Request Forest: Jointly Growing Verification Trees for Speculative Decoding

Anonymous authors
Paper under double-blind review

Abstract

Block-parallel draft models can predict multiple future tokens in a single forward pass, yet the benefits of speculative decoding under high concurrency depend on how limited verification resources are allocated. Collapsing candidates into a single path and assigning a fixed verification window to each request limits the ability to accommodate differences in draft quality across requests and exploit alternative branches within each request, while increasing intermediate verification-state overhead in hybrid architectures. We propose Request Forest (RF), which jointly grows verification trees across requests under a shared budget. RF reuses the candidates and predecessor-conditioned scores provided by a block-parallel drafter, allowing eligible nodes across requests to compete for the budget according to their path priorities. This process jointly determines tree size, depth, and branching structure, yielding a prefix-closed verification forest. RF then packs the forest into a compact, branch-isolated layout, verifies it in a single target-model forward pass, and commits only the states along accepted paths. Theoretical analysis establishes that, under the specified scoring assumptions and budget constraints, this growth strategy maximizes the sum of priorities of selected candidates. Experiments on four hybrid-architecture models and four benchmarks demonstrate the effectiveness of RF. On Qwen3.8-27B at concurrency levels of 64–160, RF improves geometric-mean throughput by 56.6% over DFlash and 13.9% over DFlash2. At a concurrency level of 128, it reduces GPU Memory reserved for intermediate recurrent verification states per tensor-parallel rank by 50.4% relative to DFlash2.These results show that shared-budget tree growth effectively translates diffusion draft diversity into throughput gains under high concurrency.Code is available at https://anonymous.4open.science/r/Request-Forest-2844/.

1 Introduction

Speculative decoding accelerates large language model inference by generating candidate tokens with a lightweight draft model and verifying them in parallel with a target model, allowing multiple tokens to be generated in a single target-model forward pass(Leviathan et al., 2023; Chen et al., 2023). Conventional autoregressive drafters generate candidates sequentially, introducing overhead that limits further acceleration. Multi-position prediction and tree-based verification have explored parallel candidate generation and multi-path verification, respectively(Cai et al., 2024; Miao et al., 2024). More recently, DFlash uses a diffusion drafter to predict an entire token block in a single forward pass, while DFlash2 further models compatibility between adjacent candidates to improve draft-path coherence(Chen et al., 2026; Inco AI, 2026). These advances reduce drafting costs, but translating them into throughput gains under high concurrency also depends on which candidates the target model verifies and how verification resources are allocated across concurrent requests.

This allocation involves two interdependent decisions: how much verification budget each request receives and which candidate continuations that budget supports. DFlash and DFlash2 ultimately verify a single draft path, leaving unselected alternative branches outside target-model verification. Dynamic verification windows accommodate differences in draft quality across requests, but adjusting candidate counts alone does not determine which branches to retain. Likewise, independently optimizing each request's verification tree within a preallocated budget restricts resource reallocation across requests. Under high concurrency, extending one request's path, retaining an alternative branch, and expanding another request's candidates all compete for limited verification resources.

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.

This motivates jointly growing verification trees under a shared budget to unify cross-request resource allocation with per-request depth and branch selection.

Verification-resource constraints are particularly pronounced in hybrid architectures with recurrent layers. During standard decoding, these layers summarize the history using fixed-size states, whereas speculative verification must also support state commitment following candidate acceptance or rejection. In the verification implementations considered here, intermediate recurrent states are retained at candidate positions so that the states corresponding to accepted paths can be committed(Ghantasala, 2026; jianuo-huang, 2026). The reserved capacity grows with the configured concurrency capacity and verification window, reducing the space available for the KV cache under a fixed GPU memory budget. Figure 1 compares verification-state GPU Memory and serving throughput between MTP-Eagle and DFlash2 on Qwen3.8-27B. Under the evaluated configurations, DFlash2's larger verification-state overhead does not yield a commensurate throughput advantage at high concurrency. This observation highlights the need to consider both candidate coverage and verification cost, directing limited resources toward more valuable candidates.

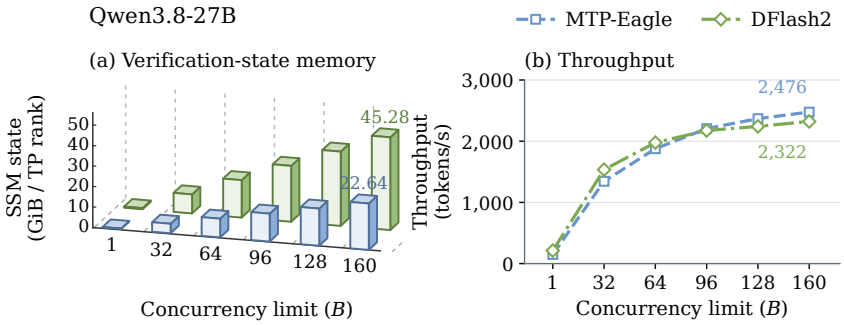

Figure 1: Verification-state GPU memory and throughput on Qwen3.8-27B: (a) startup SSM verification GPU memory per TP rank; (b) geometric-mean throughput across four benchmarks.

To this end, we propose Request Forest (RF), which jointly grows verification trees across requests under a shared verification budget, as illustrated in Figure 2. RF reuses the candidates and predecessor-conditioned scores provided by a diffusion drafter, defining node priorities by multiplying normalized scores along each path. Starting from each request's root, RF repeatedly selects the highest-priority eligible node across requests and makes its children eligible for selection, until the budget is exhausted or no eligible nodes remain. This process jointly determines each request's budget allocation, tree depth, and branching structure while retaining the ancestors required to verify every selected node. RF then packs the forest into a compact, branch-isolated layout, verifies it in a single target-model forward pass, and commits only the states along accepted paths. To maintain efficiency across serving loads, RF switches according to the number of active requests, retaining native linear verification at low load and enabling shared-budget forest verification at high load.

We evaluate RF on four hybrid-architecture models and four benchmarks. At concurrency levels of 64–160, RF improves geometric-mean throughput on Qwen3.8-27B by 56.6% over DFlash and 13.9% over DFlash2, and on Ornith-1.5-35B-A3B by 12.0% over DFlash2. These throughput gains are accompanied by lower verification-state overhead. At a concurrency level of 128, RF reduces GPU memory reserved for intermediate recurrent verification states per tensor-parallel rank by 50.4% on Qwen3.8-27B and 50.5% on Ornith-1.5-35B-A3B relative to DFlash2. On Qwen3.6-35B-A3B, the reduction reaches 63.0% at a concurrency level of 256. These results demonstrate that jointly growing verification trees under a shared budget translates diffusion draft diversity into high-concurrency throughput gains while reducing verification-state GPU memory requirements.

Our main contributions are as follows:

1. We introduce a shared-budget mechanism for jointly growing verification trees, using candidates and scores from a diffusion drafter to determine cross-request budget allocation and per-request tree structure within a unified selection process.

2. We theoretically relate candidate scores to the expected path length covered by selected candidates and establish that, under the specified scoring assumptions and uniform node costs, the proposed

strategy maximizes the sum of selected node priorities subject to a shared budget and per-request limits.

3. Experiments on four models and four benchmarks demonstrate that RF improves high-concurrency throughput over competitive methods such as DFlash2 while reducing the GPU memory overhead of intermediate verification states.

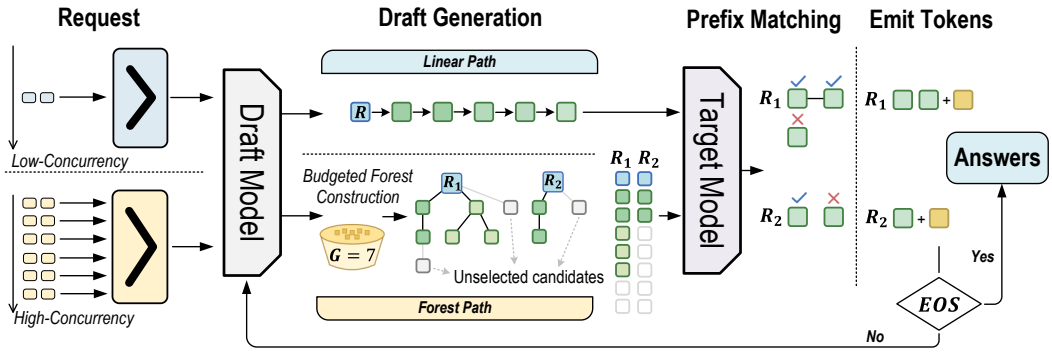

Figure 2: **Request Forest overview.** RF uses linear candidate paths at low concurrency and constructs a forest under a shared cross-request node budget at high concurrency.

2 Related Work

Tree-based speculative decoding retains multiple continuations and verifies them in parallel using tree attention(Cai et al., 2024; Miao et al., 2024). EAGLE-2, EAGLE-3, OPT-Tree, and Sequoia adapt tree structure and size using confidence, expected acceptance gains, or hardware costs(Li et al., 2024; Wang et al., 2025; Chen et al., 2024; Li et al., 2025), while SpecExec caches target distributions along high-probability draft paths(Svirschevski et al., 2024). DDTree constructs trees from position-wise distributions under a fixed budget(Ringel & Romano, 2026); BASTION and CaDDTree optimize per-request trees and budgets using acceptance gains and verification costs(Oh et al., 2026; Zhang et al., 2026). Across requests, TETRIS allocates linear prefixes using cumulative acceptance probabilities(Wu et al., 2025), D-cut trims block-parallel drafts based on execution costs(Liu et al., 2026), and DSpark selects verification lengths using prefix survival estimates and engine throughput characteristics(Cheng et al., 2026). ECHO schedules candidates under a shared budget with sparse confidence gating, prioritizing depth before branching(Hu et al., 2026). AdaSpec adapts drafting and verification to improve serving efficiency and service-level objective attainment(Huang et al., 2026). TreeWY reduces recurrent-state overhead in Gated DeltaNet tree verification through a tree-structured WY transformation that avoids full per-node state snapshots(Ghantasala, 2026).

3 Preliminaries

3.1 Candidate Space and Node Priorities

RF represents each request's candidate space as an implicit prefix tree to guide cross-request selection under a shared budget. Consider B_t active requests at round t. For each $b \in \{1, \dots, B_t\}$, let x_b denote the confirmed prefix and r_b the root corresponding to its final token. Let $\mathcal{V}_b$ be the set of non-root candidate nodes and $\mathrm{par}(v)$ the unique parent of node v. The candidate tree, directed edges, and child sets are defined as

$$\mathcal{T}_b = (\{r_b\} \cup \mathcal{V}_b, \mathcal{E}_b), \quad \mathcal{E}_b = \{(\mathrm{par}(v), v) : v \in \mathcal{V}_b\}, \quad \mathcal{C}_p = \{v \in \mathcal{V}_b : \mathrm{par}(v) = p\}. \tag{1}$$

Here, $\mathcal{T}_b$ is finite, $\mathcal{E}_b$ contains parent-to-child edges, and $\mathcal{C}_p$ contains the immediate children of p.

RF reuses DFlash2's block-parallel candidates and predecessor-conditioned scores(Chen et al., 2026; Inco AI, 2026). Let $s_\theta(p \to v)$ denote the score of edge (p, v), where θ denotes the fixed drafter

parameters. For nonempty $\mathcal{C}_p$ with finite edge scores, define

$$q_\theta(v \mid p) = \frac{\exp s_\theta(p \to v)}{\sum_{u \in \mathcal{C}_p} \exp s_\theta(p \to u)}, \qquad v \in \mathcal{C}_p. \tag{2}$$

Node priorities are products of normalized edge scores along each path:

$$\rho_\theta(r_b) = 1, \qquad \rho_\theta(v) = \rho_\theta(p) q_\theta(v \mid p) \le \rho_\theta(p), \quad p = \mathrm{par}(v). \tag{3}$$

Priorities are non-increasing along paths because $0 < q_\theta(v \mid p) \le 1$.

3.2 Selection Objective under a Shared Budget

RF assigns a shared verification-node budget to all requests, allowing their candidates to compete for selection. The surrogate objective is to maximize the total priority of selected nodes subject to the shared budget, per-request caps, and prefix closure.

Let $\mathcal{F}_b$ denote the selected nodes for request b and $\mathcal{F}$ their disjoint union, preserving request identities. Let K, K_t, and $C_{b,t}$ denote the configured budget, effective round budget, and per-request cap, respectively (Section 4.2):

$$\mathcal{F}_b \subseteq \mathcal{V}_b, \qquad \mathcal{F} = \biguplus_{b=1}^{B_t} \mathcal{F}_b, \qquad K, K_t, C_{b,t} \in \mathbb{Z}_{\ge 0}, \quad K_t \le K. \tag{4}$$

Let $\mathrm{Anc}(v)$ denote the candidate ancestors of v, excluding the root and v itself. Both K_t and $C_{b,t}$ are fixed before selection in each round. Each non-root candidate has unit cost. The prefix-closed forest selection objective is

$$\max_{\{\mathcal{F}_b \subseteq \mathcal{V}_b\}_{b=1}^{B_t}} \sum_{b=1}^{B_t} \sum_{v \in \mathcal{F}_b} \rho_\theta(v) \quad \text{s.t.} \quad \mathrm{Anc}(v) \subseteq \mathcal{F}_b, \quad |\mathcal{F}_b| \le C_{b,t}, \quad \sum_{b=1}^{B_t} |\mathcal{F}_b| \le K_t. \tag{5}$$

While candidate selection maximizes total node priority as a surrogate objective, the system aims to improve end-to-end decoding throughput, accounting for the verification size and execution overhead of the selected forest.

Each root and selected candidate contributes one verification row. Let N_{logical} denote the number of logical rows and N_{physical} the number of execution rows including padding:

$$N_{\text{logical}} = B_t + |\mathcal{F}|, \qquad N_{\text{physical}} \ge N_{\text{logical}}. \tag{6}$$

Padding rows do not participate in acceptance or state commitment. Let $\mathbf{y}_b^A$ denote the complete token sequence generated by method A for request b. Under identical requests, greedy decoding rules, and stopping criteria, the system objective is to reduce end-to-end execution time while preserving

$$\mathbf{y}_b^{\text{RF}} = \mathbf{y}_b^{\text{AR}}, \qquad b = 1, \ldots, B_t. \tag{7}$$

4 Request Forest

RF selects an execution path according to the current request load. In forest mode, it selects candidates under a shared budget, arranges them into a branch-isolated verification layout, and uses the verification results to emit tokens and commit accepted-path states. Figure 2 provides an overview.

4.1 Load-Adaptive Switching

At low load, forest selection and layout overhead may outweigh their benefits. RF therefore switches between native linear and forest verification based on the number of active requests.

Let $m_t \in \{\text{native}, \text{forest}\}$ denote the execution mode at round $t \ge 1$, with initial mode m_0. Given entry and exit thresholds B_{in} and B_{out}, where $1 \le B_{\text{out}} < B_{\text{in}}$, and a low-load confirmation period

$\tau \in \mathbb{Z}_{\geq 1}$, the switching rule is

$$m_t = \begin{cases} \text{forest}, & B_t \geq B_{\text{in}}, \\ \text{native}, & B_t < B_{\text{in}},\ m_{t-1} = \text{forest},\ t > \tau,\ \max_{1 \leq j \leq \tau} B_{t-j} \leq B_{\text{out}}, \\ m_{t-1}, & \text{otherwise}. \end{cases} \tag{8}$$

The exit condition checks only the preceding τ completed rounds.

4.2 Prefix-Closed Forest Construction under a Shared Budget

In forest mode, RF allocates a limited candidate-node budget across requests, depths, and branches using the fixed priorities in Eq. (3). To preserve verification dependencies, every selected node retains all its candidate ancestors.

Given a window length $W \in \mathbb{Z}_{\geq 2}$, the effective budget and per-request caps are

$$K_t = \min\{K, \kappa(B_t), B_t(W-1)\}, \qquad C_{b,t} = \min\{K_t, W-1\}. \tag{9}$$

Here, κ is a load-dependent budget function loaded at startup, and $C_{b,t}$ bounds the total number of selected nodes for request b.

Under these constraints, the global eligible frontier is

$$Q = \left\{ (b, v) \;\middle|\; \begin{array}{ll} 1 \leq b \leq B_t, & v \in \mathcal{V}_b \setminus \mathcal{F}_b, \\ \rho_\theta(v) > 0, & |\mathcal{F}_b| < C_{b,t}, \\ \text{par}(v) \in \mathcal{F}_b \cup \{r_b\} & \end{array} \right\}. \tag{10}$$

RF repeatedly selects the highest-priority node in Q and makes its children eligible, subject to the same constraints. See the Appendix C. for the algorithm. and Figure 3 illustrate the procedure.

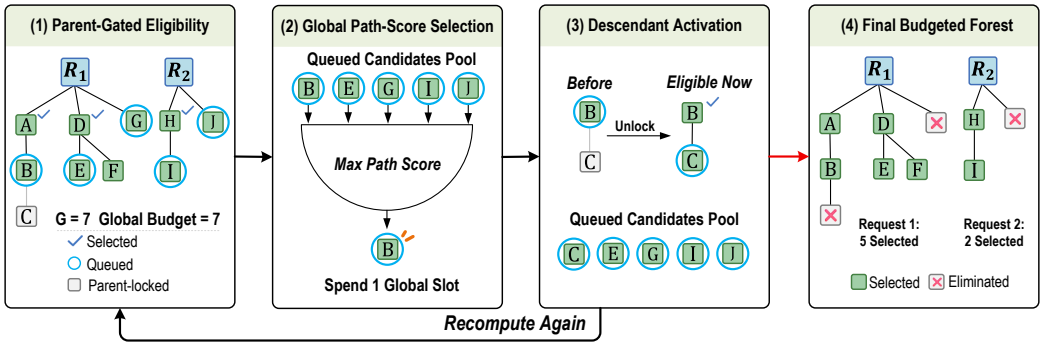

Figure 3: **Prefix-closed forest construction under a shared budget.** Eligible nodes compete across requests; the highest-priority node is selected and its children exposed until the budget is exhausted.

Surrogate Objective and Selection Guarantees. The surrogate objective is the expected candidate-prefix length under the score-induced path distribution. Consider an auxiliary random path that starts at the root and samples children according to q_θ until reaching a leaf. Let Π_b denote its non-root nodes. For any prefix-closed $\mathcal{F}_b$,

$$\Pr(v \in \Pi_b) = \rho_\theta(v), \qquad \mathbb{E}\left[\sum_{b=1}^{B_t} |\Pi_b \cap \mathcal{F}_b|\right] = \sum_{b=1}^{B_t} \sum_{v \in \mathcal{F}_b} \rho_\theta(v). \tag{11}$$

The derivation is provided in Appendix A.1.

Lemma 1 (Frontier Maximum). For finite candidate trees with fixed, nonnegative priorities that are non-increasing along parent–child paths, suppose each $\mathcal{F}_b$ is prefix-closed and respects its per-request cap. If $Q \neq \varnothing$, then

$$\max_{(b,v) \in Q} \rho_\theta(v) = \max_{\substack{b:\, |\mathcal{F}_b| < C_{b,t} \\ v \in \mathcal{V}_b \setminus \mathcal{F}_b,\ \rho_\theta(v) > 0}} \rho_\theta(v). \tag{12}$$

See Appendix A.2 for the proof.

Proposition 1 (Surrogate Optimality under a Shared Budget). For finite candidate trees with fixed, nonnegative node weights that are non-increasing along parent–child paths, Algorithm 1 optimizes the unit-cost surrogate objective in Eq. (5) for any fixed nonnegative integer budget K_t and per-request caps $C_{b,t}$. See Appendix A.3 for the proof.

4.3 Forest Verification and State Commitment

RF packs roots and selected candidates into verification rows, verifies them in one target-model forward pass, then emits tokens and commits accepted-path states.

Branch-Isolated Verification. Let x_b^- denote the confirmed prefix excluding the root token. Each node attends only to the cached prefix and its own root-to-node path, with positions assigned by path depth. Let $\mathbf{s}_b$ denote the persistent recurrent state for x_b^- and $\mathbf{s}_v$ the state after processing node v. The input state is

$$\mathbf{s}_v^{\text{in}} = \mathbf{s}_b \quad (v = r_b), \qquad \mathbf{s}_v^{\text{in}} = \mathbf{s}_{\text{par}(v)} \quad (v \in \mathcal{F}_b). \tag{13}$$

Each node writes to a separate temporary slot, preventing state contamination across branches.

Acceptance and Commitment. RF follows the target-greedy matching path with identical tie-breaking, emitting matched tokens and one bonus token subject to stopping rules. For continuing requests, it commits only path KV entries and the last matched state (the root state if none match); the unprocessed bonus token becomes the next root.

Verification Budget and GPU Memory. The shared budget bounds the logical verification size:

$$N_{\text{logical}} = B_t + |\mathcal{F}| \le B_t + K_t. \tag{14}$$

State GPU memory is reserved at startup for both execution paths, including padding; fewer candidates save GPU memory only if this reservation is reduced. Execution details and complexity analysis appear in Appendix C.

5 Experiments

5.1 Experimental Setup

Models, Benchmarks, and Implementation. We evaluate RF on four models with recurrent verification states: Qwen3.8-27B, Ornith-1.5-35B-A3B, GLM-5.3-Flash, and Qwen3.6-35B-A3B. Qwen3.8-27B is dense; the others are MoE models. We use HumanEval (Chen et al., 2021), GSM8K (Cobbe et al., 2021), MATH-500 (Hendrycks et al., 2021; Lightman et al., 2024), and MT-Bench (Zheng et al., 2023). Detailed configurations are provided in Appendix D.1.

Metrics. We report token utilization, throughput (tokens/s), and matched AR speedups using four-run means and sample standard deviations. Aggregate gains use geometric means of matched mean-throughput ratios. Memory counts only startup-reserved SSM verification states per TP rank (GiB).

5.2 Main Results

High Concurrency. Table 1 compares Qwen and Ornith at $B = 128$ and GLM at $B = 48$. RF achieves average speedups over AR across four benchmarks of $1.32\times$, $1.59\times$, $1.72\times$, and $2.09\times$ on Qwen3.8, Ornith, Qwen3.6, and GLM, respectively. Figure 4 shows throughput across concurrency levels. Across seven levels within $B = 64$–160 and four benchmarks, RF improves geometric-mean throughput on Qwen3.8 by 56.6%, 13.9%, and 11.8% over DFlash, DFlash2, and MTP, respectively, and on Ornith by 12.0% over DFlash2. On Ornith, the gain over DFlash-W4 is 4.9% across seven high-concurrency levels completed by both methods. On Qwen3.6, RF matches DFlash2 at $B \le 80$ and improves throughput by 5.7% across ten levels with $B \ge 96$. On GLM, RF outperforms DFlash2 by 2.2% at $B = 48$ and MTP by 1.3% and 3.7% at $B = 80$ and 96, respectively. Full results appear in Appendix E.

Table 1: Token utilization (Util.) and throughput speedup over matched AR at high concurrency: $B = 128$ for Qwen and Ornith, and $B = 48$ for GLM. Speedups use mean throughput over four runs; per-point sample standard deviations are reported in the Appendix E.

Model	Method	HumanEval		GSM8K		MATH-500		MT-Bench		Avg.
		Util.	Speedup	Util.	Speedup	Util.	Speedup	Util.	Speedup	
(a) Qwen / Ornith: $B = 128$										
Qwen3.8 27B	MTP	0.6533	1.10×	0.7533	1.32×	0.7200	1.17×	0.6100	1.13×	1.18×
	DFlash	0.3342	0.79×	0.4142	0.98×	0.4274	0.85×	0.2558	0.76×	0.85×
	DFlash2	0.5034	1.00×	0.6536	1.34×	0.6613	1.16×	0.4512	0.98×	1.12×
	RF	**0.8448**	**1.26×**	**0.8991**	**1.46×**	**0.9032**	**1.34×**	**0.7725**	**1.22×**	**1.32×**
Ornith-1.5 35B-A3B	DFlash	0.9030	1.64×	**0.7402**	1.53×	0.7600	1.48×	0.5260	1.18×	1.46×
	DFlash2	0.8400	1.72×	0.5414	1.31×	0.6000	1.39×	0.4657	1.00×	1.36×
	RF	**0.9038**	**1.79×**	0.7264	**1.64×**	**0.7667**	**1.65×**	**0.7600**	**1.27×**	**1.59×**
Qwen3.6 35B-A3B	MTP	0.9617	1.35×	0.8734	1.26×	0.8859	1.30×	0.7689	1.19×	1.27×
	DFlash2	0.9217	1.88×	0.7218	1.43×	0.7724	1.89×	0.5055	1.27×	1.62×
	RF	**0.9917**	**1.98×**	**0.9685**	**1.50×**	**0.9869**	**2.03×**	**0.8250**	**1.37×**	**1.72×**
(b) GLM: $B = 48$										
GLM-5.3 Flash	MTP	0.8467	1.67×	0.8067	1.42×	0.8167	1.65×	0.6800	1.41×	1.54×
	DFlash2	0.7914	2.12×	0.6200	2.60×	0.6800	2.10×	0.4500	1.37×	2.05×
	RF	**0.8515**	**2.17×**	**0.8319**	**2.63×**	**0.8689**	**2.15×**	**0.7344**	**1.41×**	**2.09×**

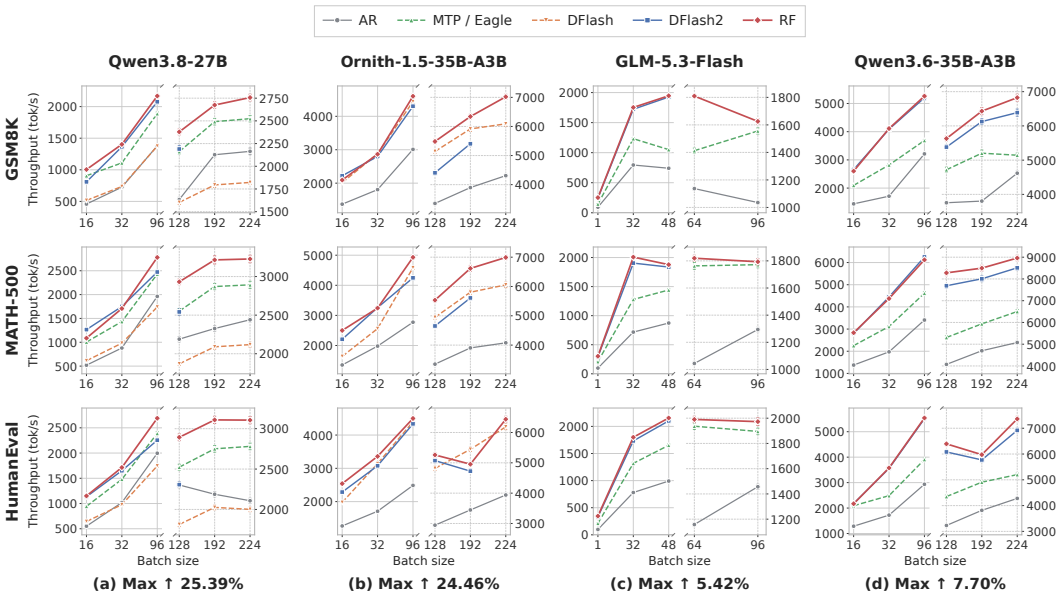

Figure 4: Serving throughput across four models; rows show GSM8K, MATH-500, and HumanEval. Left and right ranges in each subplot use separate y-scales. Max denotes RF's largest plotted gain over DFlash2. AR is target-only decoding; model-specific MTP/Eagle baselines are listed in Table 1. Curves show mean throughput.

Figure 5 reports verification-state overhead. At $B = 128$, RF reduces intermediate SSM memory by 50.4% on Qwen3.8 and 50.5% on Ornith relative to DFlash2; on Qwen3.6, the reduction reaches

63.0% at $B = 256$. On Ornith, RF runs up to $B = 256$, whereas DFlash2 encounters OOM at $B = 224, 240, 256$. These savings extend the feasible concurrency range under this configuration but do not ensure higher throughput at every point: on Qwen3.8 at $B = 256$, RF remains 4.8% slower than MTP.

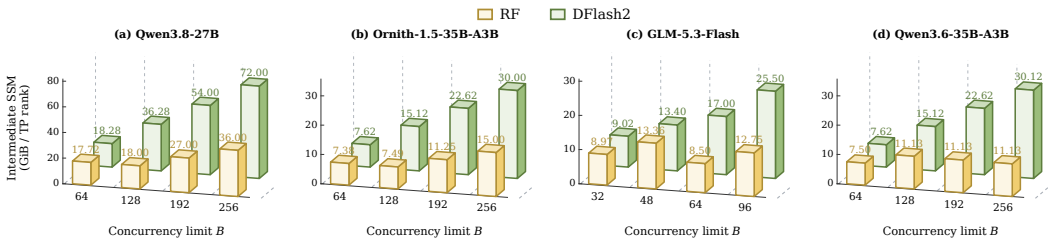

Figure 5: Intermediate SSM verification memory per TP rank (GiB) for RF (front) and DFlash2 (back). Qwen and Ornith are shown from $B = 64$; y-scales vary by panel. DFlash2 values at $B = 192, 256$ for Qwen3.8, $B = 256$ for Ornith, and $B = 64, 96$ for GLM are estimated as twice RF's allocation at the same point. Full breakdowns appear in Appendix F.

Token Utilization. In Table 1, RF achieves higher token utilization than DFlash2 on all four benchmarks across all four models. For example, on GSM8K, utilization increases from 0.6536 to 0.8991 on Qwen3.8 and from 0.7218 to 0.9685 on Qwen3.6, indicating more effective use of verified candidates. Request-level quantiles for both Qwen models appear in Appendix F.2.

5.3 Ablation Studies

We examine shared-budget allocation, branch selection, window size, and load-adaptive switching. Unless otherwise stated, component ablations use Qwen3.8-27B and Ornith-1.5-35B-A3B with the same drafter, scoring rule, total budget $K = 3B$, TP4, BF16, and a static memory fraction of 0.80. The component experiments cover $B = 64, 80, 96, 112, 128$ across four benchmarks. Table 2 summarizes the component and window-size ablations, while Table 4 reports the load-adaptive switching comparison.

Table 2: Component and window-size ablations. S: cross-request budget sharing; F: branch selection; ✓/◯: enabled/disabled. TPS is the geometric mean of 20 mean throughputs across five concurrency levels and four benchmarks (tokens/s); Δ is relative to EC. GF uses $W = 8$; GF16 is the matched $W = 16$ variant.

Variant	S	F	W	Qwen3.8-27B		Ornith-1.5-35B-A3B	
				TPS ↑	Δ ↑	TPS ↑	Δ ↑
EC	◯	◯	8	2135.3	—	3860.8	—
GC	✓	◯	8	2285.4	+7.0%	4130.7	+7.0%
GF16	✓	✓	16	2354.8	+10.3%	4253.8	+10.2%
GF	✓	✓	8	**2398.8**	**+12.3%**	**4335.7**	**+12.3%**

Shared Budget. Table 2 isolates budget sharing by comparing EC (`equal-chain`), which assigns fixed per-request quotas, with GC (`global-chain`), which reallocates the same total budget across requests while retaining single-path verification. GC increases geometric-mean throughput from 2135.3 to 2285.4 tokens/s on Qwen3.8-27B and from 3860.8 to 4130.7 tokens/s on Ornith, yielding a 7.0% gain on both models. It outperforms EC in all 20 benchmark–concurrency combinations per model. With the drafter, scoring, window size, and total budget fixed, these gains demonstrate the benefit of allocating verification resources according to candidate priorities rather than fixed per-request quotas.

Branch Selection. Allowing multiple branches within each request to compete under the shared budget yields RF's forest path, GF (`global-forest`). GF improves paired geometric-mean throughput by 5.0% over GC and 12.3% over EC on both models, outperforming GC in all 20 combinations per model. The three $W = 8$ variants reserve identical intermediate SSM memory at matched configurations, attributing the gains to improved candidate allocation under the same state reservation. Detailed concurrency trends and chain constraints are provided in Appendix G.

Window Size and State Cost. With budget sharing and branch selection fixed, $W = 8$ is 1.9% faster than $W = 16$ on both models. At $B = 128$, increasing the window raises intermediate SSM memory per GPU from 18.00 to 35.44 GiB on Qwen and from 7.49 to 14.77 GiB on Ornith. Across concurrency levels completed by both variants, $W = 8$ achieves 1.6%–3.3% and 1.6%–2.3% higher geometric-mean throughput across four benchmarks on Qwen and Ornith, respectively. It also runs up to $B = 256$ on both models, whereas $W = 16$ completes only up to $B = 192$ and 176, respectively. Additional window-size results are provided in Appendix G.3.

Table 3: Load-switching ablation. AF always uses forest verification; RF switches based on active request count. Values are geometric means of mean throughput across four benchmarks (tokens/s); Δ is relative to matched AF. Both use identical windows and budget rules.

	Qwen3.8-27B			Ornith-1.5-35B-A3B		
B	AF	RF	$\Delta\uparrow$	AF	RF	$\Delta\uparrow$
8	672.7	**723.8**	$+7.6\%$	1301.3	**1489.2**	$+14.4\%$
32	1514.1	**1591.9**	$+5.1\%$	2430.0	**2878.2**	$+18.4\%$
64	2075.7	**2089.8**	$+0.7\%$	3695.6	**3759.8**	$+1.7\%$

Load-Adaptive Switching. We fix $W = 8$, the budget rules, and the forest allocator, comparing always-forest AF with RF, which can revert to native verification using entry/exit thresholds of 64/48. At $B = 8$, RF improves throughput by 7.6% and 14.4% on Qwen and Ornith, respectively; at $B = 32$, the gains are 5.1% and 18.4%. At $B = 64$, the gap narrows to 0.7% and 1.7%. The gains are therefore concentrated at low load, supporting retention of the native path. Switching depends on the active request count B_t in each decoding round; the configured concurrency limit B does not imply a fixed execution path throughout. Full 19-point curves and per-benchmark means $\pm$ standard deviations are provided in Appendix G.4.

Overhead Breakdown. We profile EC, GC, and GF on Qwen3.8-27B with GSM8K, $B = 128$, and a 512-token generation limit. Compared with EC, GC increases full-batch token advancement by 3.98%, while target-forward time remains approximately 94 ms. End-to-end throughput, including batch-tail execution, increases by 4.00%. By additionally allowing branch competition, GF increases full-batch token advancement by 4.42% and throughput by 2.74% over GC, with similar target-forward time.

6 Conclusion

We presented Request Forest (RF), a speculative decoding framework that jointly grows verification trees across requests under a shared budget. RF reuses DFlash2's block-parallel candidates and predecessor-conditioned scores to jointly determine request budgets, tree depths, and branching structures without retraining the drafter. Compact, branch-isolated verification and accepted-path state commitment support execution on hybrid architectures, while load-adaptive switching limits forest overhead at low concurrency. Our theoretical analysis connects the selection objective to expected candidate-prefix coverage and establishes optimality under fixed, non-increasing priorities and unit node costs. Experiments on four models and four benchmarks demonstrate improved high-concurrency throughput and token utilization with lower intermediate verification-state memory. On Qwen3.8-27B, RF improves geometric-mean throughput over DFlash2 by 13.9% at concurrency levels of 64–160 and reduces reserved verification-state memory by 50.4% at concurrency 128. These findings highlight the value of jointly allocating verification resources across requests and candidate branches to translate diffusion drafting into efficient high-concurrency serving.

AI USE STATEMENT

In this work, we used generative AI tools for assistance with implementing our Request Forest (RF) prototype in SGLang and writing plotting scripts. We have not used generative AI tools for generating bibliographic references.Additionally, we used generative AI tools for checked research ideas for potential plagiarism through a manual literature survey. We have reviewed all AI-assisted work.LLM-generated code was verified and tested. We take responsibility for the final content of this work,including text, claims or artifacts produced with the aid of generative AI.

REPRODUCIBILITY STATEMENT

Reproducibility Statement. We provide an anonymous code repository containing the implementation of Request Forest (RF), including an SGLang-based prototype and experimental scripts. The repository is accessible at https://anonymous.4open.science/r/Request-Forest-2844/.

REFERENCES

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 Ruslan Salakhutdinov, Zico Kolter, Katherine Heller, Adrian Weller, Nuria Oliver, Jonathan Scarlett, and Felix Berkenkamp (eds.), *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 5209–5235. PMLR, 21–27 Jul 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.

Jian Chen, Yesheng Liang, and Zhijian Liu. DFlash: Block Diffusion for Flash Speculative Decoding. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2026. URL https://openreview.net/forum?id=Oz335dV48X.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, 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. 10.48550/arXiv.2107.03374. URL https://arxiv.org/abs/2107.03374.

Zhuoming Chen, Avner May, Ruslan Svirschevski, Yuhsun Huang, Max Ryabinin, Zhihao Jia, and Beidi Chen. Sequoia: Scalable and robust speculative decoding. In A. Globerson, L. Mackey, D. Belgrave, A. Fan, U. Paquet, J. Tomczak, and C. Zhang (eds.), *Advances in Neural Information Processing Systems*, volume 37, pp. 129531–129563. Curran Associates, Inc., 2024. 10.52202/079017-4116. URL https://proceedings.neurips.cc/paper_files/paper/2024/file/ea1f5f0878d43ff4fb8bf64ef4a2326c-Paper-Conference.pdf.

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

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. Training Verifiers to Solve Math Word Problems. *arXiv preprint arXiv:2110.14168*, 2021. 10.48550/arXiv.2110.14168. URL https://arxiv.org/abs/2110.14168.

Sneha Murthy Ghantasala. Treewy: Speculative verification for gated deltanet hybrids. *arXiv preprint arXiv:2608.20961*, 2026.

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. Measuring mathematical problem solving with the math dataset. *arXiv preprint arXiv:2103.03874*, 2021.

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. *arXiv preprint arXiv:2604.09603*, 2026.

Kaiyu Huang, Hao Wu, Zhubo Shi, Han Zou, Minchen Yu, and Qingjiang Shi. Adaspec: Adaptive speculative decoding for fast, slo-aware large language model serving. In *Proceedings of the 2025 ACM Symposium on Cloud Computing*, SoCC '25, pp. 361–374, New York, NY, USA, 2026. Association for Computing Machinery. ISBN 9798400722769. 10.1145/3772052.3772239. URL https://doi.org/10.1145/3772052.3772239.

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

jianuo-huang. [Feature] Reduce Mamba/GDN Verify Cache Memory Overhead in DFLASH Speculative Decoding. sgl-project/sglang, GitHub Issue no. 28730, June 2026. URL https://github.com/sgl-project/sglang/issues/28730. Published June 19, 2026. Accessed September 23, 2026.

Yaniv Leviathan, Matan Kalman, and Yossi Matias. Fast inference from transformers via speculative decoding. In Andreas Krause, Emma Brunskill, Kyunghyun Cho, Barbara Engelhardt, Sivan Sabato, and Jonathan Scarlett (eds.), *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pp. 19274–19286. PMLR, 23–29 Jul 2023. URL https://proceedings.mlr.press/v202/leviathan23a.html.

Yuhui Li, Fangyun Wei, Chao Zhang, and Hongyang Zhang. EAGLE-2: Faster inference of language models with dynamic draft trees. In Yaser Al-Onaizan, Mohit Bansal, and Yun-Nung Chen (eds.), *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pp. 7421–7432, Miami, Florida, USA, November 2024. Association for Computational Linguistics. 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 D. Belgrave, C. Zhang, H. Lin, R. Pascanu, P. Koniusz, M. Ghassemi, and N. Chen (eds.), *Advances in Neural Information Processing Systems*, volume 38, Main Conference, pp. 136737–136756. Curran Associates, Inc., 2025. 10.52202/085713-4562. URL https://proceedings.neurips.cc/paper_files/paper/2025/file/c7b5a35ea98b62512a869c19ea7b03cb-Paper-Conference.pdf.

Hunter Lightman, Vineet Kosaraju, Yuri Burda, Harrison Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. Let's verify step by step. In B. Kim, Y. Yue, S. Chaudhuri, K. Fragkiadaki, M. Khan, and Y. Sun (eds.), *International Conference on Learning Representations*, volume 2024, pp. 39578–39601, 2024. URL https://proceedings.iclr.cc/paper_files/paper/2024/file/aca97732e30bcf1303bc22ac3924fd16-Paper-Conference.pdf.

T. Liu, Y. Shen, R. Cen, J. Shi, J. Zhang, G. Qin, H. Liu, S. Liu, G. Yu, and J. Zhu. D-cut: Adaptive Verification Depth Pruning for Batched Speculative Decoding. *arXiv preprint arXiv:2607.14647*, 2026. 10.48550/arXiv.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*, ASPLOS '24, pp. 932–949, New York, NY, USA, 2024. Association for Computing Machinery. ISBN 9798400703867. 10.1145/3620666.3651335. URL https://doi.org/10.1145/3620666.3651335.

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

L. Ringel and Y. Romano. Accelerating Speculative Decoding with Block Diffusion Draft Trees. *arXiv preprint arXiv:2604.12989*, 2026. 10.48550/arXiv.2604.12989.

Ruslan Svirschevski, Avner May, Zhuoming Chen, Beidi Chen, Zhihao Jia, and Max Ryabinin. Specexec: Massively parallel speculative decoding for interactive llm inference on consumer devices. In A. Globerson, L. Mackey, D. Belgrave, A. Fan, U. Paquet, J. Tomczak, and C. Zhang (eds.), *Advances in Neural Information Processing Systems*, volume 37, pp. 16342–16368. Curran Associates, Inc., 2024. 10.52202/079017-0522. URL https://proceedings.neurips.cc/paper_files/paper/2024/file/1d91d5689e251d27993a3c2182dddcf7-Paper-Conference.pdf.

Jikai Wang, Yi Su, Juntao Li, Qingrong Xia, Zi Ye, Xinyu Duan, Zhefeng Wang, and Min Zhang. OPT-tree: Speculative decoding with adaptive draft tree structure. *Transactions of the Association for Computational Linguistics*, 13:188–199, 2025. 10.1162/tacl_a_00735. URL https://aclanthology.org/2025.tacl-1.8/.

Z. Wu, Z. Zhou, A. Verma, A. Prakash, D. Rus, and B. K. H. Low. TETRIS: Optimal Draft Token Selection for Batch Speculative Decoding. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics, Volume 1: Long Papers*, pp. 33329–33345, Vienna, Austria, 2025. Association for Computational Linguistics. 10.18653/v1/2025.acl-long.1598.

Zihao Ye, Lequn Chen, Ruihang Lai, Wuwei Lin, Yineng Zhang, Stephanie Wang, Tianqi Chen, Baris Kasikci, Vinod Grover, Arvind Krishnamurthy, and Luis Ceze. Flashinfer: Efficient and customizable attention engine for llm inference serving. In M. Zaharia, G. Joshi, and Y. Lin (eds.), *Proceedings of Machine Learning and Systems*, volume 7. MLSys, 2025. URL https://proceedings.mlsys.org/paper_files/paper/2025/file/dbf02b21d77409a2db30e56866a8ab3a-Paper-Conference.pdf.

S. Zhang, H. Qiu, H. He, and Y. Dai. Cost-Aware Diffusion Draft Trees for Speculative Decoding. *arXiv preprint arXiv:2606.01813*, 2026. 10.48550/arXiv.2606.01813.

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric Xing, Hao Zhang, Joseph Gonzalez, and Ion Stoica. Judging llm-as-a-judge with mt-bench and chatbot arena. In A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine (eds.), *Advances in Neural Information Processing Systems*, volume 36, pp. 46595–46623. Curran Associates, Inc., 2023. 10.52202/075280-2020. URL https://proceedings.neurips.cc/paper_files/paper/2023/file/91f18a1287b398d378ef22505bf41832-Paper-Datasets_and_Benchmarks.pdf.

Appendix Contents

A Proof of Theorem **14**
A.1 Proof of Eq. (11) . . . 14
A.2 Proof of Lemma 1 . . . 14
A.3 Proof of Proposition 1 . . . 14

B Forest Verification, State Commitment, and Runtime Overhead **15**

C Forest Construction and Execution Details **17**

D Experimental Protocols and Implementation Details **17**
D.1 Common Settings and Experiment Groups . . . 17
D.2 Model and Serving Parameters . . . 18
D.3 Workloads, Measurement, and Aggregation . . . 18

E Full Throughput Results **20**
E.1 Paired Throughput Gains . . . 20
E.2 Concurrency Sweeps Across Four Benchmarks . . . 22

F State Overhead and Candidate Utilization **24**
F.1 Persistent SSM and Intermediate Verification SSM . . . 24
F.2 Candidate Utilization . . . 27
F.3 State Layout and Capacity Limits at High Concurrency . . . 28

G Component Ablations and Parameter Analysis **30**
G.1 Incremental Component Ablations . . . 30
G.2 Full Component Ablation Results . . . 31
G.3 Window Size and Concurrency Range . . . 33
G.4 Load Switching . . . 35
G.5 Overhead Breakdown . . . 36

H Task Quality Evaluation **37**

A Proof of Theorem

A.1 Proof of Eq. (11)

Proof. Fix the candidate trees and transition weights q_θ for the current round. For a candidate node v, denote the unique path from the root to v by $v_0 = r_b, v_1, \ldots, v_d = v$. By the sampling rule for the auxiliary path and Eq. (3),

$$\Pr(v \in \Pi_b) = \prod_{i=1}^{d} q_\theta(v_i \mid v_{i-1}) = \rho_\theta(v). \tag{A.1}$$

Prefix closure ensures that $\Pi_b \cap \mathcal{F}_b$ forms a contiguous prefix of candidate nodes starting from the root. Expressing its length as a sum of indicator variables and applying linearity of expectation yields

$$\begin{aligned} \mathbb{E}\left[\sum_{b=1}^{B_t} |\Pi_b \cap \mathcal{F}_b|\right] &= \sum_{b=1}^{B_t} \sum_{v \in \mathcal{F}_b} \mathbb{E}[\mathbf{1}\{v \in \Pi_b\}] \\ &= \sum_{b=1}^{B_t} \sum_{v \in \mathcal{F}_b} \rho_\theta(v). \end{aligned} \tag{A.2}$$

□

A.2 Proof of Lemma 1

Proof. Consider any request b satisfying $|\mathcal{F}_b| < C_{b,t}$ and any unselected positive-weight candidate v in that request. Let u be the first unselected candidate node on the path from the root to v. By prefix closure and the nonincreasing priorities along each path,

$$\left.\begin{array}{r} \mathrm{par}(u) \in \mathcal{F}_b \cup \{r_b\}, \\ \rho_\theta(u) \geq \rho_\theta(v) > 0 \end{array}\right\} \quad \Longrightarrow \quad (b,u) \in Q. \tag{A.3}$$

Thus, every unselected positive-weight node in a request with remaining quota is dominated by a frontier node in the same request. Since Q is a subset of these unselected nodes, whenever $Q \neq \varnothing$,

$$\max_{\substack{b:\, |\mathcal{F}_b| < C_{b,t} \\ v \in \mathcal{V}_b \setminus \mathcal{F}_b,\ \rho_\theta(v) > 0}} \rho_\theta(v) \leq \max_{(b,u) \in Q} \rho_\theta(u) \leq \max_{\substack{b:\, |\mathcal{F}_b| < C_{b,t} \\ v \in \mathcal{V}_b \setminus \mathcal{F}_b,\ \rho_\theta(v) > 0}} \rho_\theta(v). \tag{A.4}$$

The two outer quantities are identical, establishing the claimed equality. □

A.3 Proof of Proposition 1

Proof. Fix priorities ρ_θ, budget K_t, and caps $C_{b,t}$ satisfying the assumptions of the proposition, and define

$$J(\mathcal{F}) = \sum_{b=1}^{B_t} \sum_{v \in \mathcal{F}_b} \rho_\theta(v), \qquad \mathcal{U}_b = \{v \in \mathcal{V}_b : \rho_\theta(v) > 0\}. \tag{A.5}$$

Every candidate ancestor of a positive-weight node also has positive weight. Thus, removing zero-weight nodes from a feasible solution leaves the objective unchanged and preserves prefix closure and the budget constraints. It therefore suffices to compare feasible solutions containing only positive-weight nodes. The algorithm selects a node only when its parent has been retained, its request has remaining quota, and the global budget has not been exhausted. Hence, every intermediate set produced by the algorithm is feasible.

For each request b, arrange the positive node weights in nonincreasing order and retain the first m_b entries allowed by its cap:

$$\begin{aligned} m_b &= \min\{C_{b,t}, |\mathcal{U}_b|\}, \\ \alpha_{b,1} &\geq \cdots \geq \alpha_{b,|\mathcal{U}_b|} > 0 \quad (|\mathcal{U}_b| > 0). \end{aligned} \tag{A.6}$$

Let $\mathcal{A}$ be the multiset formed from these truncated sequences, and let β_j denote its jth largest weight:

$$\begin{aligned} &\mathcal{A} = \{\{\alpha_{b,j} : 1 \le b \le B_t,\ 1 \le j \le m_b\}\}, \\ &\beta_1 \ge \cdots \ge \beta_{|\mathcal{A}|} > 0, \qquad m = \min\{K_t, |\mathcal{A}|\}. \end{aligned} \tag{A.7}$$

Equal weights are counted with their node multiplicities; an empty multiset corresponds to an empty sequence, and an empty sum is zero. If $m = 0$, both the algorithm and every feasible solution have objective value zero. Assume henceforth that $m > 0$.

Denote the set after the nth selection by $\mathcal{S}^{(n)} = \biguplus_b \mathcal{S}_b^{(n)}$, and let $n_b^{(n)} = |\mathcal{S}_b^{(n)}|$. By the per-request argument in Lemma 1, whenever the algorithm selects a node from request b, its weight is the largest among the remaining positive-weight nodes of that request. Thus, the weights already selected from request b are exactly $\alpha_{b,1}, \ldots, \alpha_{b,n_b^{(n)}}$. Since each local sequence is nonincreasing, induction gives

$$\begin{aligned} \max_{(b,v)\in Q} \rho_\theta(v) &= \max_{b:\, n_b^{(n)} < m_b} \alpha_{b,n_b^{(n)}+1} \\ &= \max_{\substack{1\le b\le B_t \\ n_b^{(n)} < j \le m_b}} \alpha_{b,j} = \beta_{n+1}, \qquad 0 \le n < m. \end{aligned} \tag{A.8}$$

Here, $n < m$ ensures that $n < K_t$ and $\sum_b n_b^{(n)} = n < \sum_b m_b$. Hence, at least one request still has both remaining quota and positive-weight candidates, so Eq. (A.3) implies $Q \neq \varnothing$. When $n = m$, either the global budget is exhausted or every request has exhausted its quota or its positive-weight candidates. The algorithm therefore selects exactly m nodes, and its final set $\mathcal{S}^\star$ satisfies

$$J(\mathcal{S}^\star) = \sum_{j=1}^{m} \beta_j. \tag{A.9}$$

For any feasible solution $\mathcal{F}$ containing only positive-weight nodes, let $\widetilde{n}_b = |\mathcal{F}_b| \le m_b$ and $\widetilde{n} = \sum_b \widetilde{n}_b \le m$. The total weight for each request is bounded above by the sum of its $\widetilde{n}_b$ largest weights. These entries form a submultiset of $\mathcal{A}$ with cardinality $\widetilde{n}$, so

$$\begin{aligned} J(\mathcal{F}) &\le \sum_{b=1}^{B_t} \sum_{j=1}^{\widetilde{n}_b} \alpha_{b,j} \\ &\le \sum_{j=1}^{\widetilde{n}} \beta_j \le \sum_{j=1}^{m} \beta_j = J(\mathcal{S}^\star). \end{aligned} \tag{A.10}$$

The feasible solution returned by the algorithm attains an upper bound on the objective value of every feasible solution and is therefore optimal. □

B Forest Verification, State Commitment, and Runtime Overhead

Forest selection determines the candidates and their prefix dependencies for the current round. RF maps roots and selected candidates to verification rows, specifying attention visibility and recurrent-state reads and writes. A single target-model forward pass then computes each node's next-token distribution to determine accepted paths and outputs.

Layout and Target Verification. Let x_b^- denote the confirmed prefix excluding the root token, and $|x_b^-|$ its length. Let $\mathrm{KVRows}(x_b^-)$ denote its cached KV row indices and $\mathcal{P}_b(v)$ the new KV row indices along the path from r_b to v, inclusive. Writing $\iota(v)$ for node v's row index in the new KV layout,

$$\mathcal{P}_b(v) = \begin{cases} \{\iota(r_b)\}, & v = r_b, \\ \{\iota(r_b), \iota(v)\} \cup \{\iota(u) : u \in \mathrm{Anc}(v)\}, & v \in \mathcal{F}_b. \end{cases} \tag{B.1}$$

Let $d(v)$ denote the path depth, with $d(r_b) = 0$. The zero-based position and attention visibility are

$$\begin{aligned} \mathrm{pos}(v) &= |x_b^-| + d(v), \\ \mathrm{Visible}(v) &= \mathrm{KVRows}(x_b^-) \cup \mathcal{P}_b(v). \end{aligned} \tag{B.2}$$

Let $\mathbf{s}_b$ denote the persistent recurrent state for x_b^- and $\mathbf{s}_v$ the state after processing node v. Each node reads

$$\mathbf{s}_v^{\text{in}} = \begin{cases} \mathbf{s}_b, & v = r_b, \\ \mathbf{s}_{\text{par}(v)}, & v \in \mathcal{F}_b. \end{cases} \tag{B.3}$$

Each node writes $\mathbf{s}_v$ to a separate temporary slot. The target model performs one verification forward pass using this layout.

Acceptance and Commitment. RF determines outputs along the matched path and updates each request's persistent state. Let $\mathbf{z}_v$ denote the next-token logits at node v and $y(u)$ the token ID of node u. Using the same tie-breaking rule as target-only decoding, the greedy prediction is

$$\hat{y}_v = \arg\max_a \mathbf{z}_v[a], \tag{B.4}$$

where a ranges over the vocabulary. Starting from $v_{b,0} = r_b$, RF follows the unique matching child:

$$v_{b,j+1} \in \left\{ u \in \mathcal{C}_{v_{b,j}} \cap \mathcal{F}_b : y(u) = \hat{y}_{v_{b,j}} \right\}. \tag{B.5}$$

Traversal stops when no matching child exists. Let ℓ_b denote the number of matched candidates, with $\ell_b = 0$ if none match.

Let Stop_b truncate at the first EOS (inclusive) or the remaining generation limit. The output for this round is

$$o_b = \text{Stop}_b \left(y(v_{b,1}), \ldots, y(v_{b,\ell_b}), \hat{y}_{v_{b,\ell_b}} \right). \tag{B.6}$$

For requests that continue generation, let KV_b denote the persistent KV cache, $\text{KV}(v_{b,0}, \ldots, v_{b,\ell_b})$ the new KV entries ordered along the matched path, and $\|$ sequential concatenation. The state updates are

$$\begin{aligned} \text{KV}_b &\leftarrow \text{KV}_b \| \text{KV}(v_{b,0}, \ldots, v_{b,\ell_b}), \\ \mathbf{s}_b &\leftarrow \mathbf{s}_{v_{b,\ell_b}}. \end{aligned} \tag{B.7}$$

The final, unprocessed token $\hat{y}_{v_{b,\ell_b}}$ becomes the root token for the next round.

Selection and Layout Overhead. RF maintains a candidate queue and layout metadata. Let $N_{\text{sel}} = |\mathcal{F}|$ denote the number of selected candidates, N_{exp} the total number of candidates exposed during selection, and H the maximum queue size. With a max-heap, the selection time T_{select} and number of logical-row metadata entries N_{meta} satisfy

$$\begin{aligned} T_{\text{select}} &= O(B_t + (N_{\text{exp}} + N_{\text{sel}}) \log(2 + H)), \\ N_{\text{meta}} &= O(B_t + N_{\text{sel}}). \end{aligned} \tag{B.8}$$

Heap entries retain request identities; entries from requests that have reached their caps may be skipped when popped. Priority ties are resolved in a fixed order.

Budget and State Capacity. Forest verification also reserves memory for intermediate recurrent states. The candidate budget bounds the logical verification size in each round. By Eq. (6),

$$\begin{aligned} N_{\text{logical}} &= B_t + |\mathcal{F}| \le B_t + K_t, \\ N_{\text{physical}} &\ge N_{\text{logical}}. \end{aligned} \tag{B.9}$$

All capacities below are measured per TP rank. Let N_{state} denote the number of intermediate SSM slots reserved at startup. Let $N_{\text{state}}^{\text{native}}$ and $N_{\text{state}}^{\text{forest}}$ denote the maximum slot requirements of the two execution paths across batch sizes and layouts permitted by the startup configuration. Let c_{state} denote bytes per slot summed over the relevant layers, S_{aux} the remaining temporary verification workspace in bytes, and $\text{scratch}_{\text{RF}}$ the total temporary verification workspace. Then

$$\begin{aligned} N_{\text{state}} &\ge \max\left\{ N_{\text{state}}^{\text{native}}, N_{\text{state}}^{\text{forest}} \right\}, \\ \text{scratch}_{\text{RF}} &\approx c_{\text{state}} N_{\text{state}} + S_{\text{aux}}. \end{aligned} \tag{B.10}$$

The reserved intermediate SSM verification memory is

$$M_{\text{SSM}} = \frac{c_{\text{state}} N_{\text{state}}}{2^{30}} \text{ GiB}. \tag{B.11}$$

C Forest Construction and Execution Details

Shared-Budget Forest Construction. Algorithm 1 summarizes the shared-budget forest construction procedure used by RF. Starting from the eligible children of all request roots, RF repeatedly selects the globally highest-priority candidate while enforcing per-request caps. Whenever a node is selected, its eligible children are exposed to the global queue, which preserves prefix closure by construction.

Algorithm 1 Request Forest: Shared-Budget Forest Construction

Input: Trees $\{\mathcal{T}_b\}$, fixed priorities ρ_θ, budget K_t, caps $\{C_{b,t}\}$
Output: Prefix-closed candidate sets $\{\mathcal{F}_b\}$
Init: $\mathcal{F}_b \leftarrow \varnothing$ for $b = 1, \ldots, B_t$; $Q \leftarrow$ eligible children of roots
while $\sum_b |\mathcal{F}_b| < K_t$ and $Q \neq \varnothing$ **do**
    Extract the highest-priority node (b, v) from Q
    $\mathcal{F}_b \leftarrow \mathcal{F}_b \cup \{v\}$
    Add (b, u) to Q for each child u of v with $\rho_\theta(u) > 0$
    Remove all candidates of request b from Q **if** $|\mathcal{F}_b| = C_{b,t}$
end while
return $\{\mathcal{F}_b\}$

Table 4: Load-switching ablation. AF always uses forest verification; RF switches based on active request count. Values are geometric means of mean throughput across four benchmarks (tokens/s); Δ is relative to matched AF. Both use identical windows and budget rules.

	Qwen3.8-27B			Ornith-1.5-35B-A3B		
B	AF	RF	$\Delta\uparrow$	AF	RF	$\Delta\uparrow$
8	672.7	**723.8**	+7.6%	1301.3	**1489.2**	+14.4%
32	1514.1	**1591.9**	+5.1%	2430.0	**2878.2**	+18.4%
64	2075.7	**2089.8**	+0.7%	3695.6	**3759.8**	+1.7%

Full 19-point throughput curves and per-benchmark means $\pm$ standard deviations are reported here.

D Experimental Protocols and Implementation Details

D.1 Common Settings and Experiment Groups

All experiments use four-way tensor parallelism (TP4). For a given model and experiment, all methods share the same target model, static GPU memory fraction, output limit, and task instances. The main performance, component, window, and load-switching experiments use a static GPU memory fraction of 0.80 and an output limit of 4096 tokens. Task quality is evaluated under a separate configuration (Table 5). All methods decode at temperature 0 with random seed 42.

Table 5: Experiment groups and evaluation scope. 27B, Ornith, GLM, and Qwen3.6 refer to Qwen3.8-27B, Ornith-1.5-35B-A3B, GLM-5.3-Flash, and Qwen3.6-35B-A3B, respectively. The table lists the tested ranges; completion status is reported separately with the results.

Experiment	Model	Concurrency B	GPU Memory Fraction	Output Limit
Main performance	27B, Ornith, Qwen3.6	1–256	0.80	4096
Main performance	GLM	1–96	0.80	4096
RF / Always-forest RF	27B, Ornith	1–256	0.80	4096
Component and window ablations	27B, Ornith	64–128	0.80	4096
Extended window experiments	27B, Ornith	64–256	0.80	4096
Task quality	27B	64	0.90	8192

The main performance experiments compare decoding methods. The three $W = 8$ variants in the component ablation share the same draft checkpoint, scoring rules, and budget rules, progressively adding budget sharing across requests and branch selection. DFlash2 serves as a system baseline using the same draft model. The window comparison evaluates `global-forest` with $W = 8$ and $W = 16$, while separate extended window experiments cover higher concurrency. The RF / Always-forest RF comparison fixes the draft model, $W = 8$, and the forest allocator, varying only whether fallback to the native path is allowed. These ablations are limited to 27B and Ornith.

D.2 Model and Serving Parameters

Qwen3.8-27B and Ornith both run on four RTX 6000D GPUs in BF16, using FA4 for target attention and FlashInfer (Ye et al., 2025) for draft attention, with Triton GDN and CUDA Graph enabled. The chunked prefill size is 2048, the KV page size is 1, and the radix cache is disabled. For both models, RF reuses the corresponding original DFlash2 draft model and uses `global-forest` with $W = 8$, with entry and exit thresholds of 64 and 48, respectively. Always-forest RF uses the forest path throughout. Model-specific differences are detailed below.

Qwen3.8-27B. The context capacity is 40960 tokens. We compare Target-only, MTP-Eagle, DFlash-W4, DFlash2-W8, and RF. MTP-Eagle uses the target model's prediction head with `num-steps`$= 3$, `topk`$= 1$, and `num-draft-tokens`$= 4$. RF and DFlash2 share the Qwen3.8-27B-DFlash2 draft model. For $B = 1, 2, 4, 8, 16$, the candidate budgets K are 7, 14, 26, 48, and 80, respectively; for $B \geq 32$, $K = 3B$. The effective per-round budget K_t for each configuration is determined by Eq. (9).

Ornith-1.5-35B-A3B. The context capacity is 32768 tokens, thinking is disabled, and the candidate budget is $K = 3B$. We compare Target-only, DFlash-W4, DFlash2-W8, and RF. DFlash-W4 is evaluated at eleven concurrency levels: $B = 1, 8, 16, 32, 64, 96, 128, 160, 192, 224, 256$.

GLM-5.3-Flash. Experiments use four RTX 6000D GPUs and NVFP4. We compare Target-only, MTP-Eagle using the target model's prediction head, DFlash2, and RF. DFlash2 and RF use a window of $W = 8$. RF reuses this model's DFlash2 draft model, with entry and exit thresholds of 48 and 32. All methods are tested up to $B = 96$. DFlash2 completes measurements up to $B = 48$, while MTP-Eagle and RF complete measurements up to $B = 96$.

Qwen3.6-35B-A3B. The target model uses BF16. RF and DFlash2 share the community DFlash2 draft model with MLX affine 4-bit quantization. The quantized weights are unpacked and run in BF16, preserving the numerical effects of 4-bit quantization. We use the perf runtime, FA4 for target attention, FlashInfer for draft attention, overlap, and decode graphs. The window is $W = 8$, the global candidate budget cap is $K = 384$, and the per-round budget is $\min(7B_t, 384)$. RF uses entry and exit thresholds of 96 and 80, with two-round confirmation and a minimum dwell time of four rounds, and immediately switches to the native path when $B_t = 1$. We compare AR, the target model's built-in MTP, DFlash2, and RF. Throughput measurements exclude cold starts. SSM values report startup reservations measured separately under the same configuration.

D.3 Workloads, Measurement, and Aggregation

The main performance experiments and the component and window ablations cover GSM8K, MATH-500, HumanEval, and MT-Bench. Task quality is evaluated on the full test sets of the first three benchmarks, following the scoring protocols in Appendix H. The main throughput experiments submit $4B$ task instances under a concurrency limit of B. At the same B, all methods share the same source samples and submission order. For $B = 1$, four source samples are each submitted once. For $B = 32, 64, 240$, we use 32 source samples, each submitted 4, 8, and 30 times, respectively. Cross-method comparisons match both B and task instances. Each MT-Bench task contains two turns, with the second turn including the corresponding method's first-turn response. Output length and latency are recorded separately for each turn. Generation stops at EOS or 4096 tokens. Budget calibration uses a separate single-turn stress workload with a 1024-token output limit.

End-to-end throughput. Timing runs from task submission until all requests have completed. Let o_i denote the output length of request i and T_{wall} the measured wall-clock duration in seconds. Then

$$\text{TPS} = \frac{\sum_i o_i}{T_{\text{wall}}}. \tag{D.1}$$

Each MT-Bench turn counts as one request. Verification state is reported as the startup allocation per TP rank; the components and accounting scope are detailed in Appendix F.

Repeated measurements. The main throughput, component, extended window, and load-switching experiments use repeated measurements and report $\bar{x} \pm s$, where

$$\bar{x} = \frac{1}{4}\sum_{r=1}^{4} x_r, \qquad s^2 = \frac{1}{3}\sum_{r=1}^{4}(x_r - \bar{x})^2. \tag{D.2}$$

Complete variance statistics for all measurement points are included in the accompanying summary data. MAT, SSM, utilization, capacity, and quality results are excluded from these four-run throughput statistics. The load-switching and main throughput experiments use separate repeat records and are aggregated independently.

Paired geometric means. Let $g_{m,s}(B) > 0$ denote the mean throughput of method m on benchmark s at concurrency B; for a point measured only once, we use that measurement. At each point, we first compute the arithmetic mean across runs, then aggregate across benchmarks and concurrency levels as follows. The equally weighted geometric mean over the four benchmarks is

$$G_m(B) = \left(\prod_{s=1}^{4} g_{m,s}(B)\right)^{1/4}. \tag{D.3}$$

When both methods complete the same four benchmarks,

$$\frac{G_{\text{RF}}(B)}{G_m(B)} = \left(\prod_{s=1}^{4} \frac{g_{\text{RF},s}(B)}{g_{m,s}(B)}\right)^{1/4}. \tag{D.4}$$

Over the set of paired concurrency levels $\mathcal{B}$, the equally weighted aggregate is

$$\begin{aligned} 1 + \Delta_{\text{RF}/m} &= \exp\left(\frac{1}{|\mathcal{B}|}\sum_{B\in\mathcal{B}} \log\frac{G_{\text{RF}}(B)}{G_m(B)}\right) \\ &= \exp\left(\frac{1}{4|\mathcal{B}|}\sum_{B\in\mathcal{B}}\sum_{s=1}^{4} \log\frac{g_{\text{RF},s}(B)}{g_{m,s}(B)}\right). \end{aligned} \tag{D.5}$$

Results are aggregated separately for each model, with a relative gain of $100\Delta\%$. The Qwen3.8-27B results reported in the abstract and the Ornith comparison against DFlash2 use $\mathcal{B} = \{64, 80, 96, 112, 128, 144, 160\}$. The Ornith comparison against DFlash-W4 uses the jointly completed levels $\{64, 96, 128, 160, 192, 224, 256\}$. The Qwen3.6 high-load summary uses the ten levels with $B \geq 96$. Per-benchmark gains fix s and take the geometric mean only over the corresponding concurrency set. The component and window ablations aggregate paired ratios across the four benchmarks at $B = 64, 80, 96, 112, 128$.

MAT, utilization, and completion status. MAT is the average number of accepted draft tokens per round plus one, including the bonus token produced by the target model. Request-level utilization u is the total number of accepted draft tokens divided by the total number of draft tokens submitted for verification within a request; neither count includes bonus tokens. The distributions at $B = 128$ are characterized by p5, p25, p50, p75, and p95. In the high-concurrency speedup table in the main text, Avg. denotes the arithmetic mean over the four benchmarks; the concurrency settings are specified there. Throughput points with completed repeat measurements report the mean and sample standard deviation over four runs, while points measured only once report a single value. OOM indicates confirmed out-of-memory failures; — indicates unavailable or incomplete measurements. Neither is included in paired aggregation.

E Full Throughput Results

This section presents the full concurrency sweeps for all four models on GSM8K, MATH-500, HumanEval, and MT-Bench. Table 6 summarizes gains at matched measurement points, and Figures 6–9 show all per-benchmark measurements. Token utilization and speedup at fixed high-concurrency points are reported in Table 1; the policy comparison with Always-forest RF is provided in Appendix G.4.

E.1 Paired Throughput Gains

For each benchmark, we first compute the geometric mean of throughput ratios at matched concurrency levels, then aggregate with equal weights across the four benchmarks. The Qwen3.8 and Ornith comparisons against DFlash2 both use seven concurrency levels spanning $B = 64$–160; the Ornith comparison against DFlash uses its seven completed high-concurrency levels. For GLM, the DFlash2 comparison uses completed measurements at $B = 48$, while the MTP comparison covers $B = 48$–96. Qwen3.6 uses the ten levels with $B \geq 96$.

Table 6: Paired throughput changes (%) of RF relative to each baseline. Both per-benchmark and overall results use geometric means. The Set column specifies the concurrency levels used; each level contributes four benchmark pairs.

Model	Baseline	Set	GSM8K	MATH-500	HumanEval	MT-Bench	Overall
Qwen3.8	MTP	A	+12.7	+12.5	+13.1	+9.1	+11.8
	DFlash	A	+52.5	+55.7	+57.1	+61.3	+56.6
	DFlash2	A	+5.3	+9.1	+19.9	+22.3	+13.9
Ornith	DFlash	B	+6.3	+7.4	+1.8	+4.1	+4.9
	DFlash2	A	+15.4	+11.4	+3.4	+18.4	+12.0
GLM	MTP	D	+25.4	+8.9	+9.6	+0.4	+10.7
	DFlash2	C	+1.1	+2.3	+2.6	+3.0	+2.2
Qwen3.6	MTP	E	+35.2	+43.8	+40.6	+12.6	+32.4
	DFlash2	E	+5.5	+5.7	+5.8	+5.8	+5.7

A: $\{64, 80, 96, 112, 128, 144, 160\}$; B: $\{64, 96, 128, 160, 192, 224, 256\}$; C: $\{48\}$; D: $\{48, 64, 80, 96\}$; E: $\{96, 112, 128, 144, 160, 176, 192, 224, 240, 256\}$.

The gains on Qwen3.8 vary across benchmarks: relative to DFlash2, RF improves throughput by 5.3% and 9.1% on GSM8K and MATH-500, respectively, and by 19.9% and 22.3% on HumanEval and MT-Bench, respectively. Over the evaluated ranges, RF improves throughput by 4.9% over DFlash-W4 on Ornith and by 5.7% over DFlash2 on Qwen3.6. At the high-concurrency end, RF is 4.8% slower than MTP on Qwen3.8 at $B = 256$, while on GLM it is 1.3% and 3.7% faster than MTP at $B = 80, 96$, respectively.

Table 7: Throughput of Ornith DFlash-W4 at eleven concurrency levels (tokens/s). Points with repeated measurements report the mean ± sample standard deviation over four runs. Acceptance rates, MAT, and state allocations are reported in Tables 13 and 9.

B	GSM8K	MATH-500	HumanEval	MT-Bench
1	346.1 ± 4.9	389.2 ± 5.8	398.7 ± 6.0	342.4 ± 3.9
8	1371.7 ± 19.7	1407.4 ± 20.3	1466.3 ± 22.1	1052.2 ± 14.7
16	2029.8 ± 23.6	1662.9 ± 32.1	1992.3 ± 33.6	1560.5 ± 21.4
32	2880.7 ± 41.5	2561.4 ± 42.8	3141.7 ± 42.3	2069.3 ± 23.9
64	4060.6 ± 48.0	3844.2 ± 45.4	3991.7 ± 60.4	3151.8 ± 44.9
96	4442.1 ± 58.1	4585.0 ± 57.0	4399.4 ± 55.4	3541.3 ± 53.1
128	5141.6 ± 68.1	4947.1 ± 68.1	4817.6 ± 72.5	4027.3 ± 53.4
160	5572.6 ± 67.7	6080.9 ± 86.8	5520.6 ± 80.7	4377.0 ± 57.3
192	5915.4 ± 85.9	5804.2 ± 70.8	5444.7 ± 71.0	4643.6 ± 67.9
224	6090.1 ± 80.6	6054.7 ± 91.8	6185.8 ± 91.0	4807.3 ± 62.7
256	6472.3 ± 95.6	6455.9 ± 86.3	6321.9 ± 95.6	5109.0 ± 70.8

E.2 Concurrency Sweeps Across Four Benchmarks

The curves show mean throughput on a linear concurrency axis, with error bars indicating one sample standard deviation and gaps at missing measurements. The concurrency grid for Qwen3.8, Ornith, and Qwen3.6 is $B = 1, 2, 4, 8, 16, 32, 48, 64, 80, 96, 112, 128, 144, 160, 176, 192, 224, 240, 256$; GLM uses the ten levels with $B \leq 96$ from this grid. Per-point throughput values, OOM outcomes, and unmeasured status are recorded in the accompanying data file `data/throughput.csv`; the configuration index is provided in `data/configurations.json`.

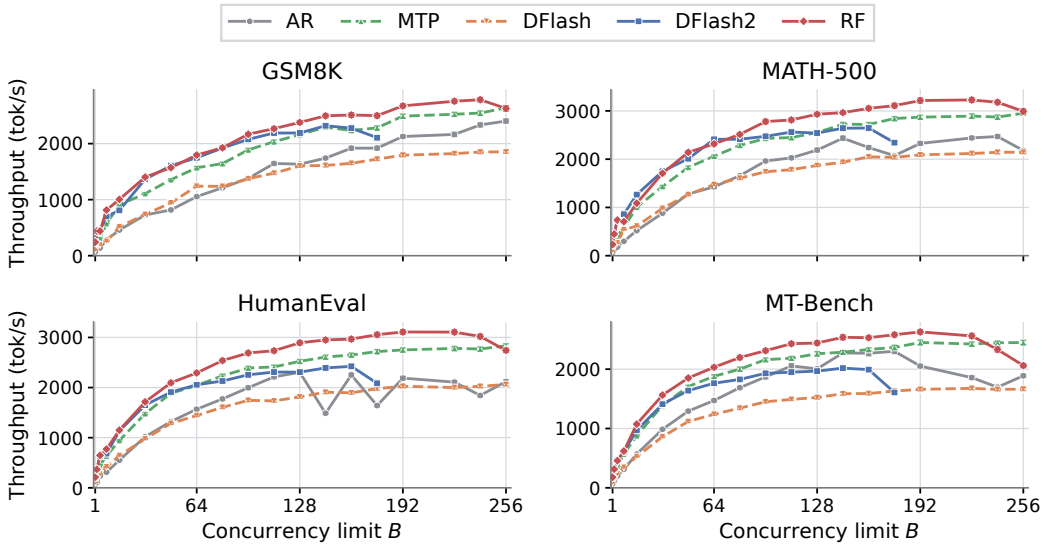

Figure 6: Throughput of Qwen3.8-27B across four benchmarks. DFlash2 encounters OOM at $B = 192, 224, 240, 256$; all other methods complete all 19 concurrency levels.

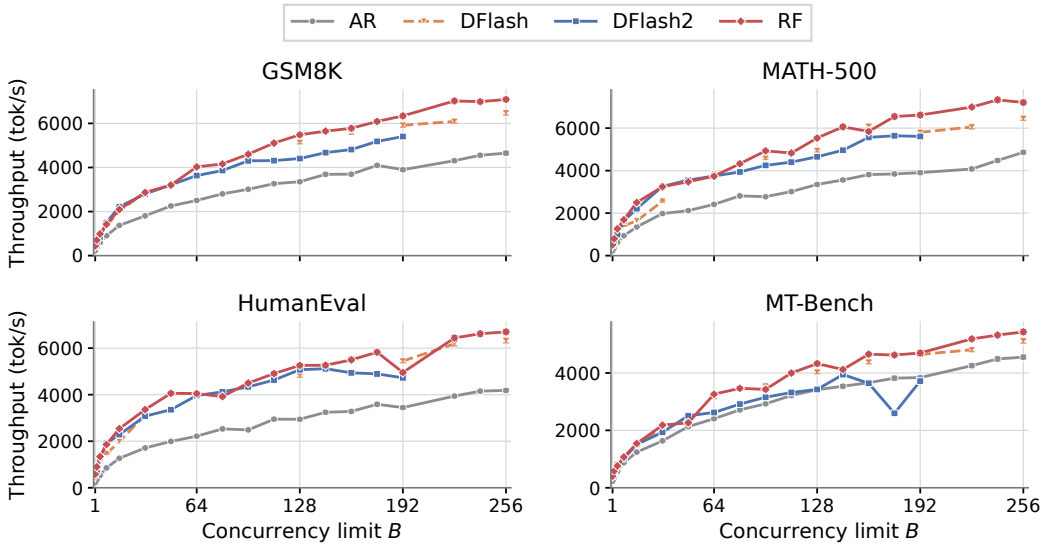

Figure 7: Throughput of Ornith-1.5-35B-A3B across four benchmarks. Measurements for DFlash-W4 at eleven concurrency levels are reported in Table 7; DFlash2 encounters OOM at $B = 224, 240, 256$.

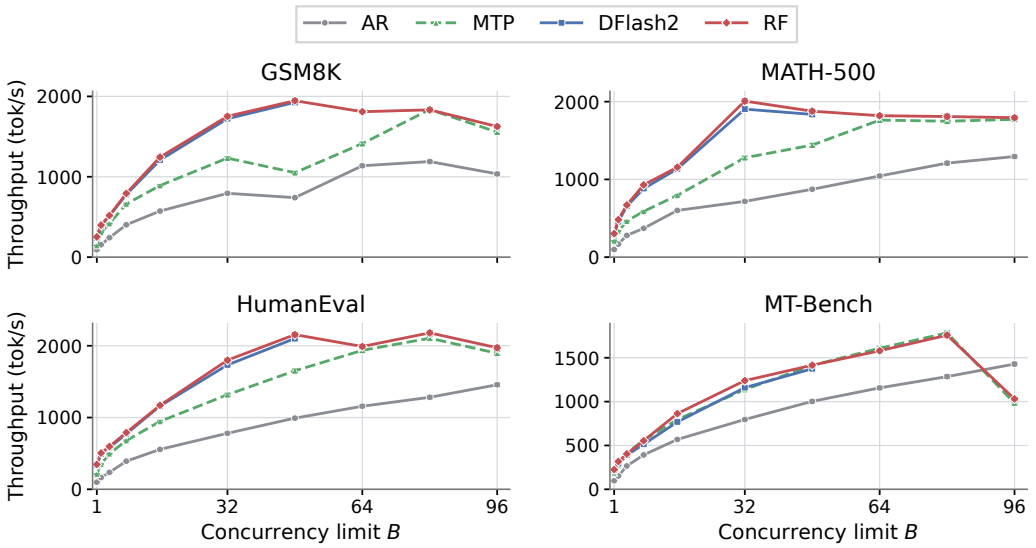

Figure 8: Throughput of GLM-5.3-Flash across four benchmarks. DFlash2 completes measurements up to $B = 48$, with measurements at $B = 64, 80, 96$ incomplete; AR, MTP-Eagle, and RF complete all ten concurrency levels.

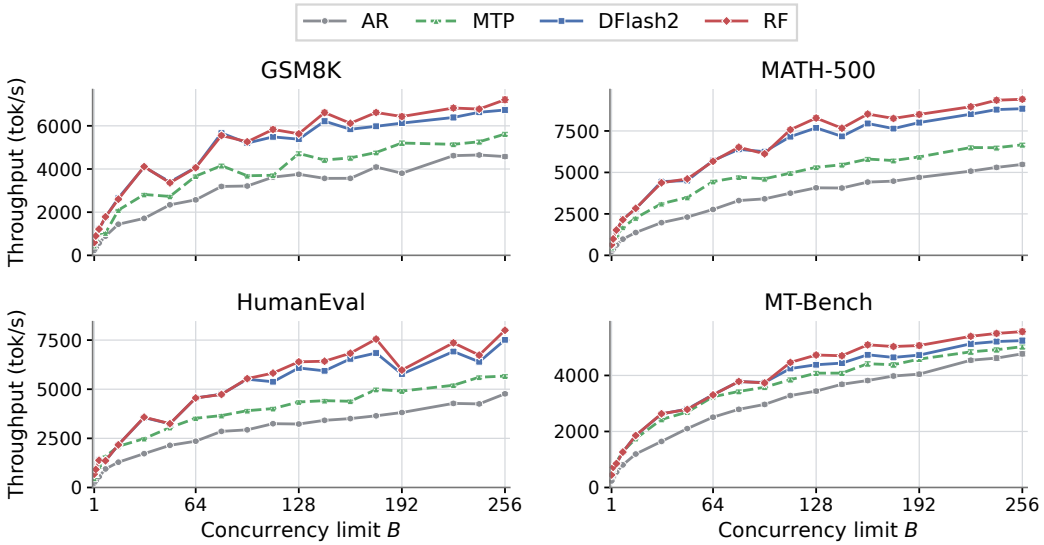

Figure 9: Throughput of Qwen3.6-35B-A3B across four benchmarks. All four methods complete all 19 concurrency levels. RF and DFlash2 share the community draft model with weights unpacked into BF16. RF uses entry and exit thresholds of 96 and 80, respectively, and a budget cap of $K = 384$.

F State Overhead and Candidate Utilization

F.1 Persistent SSM and Intermediate Verification SSM

All allocations below are reported in GiB per TP rank. For a given model, method, and concurrency configuration, all four benchmarks use the same state caches. Persistent SSM and intermediate verification SSM are accounted for separately. The intermediate component measures the SSM cache reservation at server startup; model weights, the KV cache, persistent recurrent states, and convolution caches are accounted for separately. RF switches paths according to load, while Always-forest RF uses the forest path throughout. Figure 5 shows only the intermediate verification component.

Let S_{RF} and $S_m > 0$ denote the intermediate verification SSM allocations of RF and the baseline, respectively, for the same model and concurrency level. The relative saving is

$$\delta_{S,\text{RF}/m} = \frac{S_m - S_{\text{RF}}}{S_m} = 1 - \frac{S_{\text{RF}}}{S_m}. \tag{F.1}$$

At $B = 128$, the allocation for Qwen3.8-27B decreases from 36.28 GiB with DFlash2 to 18.00 GiB with Always-forest RF, while that for Ornith decreases from 15.12 GiB to 7.49 GiB, representing reductions of 50.4% and 50.5%, respectively. For Ornith at $B = 192$, the allocation decreases from 22.62 GiB to 11.25 GiB, a reduction of 50.3%. Under this configuration, Always-forest RF achieves 15.1% higher geometric mean throughput across the four benchmarks than DFlash2 and also completes measurements at $B = 224, 240, 256$. State allocations for these three levels are reported in Table 9.

Qwen3.8-27B. Table 8 reports intermediate verification SSM allocations. MTP-Eagle and DFlash both use $W = 4$ and have identical allocations, so they are combined into one column.

Table 8: Intermediate verification SSM for Qwen3.8-27B (GiB per TP rank). MTP-Eagle and DFlash share one column; Always-forest RF is a separate baseline for the load-policy comparison. OOM indicates confirmed out-of-memory failures.

B	MTP-Eagle / DFlash	DFlash2	RF	Always-forest RF
1	0.28	0.56	0.28	0.28
2	0.42	0.84	0.56	0.56
4	0.70	1.41	1.12	1.12
8	1.27	2.53	2.25	1.97
16	2.39	4.78	4.50	3.38
32	4.64	9.28	9.00	4.50
48	6.89	13.78	13.50	6.75
64	9.14	18.28	17.72	9.00
80	11.39	22.78	17.72	11.25
96	13.64	27.28	17.72	13.50
112	15.89	31.78	17.72	15.75
128	18.14	36.28	18.00	18.00
144	20.39	40.78	20.25	20.25
160	22.64	45.28	22.50	22.50
176	24.89	49.78	24.75	24.75
192	27.14	OOM	27.00	27.00
224	31.64	OOM	31.50	31.50
240	33.89	OOM	33.75	33.75
256	36.14	OOM	36.00	36.00

Ornith-1.5-35B-A3B. Table 9 reports the intermediate verification SSM memory for all methods. For Ornith, the temporary and persistent caches correspond to `intermediate_ssm_state_cache` and `ssm_state`, respectively; the allocations are identical across all four TP ranks. All methods use the static memory fraction of 0.80 specified in Appendix D.1. The acceptance rate of DFlash-W4 is reported separately in Table 13.

Table 9: SSM memory allocation for Ornith-1.5-35B-A3B (GiB / TP rank). Only DFlash-W4 reports persistent SSM memory separately; all other columns report intermediate verification SSM memory. OOM denotes confirmed out-of-memory failure.

B	DFlash-W4 Intermediate	DFlash-W4 Persistent	DFlash2 Intermediate	RF Intermediate	Always-RF Intermediate
1	0.12	0.03	0.23	0.12	0.12
8	0.53	0.13	1.05	0.94	0.70
16	1.00	0.25	2.67	1.88	1.41
32	1.93	0.48	3.87	3.75	2.81
64	3.81	0.95	7.62	7.38	3.75
96	5.68	1.42	11.37	7.38	5.62
128	7.56	1.89	15.12	7.49	7.49
160	9.43	2.36	18.87	9.38	9.38
192	11.31	2.83	22.62	11.25	11.25
224	13.18	3.30	OOM	13.12	13.12
256	15.06	3.76	OOM	15.00	15.00

GLM-5.3-Flash. Table 10 separately reports persistent and intermediate verification SSM memory. DFlash2 does not complete at $B = 64, 80, 96$ due to insufficient memory and is therefore marked as OOM.

Table 10: SSM memory allocation per TP rank for GLM-5.3-Flash (GiB). Persistent SSM memory and intermediate verification SSM memory are reported separately for each method. Figure 5 in the main text shows only the intermediate verification SSM memory.

	MTP-Eagle		**DFlash2**		**RF**	
B	**Persistent**	**Intermediate**	**Persistent**	**Intermediate**	**Persistent**	**Intermediate**
1	0.07	0.27	0.07	0.55	0.07	0.56
2	0.10	0.41	0.10	0.82	0.10	0.81
4	0.17	0.68	0.17	1.37	0.17	1.38
8	0.31	1.23	0.31	2.46	0.30	2.48
16	0.58	2.32	0.58	4.65	0.56	4.61
32	1.13	4.51	1.13	9.02	1.10	8.97
48	1.67	6.70	1.67	13.40	1.63	13.36
64	2.22	8.89	OOM	OOM	2.16	8.50
80	2.77	11.07	OOM	OOM	2.69	10.62
96	3.32	13.26	OOM	OOM	3.22	12.75

Qwen3.6-35B-A3B. The state allocations in Table 11 were measured independently of throughput, are identical across the four TP ranks, and are rounded to 0.01 GiB. Persistent SSM allocations are identical across all four methods and are shown in a single column; intermediate verification states for AR were not recorded. For $B \geq 96$, RF's intermediate verification SSM remains at 11.13 GiB; at $B = 256$, DFlash2 and MTP allocate 30.12 and 15.06 GiB, respectively, and RF reduces the allocation by 63.0% relative to DFlash2. At this point, all four methods allocate 3.76 GiB for persistent SSM. Including persistent and intermediate verification convolution caches, the recorded cache totals for DFlash2 and RF are 34.26 and 15.69 GiB, respectively, a reduction of 54.2%; these totals exclude model weights and the KV cache.

Table 11: SSM allocations for Qwen3.6-35B-A3B (GiB / TP rank). The persistent allocation is identical for AR, MTP, DFlash2, and RF. The remaining columns report intermediate verification allocations; AR intermediate states were not recorded. Model weights, the KV cache, and convolution caches are excluded.

B	Common Persistent	AR Intermediate	MTP Intermediate	DFlash2 Intermediate	RF Intermediate
1	0.03	—	0.12	0.23	0.12
2	0.04	—	0.18	0.35	0.23
4	0.07	—	0.29	0.59	0.47
8	0.13	—	0.53	1.05	0.94
16	0.25	—	1.00	1.99	1.88
32	0.48	—	1.93	3.87	3.75
48	0.72	—	2.87	5.74	5.62
64	0.95	—	3.81	7.62	7.50
80	1.19	—	4.75	9.49	9.38
96	1.42	—	5.68	11.37	11.13
112	1.66	—	6.62	13.24	11.13
128	1.89	—	7.56	15.12	11.13
144	2.12	—	8.50	16.99	11.13
160	2.36	—	9.43	18.87	11.13
176	2.59	—	10.37	20.74	11.13
192	2.83	—	11.31	22.62	11.13
224	3.30	—	13.18	26.37	11.13
240	3.53	—	14.12	28.24	11.13
256	3.76	—	15.06	30.12	11.13

F.2 Candidate Utilization

Request-level distributions. Request-level utilization u is the ratio of the total number of accepted draft tokens to the total number of draft tokens submitted for verification within the same request, with both counts excluding bonus tokens produced by the target model. Table 12 reports request-level quantiles for Qwen3.8 and Qwen3.6 at $B = 128$; p50 matches the corresponding utilization in Table 1. Each method has $n = 512$ requests on each of HumanEval, GSM8K, and MATH-500, and $n = 1024$ records on MT-Bench. The table reports p5, p25, p50, p75, and p95, where p50 is the median.

Table 12: Quantiles of request-level token utilization, with p50 denoting the median.

Benchmark	Method	p5	p25	p50	p75	p95
Qwen3.8-27B						
HumanEval	RF	0.7647	0.8127	0.8448	0.8720	0.9082
HumanEval	DFlash2	0.3978	0.4514	0.5034	0.5660	0.7287
HumanEval	DFlash	0.2562	0.2896	0.3342	0.3899	0.5077
GSM8K	RF	0.7983	0.8660	0.8991	0.9326	0.9724
GSM8K	DFlash2	0.4996	0.5810	0.6536	0.7253	0.7884
GSM8K	DFlash	0.2973	0.3701	0.4142	0.4779	0.5358
MATH-500	RF	0.8344	0.8774	0.9030	0.9248	0.9588
MATH-500	DFlash2	0.4658	0.5848	0.6613	0.7238	0.7725
MATH-500	DFlash	0.2613	0.3601	0.4274	0.4884	0.5909
MT-Bench	RF	0.6618	0.7232	0.7725	0.8260	0.9172
MT-Bench	DFlash2	0.2948	0.3755	0.4512	0.5528	0.7279
MT-Bench	DFlash	0.1659	0.2072	0.2558	0.3501	0.4859
Qwen3.6-35B-A3B						
HumanEval	RF	0.8071	0.9870	0.9917	0.9956	1.0000
HumanEval	DFlash2	0.6684	0.8704	0.9217	0.9643	0.9768
GSM8K	RF	0.7981	0.9465	0.9685	0.9809	0.9950
GSM8K	DFlash2	0.5664	0.6518	0.7218	0.7825	0.8352
MATH-500	RF	0.8344	0.9765	0.9869	0.9918	0.9974
MATH-500	DFlash2	0.6475	0.7247	0.7724	0.8061	0.8537
MT-Bench	RF	0.4383	0.7234	0.8250	0.9001	0.9508
MT-Bench	DFlash2	0.2267	0.3456	0.5055	0.7335	0.8651

Acceptance rate and MAT of Ornith DFlash-W4. Table 13 reports token-level draft acceptance rates and MAT across the four benchmarks. The acceptance rate is the total number of accepted draft tokens divided by the total number submitted for verification, with both counts excluding bonus tokens; MAT is computed as `output_tokens/spec_verify_ct` and includes the token produced by the target model in each round. All concurrency levels follow the same protocol as the main Ornith experiments.

Table 13: Draft acceptance rate and MAT of Ornith DFlash-W4. Acceptance rates exclude bonus tokens, while MAT includes them. TP4 and a static GPU memory fraction of 0.80 are used; the remaining protocol is given in Appendix D.1.

B	GSM8K		MATH-500		HumanEval		MT-Bench	
	Acceptance Rate	MAT	Acceptance Rate	MAT	Acceptance Rate	MAT	Acceptance Rate	MAT
1	0.719	3.15	0.787	3.36	0.920	3.75	0.682	3.04
8	0.750	3.24	0.776	3.33	0.944	3.83	0.573	2.72
16	0.740	3.22	0.774	3.32	0.931	3.79	0.552	2.65
32	0.745	3.23	0.761	3.28	0.949	3.84	0.539	2.61
64	0.746	3.23	0.769	3.31	0.946	3.83	0.540	2.62
96	0.743	3.23	0.761	3.28	0.941	3.82	0.531	2.59
128	0.740	3.24	0.760	3.28	0.903	3.82	0.526	2.58
160	0.750	3.25	0.759	3.27	0.946	3.83	0.529	2.58
192	0.748	3.24	0.761	3.28	0.944	3.82	0.531	2.59
224	0.748	3.24	0.757	3.27	0.942	3.82	0.532	2.60
256	0.749	3.25	0.760	3.28	0.943	3.82	0.526	2.58

F.3 State Layout and Capacity Limits at High Concurrency

State rows and allocations. For Qwen3.8-27B, each TP rank aggregates SSM states from 48 layers into one row, which occupies 36 MiB. Let P denote the number of reserved rows; the corresponding allocation is

$$M_{\text{SSM}} = P \times \frac{36}{1024}\ \text{GiB}, \qquad 1024\,\text{MiB} = 1\,\text{GiB}. \tag{F.2}$$

With one padding request slot included, MTP/DFlash uses $P = 4(B+1)$, DFlash2 uses $P = 8(B+1)$, and Always-forest RF uses $P = 8\lceil (B + \min\{7B, K\})/8 \rceil$. At $B = 16$ and $K = 80$, Always-forest RF uses 96 rows (3.375 GiB), MTP/DFlash uses 68 rows (approximately 2.39 GiB), and DFlash2 uses 136 rows (approximately 4.78 GiB). At $B = 1$ and $K = 7$, both Always-forest RF and MTP/DFlash use 8 rows (0.28 GiB); for $B \geq 32$ and $K = 3B$, Always-forest RF uses $4B$ rows, approximately half the row count of DFlash2 at the same concurrency. These row counts are determined at startup. RF uses an upper bound covering both the P allowed by the native path and the P required by the compact forest layout, then converts it using Eq. (F.2); for Qwen at $B = 64$–112, this gives 17.72 GiB, exceeding the allocation for the $4B$ rows of Always-forest RF at the same concurrency (the row counts correspond to 9.00 GiB at $B = 64$ and 15.75 GiB at $B = 112$); from $B = 128$ onward, the two match at $4B$ rows (18.00 GiB at $B = 128$).

The following state row counts, KV capacity formulas, and serving overhead analysis correspond to the Qwen3.8-27B Always-forest RF configurations at $B = 224, 240, 256$. All three points use $W = 8$, $K = 3B$, a static GPU memory fraction of 0.80, four RTX 6000D GPUs, and TP4. Client concurrency, the server request limit, and the maximum captured batch size are all set to B.

State and KV capacity. Aligning the intermediate SSM row count for root positions and non-root candidates to the window size gives

$$P(B) = 8\left\lceil \frac{B + \min\{7B, 3B\}}{8} \right\rceil = 4B, \qquad B \in \{224, 240, 256\}. \tag{F.3}$$

Persistent states occupy an additional $B+1$ slots, and intermediate convolution states occupy $8(B+1)$ positions. Let s and c denote the bytes per SSM and convolution state slot, respectively, after aggregating all linear attention layers on each TP rank. Then

$$\begin{aligned} M_{\text{state}}(B) &= (B+1)(s+c) + 4Bs + 8(B+1)c \\ &= (5B+1)s + 9(B+1)c, \\ M_{\text{state}}(B+\Delta B) - M_{\text{state}}(B) &= \Delta B(5s + 9c). \end{aligned} \tag{F.4}$$

Each TP rank has 48 linear attention layers; each layer's SSM is an FP32 tensor of shape $(12, 128, 128)$, and its convolution state is a BF16 tensor of shape $(2560, 3)$, giving $s = 36$ MiB and $c = 0.703125$ MiB. Tree verification retains a separate convolution window. The intermediate

SSM startup allocations at the three points are 31.50, 33.75, and 36.00 GiB, respectively, consistent with Eq. (F.3).

The runtime first deducts persistent and intermediate states from the cache budget, then allocates the target and draft KV pools, and finally captures CUDA Graphs. Let $M_0(B)$ denote the cache budget after deducting other reservations, and let q denote the combined bytes per token across the two KV pools. With a page size of 1 and no other capacity limits reached,

$$C(B) = \left\lfloor \frac{M_0(B) - M_{\text{state}}(B)}{q} \right\rfloor, \qquad C(B + \Delta B) - C(B) = -\frac{\Delta B(5s + 9c)}{q} + \varepsilon, \quad |\varepsilon| < 1. \tag{F.5}$$

The second relation requires $M_0(B + \Delta B) = M_0(B)$, where ε accounts for the difference due to flooring. The target and draft KV pools require 16 and 5 KiB/token, respectively, so $q = 21$ KiB/token. Each increase of 16 in B raises the state reservation by approximately 2.91 GiB, corresponding to approximately 145371.43 KV tokens. The measured KV capacities at the three points are 530791, 385420, and 240048, decreasing by 145371 and 145372, respectively, consistent with the rounding error in the formula. These capacities are determined before graph capture and remain unchanged afterward.

Table 14 reports capacity, mean time to first token, and the number of request retractions.

Table 14: Capacity and mean time to first token of Always-forest RF on Qwen3.8-27B. The static GPU memory fraction is 0.80; SSM denotes the intermediate state allocation per TP rank, and retractions count all request retractions over the entire run, as recorded on TP0.

B	KV Capacity	Intermediate SSM	Mean TTFT (s)				Retractions
	(token)	(GiB/rank)	GSM8K	MATH-500	HumanEval	MTBench	(TP0)
224	530791	31.50	2.1	2.2	2.9	21.4	0
240	385420	33.75	7.0	7.4	14.3	67.8	0
256	240048	36.00	13.5	25.3	46.5	126.9	30

G Component Ablations and Parameter Analysis

This section examines budget sharing, branch selection, window size, and load policy separately. All experiments use Qwen3.8-27B and Ornith, TP4, BF16, and a static GPU memory fraction of 0.80; the serving and workload protocols are provided in Appendix D.1. The component experiments cover $B = 64, 80, 96, 112, 128$; the separate extended window experiments cover $B = 64$–256; the load-policy experiments cover all 19 concurrency levels.

G.1 Incremental Component Ablations

With the draft model, scoring, $W = 8$, total budget $K = 3B$, and state allocation at each matched point fixed, we progressively add budget sharing and branch selection to a single-chain baseline with fixed quotas. Table 2 in the main text summarizes the three stages and the window comparison, while Figure 10 shows trends across concurrency levels. DFlash2 serves as a system baseline using the same draft model, and the global-forest window is additionally extended to 16 for the window comparison.

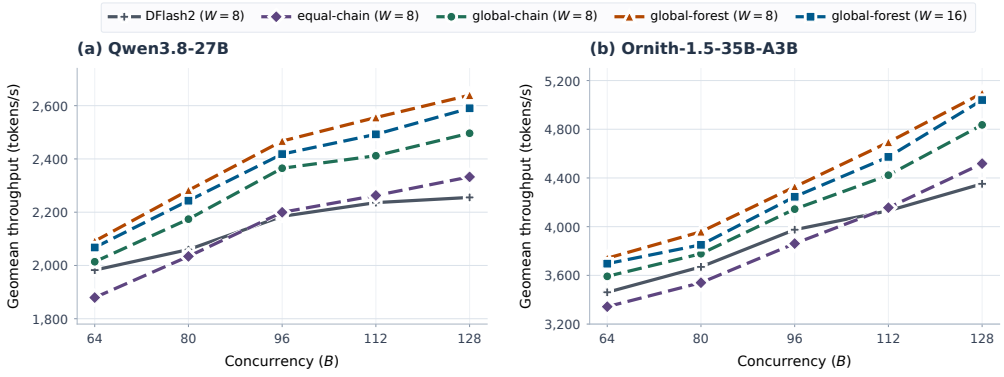

Figure 10: Throughput with components added incrementally. The first three variants are equal-chain with fixed per-request quotas, global-chain with budget sharing, and global-forest with branch selection additionally enabled (all with $W = 8$). DFlash2 serves as the system baseline, and global-forest ($W = 16$) provides the window comparison. The curves show the geometric mean of mean throughput across the four benchmarks, using TP4, BF16, and a static GPU memory fraction of 0.80.

Starting point: single chains with fixed quotas (equal-chain). Fix the candidate trees, node weights, number of active requests B_t, and effective total budget K_t for a round. Request b receives a nontransferable integer quota

$$C_{b,t}^{\text{equal}} = \lfloor K_t / B_t \rfloor + \mathbf{1}[b \le K_t \bmod B_t]. \tag{G.1}$$

The quotas sum to K_t and differ by at most 1 across requests; since $K_t \le B_t(W-1)$, each quota is at most $W - 1$. For each parent node, chain retains only the child with the highest conditional score in the original candidate set; the retained node's score remains normalized over the full set of candidate children.

Adding budget sharing (global-chain). Keeping the single-chain candidate sets, we replace fixed quotas with a shared total budget and per-request capacity caps $C_{b,t} = \min\{K_t, W-1\}$. Requests can compete for budget left unused by other requests; relative to the starting point, the paired geometric mean throughput improves by 7.0% and 7.0% for the two models, respectively.

Adding branch selection (global-forest). Keeping the shared budget and scoring rules, we allow multiple candidate children of the same parent to compete globally, further improving the paired geometric mean throughput by 5.0% and 5.0% for the two models, respectively. Both stages yield gains across all 20 benchmark–concurrency pairs. The per-round feasible sets of the three stages

satisfy

$$\mathfrak{D}_{\text{equal-chain}} \subseteq \mathfrak{D}_{\text{global-chain}} \subseteq \mathfrak{D}_{\text{global-forest}}. \tag{G.2}$$

Thus, under fixed scores, the optimal proxy value cannot decrease as the constraints are relaxed; throughput gains are established by matched measurements.

G.2 Full Component Ablation Results

Table 15 and Table 16 report absolute throughput for each benchmark at the three component stages, alongside the DFlash2 system baseline and the $W = 16$ window baseline. Table 17 reports intermediate verification SSM per TP rank; the three $W = 8$ stages have identical allocations at each matched point, shared across all four benchmarks.

Table 15: Per-benchmark throughput for the component ablations on Qwen3.8-27B (tokens/s, mean $\pm$ sample standard deviation over four runs). All columns use $W = 8$ except the last.

B	Benchmark	DFlash2	equal-chain	global-chain	global-forest ($W = 8$)	global-forest ($W = 16$)
64	GSM8K	1756.8 ± 17.4	1613.8 ± 21.1	1730.7 ± 21.8	1786.1 ± 17.6	1771.4 ± 19.9
	MATH-500	2422.5 ± 27.8	2052.8 ± 30.9	2206.2 ± 36.5	2295.9 ± 31.1	2269.7 ± 34.6
	HumanEval	2060.8 ± 28.5	2069.6 ± 29.8	2202.7 ± 28.6	2295.1 ± 29.5	2252.7 ± 34.8
	MT-Bench	1761.9 ± 25.9	1820.0 ± 29.1	1956.1 ± 25.6	2033.0 ± 30.9	2017.6 ± 26.6
80	GSM8K	1914.7 ± 20.1	1722.8 ± 18.6	1850.5 ± 28.1	1934.0 ± 23.3	1913.2 ± 14.3
	MATH-500	2394.0 ± 38.8	2248.1 ± 29.6	2393.2 ± 34.5	2538.8 ± 35.5	2474.6 ± 37.5
	HumanEval	2147.5 ± 25.5	2272.4 ± 34.6	2441.1 ± 35.8	2515.5 ± 30.6	2498.9 ± 34.6
	MT-Bench	1828.2 ± 20.2	1945.4 ± 31.0	2066.1 ± 30.4	2196.2 ± 24.8	2140.5 ± 29.4
96	GSM8K	2091.3 ± 27.3	1923.1 ± 25.3	2076.6 ± 17.0	2159.2 ± 28.7	2105.7 ± 29.1
	MATH-500	2492.2 ± 42.1	2468.0 ± 33.6	2666.0 ± 36.2	2755.3 ± 34.4	2698.4 ± 40.6
	HumanEval	2257.6 ± 32.8	2346.9 ± 36.4	2515.3 ± 28.4	2693.4 ± 34.7	2633.8 ± 40.6
	MT-Bench	1935.5 ± 29.1	2101.6 ± 29.9	2246.9 ± 37.5	2312.4 ± 25.7	2285.0 ± 32.7
112	GSM8K	2183.9 ± 19.5	2020.9 ± 26.0	2149.5 ± 26.4	2288.9 ± 27.5	2239.8 ± 24.1
	MATH-500	2544.5 ± 38.9	2496.7 ± 37.0	2649.1 ± 38.0	2826.0 ± 38.7	2725.1 ± 42.8
	HumanEval	2307.4 ± 25.5	2403.2 ± 33.9	2621.3 ± 39.6	2715.4 ± 32.8	2656.9 ± 31.0
	MT-Bench	1949.6 ± 30.3	2162.5 ± 24.2	2267.5 ± 31.0	2428.2 ± 41.7	2379.4 ± 31.6
128	GSM8K	2219.8 ± 17.0	2100.1 ± 29.3	2239.7 ± 20.8	2365.7 ± 23.3	2313.3 ± 26.1
	MATH-500	2563.4 ± 36.6	2582.4 ± 41.5	2781.8 ± 32.9	2909.9 ± 40.4	2887.2 ± 45.0
	HumanEval	2310.4 ± 28.7	2489.9 ± 33.3	2688.6 ± 41.6	2890.6 ± 41.8	2824.9 ± 34.1
	MT-Bench	1968.5 ± 21.8	2192.1 ± 38.6	2318.8 ± 33.8	2438.3 ± 29.5	2385.8 ± 40.0

Table 16: Per-benchmark throughput for the component ablations on Ornith-1.5-35B-A3B (tokens/s, mean ± sample standard deviation over four runs). All columns use $W = 8$ except the last.

B	Benchmark	DFlash2	equal-chain	global-chain	global-forest ($W = 8$)	global-forest ($W = 16$)
64	GSM8K	3648.6 ± 49.1	3556.5 ± 50.6	3815.5 ± 46.4	4008.6 ± 55.5	3977.3 ± 52.7
	MATH-500	3752.0 ± 49.0	3340.1 ± 41.9	3568.8 ± 37.7	3716.8 ± 39.1	3689.3 ± 50.0
	HumanEval	3995.7 ± 54.2	3591.3 ± 41.8	3870.5 ± 51.4	4026.8 ± 59.5	3955.5 ± 52.6
	MT-Bench	2623.3 ± 32.9	2927.0 ± 41.1	3158.2 ± 48.4	3264.1 ± 33.0	3215.8 ± 45.6
80	GSM8K	3858.8 ± 52.3	3718.7 ± 45.0	3949.0 ± 50.8	4142.7 ± 45.8	4013.0 ± 42.6
	MATH-500	3967.7 ± 51.5	3869.1 ± 46.8	4171.9 ± 46.9	4294.7 ± 55.1	4211.1 ± 58.9
	HumanEval	4092.8 ± 51.1	3508.7 ± 53.3	3730.0 ± 50.1	3951.4 ± 44.4	3805.1 ± 63.0
	MT-Bench	2894.2 ± 45.2	3110.3 ± 36.6	3313.8 ± 49.4	3491.6 ± 45.5	3419.9 ± 38.2
96	GSM8K	4301.8 ± 60.7	4113.1 ± 55.6	4407.7 ± 49.6	4600.3 ± 67.4	4558.8 ± 56.0
	MATH-500	4253.8 ± 45.1	4350.1 ± 51.2	4637.4 ± 54.0	4922.5 ± 49.4	4745.8 ± 60.8
	HumanEval	4332.8 ± 55.0	4042.6 ± 52.0	4342.2 ± 63.3	4507.8 ± 63.7	4443.9 ± 57.0
	MT-Bench	3149.2 ± 46.5	3072.6 ± 43.1	3322.5 ± 38.1	3433.5 ± 44.8	3379.0 ± 48.2
112	GSM8K	4319.3 ± 56.8	4460.8 ± 55.1	4726.0 ± 60.2	5092.2 ± 61.2	4941.7 ± 70.9
	MATH-500	4435.2 ± 64.1	4286.9 ± 46.0	4619.7 ± 50.5	4807.4 ± 49.4	4717.9 ± 56.6
	HumanEval	4601.9 ± 64.0	4386.4 ± 66.1	4633.0 ± 65.5	4902.4 ± 53.4	4802.0 ± 63.2
	MT-Bench	3301.4 ± 50.2	3556.6 ± 45.6	3783.1 ± 52.8	4042.1 ± 52.4	3906.1 ± 45.7
128	GSM8K	4430.9 ± 63.3	4763.8 ± 31.2	5134.2 ± 53.5	5452.0 ± 59.7	5358.6 ± 59.2
	MATH-500	4654.6 ± 52.4	4881.7 ± 60.0	5221.4 ± 51.6	5450.3 ± 42.6	5475.6 ± 41.5
	HumanEval	5075.6 ± 73.4	4648.5 ± 72.4	4924.2 ± 52.7	5257.9 ± 64.7	5130.9 ± 85.3
	MT-Bench	3426.5 ± 49.4	3854.0 ± 52.4	4143.0 ± 53.1	4315.3 ± 57.3	4285.7 ± 58.5

Table 17: Intermediate verification SSM for the component ablations (GiB / TP rank). EC, GC, and GF denote equal-chain, global-chain, and global-forest, respectively. The three $W = 8$ variants have identical allocations at each matched point, shared across all four benchmarks.

Model	B	DFlash2 ($W = 8$)	EC ($W = 8$)	GC ($W = 8$)	GF ($W = 8$)	GF ($W = 16$)
Qwen	64	18.28	17.72	17.72	17.72	35.44
	80	22.78	17.72	17.72	17.72	35.44
	96	27.28	17.72	17.72	17.72	35.44
	112	31.78	17.72	17.72	17.72	35.44
	128	36.28	18.00	18.00	18.00	35.44
Ornith	64	7.62	7.38	7.38	7.38	14.77
	80	9.12	7.38	7.38	7.38	14.77
	96	11.37	7.38	7.38	7.38	14.77
	112	13.52	7.38	7.38	7.38	14.77
	128	15.12	7.49	7.49	7.49	14.77

G.3 Window Size and Concurrency Range

In the window comparison at five concurrency levels spanning $B = 64$–128, with global-forest fixed, the paired geometric mean throughput of $W = 8$ is higher than that of $W = 16$ by 1.9% and 1.9% on Qwen and Ornith, respectively. Increasing the window size increases intermediate verification state allocations (Table 17) but does not improve throughput over this concurrency range.

Table 18 presents an independent extended window experiment covering $B = 64, 80, 96, 112, 128, 144, 160, 176, 192, 224, 240, 256$. The two configurations share the same draft model, $K = 3B$, and load-switching rules, changing only the forest-path window from 8 to 16; RF and GF16 in the table denote these two window settings, respectively. Measurements for this experiment are collected independently of the preceding component experiments over a smaller range, and their respective W8 values at shared B are retained separately. On Qwen, W8 is faster than W16 at all nine points spanning $B = 64$–192, with geometric mean gains across the four benchmarks of 1.6%–3.3%; W16 encounters OOM at the last three points; on Ornith, the gains at the eight points spanning $B = 64$–176 are 1.6%–2.3%, and W16 encounters OOM at the last four points.

Table 18: Throughput across window sizes and concurrency levels (tokens/s, mean ± sample standard deviation over four runs). RF and GF16 use $W = 8$ and $W = 16$, respectively; both switch paths according to load. Avg. is the geometric mean of the mean throughput values across the four benchmarks; yellow highlighting identifies RF, and superscripts indicate its gains over GF16 (%); OOM indicates measurements not completed due to insufficient GPU memory.

B	Method	GSM8K	MATH-500	HumanEval	MT-Bench	Avg.
Qwen3.8-27B						
64	RF	1794.6 ± 18.4$^{\uparrow 2.2}$	2307.4 ± 23.6$^{\uparrow 2.3}$	2287.1 ± 23.4$^{\uparrow 1.6}$	2044.7 ± 29.2$^{\uparrow 2.1}$	**2097.7**$^{\uparrow 2.0}$
	GF16	1755.1 ± 21.3	2256.1 ± 27.5	2251.8 ± 27.4	2003.6 ± 33.4	2055.9
80	RF	1942.7 ± 21.5$^{\uparrow 1.2}$	2535.1 ± 28.1$^{\uparrow 1.5}$	2540.1 ± 28.1$^{\uparrow 1.6}$	2200.5 ± 31.0$^{\uparrow 3.0}$	**2290.6**$^{\uparrow 1.8}$
	GF16	1920.3 ± 25.8	2498.8 ± 33.6	2499.5 ± 33.6	2135.6 ± 29.7	2249.7
96	RF	2165.7 ± 29.9$^{\uparrow 2.7}$	2775.4 ± 38.4$^{\uparrow 2.7}$	2688.0 ± 37.2$^{\uparrow 1.2}$	2312.0 ± 35.6$^{\uparrow 0.8}$	**2472.2**$^{\uparrow 1.8}$
	GF16	2108.9 ± 26.0	2701.1 ± 33.4	2657.2 ± 33.3	2293.8 ± 34.9	2427.4
112	RF	2279.6 ± 25.4$^{\uparrow 1.0}$	2828.6 ± 31.1$^{\uparrow 2.7}$	2728.6 ± 30.4$^{\uparrow 2.3}$	2420.4 ± 33.9$^{\uparrow 1.7}$	**2554.5**$^{\uparrow 1.9}$
	GF16	2257.1 ± 25.2	2754.8 ± 30.7	2668.1 ± 29.9	2378.9 ± 32.0	2506.4
128	RF	2377.6 ± 30.4$^{\uparrow 2.7}$	2924.9 ± 37.4$^{\uparrow 1.0}$	2889.2 ± 37.0$^{\uparrow 1.2}$	2430.0 ± 39.6$^{\uparrow 1.7}$	**2643.4**$^{\uparrow 1.6}$
	GF16	2316.0 ± 25.3	2894.8 ± 32.0	2855.4 ± 31.1	2389.6 ± 30.9	2600.7
144	RF	2516.5 ± 33.8$^{\uparrow 2.7}$	2990.1 ± 40.2$^{\uparrow 2.5}$	2954.8 ± 39.7$^{\uparrow 2.8}$	2534.7 ± 41.0$^{\uparrow 2.0}$	**2739.9**$^{\uparrow 2.5}$
	GF16	2450.0 ± 24.5	2918.4 ± 29.2	2873.1 ± 28.7	2485.9 ± 35.0	2673.2
160	RF	2498.4 ± 34.8$^{\uparrow 2.3}$	3033.3 ± 42.4$^{\uparrow 0.4}$	2973.0 ± 41.5$^{\uparrow 1.8}$	2530.3 ± 33.7$^{\uparrow 1.8}$	**2747.8**$^{\uparrow 1.6}$
	GF16	2443.2 ± 32.1	3019.8 ± 39.7	2920.5 ± 38.1	2485.0 ± 37.6	2705.1
176	RF	2514.2 ± 24.6$^{\uparrow 3.1}$	3136.3 ± 30.7$^{\uparrow 3.2}$	3055.5 ± 29.9$^{\uparrow 3.4}$	2581.1 ± 38.4$^{\uparrow 3.6}$	**2808.2**$^{\uparrow 3.3}$
	GF16	2437.9 ± 36.1	3039.0 ± 45.0	2956.3 ± 43.7	2492.5 ± 40.2	2718.2
192	RF	2698.5 ± 27.1$^{\uparrow 2.1}$	3212.5 ± 32.3$^{\uparrow 1.6}$	3111.1 ± 31.3$^{\uparrow 2.4}$	2606.6 ± 31.5$^{\uparrow 0.7}$	**2895.6**$^{\uparrow 1.7}$
	GF16	2643.0 ± 33.3	3161.0 ± 39.8	3037.0 ± 38.3	2587.3 ± 40.5	2846.4
224	RF	2760.1 ± 38.5	3233.3 ± 45.2	3106.7 ± 43.4	2584.1 ± 39.3	2909.3
	GF16	OOM	OOM	OOM	OOM	OOM
240	RF	2802.2 ± 32.6	3200.3 ± 37.3	3015.1 ± 35.1	2314.1 ± 32.3	2812.5
	GF16	OOM	OOM	OOM	OOM	OOM
256	RF	2627.4 ± 30.6	2988.5 ± 34.9	2740.6 ± 32.0	2070.2 ± 29.8	2583.5
	GF16	OOM	OOM	OOM	OOM	OOM
Ornith-1.5-35B-A3B						
64	RF	4024.2 ± 52.4$^{\uparrow 1.9}$	3730.9 ± 41.6$^{\uparrow 1.8}$	4043.2 ± 43.3$^{\uparrow 2.4}$	3280.2 ± 34.0$^{\uparrow 1.2}$	**3756.5**$^{\uparrow 1.8}$
	GF16	3950.3 ± 47.6	3666.0 ± 47.2	3948.2 ± 42.2	3240.8 ± 23.3	3689.5
80	RF	4143.9 ± 48.7$^{\uparrow 3.3}$	4320.8 ± 46.3$^{\uparrow 2.8}$	3943.4 ± 41.6$^{\uparrow 2.3}$	3488.0 ± 34.8$^{\uparrow 0.7}$	**3961.4**$^{\uparrow 2.3}$
	GF16	4011.7 ± 52.9	4202.8 ± 51.0	3856.3 ± 22.9	3464.3 ± 23.6	3874.0
96	RF	4600.2 ± 52.2$^{\uparrow 0.9}$	4917.8 ± 50.1$^{\uparrow 2.8}$	4527.2 ± 45.6$^{\uparrow 1.9}$	3443.6 ± 34.9$^{\uparrow 1.9}$	**4333.6**$^{\uparrow 1.9}$
	GF16	4560.6 ± 54.2	4783.2 ± 47.2	4441.1 ± 45.7	3380.1 ± 36.6	4253.9
112	RF	5092.2 ± 61.3$^{\uparrow 2.2}$	4824.5 ± 51.8$^{\uparrow 3.3}$	4890.0 ± 50.1$^{\uparrow 1.4}$	4020.8 ± 40.3$^{\uparrow 2.2}$	**4688.1**$^{\uparrow 2.3}$
	GF16	4982.3 ± 64.4	4669.0 ± 28.1	4822.4 ± 49.2	3932.5 ± 42.5	4582.9
128	RF	5474.8 ± 58.8$^{\uparrow 2.7}$	5514.8 ± 58.9$^{\uparrow 1.4}$	5250.6 ± 55.1$^{\uparrow 1.5}$	4343.2 ± 44.1$^{\uparrow 0.7}$	**5122.5**$^{\uparrow 1.6}$
	GF16	5332.7 ± 60.2	5438.2 ± 51.5	5173.1 ± 52.4	4313.9 ± 22.8	5043.8
144	RF	5636.7 ± 62.9$^{\uparrow 2.6}$	6038.3 ± 62.2$^{\uparrow 2.0}$	5293.8 ± 53.9$^{\uparrow 1.8}$	4138.7 ± 43.0$^{\uparrow 1.7}$	**5225.7**$^{\uparrow 2.0}$
	GF16	5493.5 ± 56.4	5922.6 ± 57.8	5197.7 ± 29.7	4070.1 ± 43.4	5122.1
160	RF	5820.5 ± 62.8$^{\uparrow 2.3}$	5835.7 ± 60.6$^{\uparrow 2.4}$	5492.0 ± 56.1$^{\uparrow 2.6}$	4672.0 ± 45.8$^{\uparrow 2.1}$	**5433.4**$^{\uparrow 2.3}$
	GF16	5692.3 ± 39.8	5697.2 ± 28.2	5355.2 ± 52.7	4576.3 ± 48.1	5309.6
176	RF	6076.5 ± 67.6$^{\uparrow 2.1}$	6581.1 ± 74.1$^{\uparrow 2.0}$	5820.5 ± 59.4$^{\uparrow 1.1}$	4658.8 ± 46.4$^{\uparrow 2.3}$	**5738.5**$^{\uparrow 1.9}$
	GF16	5952.3 ± 61.2	6451.8 ± 62.3	5758.9 ± 55.6	4552.1 ± 46.1	5632.8
192	RF	6334.6 ± 65.7	6649.9 ± 72.1	4983.2 ± 53.5	4723.7 ± 46.9	5611.5
	GF16	OOM	OOM	OOM	OOM	OOM
224	RF	7008.8 ± 74.5	6982.6 ± 76.0	6470.9 ± 67.2	5214.7 ± 51.3	6374.7
	GF16	OOM	OOM	OOM	OOM	OOM
240	RF	6974.4 ± 83.7	7323.6 ± 76.3	6607.8 ± 67.8	5358.2 ± 53.4	6521.2
	GF16	OOM	OOM	OOM	OOM	OOM
256	RF	7125.0 ± 75.3	7239.7 ± 39.0	6687.5 ± 68.2	5467.2 ± 55.0	6590.0
	GF16	OOM	OOM	OOM	OOM	OOM

G.4 Load Switching

This comparison fixes the draft model, $W = 8$, candidate budget rules, and forest allocator, varying only whether the native path is allowed. RF switches according to the number of active requests in each round, with entry and exit thresholds of 64 and 48, respectively; Always-forest RF uses the forest path throughout. Figure 11 presents the geometric mean across the four benchmarks at all concurrency levels, with DFlash2 serving as the native-path reference. Per-benchmark means and standard deviations for this experiment are provided in the accompanying data file `policy_throughput.csv`.

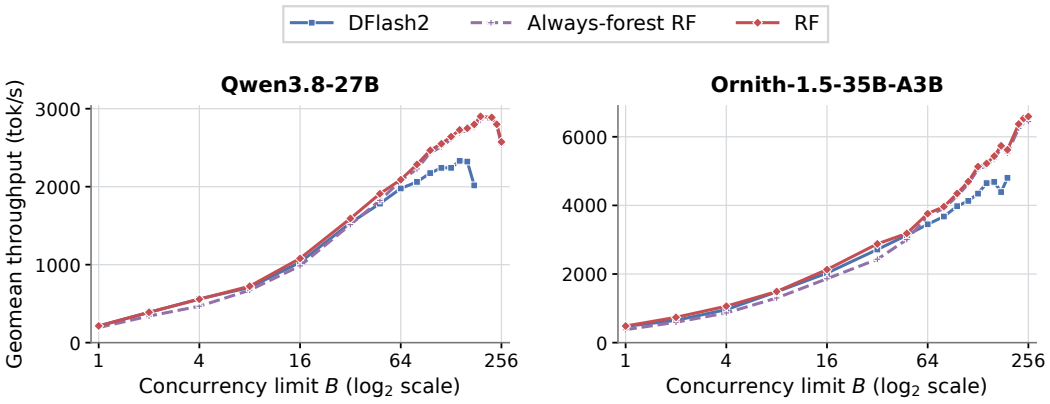

Figure 11: Load-policy comparison: Qwen3.8-27B on the left and Ornith-1.5-35B-A3B on the right. Each curve covers all 19 measured levels of B; the horizontal axis uses a base-2 logarithmic scale. Each curve is computed as the geometric mean of the four benchmark means from the corresponding experiment; RF and Always-forest RF use the load-switching experiments, while DFlash2 uses the main throughput experiments; curves are interrupted at missing or OOM points.

Table 19 reports per-benchmark results at $B = 8, 32, 64$. At $B = 8$, RF improves throughput over Always-forest RF by 7.6% and 14.4% on the two models, respectively; at $B = 64$, the gaps narrow to 0.7% and 1.7%. These results indicate that retaining the native path avoids the additional overhead of continuously constructing and verifying forests under low load. Switching depends on the number of active requests B_t in each round, so starting with a given concurrency limit B does not imply that the same path is used throughout.

Table 19: Throughput with load switching (tokens/s, mean ± sample standard deviation over four runs). Yellow highlighting identifies RF, and superscripts indicate its changes relative to Always-forest RF (%); Avg. is the geometric mean of the mean throughput values across the four benchmarks.

B	Method	GSM8K	MATH-500	HumanEval	MT-Bench	Avg.
Qwen3.8-27B						
8	RF	807.7 ± 11.1$^{\uparrow 5.4}$	704.4 ± 8.5$^{\uparrow 14.1}$	778.5 ± 11.7$^{\uparrow 8.5}$	619.6 ± 10.1$^{\uparrow 2.7}$	**723.8**$^{\uparrow 7.6}$
	Always-forest RF	766.4 ± 10.9	617.3 ± 4.3	717.4 ± 8.5	603.6 ± 8.2	672.7
32	RF	1391.9 ± 22.3$^{\uparrow 2.4}$	1710.2 ± 25.5$^{\uparrow 7.1}$	1727.3 ± 26.2$^{\uparrow 6.1}$	1561.9 ± 24.1$^{\uparrow 5.0}$	**1591.9**$^{\uparrow 5.1}$
	Always-forest RF	1359.1 ± 18.5	1596.7 ± 20.3	1628.6 ± 20.4	1487.1 ± 20.0	1514.1
64	RF	1784.8 ± 22.5$^{0.0}$	2313.0 ± 28.7$^{\uparrow 2.5}$	2275.0 ± 33.0$^{\downarrow 0.3}$	2030.8 ± 29.6$^{\uparrow 0.5}$	**2089.8**$^{\uparrow 0.7}$
	Always-forest RF	1784.8 ± 24.2	2256.3 ± 26.9	2281.0 ± 26.1	2021.0 ± 29.5	2075.7
Ornith-1.5-35B-A3B						
8	RF	1433.0 ± 5.2$^{\uparrow 9.3}$	1706.8 ± 7.1$^{\uparrow 20.8}$	1876.4 ± 7.2$^{\uparrow 27.9}$	1071.8 ± 7.2$^{\uparrow 1.5}$	**1489.2**$^{\uparrow 14.4}$
	Always-forest RF	1311.4 ± 8.1	1412.7 ± 10.4	1466.6 ± 10.3	1055.6 ± 4.5	1301.3
32	RF	2857.4 ± 19.5$^{\uparrow 8.2}$	3231.5 ± 23.4$^{\uparrow 25.1}$	3392.4 ± 13.3$^{\uparrow 24.1}$	2190.8 ± 18.3$^{\uparrow 17.2}$	**2878.2**$^{\uparrow 18.4}$
	Always-forest RF	2640.3 ± 8.6	2584.1 ± 10.5	2733.6 ± 22.2	1869.6 ± 7.1	2430.0
64	RF	4016.0 ± 26.3$^{\uparrow 1.7}$	3727.4 ± 27.7$^{\uparrow 1.4}$	4054.3 ± 15.2$^{\uparrow 2.2}$	3292.6 ± 13.9$^{\uparrow 1.6}$	**3759.8**$^{\uparrow 1.7}$
	Always-forest RF	3948.1 ± 13.3	3674.5 ± 26.8	3968.5 ± 29.1	3240.0 ± 25.5	3695.6

G.5 Overhead Breakdown

We evaluate the overhead breakdown using Qwen3.8-27B on GSM8K with $B = 128$, TP4, BF16, $W = 8$, and a static memory fraction of 0.80, using the original DFlash2 drafter checkpoint. We construct the workload by replicating 32 fixed questions eight times, yielding 256 requests per run for each variant, and repeat each experiment three times. Each variant is warmed up with 128 requests. Caches are cleared before each measurement, while CUDA Graph execution and overlap remain enabled. We use greedy decoding with natural EOS termination and a generation limit of 512 tokens; the main throughput experiments use a 4096-token generation limit. The candidate budget is capped by $K_t = \min\{7B_t, 384\}$, which becomes $384 = 3B$ at full batch size, and the number of candidate nodes selected in each round does not exceed K_t.

Progress and Timing Methodology. Table **??** reports statistics over fully instrumented rounds with $B_t = 128$ within the measurement window. All entries report means, while cells with $\pm$ additionally report the sample standard deviation across the three repeated runs. Progress is measured as the number of runtime-committed tokens per request per decoding round, including the bonus token. This quantity is recorded before the scheduler performs final truncation due to EOS or the generation-length limit.

Timing is measured on TP rank 0 using same-stream CUDA-event intervals and therefore includes host submission latency and device idle gaps within the measured intervals. *Selection* corresponds to the candidate-selection interval. *State management* is the sum of three non-overlapping intervals: drafter-state preparation, target-state commitment, and request commitment.

Table **??** reports total per-round latency as the sum of drafting, selection, target forward, state management, and other intervals. The *other* category includes layout packing, verification preparation, acceptance processing, broadcast operations, and gaps between instrumented intervals. HTTP throughput covers the complete lifetime of all requests, including the tail of each batch, and therefore uses a different measurement scope from the full-batch per-round timing.

Results. Relative to EC, GC increases full-batch progress by 3.98%, while the target-forward interval remains approximately 94 ms. End-to-end request throughput improves by 4.00%, consistent with the interpretation that shared-budget allocation increases the amount of useful progress achieved per decoding round. Compared with GC, GF further improves end-to-end request throughput by 2.74% and increases mean full-batch progress by 4.42%, while target-forward latency remains similar.

The three variants were executed sequentially in the order EC, GC, and GF rather than in a randomized interleaving. Consequently, the number of full-batch rounds and the context composition differ across variants. The workload consists of repeated copies of 32 fixed questions, and some requests reach the 512-token generation limit.

H Task Quality Evaluation

Configuration. Qwen3.8-27B is evaluated on all 1319 problems in GSM8K, all 500 problems in MATH-500, and all 164 problems in HumanEval. The five methods share the same original problems, reference answers, and zero-shot chat prompts, using $B = 64$, TP4, BF16, a static GPU memory fraction of 0.90, temperature 0, and seed 42, with one generation per problem, stopping at EOS or 8192 tokens. RF uses $W = 8$ and $K_{\max} = 192$. The quality experiments use the full problem sets and an 8192-token limit and are reported separately from the main throughput experiments, which use repeated submissions and a 4096-token limit.

Table 20: Task quality on the full datasets for Qwen3.8-27B at $B = 64$. Each cell reports the number of correctly answered problems or problems passing the tests divided by the total number of original problems (percentage).

Method	GSM8K	MATH-500	HumanEval
Target-only	1226/1319 (92.95%)	470/500 (94.00%)	153/164 (93.29%)
MTP	1225/1319 (92.87%)	468/500 (93.60%)	155/164 (94.51%)
DFlash	1224/1319 (92.80%)	469/500 (93.80%)	155/164 (94.51%)
DFlash2	1222/1319 (92.65%)	473/500 (94.60%)	156/164 (95.12%)
RF	**1222/1319 (92.65%)**	**470/500 (94.00%)**	**156/164 (95.12%)**

Scoring. For GSM8K, the last numerical value after the closed thinking segment is extracted, currency symbols, thousands-separator commas, and trailing periods are normalized, and scores are computed by exact string matching. MATH-500 uses a fixed extraction configuration and strict symbolic equivalence checking in Math-Verify 0.9.0, with no manual corrections for individual problems. For HumanEval, programs are extracted from the full code outputs and evaluated using the original tests in an isolated sandbox, without completing incomplete code. Table 20 includes all problems, with truncated outputs, extraction failures, and test failures all included in the denominator. “‘