CONTEXTUAL CORRECTION OF PARALLEL DRAFTS FOR SPECULATIVE DECODING

Anonymous authors
Paper under double-blind review

ABSTRACT

Speculative decoding accelerates large language model inference by using a lightweight drafter to propose multiple tokens that are subsequently verified in parallel by the target model. Recent diffusion-based drafters make this paradigm particularly attractive: deep attention-based neural backbones provide strong predictions, while drafting multiple future positions in parallel avoids the sequential cost of autoregressive generation. This parallelism, however, comes at a cost. Tokens within a draft block are predicted without access to the realized identities of preceding draft tokens, removing intra-block dependencies that would naturally be available under autoregressive generation. We introduce *DCOCO* (**D**Flash **Co**ntextual **Co**rrector), a lightweight mechanism that restores these dependencies while preserving parallel drafting. Rather than committing to each position's top prediction, *DCOCO* retains multiple candidate tokens and evaluates their contextual compatibility across adjacent draft positions in parallel, enabling contextually informed token selection at low additional cost. Across four verifier backbones and ten benchmarks, *DCOCO* achieves the highest average throughput in every evaluated configuration, spanning greedy and stochastic decoding with and without extended reasoning, with speedups of up to 6.6× over vanilla decoding. Its advantage persists under concurrent serving: *DCOCO* outperforms the strongest competing method by 6–23% across 4–32 concurrent requests, while other sequential correctors fall below uncorrected DFlash at these loads. These results demonstrate that recovering intra-block dependencies through parallel contextual scoring delivers consistent acceleration across decoding regimes and serving loads.

1 INTRODUCTION

Speculative decoding (SD) accelerates large language model inference by using a lightweight drafter to propose multiple future tokens, which are subsequently verified in parallel by the target model (Leviathan et al., 2023; Chen et al., 2023). Speculative verification preserves the target model's output distribution, so the benefit of a drafting method depends on the speedup it provides. This speedup depends on both how many draft tokens are accepted and the latency of draft generation. These two factors are often in tension: stronger drafting can increase acceptance, but better drafts may require more computation and therefore more latency.

This acceptance–latency trade-off has shaped the design of speculative drafters. Early SD methods use a smaller autoregressive language model to generate draft tokens sequentially (Leviathan et al., 2023; Chen et al., 2023). Subsequent approaches such as EAGLE (Li et al., 2024a;b; 2025b) improve the efficiency and quality of this drafting process by exploiting target-model features, while retaining autoregressive dependencies across draft positions. More recently, parallel diffusion-based drafters such as DFlash (Chen et al., 2026) take a different direction, generating multiple future positions within a single block pass. Autoregressive drafters condition on the realized prefix, whereas parallel diffusion drafters reduce latency by predicting the block jointly. This parallelism comes at the cost of weaker intra-block conditioning: each position is predicted without observing the realized token identities selected at preceding draft positions. As a result, individually plausible predictions may form a less consistent continuation and reduce draft acceptance.

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

Recent methods directly target this limitation. DSpark (Cheng et al., 2026) combines a parallel backbone with a lightweight sequential module, introducing intra-block dependencies through semi-autoregressive drafting. Domino (Huang et al., 2026) instead augments parallel block generation with a lightweight causal correction module that refines the draft using prefix-dependent information. Both recover acceptance, but both do so position by position. The added module is a recurrence over the block, so its latency cost sits on the critical path of every drafting cycle. This raises the central question of our work: *can intra-block dependencies be restored without reintroducing a sequential module?*

Two observations guide our approach. First, the parallel drafter has usually already suspected the correct token, even when it did not choose it: on Qwen3-8B the top-16 candidates contain the verifier's token at 59% of the slots where its top-1 is wrong (§3). Correction can therefore be cast as re-selection within the drafter's existing candidates rather than generation of new tokens. Re-selection over K candidates per slot reduces to scoring K^2 transitions between adjacent slots, all of which can be computed at once. Second, speculative verification imposes a strong asymmetry across draft positions. Verification accepts a contiguous prefix and stops at the first rejected token; therefore, an improvement at a later position is useful only if all preceding positions are also accepted. Correction should consequently prioritize the quality of the early prefix, rather than optimize an aggregate score over the entire proposed block.

These observations motivate **D**Flash **Co**ntextual **Co**rrector (***DCOCO***), a lightweight contextual corrector for parallel diffusion drafts. *DCOCO* operates on a single DFlash block pass: rather than committing immediately to the top prediction at each position, it retains multiple candidate tokens and uses the drafter's existing hidden states and logits to score their contextual compatibility across adjacent positions. All candidate interactions are scored in parallel, without an additional drafter forward pass. The resulting candidate grid is then decoded using a *prefix-first greedy selection* rule. Rather than maximizing an aggregate score over the entire path, *DCOCO* selects candidates from left to right, prioritizing each position before proceeding to the next. This aligns selection with the prefix structure of speculative verification, where tokens after the first rejection provide no benefit. Importantly, only this lightweight selection is sequential, while neural scoring remains parallel. Unlike DSpark and Domino, no learned module runs position by position.

We evaluate *DCOCO* across multiple verifier backbones and ten benchmarks spanning mathematics, code, and general-purpose generation. *DCOCO* consistently improves end-to-end decoding throughput under both greedy and stochastic decoding, outperforming the strongest evaluated speculative decoding baseline by up to 18% and vanilla autoregressive decoding by up to $6.6\times$. Because its correction is parallel, *DCOCO* keeps its advantage as serving load grows: from 1 to 32 concurrent streams its throughput stays 19–25% above DFlash under greedy decoding, while the per-cycle cost of the sequential correctors grows until both fall below the uncorrected drafter (§5.2). Beyond aggregate throughput, we separately analyze draft acceptance and correction latency to characterize where these gains arise. We further compare alternative selection rules over the same learned candidate scores, isolating the effect of the decoding objective from the quality of the corrector itself. Our contributions are threefold:

- We introduce ***DCOCO***, a lightweight contextual corrector for parallel diffusion drafts that scores candidate interactions using information from a single drafter pass, without additional drafter forward passes.
- We identify the prefix structure of speculative verification as a key consideration for correcting parallel drafts, and introduce a prefix-first greedy selection rule that aligns candidate selection with this structure.
- We establish state-of-the-art performance across multiple verifier backbones and diverse benchmarks, with and without extended thinking in both greedy and stochastic decoding and show that this advantage persists under concurrent serving load, where the sequential correctors lose theirs.

2 Related Work

Speculative decoding. Speculative decoding accelerates autoregressive LLM inference by separating generation into drafting and verification: a lightweight draft model proposes several future

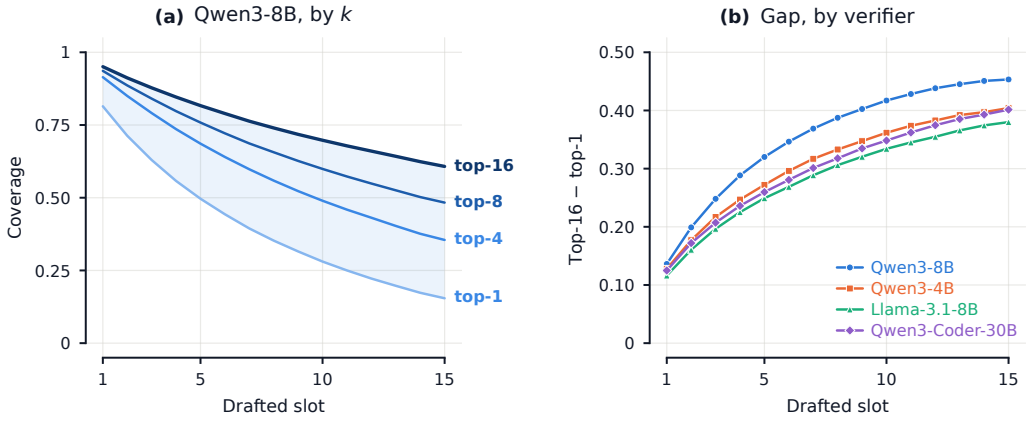


Figure 1: Candidate coverage gains grow deeper in the block. Fraction of slots where DFlash's top-k candidates contain the verifier's committed token. (a) Qwen3-8B; shading marks the top-16–top-1 coverage gap. (b) The same gap across verifiers. Averages over the 15 slots appear in Tab. 18.

tokens, which are then verified in parallel by the target model while preserving the target model's output distribution (Leviathan et al., 2023; Chen et al., 2023). Early approaches used a smaller autoregressive language model as the drafter, reducing the cost of proposing tokens while retaining autoregressive conditioning across the draft sequence (Leviathan et al., 2023; Chen et al., 2023). Subsequent work has improved the acceptance–latency trade-off by constructing lightweight drafters from representations of the target model itself. EAGLE (Li et al., 2024a) performs autoregression at the feature level, predicting target-model features rather than relying on an independent language-model drafter. EAGLE-2 (Li et al., 2024b) further introduces context-aware dynamic draft trees, allocating speculative candidates according to the drafter's confidence, while EAGLE-3 (Li et al., 2025b) replaces feature prediction with direct token prediction and fuses features from multiple target-model layers. Other approaches explore recurrent drafters conditioned on target-model representations, such as ReDrafter (Cheng et al., 2024), or improve the alignment between the drafter and target representations (Zhang et al., 2025). These methods substantially strengthen lightweight autoregressive drafting, but draft tokens along a candidate path remain sequentially dependent. This has motivated a complementary line of work that moves parallelism from verification into the drafting stage itself.

Parallel drafters. Parallel drafters reduce draft-generation latency by predicting multiple future positions within a single pass. Medusa (Cai et al., 2024) predicts future tokens in parallel using multiple lightweight heads, to which Hydra (Ankner et al., 2024) adds sequential dependency in order to increase performance. PARD (An et al., 2026) adapts autoregressive drafters to parallel block prediction, while CTC-based drafting (Wen et al., 2024) uses a non-autoregressive sequence generation technique to predict the speculated sequence in parallel. Diffusion models provide a natural foundation for this setting because multiple positions can be denoised simultaneously. DFlash (Chen et al., 2026) exploits this property with a lightweight diffusion drafter conditioned on target-model features, generating an entire draft block in one forward pass. However, unlike autoregressive drafting, parallel block prediction does not condition each position on the realized tokens selected at earlier positions, weakening dependencies within the draft and motivating methods that restore this missing context.

Refining parallel drafts. Recent methods address this limitation by augmenting parallel drafts with lightweight causal correction. DSpark (Cheng et al., 2026) combines a parallel backbone with a sequential module that introduces dependencies across draft positions, while Domino (Huang et al., 2026) applies recurrent causal correction to refine the parallel proposal using prefix-dependent information.

These approaches highlight the central challenge: recovering the dependencies lost during parallel drafting without sacrificing its latency advantage. *DCOCO* takes a different approach: rather than

sequentially refining the draft or iteratively updating the block, it operates on the candidate set produced by a single DFlash pass. *DCOCO* scores all adjacent candidate interactions in one batched operation using the drafter's hidden states and logits, without an additional drafter pass. Only the subsequent candidate selection, reduced to a plain walk on a grid, is sequential.

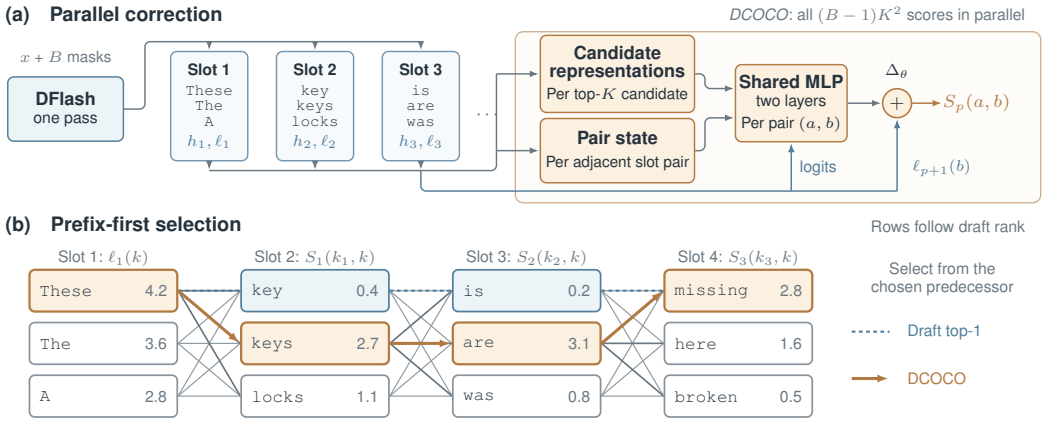


Figure 2: *DCOCO*: parallel scoring, prefix-first selection. (a) One DFlash pass supplies slot states, candidate IDs, and logits. Candidate and pair representations feed a shared MLP, computing all residual scores in parallel: $S_p(a, b) = \ell_{p+1}(b) + \Delta_\theta$. (b) Starting from the first-slot draft argmax, selection follows the best transition from each chosen predecessor. Rows retain draft rank; scores in slots 2–4 condition on the selected predecessor. Tokens and scores are illustrative.

3 Parallel Drafts Already Contain the Right Token

A parallel drafter predicts all block positions without conditioning on the tokens selected at earlier positions. This can produce incoherent combinations of individually plausible predictions, a known limitation of non-autoregressive and discrete-diffusion language modeling (Gu et al., 2018; Lee et al., 2026). For example, a drafter assigning probability to both "New York" and "San Diego" may emit "New Diego". Such errors raise the question of whether the correct token remains among the drafter's leading candidates. To test whether correct alternatives remain available for re-selection, we measure *top-k coverage*: the fraction of drafted slots whose k highest-scoring candidates contain the token eventually committed by the verifier. We record DFlash's candidates during speculative decoding at $T = 0$ and compare them with the realized continuation. We observe two regularities.

Fig. 1 shows that on Qwen3-8B, top-16 coverage stays above 0.60 at the last slot while top-1 falls below 0.16, and the top-16 $-$ top-1 gap grows from 0.12–0.14 at the first slot to 0.38–0.45 at the last on *every* verifier (averages in Tab. 18). The drafter does not lack the right tokens; it fails to *select* them, increasingly so deeper in the block. That headroom is conditional, however: under prefix acceptance a correction at slot p counts only if every earlier slot is accepted, hence our prefix-first rule, selecting candidates left to right, each conditioned on the choice just made.

4 Method

We consider speculative decoding with a parallel block drafter: DFlash proposes a block of B tokens in a single forward pass, and a verifier accepts the longest correct prefix of the proposal. Our proposed corrector, *DCOCO*, works in two steps. It first reassigns scores to all drafter's per-slot candidates based on stronger contextual information given the *realized* tokens and second, it re-selects the proposed block based on the scoring. Figure 2 gives an overview of the architecture and the flow from drafted block to corrected proposal.

4.1 Block Drafting and Prefix Acceptance

The verifier scores the proposed block $\hat{y}_{1:B}$ in a single pass and accepts a prefix of it, judging slots left to right, then commits one further token of its own. Under greedy decoding, slot j is accepted iff $\hat{y}_j$ matches the verifier's greedy choice $\arg\max_v q(v \mid x, \hat{y}_{<j})$; under stochastic decoding it

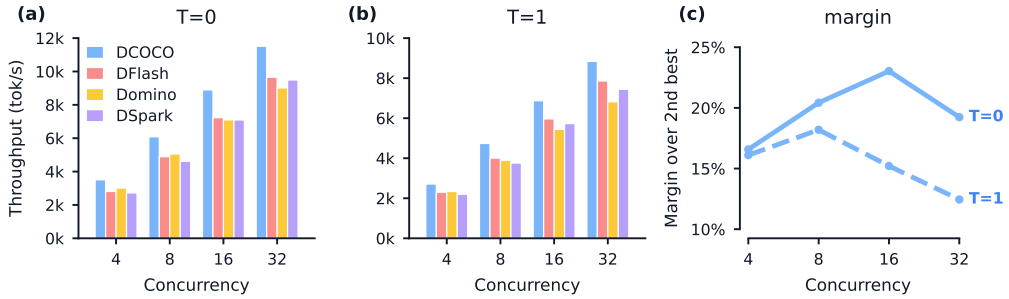


Figure 3: At high concurrency, *DCOCO* achieves the highest throughput, while Domino and DSpark fall below DFlash. Qwen3-8B throughput at (a) $T = 0$ and (b) $T = 1$. (c) *DCOCO*'s relative throughput gain over the strongest baseline at each concurrency.

is accepted by the standard speculative-sampling rule in Leviathan et al. (2023) which accepts $\hat{y}_j$ with probability $\min\{1, q(\hat{y}_j \mid x, \hat{y}_{<j})/p(\hat{y}_j \mid x, \hat{y}_{<j})\}$ and otherwise resamples from the residual. *DCOCO* selects proposals deterministically in both decoding regimes, so its proposal distribution is a point mass at each selected token; the resulting lossless stochastic verification rule is detailed in Appendix G. In both regimes, verification halts at the first rejection. Writing $a_j \in \{0, 1\}$ for slot j's acceptance, the accepted length is

$$A(\hat{y}) \;=\; \max\{\, m : \, a_1 = \cdots = a_m = 1 \,\}, \tag{1}$$

and each round commits $A(\hat{y}) + 1$ tokens. Throughput is thus governed by the expected accepted length together with the wall-clock cost of a round. The one structural fact that drives every design choice below is that verification halts at the first rejection: a corrected token at slot 5 standing behind a rejection at slot 1 is never even examined. Expected accepted length depends on the probability that each successive prefix survives verification. This motivates prioritizing early positions.

4.2 A LEARNED CORRECTOR OVER THE BLOCK GRID

At each slot $i \in \{1, \ldots, B\}$, DFlash produces a hidden state $h_i \in \mathbb{R}^d$ and retains its top-K candidate tokens. Let c_{ik} denote the token id of the k-th retained candidate and $\ell_i(k)$ its draft logit. These candidates form a dense first-order grid with $B \times K$ nodes, in which every candidate at slot p connects to every candidate at slot $p + 1$. The *DCOCO* corrector assigns each transition a score

$$S_p(a, b) = \underbrace{\ell_{p+1}(b)}_{\text{draft unary logit}} + \underbrace{\Delta_\theta\big(s_p,\; z_{pa},\; z_{p+1,b},\; \ell_p(a),\; \ell_{p+1}(b)\big)}_{\text{learned correction}}, \tag{2}$$

where $a, b \in \{1, \ldots, K\}$ index the candidates at the two slots, z_{ik} is a candidate representation (Eq. 4), s_p is a candidate-independent pair state (Eq. 5), and Δ_θ is a shared two-layer MLP. The corrector uses only the hidden states, candidate ids, and logits available from the single DFlash pass.

Candidate representation. Each candidate representation combines the candidate's token embedding with its slot's hidden state. Using the frozen token-embedding table E and layer normalization LN, we compute

$$b_{ik} = \mathrm{LN}\big(W_{\text{cand}} E[c_{ik}]\big), \qquad g_i = \mathrm{LN}\big(W_{\text{ctx}} h_i\big), \tag{3}$$

$$z_{ik} = \mathrm{LN}_{\gamma,\beta}\big(W_{\text{fuse}}[\, b_{ik} \,;\, g_i \,]\big), \tag{4}$$

where b_{ik} is the projected and normalized candidate embedding, g_i is the projected and normalized slot context shared by all candidates at slot i, and z_{ik} is their fused representation. Here, $[\cdot\,;\,\cdot]$ denotes concatenation, $W_{\text{cand}}, W_{\text{ctx}} \in \mathbb{R}^{r \times d}$, and $W_{\text{fuse}} \in \mathbb{R}^{r \times 2r}$, with corrector width $r \ll d$. The token embeddings and hidden states differ in scale by more than two orders of magnitude. We therefore use non-affine layer normalization for both projections in Eq. 3. The output normalization in Eq. 4 includes learnable scale and bias parameters γ and β.

Pair state. For each pair of adjacent slots, we compute a shared state from their hidden states:

$$s_p = W_{\text{pair}}[\, h_p \,;\, h_{p+1} \,], \qquad W_{\text{pair}} \in \mathbb{R}^{r \times 2d}. \tag{5}$$

Because s_p depends only on the slots, it is computed once and reused across all K^2 candidate transitions between them.

Transition score. The correction Δ_θ in Eq. 2 is computed by a two-layer MLP with a GELU activation between the layers and a scalar output. Its input is the concatenation $[\, s_p \,;\, z_{pa} \,;\, z_{p+1,b} \,;\, \ell_p(a) \,;\, \ell_{p+1}(b) \,]$. The successor's draft logit provides the base score, while Δ_θ learns an additive correction conditioned on both candidates and their slot contexts. The MLP's output layer is zero-initialized, so initially $\Delta_\theta \equiv 0$ and every transition score reduces to the successor's draft logit, $S_p(a, b) = \ell_{p+1}(b)$. Prefix-first greedy selection therefore reproduces the drafter's original proposal at initialization. Finally, all projection and MLP parameters are shared across slots and transitions, and all $(B-1)K^2$ transition scores are computed in one batched operation before path selection begins.

Training. The drafter and corrector are optimized together using the sum of the original drafter loss and the correction loss. The token-embedding table remains frozen throughout training. For each transition p, the label $(a_p^\star, b_p^\star)$ is the pair of verifier-argmax tokens at the two slots, computed over the teacher-forced context and located within the slots' top-K sets, with mask $m_p \in \{0, 1\}$ zeroing transitions where either token is absent. The corrector minimizes an exponentially down-weighted cross-entropy over the K^2 jointly softmax-normalized transition scores,

$$\mathcal{L}_{\text{corr}} \;\propto\; -\sum_{p=1}^{B-1} e^{-(p-1)/\rho}\, m_p \,\log \frac{\exp S_p(a_p^\star, b_p^\star)}{\sum_{a,b=1}^{K} \exp S_p(a, b)}, \qquad \rho = 4. \tag{6}$$

This softmax exists only in training; at inference S_p is used unnormalized. The complete loss term along with further information about the training procedure are detailed in App. A.

4.3 Prefix-first path selection

Given the scored grid, one possible decoding strategy returns the path maximizing the sum of transition scores. This sum does not explicitly account for prefix survival: a higher score at a later position can compensate for a lower score at an earlier position, even though later tokens contribute only if the preceding prefix is accepted. Motivated by this asymmetry, we use a simple prefix-first greedy rule. The first slot keeps the drafter's top candidate, and each subsequent slot selects the highest-scoring successor of the chosen predecessor,

$$k_1 = \arg\max_k \ell_1(k), \qquad k_{p+1} = \arg\max_b S_p(k_p, b) \quad \text{for } p = 1, \ldots, B{-}1. \tag{7}$$

This rule makes each selection using the transition scores conditioned on the chosen predecessor, without lookahead over future transitions. Its main advantage is computational: the walk costs $(O(BK))$ on top of the $(O(BK^2))$ batched scoring pass. Section 5.3 shows that it achieves acceptance comparable to geometrically discounted global selection at higher throughput. With $(\Delta_\theta \equiv 0)$, it reduces to the drafter's own argmax at every slot, preserving the step-0 equivalence noted above.

5 Experiments

Setup. We use the vLLM framework (Kwon et al., 2023) to evaluate *DCOCO* against vanilla decoding, DFlash, DSpark, and Domino for four verifier backbones: Qwen3-(4B/8B/Coder-30B-A3B) and Llama-3.1-8B in production-like settings. Evaluation covers ten benchmarks spanning mathematics, code, and general-purpose generation, under greedy ($T = 0$) and stochastic ($T = 1$) decoding, with and without extended thinking where supported. All drafters use block size 16 and are trained on a common corpus of approximately 800K prompts, with continuations generated by their respective verifiers. *DCOCO* starts from a six-epoch DFlash checkpoint and trains for six additional epochs with the corrector; training-budget comparisons are reported in App. B. Training and implementation details appear in App. A, and evaluation and serving settings in App. C.

Baselines. Beyond improvement achieved over vanilla decoding we compare against the uncorrected DFlash draft, DSpark, and Domino. Each is retrained by us on the same corpus with the same five-layer drafter backbone reading the same verifier layers, and served with the same stack, hardware, and generation limits. Domino uses its published configuration (causal-encoder width 1024, correction rank 256) and DSpark a rank-256 Markov head with its confidence head.

(a) Greedy decoding, T=0 (single deterministic run)

Domain	Dataset	No-spec	DFlash	Domino	DSpark	*DCOCO*
Math	MATH-500	265.2	1293.8 (4.88×)	1412.3 (5.33×)	1257.9 (4.74×)	**1625.2 (6.13×)**
	AIME-25	263.7	1133.7 (4.30×)	1279.9 (4.85×)	1188.4 (4.51×)	**1436.8 (5.45×)**
	GSM8K	267.0	936.9 (3.51×)	1018.1 (3.81×)	921.6 (3.45×)	**1175.8 (4.40×)**
Code	HumanEval	266.6	873.9 (3.28×)	950.5 (3.57×)	857.5 (3.22×)	**1075.6 (4.04×)**
	MBPP	267.0	800.9 (3.00×)	891.7 (3.34×)	778.2 (2.92×)	**979.1 (3.67×)**
	LiveCodeBench	264.2	1080.1 (4.09×)	1161.6 (4.40×)	1084.7 (4.11×)	**1328.5 (5.03×)**
General	MT-Bench	266.0	434.1 (1.63×)	425.6 (1.60×)	386.5 (1.45×)	**494.7 (1.86×)**
	Arena-Hard-v2	264.1	423.7 (1.60×)	390.1 (1.48×)	386.8 (1.46×)	**486.9 (1.84×)**
	Alpaca	265.9	428.8 (1.61×)	436.4 (1.64×)	392.9 (1.48×)	**496.1 (1.87×)**
	SpeedBench	266.5	425.3 (1.60×)	426.8 (1.60×)	401.7 (1.51×)	**498.4 (1.87×)**
Average		265.6	783.1 (2.95×)	839.3 (3.16×)	765.6 (2.88×)	**959.7 (3.62×)**

(b) Sampling, T=1 (mean±std over seeds 42/43/44)

Domain	Dataset	No-spec	DFlash	Domino	DSpark	*DCOCO*
Math	MATH-500	264.5 ±0.1	943.9 ±37.0 (3.57×)	1013.7 ±25.3 (3.83×)	956.5 ±33.2 (3.62×)	**1145.5 ±15.5 (4.33×)**
	AIME-25	263.6 ±0.0	767.5 ±25.5 (2.91×)	781.2 ±74.1 (2.96×)	698.5 ±14.3 (2.65×)	**828.7 ±68.3 (3.14×)**
	GSM8K	266.4 ±0.0	852.8 ±16.0 (3.20×)	920.2 ±5.6 (3.45×)	834.5 ±13.9 (3.13×)	**1081.3 ±28.5 (4.06×)**
Code	HumanEval	266.4 ±0.0	770.4 ±2.7 (2.89×)	813.7 ±18.7 (3.06×)	755.0 ±6.1 (2.83×)	**910.0 ±13.9 (3.42×)**
	MBPP	266.6 ±0.3	703.1 ±4.5 (2.64×)	756.9 ±24.6 (2.84×)	678.7 ±12.3 (2.55×)	**848.3 ±6.1 (3.18×)**
	LiveCodeBench	265.3 ±0.3	874.8 ±36.1 (3.30×)	861.8 ±23.2 (3.25×)	889.7 ±4.4 (3.35×)	**1013.7 ±62.6 (3.82×)**
General	MT-Bench	265.3 ±0.0	401.9 ±5.1 (1.52×)	402.4 ±7.9 (1.52×)	369.4 ±8.9 (1.39×)	**450.8 ±8.0 (1.70×)**
	Arena-Hard-v2	264.2 ±0.1	379.4 ±1.2 (1.44×)	366.6 ±11.4 (1.39×)	350.2 ±9.0 (1.33×)	**436.8 ±6.5 (1.65×)**
	Alpaca	265.5 ±0.0	402.1 ±2.7 (1.51×)	393.9 ±8.7 (1.48×)	377.4 ±1.4 (1.42×)	**447.5 ±7.0 (1.69×)**
	SpeedBench	266.2 ±0.4	401.9 ±7.9 (1.51×)	398.5 ±6.9 (1.50×)	376.7 ±2.9 (1.42×)	**459.5 ±14.9 (1.73×)**
Average		265.4	649.8 (2.45×)	670.9 (2.53×)	628.7 (2.37×)	**762.2 (2.87×)**

Table 1: Throughput (output tokens/s, per-stream) and speedup over no-spec decoding, Qwen3-8B, thinking off, concurrency 1, max 2048 generated tokens. Each speculative cell reports TPS with its speedup over the no-spec column in parentheses. Bold: best method per row.

Metrics. Our primary metric is end-to-end decoding throughput in **tokens per second** (TPS), measured on a single B200 GPU per system at concurrency 1 unless stated otherwise; alongside each system's TPS we report its speedup over vanilla autoregressive decoding with the verifier alone ($\mathrm{TPS}/\mathrm{TPS}_{\text{vanilla}}$). TPS is the quantity a deployment actually experiences, and it is the appropriate one for a method that adds compute to the decode loop: a corrector can raise the accepted length while losing wall-clock time to its own scoring cost, and only TPS exposes that trade. Every design constraint in §4, the small corrector width, the single batched scoring pass, the $O(BK)$ walk, exists in service of this metric. We additionally report the mean accepted length $\tau = \mathbb{E}[A(\hat{y})] + 1$ of Eq. 1 as the *mechanism* metric: it isolates proposal quality from serving overheads and explains *where* a TPS change comes from (τ up $\Rightarrow$ fewer verifier rounds per token).

5.1 Main results

Conditions. Tab. 1 reports per-suite results for Qwen3-8B with thinking off, under greedy decoding (T=0) and sampling (T=1); Tab. 2 summarizes all four verifiers, with and without extended thinking, by their ten-suite averages. All numbers are output tokens/s at concurrency 1, with the system served on a single B200 GPU, generating at most 2048 tokens with thinking off and 4096 with thinking on, over the ten suites above. T=0 entries are single deterministic runs; T=1 entries are mean±std over *three* seeds. Bold marks the best speculative method per row and temperature; vanilla decoding is the speedup denominator. Per-suite tables for every other configuration, the accepted-length companion of every throughput table, and evaluation details are in App. D and C.

Discussion. On Qwen3-8B without thinking, *DCOCO* improves average accepted length over Domino by 8.1% at T=0 and 7.5% at T=1, while improving request throughput by 14.3% and 13.6%. Accepted length measures proposal quality, whereas request throughput also includes prefill and serving overhead.

Tab. 2 shows the same ordering across verifiers and thinking modes: *DCOCO* has the best average speedup in all twelve cells, by 8–19% over the strongest baseline in each; it also has the best average

Verifier	Think.	DFlash $T{=}0$	DFlash $T{=}1$	Domino $T{=}0$	Domino $T{=}1$	DSpark $T{=}0$	DSpark $T{=}1$	*DCOCO* $T{=}0$	*DCOCO* $T{=}1$
Qwen3-4B	off	2.98×	2.52×	3.27×	2.71×	2.66×	2.28×	**3.62**×	**2.95**×
	on	2.08×	1.87×	2.17×	1.93×	1.85×	1.69×	**2.38**×	**2.12**×
Qwen3-8B	off	2.95×	2.45×	3.16×	2.53×	2.88×	2.37×	**3.62**×	**2.87**×
	on	1.95×	1.78×	1.97×	1.79×	1.83×	1.68×	**2.26**×	**2.03**×
Qwen3-Coder-30B-A3B	off	1.89×	1.67×	1.94×	1.67×	1.84×	1.62×	**2.31**×	**1.98**×
Llama-3.1-8B	off	3.03×	1.75×	3.24×	1.79×	3.01×	1.65×	**3.75**×	**1.93**×

Table 2: All configurations at a glance: average speedup over vanilla decoding, unweighted mean over the ten evaluation suites. Bold: best per row and temperature. Qwen3-Coder-30B-A3B has no thinking mode. Per-dataset throughput and accepted length: App. D.

accepted length in eleven of the twelve (App. D). On the Qwen3 verifiers the margin under sampling stays within two points of the greedy margin; on Llama-3.1-8B it halves ($+16\%$ at $T{=}0$, $+8\%$ at $T{=}1$).

Throughput is the number of tokens committed per verification cycle divided by the time a cycle takes, so a method can gain on either axis, and *DCOCO* gains on both. On Qwen3-8B it attains the highest τ in every cell of both configurations (Tab. 7 and 9). Relative to the other correction methods, *DCOCO* both accepts longer prefixes and runs shorter cycles: against the strongest baseline, 7–8% longer accepted prefixes and 6% shorter cycles multiply to the 13–15% throughput gain. Relative to the uncorrected drafter, it pays 3–4% per cycle for 18–27% longer accepted prefixes (App. C). Absolute speedups follow the drafter's base acceptance: with thinking off, math and code reach 3.7–6.1× while open-ended chat stays at 1.8–1.9× (2.3–2.8× and 1.7–2.1× at $T{=}0$ in extended thinking).

App. B ablates the corrector's two input pathways and its training: removing the pair state costs 7.3%/5.6% of accepted length ($T{=}0$/$T{=}1$) and removing the candidate representations 4.8%/3.7%, uniformly across the suites, while freezing the drafter, the corrector as a pure drop-in, still improves DFlash by 4.4%, with joint training worth a further $\approx 15\%$.

5.2 Scaling with concurrency

LLM serving systems batch concurrent requests to improve GPU utilization and throughput. We therefore examine whether the benefits of draft correction persist as serving load increases. Fig. 3 reports aggregate throughput from 4 to 32 concurrent streams on Qwen3-8B with thinking off, averaged over all ten benchmark suites. *DCOCO* achieves the highest throughput at every concurrency, with margins over the strongest alternative reaching 23% under greedy decoding and 18% under sampling, and remaining 19% and 12%, respectively, at 32 streams. The comparison with DFlash in Fig. 4 reveals how these gains evolve under load. Although accepted length remains nearly unchanged, Domino and DSpark progressively lose their throughput advantage and both become slower than the uncorrected drafter at the highest concurrency as their correction overhead outweighs the benefit in this regime. In contrast, *DCOCO* remains faster than DFlash throughout, preserving the benefit of correction even when the sequential alternatives incur a net slowdown. Additional results for the distinct verifiers are reported in App. F.

5.3 Path selection strategy

We compare three selection strategies over the same grid of transition scores produced by a trained corrector. Global-Sum maximizes the sum of transition scores across the block, while prefix-first greedy(Eq. 7) selects each successor using only the scores associated with the chosen predecessor. To test whether emphasizing earlier transitions improves global selection, we also evaluate a geometrically discounted variant,

$$\hat{k}_{1:B} \;=\; \arg\max_{k_1,\dots,k_B} \Big[\, \ell_1(k_1) \;+\; \sum_{p=1}^{B-1} \lambda^{\,p-1}\, S_p(k_p, k_{p+1}) \Big]\,, \qquad \textsc{geom}(\lambda)\,, \tag{8}$$

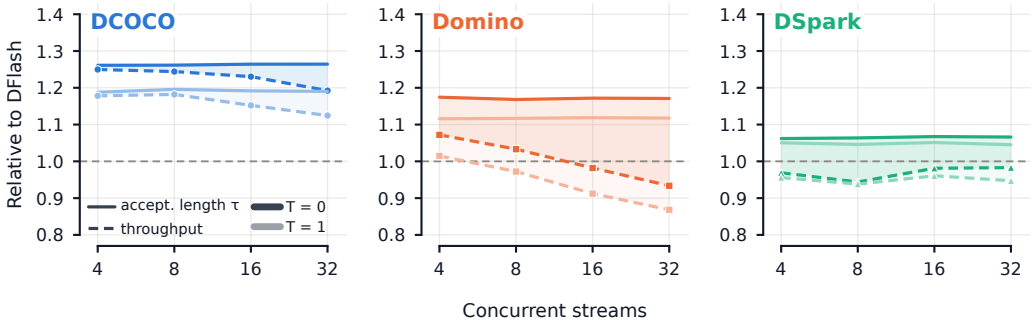


Figure 4: At high concurrency, overhead outweighs acceptance gains for Domino and DSpark, while *DCOCO* remains faster than DFlash. Accepted length τ (solid) and throughput (dashed), relative to DFlash; Shading highlights the separation between the accepted-length and throughput ratios. Qwen3-8B, thinking off, $T = 0$ (dark) and $T = 1$ (light), averaged over all ten benchmark suites. Absolute values: Tab. 19.

		Accepted length τ			Throughput (tok/s)		
Domain	Dataset	Greedy	Geometric	Global-Sum	Greedy	Geometric	Global-Sum
Math	MATH-500	8.039	**8.053**	7.999	**1627.2**	1543.6	1565.9
	AIME-25	7.263	**7.292**	7.226	**1437.7**	1404.1	1392.3
	GSM8K	**6.118**	6.115	6.075	**1176.9**	1160.5	1169.4
Code	HumanEval	5.443	**5.445**	5.401	**1077.0**	1051.8	1059.0
	MBPP	5.215	**5.231**	5.181	**979.3**	971.1	969.7
	LiveCodeBench	**6.982**	6.980	6.930	**1326.4**	1282.4	1275.5
General	MT-Bench	**2.930**	2.913	2.902	**498.1**	487.9	488.3
	Arena-Hard-v2	2.736	**2.738**	2.731	**484.8**	470.9	477.3
	Alpaca	**2.560**	2.551	2.543	**497.1**	476.9	484.2
Average		5.254	**5.258**	5.221	**1011.6**	983.2	986.8

Table 3: Path selection over identical trained scores (Qwen3-8B from-pretrained corrector, T=0, no thinking, 2048 max tokens). Bold: best rule per row and metric. Between greedy and geometric the τ differences carry both signs and are an order of magnitude smaller than either rule's gap to Global-Sum; the throughput ordering is uniform.

which keeps the global argmax but shifts weight toward earlier transitions; $\lambda = 1$ recovers the plain sum. We report $\lambda = 0.75$.

Tab. 3 shows that Global-Sum yields lower accepted length and throughput than Greedy on all nine evaluated suites. Geometric improves accepted length over Global-Sum on every suite and achieves acceptance comparable to Greedy: their per-suite differences have both signs, with average accepted lengths of 5.258 and 5.254, respectively. Greedy nevertheless achieves higher throughput on every suite, averaging a 2.8% improvement over Geometric. These results support greedy selection as an effective acceptance–latency trade-off for the learned scores evaluated here, with an $(O(BK))$ walk and no discount hyperparameter.

6 Conclusion

Parallel block drafters leave two things on the table: their top-K lists provide a significant coverage boost over the top-1 choice, and the identity of each drafted token is information no other slot ever gets to use. *DCOCO* is a lightweight corrector built on exactly these two facts. Reading only what the drafter already produced, it scores every transition between adjacent candidates in parallel with a cheap neural scorer, then greedily takes the best transition at each step to form the proposed continuation, mirroring the way verification consumes a draft, front to back, stopping at the first miss. Across several backbone models and ten benchmarks, the corrected system leads in both accepted length and decoding throughput at almost every point of measurement, and its advantage persists or widens at higher serving concurrencies, where competing correctors fall below the uncorrected drafter. Correction, made cheap enough, is pure profit for speculative decoding.

DISCLOSURE OF LLM USE

We used generative AI tools to assist with code implementation and to refine the clarity and presentation of the manuscript. All research ideas and hypotheses underlying the final method were developed by the authors. All AI-assisted code was reviewed, verified, and executed by the authors, and all AI-assisted text was reviewed and edited by the authors. We take full responsibility for the final content of this work, including its code, experimental results, and scientific claims.

REFERENCES

Talor Abramovich, Maor Ashkenazi, Izzy Putterman, Benjamin Chislett, Tiyasa Mitra, Bita Darvish Rouhani, Ran Zilberstein, and Yonatan Geifman. Speed-bench: A unified and diverse benchmark for speculative decoding. In *Proceedings of the 43rd International Conference on Machine Learning (ICML)*, 2026.

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

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

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

Tianle Cai, Yuhong Li, Zhengyang Geng, Hongwu Peng, Jason D. Lee, Deming Chen, and Tri Dao. Medusa: Simple LLM inference acceleration framework with multiple decoding heads. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 5209–5235. PMLR, 21–27 Jul 2024.

Sahil Chaudhary. Code alpaca: An instruction-following llama model for code generation. https://github.com/sahil280114/codealpaca, 2023.

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

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

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

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

Yunfei Cheng, Aonan Zhang, Xuanyu Zhang, Chong Wang, and Yi Wang. Recurrent drafter for fast speculative decoding in large language models, 2024. URL https://arxiv.org/abs/2403.09919.

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.

Jiatao Gu, James Bradbury, Caiming Xiong, Victor O.K. Li, and Richard Socher. Non-autoregressive neural machine translation. In *International Conference on Learning Representations*, 2018.

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

Naman Jain, King Han, Alex Gu, Wen-Ding Li, Fanjia Yan, Tianjun Zhang, Sida Wang, Armando Solar-Lezama, Koushik Sen, and Ion Stoica. Livecodebench: Holistic and contamination free evaluation of large language models for code. In *The International Conference on Learning Representations (ICLR)*, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/hash/94074dd5a072d28ff75a76dabed43767-Abstract-Conference.html.

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

Chanhyuk Lee, Jaehoon Yoo, Manan Agarwal, Sheel Shah, Jerry Huang, Aditi Raghunathan, Seunghoon Hong, Nicholas M. Boffi, and Jinwoo Kim. Flow map language models: One-step language modeling via continuous denoising. *arXiv preprint arXiv:2602.16813*, 2026.

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

Tianle Li, Wei-Lin Chiang, Evan Frick, Lisa Dunlap, Tianhao Wu, Banghua Zhu, Joseph E Gonzalez, and Ion Stoica. From crowdsourced data to high-quality benchmarks: Arena-hard and benchbuilder pipeline. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2025a.

Yuhui Li, Fangyun Wei, Chao Zhang, and Hongyang Zhang. EAGLE: Speculative sampling requires rethinking feature uncertainty. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 28935–28948. PMLR, 21–27 Jul 2024a.

Yuhui Li, Fangyun Wei, Chao Zhang, and Hongyang Zhang. Eagle-2: Faster inference of language models with dynamic draft trees. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, 2024b.

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

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

NVIDIA. Nvidia nemotron nano 2: An accurate and efficient hybrid mamba-transformer reasoning model, 2025. URL https://arxiv.org/abs/2508.14444.

Rohan Taori, Ishaan Gulrajani, Tianyi Zhang, Yann Dubois, Xuechen Li, Carlos Guestrin, Percy Liang, and Tatsunori B. Hashimoto. Stanford alpaca: An instruction-following llama model. https://github.com/tatsu-lab/stanford_alpaca, 2023.

Zhuofan Wen, Shangtong Gui, and Yang Feng. Speculative decoding with ctc-based draft model for llm inference acceleration. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2024.

Lefan Zhang, Xiaodan Wang, Yanhua Huang, and Ruiwen Xu. Learning harmonized representations for speculative sampling. In *International Conference on Learning Representations*, 2025.

Yifan Zhang and Team Math-AI. American invitational mathematics examination (aime) 2025, 2025.

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

A TRAINING DETAILS

Training loss. For a single block the correction term is a weight-normalised cross-entropy over the K^2 jointly softmax-normalised transition scores,

$$\mathcal{L}_{\text{corr}} \;=\; -\frac{1}{Z}\sum_{p=1}^{B-1} e^{-(p-1)/\rho}\, m_p \,\log \frac{\exp S_p(a_p^\star, b_p^\star)}{\sum_{a,b=1}^{K} \exp S_p(a,b)}, \qquad Z = \sum_{p=1}^{B-1} e^{-(p-1)/\rho}\, m_p, \quad \rho = 4, \tag{9}$$

where the label $(a_p^\star, b_p^\star)$ is the pair of verifier-argmax tokens at the two slots, computed over the teacher-forced context. The mask $m_p \in \{0, 1\}$ is 1 only when both verifier tokens lie inside their slots' top-K candidate sets and both slots pass the alignment mask; transitions the drafter cannot reach are dropped from supervision rather than taught. In a batch, the sums in equation 9 and Z extend jointly over all block, transition pairs, a single weighted mean, not a mean of per-block means, and Z is clamped below by 1, which bites only when a block has no supervised transition. The complete loss is given by the adding the correction loss, with unit weight, to DFlash's own loss.

Training data. Following DFlash, we use approximately 800K prompts from the stem, math, and code splits of the Nemotron Post-Training Dataset v2 (NVIDIA, 2025) and CodeAlpaca-20k (Chaudhary, 2023). Each verifier regenerates its own continuations using greedy decoding ($T = 0$), with thinking disabled and at most 4096 generated tokens. For each verifier, all methods train on the same resulting corpus.

Training protocol. For each verifier, *DCOCO* is initialized from a DFlash drafter trained for six epochs, then trained for six further epochs alongside the corrector. Baseline results use six training epochs under each method's published objective; comparisons with extended training are reported in Appendix B.

Optimization and checkpoint selection. All methods use AdamW with learning rate 6×10^{-4}, no weight decay, and a cosine learning-rate schedule with 4% warmup. Training uses token-budgeted sequence packing with a budget of 4096 tokens per GPU per micro-step across eight GPUs and 23 gradient-accumulation steps. This corresponds to a nominal budget of $4096 \times 8 \times 23 = 753{,}664$ tokens per optimizer step; the number of sequences varies with their lengths. Validation metrics are diagnostic and are not used for checkpoint selection. Each method is trained once with seed 42, so variability across training seeds is not measured. The drafter loss used by *DCOCO* is specified above; baselines use their published losses.

Relation of equation 9 to the decode rule. Because the label fixes the true predecessor, the joint cross-entropy orders the true successor above every alternative in the predecessor's row, precisely the conditional decision that equation 7 executes at decode time, while additionally suppressing every competing row, so the training objective is strictly stronger than the decision rule it must support. The exponential training weight emphasizes earlier transitions, motivated by the requirement that preceding tokens must be accepted before a later transition can contribute. It serves as a fixed proxy for prefix survival rather than an estimated acceptance probability. This training weight is distinct from the geometric discount evaluated in Section 5.3: the former controls the relative contribution of positions to the training loss, while the latter changes path selection at inference.

Label extraction. Training does not execute an online speculative loop. The context is a fixed, teacher-forced corpus; the verifier's hidden states over it are precomputed once, its output head is applied under no gradient, and the argmax at each slot, when present in that slot's top-K, supplies the label. Supervision is therefore conditioned exclusively on verifier-correct prefixes. This is not a train/serve mismatch but a consequence of prefix acceptance: a transition's decision is only ever consumed when every earlier slot was accepted, i.e. matches the verifier, so verifier-correct prefixes are precisely the states in which the corrector's output matters.

Corrector architecture. The corrector uses candidate-representation width $d_e = 128$, pair-state width $d_h = 128$, and $K = 16$ candidates per drafted slot. These widths correspond to $r = 128$ in the method description. The two-layer scoring MLP has input width $d_h + 2d_e + 2 = 386$, hidden

width 256 with GELU activation, and a scalar output. The pair projection has shape $128 \times (2d)$ and the candidate–context fusion projection has shape 128×256. On Qwen3-8B, the five-layer drafter has hidden width $d = 4096$ and the corrector has 2,229,505 parameters. The same corrector architecture has 1,443,073 parameters on Qwen3-4B, reflecting the smaller input hidden width. The frozen token-embedding table is shared with the underlying model and is not an additional corrector parameter allocation. Inference uses BF16 weights.

Draft-length convention. The nominal block-size setting is 16 for all methods $B = 15$ refers to *DCOCO*'s drafted positions, excluding the anchor supplied as context. *DCOCO* therefore scores $(B-1)K^2 = 14 \times 16^2 = 3584$ adjacent candidate transitions and commits at most 16 tokens per round, including the verifier's additional token. The comparisons use these native configurations rather than a strictly matched number of drafted tokens.

B TRAINING BUDGET AND CONVERGENCE

Our arm trains for twelve epochs in total, six drafter-only, matching the published DFlash recipe, followed by six joint epochs with the corrector, while every baseline trains for its standard six. To rule out budget rather than the corrector as the source of the gains, we continue each baseline (DFlash, Domino, DSpark) to twelve epochs under its own objective and recipe, unchanged. Table 4 reports the result: all three are converged at six epochs, doubling their training moves accepted length by no more than 1.9%, whereas the joint phase is where our arm's gain accrues. The comparison in the main tables is therefore between converged systems, not between budgets.

Arm	τ @ 6 epochs	τ @ 12 epochs	$\Delta\tau$	ΔTPS
DFlash	3.913	3.948	+0.035 (+0.9%)	+11.0 (+1.4%)
Domino	4.613	4.701	+0.088 (+1.9%)	+10.9 (+1.3%)
DSpark	4.254	4.321	+0.067 (+1.6%)	+3.1 (+0.4%)
DCOCO (6 + 6 joint)	3.913 (drafter-only @ 6)	4.985	+1.072 (+27.4%)	+176.6 (+22.6%)

Table 4: Extending baseline training to match our total budget, Q8B no-thinking, max generated tokens 2048, T=0 averaged across the 10 benchmarks. Baselines show no more than 2% improvement from a second six epochs; the joint phase accounts for our arm's improvement.

Component ablations. The corrector's correction term Δ_θ reads two learned input pathways: the candidate representations z_p^a, z_{p+1}^b and the candidate-independent pair state s_p. We retrain the corrector with each pathway removed, under the recipe of App. A unchanged: the $-$pair state variant drops the pair state (transitions scored from candidate representations and logits alone) and the $-$cand state variant drops the candidate representations (a candidate-blind scorer that recalibrates the draft logits from the pair state alone). The Frozen backbone ablation keeps the corrector intact but freezes the drafter at its six-epoch DFlash checkpoint, so the corrector trains as a pure drop-in and the backbone never co-adapts. Tables 5 and 6 report throughput and accepted length, respectively. Removing either input pathway hurts both metrics across all ten suites and both temperatures. The pair state carries the larger share: without it, accepted length drops by $7.3\%/5.6\%$ ($T{=}0/T{=}1$ averages) and throughput by $6.3\%/4.6\%$; without candidate representations, accepted length drops by $4.8\%/3.7\%$ and throughput by $4.6\%/3.3\%$. Neither pathway is redundant, and their contributions are consistent across domains, largest on math and code. Even with the drafter frozen, training the corrector increases average throughput from 783.1 to 817.7 tok/s at $T{=}0$ ($+4.4\%$). Freezing the drafter nevertheless reduces average throughput by $15.9\%/12.9\%$ at $T{=}0/T{=}1$, respectively, relative to joint training. By comparison, extending drafter training alone improves throughput by at most 2% (Tab. 4), suggesting that the gains from joint training cannot be explained by additional drafter training alone.

(a) Greedy decoding, T=0 (mean of 3 runs)

Domain	Dataset	*DCOCO*	−pair state	−cand state	Frozen backbone
Math	MATH-500	**1599.3**	1497.7 (−6.4%)	1521.8 (−4.8%)	1333.6 (−16.6%)
	AIME-25	**1475.6**	1382.0 (−6.3%)	1387.9 (−5.9%)	1236.1 (−16.2%)
	GSM8K	**1130.3**	1042.3 (−7.8%)	1073.2 (−5.0%)	920.2 (−18.6%)
Code	HumanEval	**1113.9**	1035.6 (−7.0%)	1055.9 (−5.2%)	929.5 (−16.6%)
	MBPP	**1121.6**	1048.7 (−6.5%)	1067.8 (−4.8%)	929.5 (−17.1%)
	LiveCodeBench	**1202.9**	1121.2 (−6.8%)	1139.7 (−5.3%)	1019.2 (−15.3%)
General	MT-Bench	**575.2**	548.4 (−4.7%)	565.2 (−1.8%)	499.6 (−13.2%)
	Arena-Hard-v2	**516.2**	493.4 (−4.4%)	504.7 (−2.2%)	450.1 (−12.8%)
	Alpaca	**485.2**	461.5 (−4.9%)	472.2 (−2.7%)	429.3 (−11.5%)
	SpeedBench	**498.0**	475.8 (−4.5%)	481.1 (−3.4%)	430.3 (−13.6%)
Average		**971.8**	910.7 (−6.3%)	927.1 (−4.6%)	817.7 (−15.9%)

(b) Sampling, T=1 (mean±std over seeds 42/43/44)

Domain	Dataset	*DCOCO*	−pair state	−cand state	Frozen backbone
Math	MATH-500	**1133.8** ±5.0	1094.1 ±3.5 (−3.5%)	1101.1 ±6.5 (−2.9%)	988.6 ±10.2 (−12.8%)
	AIME-25	**884.8** ±13.7	848.2 ±22.9 (−4.1%)	859.7 ±6.4 (−2.8%)	785.3 ±13.2 (−11.2%)
	GSM8K	**1007.8** ±18.7	940.7 ±6.0 (−6.7%)	971.8 ±11.5 (−3.6%)	837.5 ±10.5 (−16.9%)
Code	HumanEval	**945.3** ±4.4	882.5 ±1.4 (−6.6%)	901.9 ±8.0 (−4.6%)	806.4 ±9.6 (−14.7%)
	MBPP	**894.9** ±6.4	852.4 ±2.4 (−4.7%)	855.2 ±3.5 (−4.4%)	771.8 ±2.7 (−13.8%)
	LiveCodeBench	**766.5** ±9.5	736.2 ±4.2 (−4.0%)	738.2 ±5.2 (−3.7%)	678.2 ±6.0 (−11.5%)
General	MT-Bench	**507.9** ±1.5	492.1 ±3.3 (−3.1%)	496.9 ±5.2 (−2.2%)	456.3 ±4.8 (−10.2%)
	Arena-Hard-v2	**439.7** ±3.0	422.5 ±1.4 (−3.9%)	430.9 ±1.5 (−2.0%)	393.4 ±1.5 (−10.5%)
	Alpaca	**444.2** ±1.5	425.6 ±2.0 (−4.2%)	432.6 ±3.0 (−2.6%)	402.0 ±2.4 (−9.5%)
	SpeedBench	**460.0** ±16.9	442.5 ±3.4 (−3.8%)	447.0 ±5.3 (−2.8%)	398.5 ±2.7 (−13.4%)
Average		**748.5**	713.7 (−4.6%)	723.5 (−3.3%)	651.8 (−12.9%)

Table 5: Ablations: throughput (output tokens/s, per-stream), Qwen3-8B, thinking off, concurrency 1, max 2048 generated tokens, all ten suites. Each ablation cell reports its value with the change relative to *DCOCO* in parentheses. −pair state removes the corrector's dedicated previous-state channel; −cand state zeroes the candidate representations entering the transition scorer; Frozen backbone trains only the corrector on the fixed six-epoch DFlash backbone. Bold: best measured system per row.

(a) Greedy decoding, T=0 (mean of 3 runs)

Domain	Dataset	*DCOCO*	−pair state	−cand state	Frozen backbone
Math	MATH-500	**8.256**	7.588 (−8.1%)	7.802 (−5.5%)	6.801 (−17.6%)
	AIME-25	**7.506**	6.935 (−7.6%)	7.043 (−6.2%)	6.250 (−16.7%)
	GSM8K	**6.010**	5.482 (−8.8%)	5.697 (−5.2%)	4.845 (−19.4%)
Code	HumanEval	**5.746**	5.340 (−7.1%)	5.439 (−5.3%)	4.764 (−17.1%)
	MBPP	**5.858**	5.379 (−8.2%)	5.575 (−4.8%)	4.800 (−18.1%)
	LiveCodeBench	**6.153**	5.712 (−7.2%)	5.790 (−5.9%)	5.196 (−15.5%)
General	MT-Bench	**2.930**	2.760 (−5.8%)	2.860 (−2.4%)	2.539 (−13.4%)
	Arena-Hard-v2	**2.605**	2.479 (−4.9%)	2.550 (−2.1%)	2.257 (−13.3%)
	Alpaca	**2.474**	2.344 (−5.3%)	2.407 (−2.7%)	2.173 (−12.2%)
	SpeedBench	**2.561**	2.434 (−5.0%)	2.499 (−2.4%)	2.207 (−13.8%)
Average		**5.010**	4.645 (−7.3%)	4.767 (−4.8%)	4.183 (−16.5%)

(b) Sampling, T=1 (mean±std over seeds 42/43/44)

Domain	Dataset	*DCOCO*	−pair state	−cand state	Frozen backbone
Math	MATH-500	**5.801** ±0.025	5.495 ±0.019 (−5.3%)	5.589 ±0.037 (−3.7%)	5.004 ±0.052 (−13.7%)
	AIME-25	**4.461** ±0.070	4.235 ±0.114 (−5.1%)	4.318 ±0.039 (−3.2%)	3.951 ±0.068 (−11.4%)
	GSM8K	**5.324** ±0.104	4.912 ±0.048 (−7.7%)	5.089 ±0.061 (−4.4%)	4.392 ±0.057 (−17.5%)
Code	HumanEval	**4.839** ±0.022	4.527 ±0.006 (−6.5%)	4.618 ±0.031 (−4.6%)	4.109 ±0.050 (−15.1%)
	MBPP	**4.636** ±0.033	4.339 ±0.014 (−6.4%)	4.422 ±0.023 (−4.6%)	3.962 ±0.016 (−14.5%)
	LiveCodeBench	**3.891** ±0.050	3.714 ±0.010 (−4.5%)	3.719 ±0.026 (−4.4%)	3.436 ±0.032 (−11.7%)
General	MT-Bench	**2.581** ±0.007	2.471 ±0.017 (−4.3%)	2.507 ±0.026 (−2.9%)	2.315 ±0.023 (−10.3%)
	Arena-Hard-v2	**2.218** ±0.015	2.119 ±0.007 (−4.4%)	2.175 ±0.009 (−2.0%)	1.972 ±0.008 (−11.1%)
	Alpaca	**2.265** ±0.009	2.161 ±0.010 (−4.6%)	2.205 ±0.014 (−2.6%)	2.034 ±0.012 (−10.2%)
	SpeedBench	**2.373** ±0.101	2.252 ±0.024 (−5.1%)	2.313 ±0.026 (−2.5%)	2.050 ±0.011 (−13.6%)
Average		**3.839**	3.623 (−5.6%)	3.695 (−3.7%)	3.322 (−13.5%)

Table 6: Ablations: accepted length τ, Qwen3-8B, thinking off, concurrency 1, max 2048 generated tokens, all ten suites. Each ablation cell reports its value with the change relative to *DCOCO* in parentheses. −pair state removes the corrector's dedicated previous-state channel; −cand state zeroes the candidate representations entering the transition scorer; Frozen backbone trains only the corrector on the fixed six-epoch DFlash backbone. Bold: best measured system per row.

C EVALUATION DETAILS

Benchmarks and generation settings. We evaluate ten benchmarks across three domains: mathematics (GSM8K (Cobbe et al., 2021), MATH-500 (Lightman et al., 2024), AIME-25 (Zhang & Math-AI, 2025)), code (HumanEval (Chen et al., 2021), MBPP (Austin et al., 2021), LiveCodeBench (Jain et al., 2025)), and general tasks and chat (SpeedBench (Abramovich et al., 2026), Alpaca (Taori et al., 2023), MT-Bench (Zheng et al., 2023), Arena-Hard-v2 (Li et al., 2025a)). We evaluate greedy decoding at $T = 0$ and stochastic decoding at $T = 1$. The maximum number of generated tokens is 2048 without thinking and 4096 with extended thinking. Both thinking settings are evaluated for Qwen3-4B and Qwen3-8B; Qwen3-Coder-30B-A3B and Llama-3.1-8B are evaluated without a separate thinking mode.

Serving configuration. All systems run on a single NVIDIA B200 GPU with tensor parallelism 1, using vLLM 0.27.0 and PyTorch 2.13.0+cu130. Model precision is BF16 (`-dtype auto`); KV-cache dtype uses the automatic setting (`-kv-cache-dtype auto`). Prefix caching is disabled (`-no-enable-prefix-caching`). CUDA Graphs are enabled, eager execution is not forced, and no optimization-level override is supplied. Evaluation begins after server initialization and CUDA Graph capture complete.

Timing and aggregation. For request i, let n_i denote the number of generated output tokens and L_i the request latency in seconds, including prefill. Its reported rate is n_i/L_i, rather than the inverse inter-token latency. At concurrency 1, we report the median request-level rate within each benchmark. The reported ten-suite averages give each benchmark equal weight. No initial evaluation requests are discarded as warmup. For $T = 1$, the reported mean and standard deviation are computed across generation seeds 42, 43, and 44, after computing the benchmark statistic for each seed. These quantify evaluation variability for fixed checkpoints, not variability across independent training runs.

Evaluation subsets. We use fixed benchmark subsets selected with subsampling seed 1234, shared across methods. GSM8K, MATH-500, HumanEval, MBPP, LiveCodeBench, Alpaca, and Arena-Hard-v2 each contain 128 selected rows. AIME-25 uses all 30 rows and MT-Bench all 80 rows. SPEED-Bench contains 80 rows from each of 11 subsets, for 880 rows in total, and is treated as one of the ten benchmark suites. The evaluation manifest records the exact selected row indices. For LiveCodeBench, we sample from the 1,055 available rows in the `code_generation_lite` v5 and v6 release files, `test5.jsonl` and `test6.jsonl` (`version_tag=v5_v6`).

D FULL PER-DATASET RESULTS

Per-dataset throughput and accepted length for each configuration not shown in the main text, in the format of Tables 1 and 7.

Domain	Dataset	DFlash T=0	DFlash T=1	Domino T=0	Domino T=1	DSpark T=0	DSpark T=1	*DCOCO* T=0	*DCOCO* T=1
Math	MATH-500	6.173	4.751 ±0.024	7.500	5.518 ±0.022	6.815	5.245 ±0.053	**8.039**	**5.755** ±0.073
	AIME-25	5.554	3.751 ±0.064	6.803	4.246 ±0.076	6.289	3.975 ±0.059	**7.263**	**4.486** ±0.037
	GSM8K	4.632	4.217 ±0.024	5.563	4.960 ±0.122	4.992	4.526 ±0.019	**6.118**	**5.440** ±0.141
Code	HumanEval	4.245	3.771 ±0.009	4.971	4.319 ±0.075	4.576	4.046 ±0.033	**5.443**	**4.646** ±0.043
	MBPP	3.992	3.507 ±0.014	4.857	4.121 ±0.117	4.345	3.767 ±0.009	**5.215**	**4.428** ±0.008
	LiveCodeBench	5.530	4.398 ±0.140	6.503	4.908 ±0.083	6.342	4.920 ±0.089	**6.982**	**5.431** ±0.160
General	MT-Bench	2.444	2.249 ±0.010	2.736	2.475 ±0.044	2.456	2.318 ±0.016	**2.930**	**2.581** ±0.007
	Arena-Hard-v2	2.288	1.967 ±0.029	2.473	2.101 ±0.036	2.288	2.019 ±0.009	**2.736**	**2.286** ±0.020
	Alpaca	2.147	1.983 ±0.025	2.393	2.169 ±0.026	2.242	2.051 ±0.009	**2.560**	**2.313** ±0.024
	SpeedBench	2.124	1.994 ±0.032	2.327	2.156 ±0.036	2.195	2.070 ±0.016	**2.561**	**2.373** ±0.101
Average		3.913	3.259	4.613	3.697	4.254	3.494	**4.985**	**3.974**

Table 7: Mechanism: mean accepted length τ, Qwen3-8B, thinking off, companion to Table 1; no-spec commits one token per round by construction and is omitted. τ explains the TPS movement; the corrector's own overhead is visible only in throughput.

(a) Greedy decoding, T=0 (single deterministic run)

Domain	Dataset	No-spec	DFlash	Domino	DSpark	*DCOCO*
Math	MATH-500	262.0	608.8 (2.32×)	617.1 (2.36×)	567.4 (2.17×)	**724.9 (2.77×)**
	AIME-25	260.3	586.9 (2.26×)	612.5 (2.35×)	574.9 (2.21×)	**703.1 (2.70×)**
	GSM8K	264.1	555.6 (2.10×)	559.7 (2.12×)	518.6 (1.96×)	**646.5 (2.45×)**
Code	HumanEval	260.6	562.9 (2.16×)	572.8 (2.20×)	524.7 (2.01×)	**654.4 (2.51×)**
	MBPP	262.8	535.6 (2.04×)	548.2 (2.09×)	498.2 (1.90×)	**618.1 (2.35×)**
	LiveCodeBench	259.2	541.9 (2.09×)	554.5 (2.14×)	506.6 (1.95×)	**624.2 (2.41×)**
General	MT-Bench	264.4	477.4 (1.81×)	481.9 (1.82×)	465.3 (1.76×)	**545.6 (2.06×)**
	Arena-Hard-v2	260.4	439.1 (1.69×)	421.9 (1.62×)	404.8 (1.56×)	**508.3 (1.95×)**
	Alpaca	265.8	388.1 (1.46×)	389.0 (1.46×)	364.2 (1.37×)	**440.8 (1.66×)**
	SpeedBench	264.3	403.3 (1.53×)	403.3 (1.53×)	380.9 (1.44×)	**458.7 (1.74×)**
Average		262.4	510.0 (1.95×)	516.1 (1.97×)	480.6 (1.83×)	**592.5 (2.26×)**

(b) Sampling, T=1 (mean±std over seeds 42/43/44)

Domain	Dataset	No-spec	DFlash	Domino	DSpark	*DCOCO*
Math	MATH-500	261.8 ±0.0	567.0 ±3.3 (2.17×)	566.2 ±5.3 (2.16×)	538.8 ±3.0 (2.06×)	**664.1 ±7.6 (2.54×)**
	AIME-25	259.7 ±0.0	532.1 ±4.7 (2.05×)	570.5 ±13.0 (2.20×)	514.4 ±17.1 (1.98×)	**629.3 ±15.4 (2.42×)**
	GSM8K	264.0 ±0.1	528.1 ±3.6 (2.00×)	522.7 ±1.2 (1.98×)	495.6 ±5.0 (1.88×)	**599.6 ±6.2 (2.27×)**
Code	HumanEval	259.9 ±0.1	502.0 ±3.1 (1.93×)	508.6 ±5.4 (1.96×)	479.4 ±4.9 (1.85×)	**572.4 ±8.6 (2.20×)**
	MBPP	262.3 ±0.2	479.1 ±1.0 (1.83×)	484.1 ±7.1 (1.85×)	446.2 ±0.9 (1.70×)	**541.7 ±6.6 (2.07×)**
	LiveCodeBench	258.6 ±0.0	474.2 ±0.2 (1.83×)	479.5 ±3.8 (1.85×)	446.6 ±2.0 (1.73×)	**535.4 ±2.0 (2.07×)**
General	MT-Bench	264.2 ±0.1	427.2 ±9.8 (1.62×)	436.6 ±4.7 (1.65×)	404.5 ±8.6 (1.53×)	**475.4 ±7.7 (1.80×)**
	Arena-Hard-v2	259.8 ±0.0	391.4 ±5.5 (1.51×)	383.9 ±5.9 (1.48×)	366.7 ±5.6 (1.41×)	**441.4 ±13.3 (1.70×)**
	Alpaca	265.6 ±0.1	374.9 ±1.4 (1.41×)	371.9 ±7.3 (1.40×)	352.2 ±0.9 (1.33×)	**421.5 ±3.1 (1.59×)**
	SpeedBench	263.9 ±0.1	375.8 ±5.7 (1.42×)	371.0 ±5.7 (1.41×)	344.6 ±10.0 (1.31×)	**423.2 ±11.3 (1.60×)**
Average		262.0	465.2 (1.78×)	469.5 (1.79×)	438.9 (1.68×)	**530.4 (2.03×)**

Table 8: Throughput (output tokens/s, per-stream) and speedup over no-spec decoding, Qwen3-8B, thinking on, concurrency 1, max 4096 generated tokens. Same format as Table 1.

Domain	Dataset	DFlash T=0	DFlash T=1	Domino T=0	Domino T=1	DSpark T=0	DSpark T=1	*DCOCO* T=0	*DCOCO* T=1
Math	MATH-500	2.947	2.759 ±0.017	3.298	3.057 ±0.018	3.102	2.884 ±0.016	**3.590**	**3.304 ±0.021**
	AIME-25	2.887	2.625 ±0.035	3.250	2.981 ±0.015	3.039	2.788 ±0.021	**3.526**	**3.170 ±0.013**
	GSM8K	2.717	2.560 ±0.004	3.025	2.820 ±0.005	2.788	2.649 ±0.012	**3.287**	**3.040 ±0.015**
Code	HumanEval	2.786	2.443 ±0.004	3.099	2.702 ±0.016	2.876	2.561 ±0.004	**3.331**	**2.879 ±0.018**
	MBPP	2.588	2.326 ±0.003	2.892	2.555 ±0.020	2.681	2.420 ±0.009	**3.093**	**2.703 ±0.011**
	LiveCodeBench	2.631	2.325 ±0.010	2.934	2.562 ±0.014	2.720	2.414 ±0.015	**3.152**	**2.715 ±0.008**
General	MT-Bench	2.307	2.096 ±0.013	2.541	2.288 ±0.007	2.432	2.163 ±0.009	**2.738**	**2.435 ±0.016**
	Arena-Hard-v2	2.160	1.905 ±0.010	2.333	2.045 ±0.002	2.197	1.968 ±0.013	**2.576**	**2.217 ±0.029**
	Alpaca	1.982	1.848 ±0.007	2.153	2.012 ±0.028	2.028	1.917 ±0.020	**2.314**	**2.146 ±0.029**
	SpeedBench	2.008	1.849 ±0.026	2.193	1.997 ±0.025	2.085	1.878 ±0.043	**2.380**	**2.159 ±0.049**
Average		2.501	2.274	2.772	2.502	2.595	2.364	**2.999**	**2.677**

Table 9: Mean accepted length τ, Qwen3-8B, thinking on, companion to Table 8; *DCOCO* attains the best τ in every cell in this configuration.

(a) Greedy decoding, T=0 (single deterministic run)

Domain	Dataset	No-spec	DFlash	Domino	DSpark	*DCOCO*
Math	MATH-500	278.4	1356.9 (4.88×)	1477.5 (5.31×)	1367.6 (4.91×)	**1636.0 (5.88×)**
	AIME-25	274.8	1465.6 (5.33×)	1601.0 (5.83×)	1491.2 (5.43×)	**1811.7 (6.59×)**
	GSM8K	278.4	891.0 (3.20×)	915.5 (3.29×)	865.8 (3.11×)	**1160.8 (4.17×)**
Code	HumanEval	276.4	1049.9 (3.80×)	1144.7 (4.14×)	1038.0 (3.76×)	**1260.3 (4.56×)**
	MBPP	282.3	1036.3 (3.67×)	**1128.5 (4.00×)**	1039.0 (3.68×)	989.7 (3.51×)
	LiveCodeBench	278.7	990.0 (3.55×)	1050.5 (3.77×)	972.1 (3.49×)	**1448.8 (5.20×)**
General	MT-Bench	281.6	420.8 (1.49×)	436.4 (1.55×)	416.9 (1.48×)	**572.5 (2.03×)**
	Arena-Hard-v2	279.0	409.0 (1.47×)	407.3 (1.46×)	387.6 (1.39×)	**570.5 (2.05×)**
	Alpaca	279.2	409.8 (1.47×)	426.7 (1.53×)	402.1 (1.44×)	**478.8 (1.72×)**
	SpeedBench	279.2	413.0 (1.48×)	419.6 (1.50×)	399.7 (1.43×)	**494.6 (1.77×)**
Average		278.8	844.2 (3.03×)	900.8 (3.24×)	838.0 (3.01×)	**1042.4 (3.75×)**

(b) Sampling, T=1 (mean±std over seeds 42/43/44)

Domain	Dataset	No-spec	DFlash	Domino	DSpark	*DCOCO*
Math	MATH-500	277.8 ±0.3	429.1 ±52.7 (1.55×)	373.8 ±98.2 (1.35×)	343.2 ±14.7 (1.24×)	**545.5 ±23.7 (1.96×)**
	AIME-25	274.9 ±0.6	296.1 ±81.6 (1.08×)	253.1 ±18.9 (0.92×)	247.1 ±9.3 (0.90×)	**354.4 ±27.7 (1.29×)**
	GSM8K	277.7 ±0.1	610.7 ±66.2 (2.20×)	**692.5 ±66.5 (2.49×)**	569.3 ±66.7 (2.05×)	681.6 ±34.1 (2.45×)
Code	HumanEval	275.9 ±0.0	814.5 ±21.0 (2.95×)	883.3 ±15.3 (3.20×)	807.0 ±31.2 (2.92×)	**895.8 ±67.9 (3.25×)**
	MBPP	282.1 ±0.0	776.8 ±2.2 (2.75×)	**841.3 ±12.8 (2.98×)**	770.8 ±7.9 (2.73×)	793.4 ±64.4 (2.81×)
	LiveCodeBench	277.4 ±0.1	**643.3 ±36.7 (2.32×)**	619.9 ±41.3 (2.24×)	586.4 ±45.6 (2.11×)	545.3 ±21.3 (1.97×)
General	MT-Bench	281.4 ±0.1	364.3 ±24.3 (1.30×)	365.8 ±5.3 (1.30×)	346.3 ±8.4 (1.23×)	**442.3 ±9.2 (1.57×)**
	Arena-Hard-v2	277.8 ±0.1	224.7 ±5.7 (0.81×)	221.8 ±1.0 (0.80×)	221.8 ±7.6 (0.80×)	**310.7 ±0.8 (1.12×)**
	Alpaca	278.9 ±0.1	365.5 ±3.9 (1.31×)	370.0 ±7.2 (1.33×)	364.2 ±6.2 (1.31×)	**401.6 ±5.0 (1.44×)**
	SpeedBench	278.3 ±0.1	347.5 ±5.0 (1.25×)	351.1 ±12.8 (1.26×)	340.5 ±17.4 (1.22×)	**392.7 ±5.5 (1.41×)**
Average		278.2	487.2 (1.75×)	497.3 (1.79×)	459.6 (1.65×)	**536.3 (1.93×)**

Table 10: Throughput (output tokens/s, per-stream) and speedup over no-spec decoding, Llama-3.1-8B, thinking off, concurrency 1, max 2048 generated tokens. Same format as Table 1.

Domain	Dataset	DFlash		Domino		DSpark		*DCOCO*	
		T=0	T=1	T=0	T=1	T=0	T=1	T=0	T=1
Math	MATH-500	7.194	2.514 ±0.075	7.837	2.393 ±0.278	7.540	2.314 ±0.067	**8.141**	**2.635 ±0.18**
	AIME-25	7.937	**1.758 ±0.149**	8.470	1.662 ±0.145	8.306	1.573 ±0.137	**8.916**	1.714 ±0.15
	GSM8K	4.529	3.071 ±0.374	4.692	**3.496 ±0.339**	4.543	2.897 ±0.242	**5.984**	3.404 ±0.25
Code	HumanEval	5.126	4.186 ±0.092	5.545	**4.523 ±0.059**	5.254	4.258 ±0.179	**6.454**	4.490 ±0.39
	MBPP	5.293	3.929 ±0.129	**5.806**	**4.276 ±0.091**	5.503	4.000 ±0.201	5.560	4.274 ±0.43
	LiveCodeBench	5.292	**3.468 ±0.176**	5.594	3.283 ±0.344	5.411	3.142 ±0.293	**7.283**	2.649 ±0.13
General	MT-Bench	2.546	2.090 ±0.120	2.644	**2.155 ±0.079**	2.593	2.065 ±0.056	**2.802**	2.153 ±0.01
	Arena-Hard-v2	2.466	1.449 ±0.024	2.457	1.455 ±0.003	2.401	1.474 ±0.029	**2.773**	**1.499 ±0.02**
	Alpaca	2.161	1.872 ±0.018	2.242	1.924 ±0.033	2.191	1.950 ±0.011	**2.341**	**1.957 ±0.0**
	SpeedBench	2.253	1.837 ±0.048	2.270	1.852 ±0.054	2.252	1.857 ±0.075	**2.459**	**1.932 ±0.031**
Average		4.480	2.617	4.756	**2.702**	4.599	2.553	**5.271**	2.671

Table 11: Mean accepted length τ, Llama-3.1-8B, thinking off, companion to Table 10. Domino attains best τ on some datasets here, yet the lower average throughput (Table 10), the τ-vs-TPS trade at the aggregate level.

(a) Greedy decoding, T=0 (single deterministic run)

Domain	Dataset	No-spec	DFlash	Domino	DSpark	*DCOCO*
Math	MATH-500	343.2	1043.3 (3.04×)	1138.2 (3.32×)	1050.8 (3.06×)	**1231.8 (3.59×)**
	AIME-25	342.9	928.1 (2.71×)	941.5 (2.75×)	906.7 (2.64×)	**1074.5 (3.13×)**
	GSM8K	348.0	733.5 (2.11×)	728.3 (2.09×)	698.2 (2.01×)	**854.8 (2.46×)**
Code	HumanEval	335.2	874.8 (2.61×)	880.9 (2.63×)	847.7 (2.53×)	**1003.8 (3.00×)**
	MBPP	333.5	759.5 (2.28×)	779.0 (2.34×)	734.6 (2.20×)	**974.0 (2.92×)**
	LiveCodeBench	338.9	684.5 (2.02×)	743.9 (2.20×)	698.9 (2.06×)	**890.8 (2.63×)**
General	MT-Bench	337.0	317.6 (0.94×)	316.9 (0.94×)	298.8 (0.89×)	**510.1 (1.51×)**
	Arena-Hard-v2	333.5	357.1 (1.07×)	347.0 (1.04×)	336.2 (1.01×)	**456.5 (1.37×)**
	Alpaca	348.0	298.3 (0.86×)	302.7 (0.87×)	296.9 (0.85×)	**382.7 (1.10×)**
	SpeedBench	341.5	427.2 (1.25×)	422.8 (1.24×)	403.7 (1.18×)	**472.0 (1.38×)**
Average		340.2	642.4 (1.89×)	660.1 (1.94×)	627.2 (1.84×)	**785.1 (2.31×)**

(b) Sampling, T=1 (mean±std over seeds 42/43/44)

Domain	Dataset	No-spec	DFlash	Domino	DSpark	*DCOCO*
Math	MATH-500	337.5 ±0.2	923.6 ±17.5 (2.74×)	945.3 ±16.2 (2.80×)	902.2 ±8.7 (2.67×)	**1060.9 ±2.7 (3.14×)**
	AIME-25	337.2 ±0.1	646.8 ±59.5 (1.92×)	666.5 ±67.1 (1.98×)	632.3 ±48.4 (1.88×)	**808.8 ±19.1 (2.40×)**
	GSM8K	342.1 ±0.2	698.8 ±15.6 (2.04×)	687.1 ±13.3 (2.01×)	676.2 ±4.7 (1.98×)	**793.8 ±8.5 (2.32×)**
Code	HumanEval	329.5 ±0.1	827.3 ±25.7 (2.51×)	816.0 ±3.5 (2.48×)	782.2 ±16.1 (2.37×)	**908.3 ±14.9 (2.76×)**
	MBPP	327.8 ±0.1	728.1 ±5.5 (2.22×)	749.8 ±12.0 (2.29×)	708.8 ±4.6 (2.16×)	**812.2 ±9.6 (2.48×)**
	LiveCodeBench	333.1 ±0.3	440.2 ±26.2 (1.32×)	446.6 ±10.4 (1.34×)	444.7 ±10.4 (1.34×)	**559.5 ±4.8 (1.68×)**
General	MT-Bench	331.0 ±0.2	308.0 ±8.6 (0.93×)	313.2 ±13.2 (0.95×)	301.9 ±5.8 (0.91×)	**475.8 ±10.3 (1.44×)**
	Arena-Hard-v2	327.9 ±0.0	331.0 ±8.1 (1.01×)	290.9 ±0.9 (0.89×)	298.2 ±10.4 (0.91×)	**386.1 ±2.7 (1.18×)**
	Alpaca	342.1 ±0.2	289.9 ±6.2 (0.85×)	296.5 ±6.9 (0.87×)	291.4 ±1.1 (0.85×)	**377.3 ±1.8 (1.10×)**
	SpeedBench	336.3 ±0.9	375.3 ±44.2 (1.12×)	365.7 ±43.5 (1.09×)	362.0 ±43.0 (1.08×)	**440.1 ±29.5 (1.31×)**
Average		334.4	556.9 (1.67×)	557.8 (1.67×)	540.0 (1.62×)	**662.3 (1.98×)**

Table 12: Throughput (output tokens/s, per-stream) and speedup over no-spec decoding, Qwen3-Coder-30B-A3B, thinking off, concurrency 1, max 2048 generated tokens. Same format as Table 1.

Domain	Dataset	DFlash T=0	DFlash T=1	Domino T=0	Domino T=1	DSpark T=0	DSpark T=1	*DCOCO* T=0	*DCOCO* T=1
Math	MATH-500	6.372	5.639 ±0.062	6.947	6.017 ±0.104	6.536	5.626 ±0.055	**7.224**	**6.221 ±0.016**
	AIME-25	5.566	4.276 ±0.122	6.057	4.439 ±0.091	5.695	4.309 ±0.091	**6.312**	**4.693 ±0.114**
	GSM8K	4.717	4.457 ±0.138	4.998	4.616 ±0.109	4.652	4.372 ±0.029	**5.322**	**5.036 ±0.064**
Code	HumanEval	5.805	5.279 ±0.155	6.189	5.598 ±0.031	5.789	5.285 ±0.044	**6.435**	**5.803 ±0.103**
	MBPP	4.981	4.787 ±0.023	5.391	**5.208 ±0.048**	4.989	4.828 ±0.029	**6.141**	4.987 ±0.063
	LiveCodeBench	4.498	3.200 ±0.045	4.985	3.279 ±0.079	4.668	3.221 ±0.076	**5.441**	**3.331 ±0.032**
General	MT-Bench	2.679	2.516 ±0.024	2.807	2.674 ±0.054	2.669	2.531 ±0.038	**2.997**	**2.762 ±0.069**
	Arena-Hard-v2	2.476	2.124 ±0.026	2.524	2.143 ±0.020	2.400	2.073 ±0.016	**2.648**	**2.205 ±0.018**
	Alpaca	2.115	2.028 ±0.049	2.228	2.112 ±0.030	2.119	2.044 ±0.049	**2.234**	**2.182 ±0.010**
	SpeedBench	2.704	2.393 ±0.299	2.837	2.458 ±0.298	2.640	2.404 ±0.268	**2.989**	**2.762 ±0.221**
Average		4.191	3.670	4.496	3.854	4.216	3.669	**4.774**	**3.998**

Table 13: Mean accepted length τ, Qwen3-Coder-30B-A3B, thinking off, companion to Table 12.

(a) Greedy decoding, T=0 (single deterministic run)

Domain	Dataset	No-spec	DFlash	Domino	DSpark	*DCOCO*
Math	MATH-500	366.2	1764.7 (4.82×)	1974.6 (5.39×)	1599.5 (4.37×)	**2165.5 (5.91×)**
	AIME-25	364.4	1640.5 (4.50×)	1922.1 (5.27×)	1566.0 (4.30×)	**2085.9 (5.72×)**
	GSM8K	369.6	1330.9 (3.60×)	1450.5 (3.93×)	1127.0 (3.05×)	**1638.4 (4.43×)**
Code	HumanEval	370.6	1226.4 (3.31×)	1348.7 (3.64×)	1075.4 (2.90×)	**1475.8 (3.98×)**
	MBPP	370.2	1137.6 (3.07×)	1267.5 (3.42×)	978.6 (2.64×)	**1435.3 (3.88×)**
	LiveCodeBench	369.2	1482.9 (4.02×)	1620.5 (4.39×)	1309.8 (3.55×)	**1754.1 (4.75×)**
General	MT-Bench	369.0	580.4 (1.57×)	593.4 (1.61×)	528.2 (1.43×)	**661.3 (1.79×)**
	Arena-Hard-v2	369.6	600.8 (1.63×)	610.5 (1.65×)	527.7 (1.43×)	**692.3 (1.87×)**
	Alpaca	368.1	559.0 (1.52×)	595.6 (1.62×)	513.0 (1.39×)	**658.7 (1.79×)**
	SpeedBench	369.1	645.6 (1.75×)	643.9 (1.74×)	575.5 (1.56×)	**759.0 (2.06×)**
Average		368.6	1096.9 (2.98×)	1202.7 (3.27×)	980.1 (2.66×)	**1332.6 (3.62×)**

(b) Sampling, T=1 (mean±std over seeds 42/43/44)

Domain	Dataset	No-spec	DFlash	Domino	DSpark	*DCOCO*
Math	MATH-500	365.2 ±0.0	1360.4 ±42.2 (3.73×)	1483.8 ±47.3 (4.06×)	1277.3 ±33.5 (3.50×)	**1644.7 ±21.0 (4.50×)**
	AIME-25	364.0 ±0.1	1017.5 ±78.9 (2.80×)	1149.8 ±18.1 (3.16×)	966.8 ±44.1 (2.66×)	**1194.1 ±142.7 (3.28×)**
	GSM8K	368.6 ±0.1	1270.6 ±14.1 (3.45×)	1375.3 ±25.2 (3.73×)	1080.4 ±10.8 (2.93×)	**1525.7 ±19.7 (4.14×)**
Code	HumanEval	370.1 ±0.0	1150.9 ±10.6 (3.11×)	1231.8 ±10.8 (3.33×)	1026.9 ±18.0 (2.77×)	**1372.1 ±14.1 (3.71×)**
	MBPP	369.4 ±0.1	1032.1 ±26.3 (2.79×)	1133.5 ±35.7 (3.07×)	928.2 ±14.9 (2.51×)	**1260.7 ±15.5 (3.41×)**
	LiveCodeBench	369.2 ±0.8	1184.6 ±22.8 (3.21×)	**1302.5 ±29.7 (3.53×)**	1065.5 ±32.3 (2.89×)	1301.1 ±39.4 (3.52×)
General	MT-Bench	369.0 ±0.1	559.4 ±3.7 (1.52×)	570.1 ±5.1 (1.55×)	512.1 ±7.9 (1.39×)	**639.1 ±17.1 (1.73×)**
	Arena-Hard-v2	369.9 ±0.4	568.5 ±12.0 (1.54×)	554.5 ±9.5 (1.50×)	490.8 ±8.7 (1.33×)	**631.4 ±25.0 (1.71×)**
	Alpaca	367.2 ±0.0	548.3 ±6.8 (1.49×)	572.1 ±7.0 (1.56×)	517.3 ±3.0 (1.41×)	**631.3 ±8.6 (1.72×)**
	SpeedBench	368.8 ±0.2	583.4 ±11.6 (1.58×)	583.4 ±6.5 (1.58×)	518.6 ±11.4 (1.41×)	**668.3 ±21.9 (1.81×)**
Average		368.1	927.6 (2.52×)	995.7 (2.71×)	838.4 (2.28×)	**1086.8 (2.95×)**

Table 14: Throughput (output tokens/s, per-stream) and speedup over no-spec decoding, Qwen3-4B, thinking off, concurrency 1, max 2048 generated tokens. Same format as Table 1.

Domain	Dataset	DFlash		Domino		DSpark		*DCOCO*	
		T=0	T=1	T=0	T=1	T=0	T=1	T=0	T=1
Math	MATH-500	6.443	5.104 ±0.039	7.719	5.799 ±0.119	6.125	4.910 ±0.068	**7.967**	**6.041 ±0.098**
	AIME-25	5.996	3.894 ±0.084	7.176	4.520 ±0.067	5.729	3.872 ±0.068	**7.368**	**4.623 ±0.178**
	GSM8K	4.872	4.604 ±0.036	5.825	5.422 ±0.084	4.474	4.230 ±0.021	**6.082**	**5.610 ±0.065**
Code	HumanEval	4.416	4.163 ±0.012	5.263	4.904 ±0.034	4.091	3.848 ±0.021	**5.351**	**4.949 ±0.046**
	MBPP	4.270	3.812 ±0.084	5.085	4.508 ±0.062	3.933	3.541 ±0.004	**5.297**	**4.603 ±0.007**
	LiveCodeBench	6.075	4.444 ±0.053	7.103	**5.178 ±0.044**	5.637	4.143 ±0.076	**7.127**	5.110 ±0.139
General	MT-Bench	2.564	2.351 ±0.017	2.851	2.592 ±0.010	2.415	2.250 ±0.023	**2.946**	**2.669 ±0.033**
	Arena-Hard-v2	2.408	2.173 ±0.036	2.596	2.278 ±0.005	2.200	1.980 ±0.011	**2.716**	**2.393 ±0.036**
	Alpaca	2.199	2.071 ±0.012	2.420	2.278 ±0.032	2.086	2.026 ±0.029	**2.515**	**2.357 ±0.018**
	SpeedBench	2.381	2.143 ±0.029	2.566	2.290 ±0.030	2.200	2.005 ±0.031	**2.753**	**2.425 ±0.052**
Average		4.162	3.476	4.861	3.977	3.889	3.281	**5.012**	**4.078**

Table 15: Mean accepted length τ, Qwen3-4B, thinking off, companion to Table 14. *DCOCO* leads τ in 19 of 20 cells and both averages.

(a) Greedy decoding, T=0 (single deterministic run)

Domain	Dataset	No-spec	DFlash	Domino	DSpark	*DCOCO*
Math	MATH-500	361.0	868.8 (2.41×)	921.7 (2.55×)	785.8 (2.18×)	**1000.4 (2.77×)**
	AIME-25	359.5	861.5 (2.40×)	909.9 (2.53×)	755.0 (2.10×)	**989.8 (2.75×)**
	GSM8K	362.0	846.8 (2.34×)	869.7 (2.40×)	735.9 (2.03×)	**974.0 (2.69×)**
Code	HumanEval	357.9	824.4 (2.30×)	875.9 (2.45×)	739.7 (2.07×)	**934.4 (2.61×)**
	MBPP	359.5	797.0 (2.22×)	836.3 (2.33×)	731.1 (2.03×)	**923.9 (2.57×)**
	LiveCodeBench	361.2	780.1 (2.16×)	842.8 (2.33×)	706.5 (1.96×)	**896.5 (2.48×)**
General	MT-Bench	363.4	741.7 (2.04×)	745.6 (2.05×)	651.0 (1.79×)	**839.0 (2.31×)**
	Arena-Hard-v2	362.7	682.9 (1.88×)	702.1 (1.94×)	574.1 (1.58×)	**796.0 (2.20×)**
	Alpaca	367.9	543.2 (1.48×)	551.2 (1.50×)	497.4 (1.35×)	**604.8 (1.64×)**
	SpeedBench	365.7	588.7 (1.61×)	596.9 (1.63×)	529.2 (1.45×)	**663.5 (1.81×)**
Average		362.1	753.5 (2.08×)	785.2 (2.17×)	670.6 (1.85×)	**862.2 (2.38×)**

(b) Sampling, T=1 (mean±std over seeds 42/43/44)

Domain	Dataset	No-spec	DFlash	Domino	DSpark	*DCOCO*
Math	MATH-500	360.6 ±0.0	812.2 ±1.5 (2.25×)	861.0 ±10.2 (2.39×)	731.0 ±4.3 (2.03×)	**915.0 ±9.1 (2.54×)**
	AIME-25	359.1 ±0.0	775.0 ±26.0 (2.16×)	799.8 ±10.0 (2.23×)	692.2 ±4.3 (1.93×)	**905.8 ±20.8 (2.52×)**
	GSM8K	360.5 ±0.2	756.2 ±3.0 (2.10×)	777.3 ±6.0 (2.16×)	669.0 ±2.3 (1.86×)	**859.6 ±5.0 (2.38×)**
Code	HumanEval	357.9 ±0.4	735.6 ±2.8 (2.06×)	775.1 ±4.7 (2.17×)	672.1 ±2.9 (1.88×)	**832.5 ±4.3 (2.33×)**
	MBPP	358.4 ±0.6	697.7 ±2.5 (1.95×)	739.1 ±2.0 (2.06×)	647.6 ±2.4 (1.81×)	**804.9 ±4.3 (2.25×)**
	LiveCodeBench	360.7 ±0.0	689.4 ±4.7 (1.91×)	726.2 ±6.0 (2.01×)	632.6 ±1.4 (1.75×)	**778.1 ±8.7 (2.16×)**
General	MT-Bench	361.2 ±0.5	622.4 ±9.1 (1.72×)	637.3 ±4.2 (1.76×)	566.7 ±8.4 (1.57×)	**697.9 ±14.5 (1.93×)**
	Arena-Hard-v2	362.5 ±0.1	584.5 ±6.4 (1.61×)	579.2 ±3.6 (1.60×)	519.6 ±4.8 (1.43×)	**657.2 ±6.8 (1.81×)**
	Alpaca	367.7 ±0.2	512.7 ±2.0 (1.39×)	527.6 ±3.1 (1.44×)	477.3 ±1.8 (1.30×)	**576.8 ±2.6 (1.57×)**
	SpeedBench	364.4 ±0.2	549.3 ±9.5 (1.51×)	544.7 ±3.1 (1.50×)	493.6 ±2.2 (1.36×)	**625.2 ±5.4 (1.72×)**
Average		361.3	673.5 (1.87×)	696.7 (1.93×)	610.2 (1.69×)	**765.3 (2.12×)**

Table 16: Throughput (output tokens/s, per-stream) and speedup over no-spec decoding, Qwen3-4B, thinking on, concurrency 1, max 4096 generated tokens. Same format as Table 1.

		DFlash		Domino		DSpark		*DCOCO*	
Domain	Dataset	T=0	T=1	T=0	T=1	T=0	T=1	T=0	T=1
Math	MATH-500	3.216	2.957 ±0.008	3.607	3.289 ±0.005	2.969	2.771 ±0.001	**3.644**	**3.317 ±0.012**
	AIME-25	3.140	2.845 ±0.032	3.520	3.134 ±0.026	2.900	2.664 ±0.017	**3.568**	**3.234 ±0.020**
	GSM8K	3.099	2.753 ±0.008	3.437	3.030 ±0.010	2.848	2.563 ±0.018	**3.538**	**3.088 ±0.010**
Code	HumanEval	3.100	2.689 ±0.010	**3.517**	**3.020 ±0.020**	2.858	2.524 ±0.012	3.451	2.968 ±0.002
	MBPP	2.961	2.564 ±0.004	**3.337**	**2.850 ±0.005**	2.749	2.425 ±0.012	3.327	2.847 ±0.000
	LiveCodeBench	2.880	2.553 ±0.019	**3.267**	**2.830 ±0.014**	2.674	2.403 ±0.012	3.246	2.816 ±0.012
General	MT-Bench	2.531	2.246 ±0.014	2.784	2.440 ±0.018	2.356	2.118 ±0.006	**2.843**	**2.471 ±0.004**
	Arena-Hard-v2	2.482	2.064 ±0.016	2.695	2.223 ±0.007	2.230	1.934 ±0.003	**2.807**	**2.279 ±0.008**
	Alpaca	2.148	1.921 ±0.007	2.333	2.075 ±0.013	2.013	1.841 ±0.006	**2.385**	**2.134 ±0.023**
	SpeedBench	2.225	2.010 ±0.026	2.393	2.132 ±0.013	2.053	1.886 ±0.007	**2.478**	**2.256 ±0.007**
Average		2.778	2.460	3.089	2.702	2.565	2.313	**3.129**	**2.741**

Table 17: Mean accepted length τ, Qwen3-4B, thinking on, companion to Table 16.

E COVERAGE BY VERIFIER

Table 18 summarises Figure 1 for all four verifiers, and the pattern is strikingly uniform: at the first drafted slot the top-16 pool contains the committed token about 95% of the time and the drafter's own first choice about 81%, while by the last slot the pool still holds it in 57–61% of cases but the first choice finds it only about one time in five. In other words, deep in the block roughly every second slot still has the right token waiting in the candidate pool, the drafter simply stops picking it. This is the headroom the corrector re-selects from.

	top-1	top-4	top-8	top-16
Qwen3-8B, mean over drafted slots	0.400	0.587	0.676	0.755
first drafted slot	0.814	0.915	0.936	0.951
last drafted slot	0.154	0.355	0.483	0.608
Qwen3-4B, mean over drafted slots	0.413	0.574	0.651	0.722
first drafted slot	0.815	0.907	0.928	0.943
last drafted slot	0.169	0.348	0.461	0.573
Qwen3-Coder-30B-A3B, mean over drafted slots	0.423	0.584	0.657	0.723
first drafted slot	0.811	0.900	0.920	0.936
last drafted slot	0.181	0.366	0.476	0.582
Llama-3.1-8B, mean over drafted slots	0.449	0.602	0.672	0.735
first drafted slot	0.830	0.915	0.933	0.946
last drafted slot	0.205	0.377	0.482	0.585

Table 18: Coverage by verifier: means over the 15 drafted slots of Figure 1, with the first and last slot shown per verifier. Top-1 is the drafter's own per-slot accuracy; *DCOCO* uses $K{=}16$.

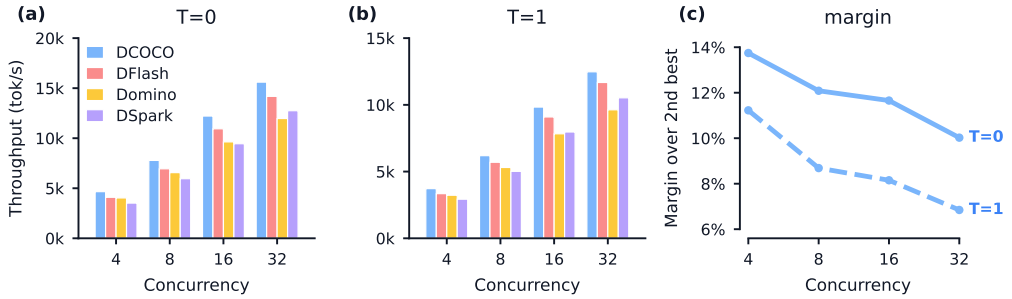


Figure 5: At high concurrency, *DCOCO* achieves the highest throughput, while Domino and DSpark fall below DFlash. Qwen3-4B throughput at (a) $T = 0$ and (b) $T = 1$, averaged over the ten suites. (c) *DCOCO*'s relative throughput gain over the strongest baseline at each concurrency. *DCOCO*'s margin is 10–14% at T=0 and 7–11% at T=1.

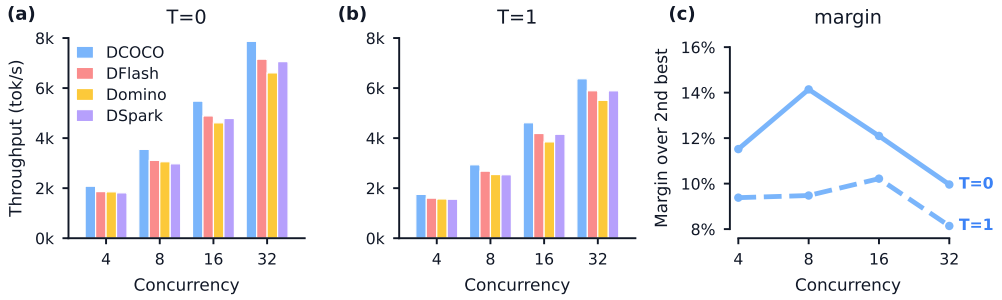


Figure 6: *DCOCO* achieves the highest throughput at every concurrency, while Domino and DSpark stay at or below DFlash. Qwen3-Coder-30B-A3B throughput at (a) $T = 0$ and (b) $T = 1$, averaged over the ten suites. (c) *DCOCO*'s relative throughput gain over the strongest baseline at each concurrency. *DCOCO*'s margin is 10–14% at T=0 and 8–10% at T=1.

F THROUGHPUT AT HIGHER CONCURRENCY

Batched serving is where correction overhead either pays or does not: as concurrency grows, the verifier's cycle time is amortised across streams, so whatever a corrector adds per cycle weighs ever more heavily against the acceptance it buys. This appendix extends the main-text view of Figures 3 and 4 to the remaining backbones. Figures 5–7 plot aggregate throughput with *DCOCO*'s margin over the strongest baseline at each load, and Tables 19–22 give the absolute numbers from concurrency 1: *DCOCO* is the fastest arm in every cell, all four backbones, both temperatures, by 6–23% over the strongest baseline, while Domino drops below even the uncorrected drafter as load grows and DSpark sits at or below it throughout. Figures 8–10 show why, plotting each corrector's accepted length and throughput relative to DFlash with its per-cycle overhead as the shaded gap between the two curves: Domino's acceptance gain persists at every load, but the widening overhead consumes it, the τ curve stays above parity while the throughput curve slides below, DSpark buys little or no acceptance to begin with, and *DCOCO*'s two curves move together above parity because its shading stays thin.

G STOCHASTIC VERIFICATION WITH DETERMINISTIC PROPOSALS

DCOCO selects draft tokens deterministically using the prefix-first rule, including when the verifier uses stochastic decoding. Conditional on the current context and the selected draft prefix, its proposal distribution is therefore a point mass:

$$p_j(v \mid x, \hat{y}_{<j}) = \mathbf{1}[v = \hat{y}_j]. \tag{10}$$

The probability one refers to proposal selection, not verifier acceptance. Let $q_j(v) = q(v \mid x, \hat{y}_{<j})$ denote the verifier distribution after applying the target sampling settings, including temperature.

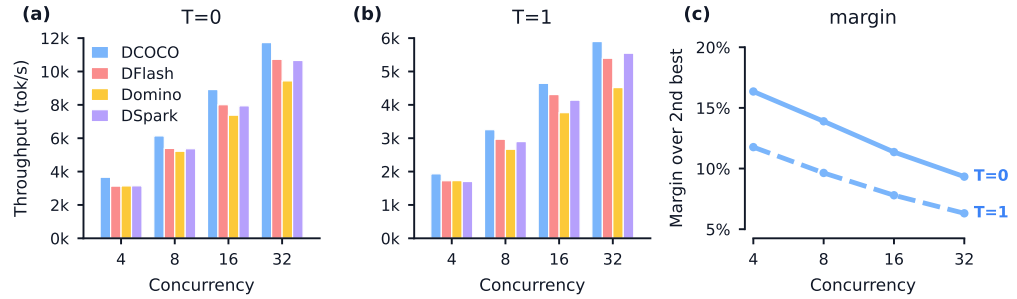


Figure 7: *DCOCO* achieves the highest throughput at every concurrency, while Domino falls increasingly behind DFlash as load grows. Llama-3.1-8B throughput at (a) $T = 0$ and (b) $T = 1$, averaged over the ten suites. (c) *DCOCO*'s relative throughput gain over the strongest baseline at each concurrency. *DCOCO*'s margin is 9–16% at T=0 and 6–12% at T=1.

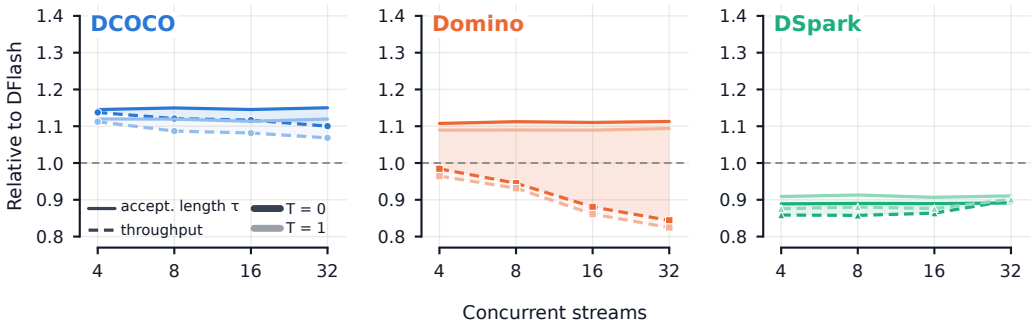


Figure 8: Domino's acceptance gain is consumed by per-cycle overhead as concurrency grows, and DSpark never matches DFlash's acceptance, while *DCOCO* remains faster than DFlash. Accepted length τ (solid) and throughput (dashed), relative to DFlash; Shading highlights the separation between the accepted-length and throughput ratios.. Qwen3-4B, thinking off, $T = 0$ (dark) and $T = 1$ (light), averaged over all ten benchmark suites. Absolute values: Tab. 20.

Standard speculative sampling accepts $\hat{y}_j$ with probability $\min\{1, q_j(\hat{y}_j)/p_j(\hat{y}_j)\} = q_j(\hat{y}_j)$. On rejection, its residual distribution simplifies to

$$r_j(v) = \frac{q_j(v)\,\mathbf{1}[v \neq \hat{y}_j]}{1 - q_j(\hat{y}_j)}. \tag{11}$$

When $q_j(\hat{y}_j) = 1$, rejection never occurs and the residual is not needed.

This rule preserves the verifier distribution: the proposed token is emitted with probability $q_j(\hat{y}_j)$, while any other token v is emitted with probability $(1-q_j(\hat{y}_j))r_j(v) = q_j(v)$. Verification continues through accepted draft tokens, commits the residual sample at the first rejection, and discards the remaining draft suffix. If all draft tokens are accepted, one additional token is sampled from the verifier. Applying the same argument at each committed position preserves the target autoregressive distribution.

Consequently, stochastic target decoding does not require stochastic draft selection: *DCOCO*'s transition scores need not be softmax-normalized, and no distribution over the retained candidates needs to be sampled or passed to the verifier. The verifier must use the point-mass proposal semantics above.

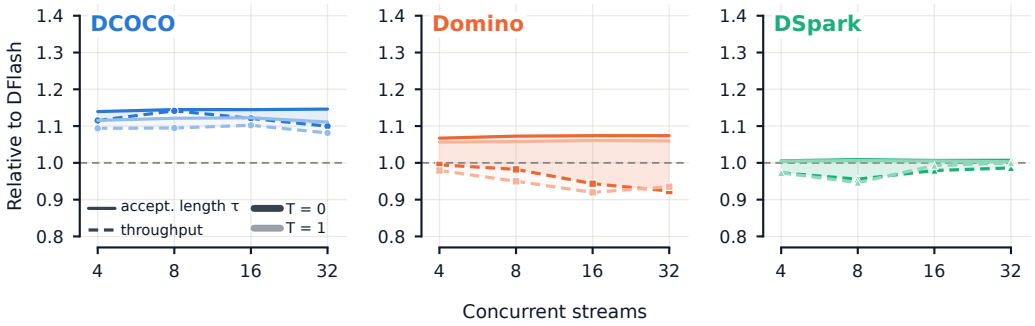


Figure 9: At high concurrency, overhead outweighs Domino's acceptance gain and DSpark gains no acceptance over DFlash, while *DCOCO* remains faster than DFlash. Accepted length τ (solid) and throughput (dashed), relative to DFlash; Shading highlights the separation between the accepted-length and throughput ratios. Qwen3-Coder-30B-A3B, thinking off, $T = 0$ (dark) and $T = 1$ (light), averaged over all ten benchmark suites. Absolute values: Tab. 21.

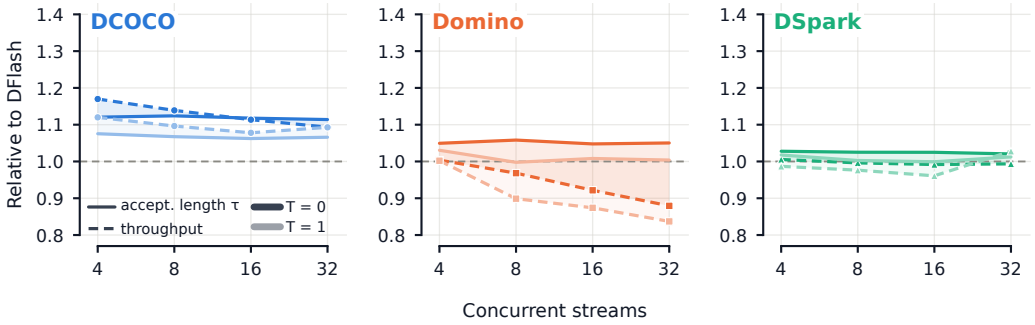


Figure 10: As concurrency grows, Domino's acceptance gain is consumed by per-cycle overhead, while *DCOCO* remains faster than DFlash. Accepted length τ (solid) and throughput (dashed), relative to DFlash; Shading highlights the separation between the accepted-length and throughput ratios. Llama-3.1-8B, thinking off, $T = 0$ (dark) and $T = 1$ (light), averaged over all ten benchmark suites. Absolute values: Tab. 22.

Concurrency	DFlash	Domino	DSpark	*DCOCO*	*DCOCO*/ DFlash	*DCOCO* vs. best
Greedy decoding, T=0						
1	783.1	839.3	765.6	**959.7**	1.23×	+14.3%
4	2806.4	3008.6	2719.0	**3507.3**	1.25×	+16.6%
8	4889.1	5051.0	4614.4	**6082.6**	1.24×	+20.4%
16	7229.1	7098.4	7092.9	**8894.3**	1.23×	+23.0%
32	9655.0	9013.9	9490.3	**11512.4**	1.19×	+19.2%
Sampling, T=1 (mean over three seeds)						
1	649.8	670.9	628.7	**762.2**	1.17×	+13.6%
4	2300.3	2334.1	2198.9	**2710.0**	1.18×	+16.1%
8	4006.3	3894.9	3758.1	**4735.3**	1.18×	+18.2%
16	5964.5	5441.1	5730.3	**6871.9**	1.15×	+15.2%
32	7857.9	6821.8	7443.4	**8836.1**	1.12×	+12.4%

Table 19: Aggregate throughput (output tokens/s) as concurrency grows, Qwen3-8B, thinking off, averaged over the ten suites per concurrency workload.

Concurrency	DFlash	Domino	DSpark	*DCOCO*	*DCOCO*/ DFlash	*DCOCO* vs. best
Greedy decoding, $T{=}0$						
1	1096.9	1202.7	980.1	**1332.6**	1.21×	+10.8%
4	4097.1	4032.1	3517.5	**4660.3**	1.14×	+13.7%
8	6942.6	6560.9	5953.9	**7781.7**	1.12×	+12.1%
16	10940.3	9633.9	9448.6	**12215.5**	1.12×	+11.7%
32	14181.1	11966.5	12746.7	**15602.7**	1.10×	+10.0%
Sampling, $T{=}1$ *(mean over three seeds)*						
1	927.6	995.7	838.4	**1086.8**	1.17×	+9.1%
4	3345.6	3226.4	2929.6	**3721.2**	1.11×	+11.2%
8	5697.0	5306.7	5015.9	**6191.9**	1.09×	+8.7%
16	9092.8	7830.4	7965.6	**9834.0**	1.08×	+8.2%
32	11676.9	9628.3	10525.8	**12476.2**	1.07×	+6.8%

Table 20: Aggregate throughput (output tokens/s) as concurrency grows, Qwen3-4B, thinking off, averaged over the ten suites. Bold: best method per row.

Concurrency	DFlash	Domino	DSpark	*DCOCO*	*DCOCO*/ DFlash	*DCOCO* vs. best
Greedy decoding, $T{=}0$						
1	642.4	660.1	627.2	**785.1**	1.22×	+18.9%
4	1862.6	1852.5	1812.2	**2077.2**	1.12×	+11.5%
8	3110.6	3055.5	2972.4	**3550.5**	1.14×	+14.1%
16	4886.7	4610.1	4784.7	**5477.8**	1.12×	+12.1%
32	7155.2	6606.8	7057.2	**7867.9**	1.10×	+10.0%
Sampling, $T{=}1$ *(mean over three seeds)*						
1	556.9	557.8	540.0	**662.3**	1.19×	+18.7%
4	1601.0	1567.8	1557.3	**1751.2**	1.09×	+9.4%
8	2679.3	2544.9	2535.8	**2933.2**	1.09×	+9.5%
16	4185.4	3849.4	4154.3	**4613.0**	1.10×	+10.2%
32	5892.4	5511.0	5890.4	**6371.9**	1.08×	+8.1%

Table 21: Aggregate throughput (output tokens/s) as concurrency grows, Qwen3-Coder-30B-A3B, thinking off, averaged over the ten suites. Bold: best method per row.

Concurrency	DFlash	Domino	DSpark	*DCOCO*	*DCOCO*/ DFlash	*DCOCO* vs. best
Greedy decoding, $T{=}0$						
1	844.2	900.8	838.0	**1042.4**	1.23×	+15.7%
4	3128.0	3139.2	3145.0	**3659.2**	1.17×	+16.3%
8	5388.7	5217.4	5367.7	**6137.4**	1.14×	+13.9%
16	8007.2	7379.9	7940.5	**8916.8**	1.11×	+11.4%
32	10730.3	9435.8	10660.3	**11731.7**	1.09×	+9.3%
Sampling, $T{=}1$ *(mean over three seeds)*						
1	487.2	497.3	459.6	**536.3**	1.10×	+7.8%
4	1724.9	1728.4	1702.1	**1931.8**	1.12×	+11.8%
8	2969.0	2668.5	2898.7	**3255.2**	1.10×	+9.6%
16	4308.5	3765.6	4140.3	**4644.4**	1.08×	+7.8%
32	5396.9	4517.5	5547.6	**5897.7**	1.09×	+6.3%

Table 22: Aggregate throughput (output tokens/s) as concurrency grows, Llama-3.1-8B, thinking off, averaged over the ten suites. Bold: best method per row.