads with Cartesian or calibration-pruned sparse trees, and Sequoia (Chen et al., 2024) studies hardware-aware speculative trees for serving.

**Adaptive tree and budget objectives.** EAGLE-2 (Li et al., 2024b) expands the current frontier, scores nodes by pathwise draft confidence, and reranks shallow and deep nodes into a connected tree. OPT-Tree (Wang et al., 2025) maximizes a draft-probability proxy for expected acceptance and stops draft deepening when the gain falls below a threshold tied to draft-step cost. Both optimize before the target pass for AR or EAGLE-style drafters.

**Verification rules and traversal.** Block verification (Sun et al., 2024) jointly verifies a sampled draft block, HSD (Zhou et al., 2026) adds hierarchical branch resampling, Traversal Verification (Weng et al., 2025) accepts leaf-to-root sequences, and SpecTr (Sun et al., 2023) and Towards Optimal Multi-draft SD (Hu et al., 2025b) formulate distribution-preserving multi-candidate acceptance through optimal transport. For the greedy walk, acceptance certificates show that a tree with branching factor $m$ certifies acceptance at a target margin about $\sqrt{(m+1)/2}$ times smaller than single-token greedy under the same divergence budget (Sharma, 2026).

**Adaptive controllers.** At the serving level, MagicDec maps where speculation pays (Sadhukhan et al., 2024), TETRIS allocates a batch's draft-token budget (Wu et al., 2025), FASER manages draft and verify phases under dynamic load (Chen et al., 2026b), a utility gate bounds worst-case slowdown for MoE targets (Saxena et al., 2025), and MoESD identifies when speculation accelerates sparse MoE targets (Huang et al., 2025).

Three concurrent controllers adapt scalar draft budgets for the same one-pass block drafter. D-cut keeps the top survival-ranked prefixes under a profiled latency table (Liu et al., 2026b), AdaFlash distills on policy and truncates draft length with a learned head (Qian et al., 2026), and BlockPilot classifies a sample-adaptive block size (Zhang et al., 2026a). All adapt the depth or granularity of one chain.

**Objectives and theory.** Acceptance-aligned objectives (Zhou et al., 2024; Goel et al., 2024; Zhang et al., 2025) change the trained drafter, DistillSpec by aligning draft and target distributions and HASS by targeting the training and decoding mismatch. Yin et al. (2024) analyze limits of unbiased rejection-based speculative decoding. Surveys of speculative decoding round out the context (Xia et al., 2024; Hu et al., 2025a).