SODA: SUBSPACE ONLINE DRAFTER ADAPTATION FOR SPECULATIVE DECODING

Anonymous authors
Paper under double-blind review

ABSTRACT

Speculative decoding accelerates large language model inference by verifying tokens proposed by a lightweight drafter, but a drafter trained offline on a fixed corpus may not match the local request stream encountered during serving, thereby reducing the number of accepted draft tokens. Online drafter adaptation can specialize the drafter to this stream, yet optimizing the full drafter introduces computation and communication overhead that can offset the resulting decoding gains or require substantial additional resources. We introduce SODA, which restricts stream-local drafter correction to an offline-learned low-dimensional subspace, thereby substantially reducing the state that must be optimized online. Across four mathematical reasoning and code generation benchmarks with Qwen3-8B and Gemma-4-12B-IT targets, SODA improves mean accepted tokens (MAT) by 9.9%–38.8% and reduces end-to-end latency by 6.5%–21.6% relative to the corresponding fixed drafters. On Qwen3-8B, SODA updates only 65.5K parameters online, reaching 97.1%–99.7% of the absolute MAT of full-drafter fine-tuning, which updates up to 1.05B parameters. These results show that compact stateful adaptation is able to improve the serving efficiency while requiring only a small online-updated state.

1 INTRODUCTION

Large language models (LLMs) are widely used for tasks that involve long response sequences, including mathematical reasoning, code generation, and agentic workflows. However, the token-by-token dependency of autoregressive decoding leads to substantial generation latency. Speculative decoding mitigates this bottleneck by using a lightweight draft model (drafter) to propose candidate tokens, which the target model then verifies in parallel. This approach reduces the number of serial decoding steps without changing the target model's output distribution (Leviathan et al., 2023; Chen et al., 2023). Its practical speedup, however, depends on a delicate balance: the drafter must be lightweight enough to keep drafting overhead low, yet sufficiently aligned with the target distribution to produce long accepted sequences.

Maintaining this alignment throughout continuous serving is challenging for two reasons: the inherent trade-off between drafter capacity and efficiency, and shifts in the deployment distribution. First, a drafter is typically much smaller than its target model. Scaling it up may improve prediction quality, but also increases drafting latency and can erode the end-to-end benefits of speculative decoding (Bhendawade et al., 2025; Qian et al., 2026b). A general-purpose drafter constrained by strict latency requirements is therefore unlikely to approximate the target distribution equally well across all domains and request patterns, leaving substantial room for specialization to the current request stream.

Distribution shifts during deployment further exacerbate this mismatch. Offline training data often differ from actual user requests, and adapting a drafter using historical requests or domain-specific data can improve its prediction accuracy (Liu et al., 2024; Hong et al., 2025). More broadly, a self-evolving system may update the model itself or modify its harness, including prompts, tools, memory, and skills (Hebbar et al., 2026; Chen et al., 2026b). ReSpec (Chen et al., 2026c) shows that continual actor updates progressively misalign a parameterized drafter with the evolving actor distribution. Similarly, DAS (Shao et al., 2026) observes that older rollouts become less predictive as the policy changes and therefore continually refreshes its nonparametric drafter with recent trajectories. Changes in model weights, deployment domains, or the surrounding harness can thus shift the conditional

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.

distribution that the drafter must match. Motivated by this observation, we study continuous request streams under a fixed target model and use data generated naturally during serving to specialize the drafter online.

Crucially, speculative verification already provides the supervision needed for such adaptation. When verifying candidate tokens, the target model produces predictions for the corresponding draft positions, exposing discrepancies between the drafter and target distributions without requiring additional target-model forward passes (Qian et al., 2026b; Kumar et al., 2026). Online Speculative Decoding (Liu et al., 2024), OnlineSpec (Qian et al., 2026b), and Test-Time Speculation (Kumar et al., 2026) collectively show that requests and verification signals observed during serving can be used to continually adapt the drafter, improving acceptance on the current request distribution, domain-specific workloads, and long generations.

The central challenge is therefore not how to obtain supervision, but how to make online adaptation sufficiently lightweight. Updating the entire drafter requires expensive backpropagation during serving, persistent optimizer states, and parameter synchronization. Even when training is periodic or asynchronous, its computation and communication costs may offset the gains from higher acceptance (Kumar et al., 2026; Wang et al., 2026). Moreover, high-performance drafters are evolving from simple prediction heads toward multilayer parallel-generation architectures like DFlash (Chen et al., 2026a), Domino (Huang et al., 2026), and DSpark (Cheng et al., 2026) using block diffusion. Their larger parameter footprints make continual full-backbone updates costly, motivating a compact representation of the stream-dependent adaptation state.

To address this challenge, we propose SODA, a low-overhead method for online drafter adaptation over continuous request streams. Building on low-rank adaptation (Hu et al., 2022), SODA separates an offline-learned shared correction subspace from a compact stream-local state that is updated online. We assume that useful adaptations across request streams share a set of low-rank correction directions, while each stream need only adjust its coefficients within this subspace. Specifically, under the column-vector convention, SODA parameterizes the weight correction as $\Delta W_t = A_0 B_t$, where A_0 is a shared basis learned offline and then frozen, and B_t is the stream-local state updated online. Thus, only B_t requires online gradients, optimizer state, and parameter transfer. An independent update worker optimizes B_t asynchronously, and the inference process loads the latest state at update boundaries, keeping adaptation off the critical serving path.

We evaluate SODA on four mathematical reasoning and code generation benchmarks with Qwen3-8B and Gemma-4-12B-IT targets, each paired with its own drafter and offline basis. Using the same online adaptation hyperparameters for both configurations, SODA improves mean accepted tokens (MAT) by 9.9%–38.8% and reduces end-to-end latency by 6.5%–21.6% relative to the corresponding fixed drafters, while updating only 61.4K–65.5K parameters online. On Qwen3-8B, its 65.5K online parameters reach 97.1%–99.7% of the absolute MAT of full-drafter fine-tuning, which updates up to 1.05B parameters. In uninterrupted Qwen3-8B cross-domain stress tests with shuffled requests and no boundary signals, SODA further achieves overall MAT improvements of 12.5%–13.1% under opposite domain orders. These results show that low-dimensional stream-local updates can effectively balance adaptation quality and serving overhead across the tested target configurations and request streams.

The main contributions of this paper are as follows:

- We propose SODA, which adapts drafters through low-dimensional stream-local updates within an offline-learned correction subspace. This design substantially reduces the number of online-updated parameters while preserving the ability to specialize to individual request streams.
- We design a lightweight asynchronous training–serving pipeline that updates and transfers only the stream-local state, measure its adaptation overhead, and estimate update-worker capacity under a serial sharing model.
- We validate SODA on four mathematical reasoning and code generation benchmarks under two target configurations, demonstrating acceptance and end-to-end latency improvements with identical online adaptation hyperparameters. On Qwen3-8B, compact state updates approach the acceptance performance of full-drafter fine-tuning with more than four orders of magnitude fewer online-updated parameters.

2 Related Work

2.1 Speculative Decoding and Drafter Design

Speculative decoding uses a lightweight draft model to generate candidate sequences verified in parallel by the target model (Leviathan et al., 2023; Chen et al., 2023). To improve the draft-token acceptance rate, Medusa adds multiple prediction heads to the target model and uses tree-structured attention (Cai et al., 2024); the EAGLE series successively introduces target-model feature prediction, context-aware dynamic draft trees, and direct token prediction with multi-layer feature fusion (Li et al., 2024a;b; 2025). These methods improve acceptance length by refining the prediction structure or expanding candidate coverage, but may also increase draft-construction and verification overhead.

Recent work further adopts parallel block generation to reduce the serial cost of autoregressive drafting. DFlash uses a block diffusion drafter to generate candidate blocks in parallel, conditioned on target-model hidden states (Chen et al., 2026a); Domino and DSpark introduce lightweight causal correction after a parallel backbone to strengthen intra-block dependencies and improve the acceptance of candidate suffixes (Huang et al., 2026; Cheng et al., 2026). These multi-layer parallel drafters improve generation capability through offline training, but typically remain fixed after deployment and cannot continuously adapt to changing local distributions in request streams.

2.2 Test-Time Training and Online Drafter Adaptation

Test-time training (TTT) updates model parameters or internal states using data available during inference (Sun et al., 2020); in large language models, related methods primarily improve the context utilization and the task performance of the model (Hardt & Sun, 2024; Sun et al., 2025; Feng et al., 2026). For draft models, Online Speculative Decoding and OnlineSpec continually distills the drafter using real requests to adapt to deployment-time request distributions and Test-Time Speculation updates the drafter during long-sequence generation (Liu et al., 2024; Qian et al., 2026b; Kumar et al., 2026). AdaFlash applies on-policy distillation and adaptive draft length to diffusion drafters, while systems such as Aurora use asynchronous training to reduce the impact of updates on the inference critical path (Qian et al., 2026a; Wang et al., 2026). However, existing methods typically still need to update the full drafter, incurring considerable online adaptation overhead. Therefore, a more direct related question is how to preserve the existing drafter's generation capability while compressing the alignment state into a sufficiently small parameter space.

3 Method

3.1 Problem Setup

We formulate online adaptation for a general speculative drafter. Let p_θ be a frozen autoregressive target model and q_{ϕ_t} a drafter with online state ϕ_t. Given a prompt x and the tokens $y_{<s_t}$ committed before speculation round t, the current context is $c_t = (x, y_{<s_t})$. The drafter proposes K candidate tokens $\hat{\mathbf{y}}_t = (\hat{y}_{t,1}, \ldots, \hat{y}_{t,K})$, which the target verifies in parallel to produce

$$p_{t,j}(\cdot) = p_\theta(\cdot \mid c_t, \hat{y}_{t,<j}), \qquad j = 1, \ldots, K. \tag{1}$$

We call $v_t = (c_t, \hat{\mathbf{y}}_t, \{p_{t,j}\}_{j=1}^K)$ a *verified block*. It provides both the accepted output and the target-supervised signal used for subsequent drafter adaptation.

We instantiate this setting on DFlash, a parameter-rich block-diffusion draft model that predicts candidate blocks in parallel from target-model features (Chen et al., 2026a). It is straightforward to extend SODA to other drafter methods.

Our objective is to improve target–drafter alignment without heavy online optimization or communication that may block the critical inference path. Because adaptation changes only the drafter and leaves target verification unchanged, the output distribution of the target model is preserved.

3.2 Subspace Online Drafter Adaptation

SODA reduces the online optimization target by separating drafter correction into a reusable component learned offline and a stream-specific component updated during serving. Let $\mathcal{S}$ denote the

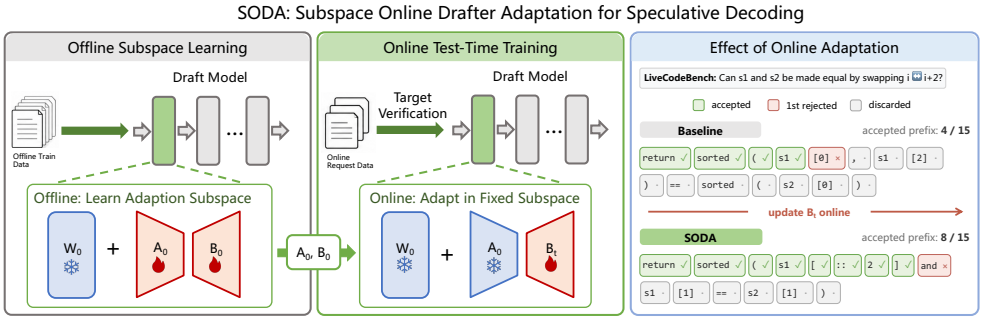

Figure 1: Overview of SODA. Offline training learns a reusable LoRA basis A_0 and initial coefficients B_0. During serving, verified blocks enter a replay buffer, and an asynchronous worker updates only the stream-local coefficients B_t.

selected fully connected layers. We use column-vector activations throughout: for each $\ell \in \mathcal{S}$, $h_{\text{in}}^{(\ell)} \in \mathbb{R}^{d_i^{(\ell)}}$ is mapped to $h_{\text{out}}^{(\ell)} \in \mathbb{R}^{d_o^{(\ell)}}$ by

$$\begin{aligned} h_{\text{out}}^{(\ell)} &= W_t^{(\ell)} h_{\text{in}}^{(\ell)}, \\ W_t^{(\ell)} &= W^{(\ell)} + \Delta W_t^{(\ell)}, \quad \Delta W_t^{(\ell)} = A_0^{(\ell)} B_t^{(\ell)}, \end{aligned} \tag{2}$$

where $W^{(\ell)} \in \mathbb{R}^{d_o^{(\ell)} \times d_i^{(\ell)}}$, $A_0^{(\ell)} \in \mathbb{R}^{d_o^{(\ell)} \times r}$ and $B_0^{(\ell)} \in \mathbb{R}^{r \times d_i^{(\ell)}}$ are learned through offline low-rank adaptation and then frozen, and $B_t^{(\ell)}$ is initialized from $B_0^{(\ell)}$ and updated independently for each request stream. The column space of $A_0^{(\ell)}$ defines the output-space correction directions available during serving, while $B_t^{(\ell)}$ specifies their coefficients for each input dimension. We refer to $B_t = \{B_t^{(\ell)}\}_{\ell \in \mathcal{S}}$ as the online state that evolves with verification feedback. The complete procedure therefore consists of two stages: jointly learning (A_0, B_0) offline, followed by B_t-only adaptation online.

Figure 1 summarizes the data path. The verified block defined in Section 3.1 supplies both committed tokens and teacher tokens for later adaptation. Only B_t changes online; the basis A_0, pretrained DFlash weights, and target model remain frozen.

Verification loss. The offline and online stages use the same target-supervised objective but optimize different variables. For each position in a verified block v_t, let $y_{t,j}^T = \arg\max_y p_{t,j}(y)$ under greedy decoding; under stochastic decoding, $y_{t,j}^T$ is sampled from $p_{t,j}$ using the serving sampling configuration. Because the target logits are already produced during verification, obtaining these teacher tokens requires no extra target-model forward pass. We minimize the position-weighted hard-distillation loss

$$\begin{aligned} \mathcal{L}(v_t; A, B) &= -\frac{1}{Z} \sum_{j=1}^{K} \omega_j \ell_{t,j}, \\ \ell_{t,j} &= \log q_{\phi,A,B}(y_{t,j}^T \mid c_t, j), \\ Z &= \sum_{j=1}^{K} \omega_j, \end{aligned} \tag{3}$$

where $q_{\phi,A,B}(\cdot \mid c_t, j)$ denotes the DFlash prediction associated with position j of the candidate block under the low-rank residuals AB. We use the same position decay as DFlash, $\omega_j = \exp(-(j-1)/\gamma)$, because an early error invalidates the remaining draft suffix. For the default 16-position block, $K = 15$ and $\gamma = 7$. Hard labels allow the replay buffer to retain token IDs instead of vocabulary-sized target distributions. The loss uses every position evaluated by the target: positions through the first rejection directly supervise the realized continuation, while later positions provide target feedback on drafter-induced candidate contexts that have already been evaluated during verification.

Offline low-rank adaptation. Before deployment, we learn the correction subspace by standard low-rank adaptation on an offline set $\mathcal{D}_{\text{off}}$. This set is a held-out split of the target-generated response mixture used to train the corresponding DFlash checkpoint and contains no evaluation prompts. We form verified blocks with the same drafting and target-verification procedure used during serving, and jointly optimize both low-rank factors:

$$(A_0, B_0) = \arg\min_{A,B} \frac{1}{|\mathcal{D}_{\text{off}}|} \sum_{v \in \mathcal{D}_{\text{off}}} \mathcal{L}(v; A, B). \tag{4}$$

Here A and B collect the factors from all selected layers; the pretrained drafter weights and the target model remain frozen. Equation (4) is an ordinary LoRA-style training objective (Hu et al., 2022): joint optimization allows A_0 to learn reusable correction directions and B_0 to provide the initial coefficient state. After training, we freeze A_0 and initialize each online request stream with a separate copy of B_0. This design assumes that the offline-learned column spaces contain directions useful for stream-local correction; online adaptation may change the correction coefficients but cannot introduce directions outside those spaces.

Online state update. At the beginning of a request stream, we copy B_0 and reset the optimizer states. The resulting B_t persists across requests in that stream, whereas an explicit stream boundary restores the initial state B_0 and prevents corrections from leaking between unrelated workloads. We retain verified blocks in a bounded FIFO buffer $\mathcal{R}_t$ until the next online update. The buffer stores drafter inputs, indices into the existing target-feature cache, and teacher token IDs rather than full target logits.

After every U verification rounds, we replay the buffer once and update only B_t:

$$\begin{aligned} B_{t+1} &= \mathrm{AdamW}(B_t, g_t; \eta), \\ g_t &= \nabla_B \frac{1}{|\mathcal{R}_t|} \sum_{v \in \mathcal{R}_t} \mathcal{L}(v; A_0, B_t). \end{aligned} \tag{5}$$

On rounds without an update, $B_{t+1} = B_t$. No gradient or optimizer state is constructed for A_0, W, or any target parameter. For each selected layer,

$$\nabla_{B^{(\ell)}} \mathcal{L} = \left(A_0^{(\ell)}\right)^\top \nabla_{W_t^{(\ell)}} \mathcal{L}, \tag{6}$$

and any induced weight change has the form $A_0^{(\ell)} \Delta B^{(\ell)}$. Thus, irrespective of the optimizer preconditioning, every column of the online weight update remains within the column space learned offline; it should not be interpreted as an exact Euclidean projection of the dense AdamW step. The update occurs only after the current output has been verified and committed and therefore affects future draft proposals, never the correctness of the current round. We report the concrete adapter placement, optimization hyperparameters, and serving configuration used in our experiments in Section 4.1. Appendix A summarizes the complete offline and online procedure.

4 Experiments

4.1 Experimental Setup

We evaluate SODA with Qwen3-8B and Gemma-4-12B-IT, each paired with its own drafter and separately trained offline basis; no adapter is transferred between targets. The Qwen3-8B drafter (Chen et al., 2026a) is trained with SpecForge (Li et al., 2026) on PerfectBlend (Xu et al., 2024) and OpenCodeInstruct (Ahmad et al., 2025). The Gemma-4-12B-IT configuration (Gemma Team, 2026) uses an 11.96B-parameter target and a 727.5M-parameter drafter. We compare against a fixed-drafter baseline with identical serving settings but no online updates. Unless stated otherwise, ablations and stress tests use Qwen3-8B.

We use GSM8K (Cobbe et al., 2021), Math500 (Hendrycks et al., 2021; Lightman et al., 2024), MBPP (Austin et al., 2021), and LiveCodeBench (Jain et al., 2025) as sequential request streams, following Liu et al. (2024). For the main results, all variants process each benchmark in the same fixed order, and we reset the online state between benchmarks to isolate within-domain specialization.

Table 1: Acceptance and latency under two target–drafter configurations. Each entry reports fixed drafter → SODA (relative change, computed before rounding); online adaptation hyperparameters are shared.

Dataset	MAT ↑	Generation (s) ↓	E2E (s) ↓
Qwen3-8B			
GSM8K	5.31 → 6.22 (+17.3%)	629.6 → 520.5 (-17.3%)	638.9 → 538.7 (-15.7%)
Math500	5.22 → 5.99 (+14.7%)	566.3 → 502.5 (-11.3%)	579.0 → 532.0 (-8.1%)
MBPP	4.06 → 4.49 (+10.5%)	165.1 → 153.4 (-7.1%)	180.8 → 169.2 (-6.5%)
LiveCodeBench	3.53 → 3.88 (+9.9%)	1808.6 → 1612.9 (-10.8%)	1822.2 → 1636.1 (-10.2%)
Gemma-4-12B-IT			
GSM8K	3.59 → 4.99 (+38.8%)	1434.0 → 1102.1 (-23.1%)	1447.2 → 1137.0 (-21.4%)
Math500	3.54 → 4.90 (+38.3%)	1089.5 → 832.6 (-23.6%)	1102.5 → 864.9 (-21.6%)
MBPP	3.09 → 4.10 (+32.7%)	976.5 → 792.1 (-18.9%)	992.0 → 825.0 (-16.8%)
LiveCodeBench	2.87 → 3.76 (+30.8%)	3252.6 → 2632.2 (-19.1%)	3267.4 → 2668.7 (-18.3%)

Earlier verification feedback can affect only later speculation rounds and requests; adaptation uses neither reference answers nor future prompts. Section 4.3 instead retains all online state across unannounced benchmark boundaries. We report mean accepted tokens (MAT), the average number of consecutive draft tokens accepted per verification round, together with generation and end-to-end (E2E) latency.

Both configurations use the same hardware, decoding protocol, and online hyperparameters. We apply rank-16 LoRA to the target-feature projection and update only B_t: 65.5K parameters for Qwen3-8B and 61.4K for Gemma-4-12B-IT. An asynchronous worker updates after every 64 verified blocks and returns the latest B_t through a single-slot latest-only queue. Appendix B provides the complete configuration.

4.2 Main Results

We conduct end-to-end experiments on four public benchmarks under both target configurations to compare each fixed-drafter baseline with SODA. Within each configuration, both methods process the same prompt streams under the same decoding setup. Table 1 summarizes acceptance and latency for both configurations, while Figure 2 shows how acceptance evolves throughout each Qwen3-8B stream.

SODA improves acceptance and reduces E2E latency in all eight benchmark–target settings. On Qwen3-8B, MAT increases by 9.9%–17.3% and E2E latency decreases by 6.5%–15.7%; Figure 2 shows that the running advantage persists throughout all four streams. On Gemma-4-12B-IT, MAT increases by 30.8%–38.8% and E2E latency decreases by 16.8%–21.6%. Thus, the acceptance gains translate into end-to-end improvements under both target–drafter configurations without retuning the online hyperparameters.

4.3 Robustness to Request Ordering and Cross-Domain Transitions

Because SODA carries adaptation states across requests, its performance may depend on request ordering, and corrections accumulated on one domain may interfere with subsequent domains. Using the Qwen3-8B configuration, we therefore concatenate all four benchmarks into a single continuous stream using two domain orders: MBPP → Math500 → GSM8K → LiveCodeBench and its reverse. For each domain order, we evaluate both the canonical within-benchmark request order and a shuffled request order. We treat each concatenation as one uninterrupted stream: B_t, the AdamW states, and the replay buffer are retained across benchmark boundaries, the server receives no boundary indicators, and all adaptation hyperparameters remain unchanged.

Under the canonical request orders, per-benchmark MAT differs from the independently reset runs in Table 1 by at most 0.20, with no cumulative deterioration in either tested domain order; Appendix C reports the individual values.

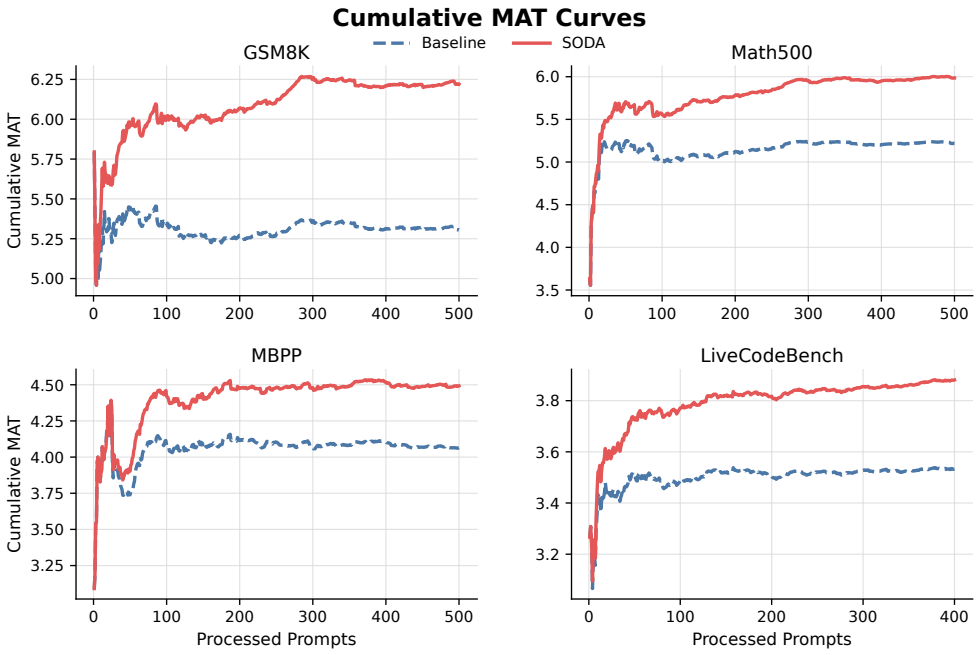

Figure 2: Cumulative MAT over the four fixed-order Qwen3-8B request streams. SODA updates after every 64 verified blocks.

Table 2: Robustness of SODA to request ordering and cross-domain transitions on Qwen3-8B. Forward concatenates MBPP → Math500 → GSM8K → LiveCodeBench; reverse uses the opposite order. Requests are shuffled within each benchmark, while all online states persist across hidden boundaries. Gains are relative to the fixed drafter and computed before rounding.

Dataset	Baseline MAT	Shuffled forward MAT	Gain	Shuffled reverse MAT	Gain
MBPP	4.0640	4.3954	+8.15%	4.5242	+11.32%
Math500	5.2202	6.0316	+15.54%	6.2005	+18.78%
GSM8K	5.3052	6.3400	+19.51%	6.2851	+18.47%
LiveCodeBench	3.5324	3.8736	+9.66%	3.8769	+9.75%
Overall	4.2041	4.7282	+12.47%	4.7527	+13.05%

Shuffling requests within each benchmark leads to the same conclusion. As shown in Table 2, SODA improves MAT on every benchmark under both domain orders, with per-benchmark gains ranging from 8.15% to 19.51%. When MAT is pooled over all verification rounds in each concatenated stream, it improves by 12.47% in the forward stream and 13.05% in the reverse stream; the two gains differ by only 0.58 percentage points. The gains remain positive under both tested domain orders and their shuffled request realizations, with no material negative transfer across these boundaries.

4.4 Ablation Study

4.4.1 Ablation on Online Adaptation Subspace

This ablation examines two design choices in SODA: whether the correction basis must be learned offline and whether it must continue to be optimized during online serving. Table 3 compares the fixed drafter with four online adaptation variants. Full-drafter fine-tuning updates every drafter parameter as a high-capacity reference; the random-basis control replaces A_0 with a frozen random matrix while preserving the B-only update; and Online LoRA jointly updates both FC-layer LoRA factors.

Jointly updating both LoRA factors provides only modest additional acceptance gains over SODA. Online LoRA improves MAT by 0.11, 0.18, 0.02, and 0.00 over SODA on GSM8K, Math500,

Table 3: Effect of the online adaptation subspace on Qwen3-8B. All variants use the same verification feedback; entries report MAT (↑) and the number of online-updated parameters.

Variant	Online parameters	GSM8K	Math500	MBPP	LiveCodeBench
Fixed drafter	0	5.31	5.22	4.06	3.53
SODA (FC B-only, learned basis)	65.5K	6.22	5.99	4.49	3.88
SODA (FC B-only, random basis)	65.5K	5.50	5.43	4.16	3.64
Online LoRA (FC, A+B)	328K	6.33	6.17	4.51	3.88
Full-drafter fine-tuning	up to 1.05B	6.24	6.16	4.52	3.92

Table 4: Effect of synchronous and asynchronous execution on Qwen3-8B. Each cell reports MAT / E2E latency (s); no-apply overhead is relative to the fixed drafter.

Dataset	Fixed drafter	SODA (async)	SODA (sync)	SODA (async, no apply)	No-apply overhead
GSM8K	5.31 / 638.9	6.22 / 538.7	6.22 / 762.3	5.31 / 651.7	+2.0%
Math500	5.22 / 579.0	5.99 / 532.0	6.10 / 700.2	5.22 / 611.7	+5.6%
MBPP	4.06 / 180.8	4.49 / 169.2	4.61 / 217.7	4.06 / 185.3	+2.5%
LiveCodeBench	3.53 / 1822.2	3.88 / 1636.1	3.87 / 2802.3	3.53 / 1890.9	+3.8%

MBPP, and LiveCodeBench, respectively, despite updating approximately five times as many online parameters. SODA also reaches 97.1%–99.7% of the absolute MAT achieved by full-drafter fine-tuning while updating only 65.5K parameters rather than up to 1.05B. These results indicate that, on the evaluated streams, the offline-learned basis already captures most correction directions needed during serving, while updating the compact coefficient state recovers most of the benefit available from adapting a substantially larger parameter space.

The random-basis control, however, lowers MAT by 6.1%–11.6% relative to SODA and improves over the fixed drafter by only 2.3%–4.1%, compared with the 9.9%–17.3% gains obtained with the learned basis. This result shows that restricting online adaptation is effective only when offline training identifies useful correction directions. Together, the two controls isolate both sides of SODA's design: the random-basis comparison shows why A_0 must be learned, whereas the Online LoRA comparison shows that A_0 need not be repeatedly optimized online to recover most of the attainable MAT. SODA therefore provides a favorable trade-off between stream adaptation quality and the amount of state requiring online gradients, optimizer state, and parameter transfer.

4.4.2 Ablation on Asynchronous Implementation

We next isolate the adaptation cost removed from the critical path and the communication and weight-loading overhead introduced by asynchronous execution. Table 4 compares asynchronous SODA with a synchronous variant and a no-apply variant that performs asynchronous updates but does not load the returned states.

A synchronous variant achieves comparable MAT but increases E2E latency by 19.3%–53.8% relative to the fixed drafter. The no-apply variant matches the fixed-drafter MAT while adding 2.0%–5.6% E2E latency, isolating the visible overhead of asynchronous training and parameter transfer. Applying the returned states more than offsets this overhead, yielding net E2E reductions of 6.5%–15.7%.

Update-worker profiling gives occupancy ratios of 4.8%–6.3% on GSM8K and LiveCodeBench. Under a serial worker-sharing model, these measurements project to 14.3–18.8 inference processes per update accelerator; Appendix D provides the assumptions and calculation.

5 Conclusion

This paper presents SODA, which learns a reusable low-rank correction basis offline and asynchronously updates only compact stream-local coefficients from verification feedback. On four mathematical reasoning and code-generation benchmarks with Qwen3-8B and Gemma-4-12B-IT

targets, each using its own drafter and offline basis, SODA improves MAT by 9.9%–38.8% and reduces E2E latency by 6.5%–21.6% with identical online adaptation hyperparameters and only 61.4K–65.5K online-updated parameters. On Qwen3-8B, its compact updates achieve 97.1%–99.7% of the absolute MAT of full-drafter fine-tuning, and its gains remain positive across two uninterrupted cross-domain orders with shuffled requests. These results show that low-dimensional, asynchronous drafter adaptation can improve serving efficiency with a small online-updated state.

6 LIMITATIONS

Our evaluation covers Qwen3-8B and Gemma-4-12B-IT with separately trained adapters; it does not evaluate transfer of a shared basis between targets. The ablations, cross-domain ordering tests, and hardware analysis use only Qwen3-8B; the ordering tests cover two domain orders with one shuffled realization per order. The results do not establish robustness to arbitrary domain mixtures or longer-term drift, and gains may differ for other target sizes, drafter architectures, or request distributions. The method also assumes that an offline-learned correction subspace provides useful directions for online adaptation; when the frozen subspace has insufficient capacity or poor layer coverage, the B-only update may yield limited improvement. Finally, the serving-efficiency analysis assumes a persistent update worker and a serial worker-sharing model; deployments with different accelerator interconnects, worker cold-start costs, or scheduling policies may observe different latency and hardware trade-offs.

AI USE STATEMENT

We used generative AI tools to assist with language polishing of the manuscript.

REFERENCES

Wasi Uddin Ahmad, Aleksander Ficek, Mehrzad Samadi, Jocelyn Huang, Vahid Noroozi, Somshubra Majumdar, and Boris Ginsburg. OpenCodeInstruct: A large-scale instruction tuning dataset for code LLMs. *arXiv preprint arXiv:2504.04030*, 2025. URL https://arxiv.org/abs/2504.04030.

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

Nikhil Bhendawade, Irina Belousova, Qichen Fu, Henry Mason, Antonie Lin, Mohammad Rastegari, and Mahyar Najibi. Speculative streaming: Efficient and scalable speculative decoding with multi-stream attention. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 19547–19570, 2025.

Tianle Cai, Yuhong Li, Zhengyang Geng, Hongwu Peng, Jason D. Lee, Deming Chen, and Tri Dao. Medusa: Simple LLM inference acceleration framework with multiple decoding heads. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 5209–5235. PMLR, 2024. URL https://proceedings.mlr.press/v235/cai24b.html.

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

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

Mingguang Chen, Licheng Wang, and Bo Qu. Recursive self-improvement in AI: From bounded self-refinement to autonomous research loops. *arXiv preprint arXiv:2607.07663*, 2026b. doi: 10.48550/arXiv.2607.07663. URL https://arxiv.org/abs/2607.07663.

Qiaoling Chen, Zijun Liu, Peng Sun, Shenggui Li, Guoteng Wang, Ziming Liu, Yonggang Wen, Siyuan Feng, and Tianwei Zhang. Respec: Towards optimizing speculative decoding in reinforcement learning systems. *Proceedings of Machine Learning and Systems*, 8:367–379, 2026c.

Xin Cheng, Xingkai Yu, Chenze Shao, Jiashi Li, Yunfan Xiong, Yi Qian, Jiaqi Zhu, Shirong Ma, Xiaokang Zhang, Jiasheng Ye, Qinyu Chen, Chengqi Deng, Jiping Yu, Damai Dai, Zhengyan Zhang, Yixuan Wei, Yixuan Tan, Wenkai Yang, Runxin Xu, Yu Wu, Zhean Xu, Xuanyu Wang, Muyang Chen, Rui Tian, Xiao Bi, Zhewen Hao, Shaoyuan Chen, Huanqi Cao, Wentao Zhang, Anyi Xu, Huishuai Zhang, Dongyan Zhao, and Wenfeng Liang. DSpark: Confidence-scheduled speculative decoding with semi-autoregressive generation. *arXiv preprint arXiv:2607.05147*, 2026. doi: 10.48550/arXiv.2607.05147. URL https://arxiv.org/abs/2607.05147.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*, 2021. URL https://arxiv.org/abs/2110.14168.

Guhao Feng, Shengjie Luo, Kai Hua, Ge Zhang, Wenhao Huang, Di He, and Tianle Cai. In-place test-time training. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=dTWfCLSoyl.

Gemma Team. Gemma 4 Technical Report. *arXiv preprint arXiv:2607.02770*, 2026. doi: 10.48550/arXiv.2607.02770. URL https://arxiv.org/abs/2607.02770.

Moritz Hardt and Yu Sun. Test-time training on nearest neighbors for large language models. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=CNL2bku4ra.

Prannay Hebbar, Yogendra Manawat, Samuel Verboomen, Alesia Ivanova, Selvam Palanimalai, Kunal Bhatia, and Vignesh Baskaran. SIA: Self improving AI with harness & weight updates. *arXiv preprint arXiv:2605.27276*, 2026. doi: 10.48550/arXiv.2605.27276. URL https://arxiv.org/abs/2605.27276.

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. Measuring mathematical problem solving with the MATH dataset. In *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks*, 2021. URL https://arxiv.org/abs/2103.03874.

Fenglu Hong, Ravi Raju, Jonathan Lingjie Li, Bo Li, Urmish Thakker, Avinash Ravichandran, Swayambhoo Jain, and Changran Hu. Training domain draft models for speculative decoding: Best practices and insights. In *SCOPE Workshop at the International Conference on Learning Representations*, 2025. URL https://arxiv.org/abs/2503.07807.

Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. LoRA: Low-rank adaptation of large language models. *International Conference on Learning Representations*, 2022. URL https://openreview.net/forum?id=nZeVKeeFYf9.

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. URL https://arxiv.org/abs/2605.29707.

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 Thirteenth International Conference on Learning Representations*, 2025. URL https://arxiv.org/abs/2403.07974.

Avinash Kumar, Sujay Sanghavi, and Poulami Das. Test-time speculation. *arXiv preprint arXiv:2605.09329*, 2026. doi: 10.48550/arXiv.2605.09329. URL https://arxiv.org/abs/2605.09329.

Yaniv Leviathan, Matan Kalman, and Yossi Matias. Fast inference from transformers via speculative decoding. In *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pp. 19274–19286. PMLR, 2023. URL https://proceedings.mlr.press/v202/leviathan23a.html.

Shenggui Li, Chao Wang, Yikai Zhu, Yubo Wang, Fan Yin, Shuai Shi, Yefei Chen, Xiaomin Dong, Qiaoling Chen, Jin Pan, Ji Li, Laixin Xie, Yineng Zhang, Lei Yu, Yonggang Wen, Ivor Tsang, and Tianwei Zhang. SpecForge: A flexible and efficient open-source training framework for speculative decoding. *arXiv preprint arXiv:2603.18567*, 2026. URL https://arxiv.org/abs/2603.18567.

Yuhui Li, Fangyun Wei, Chao Zhang, and Hongyang Zhang. EAGLE: Speculative sampling requires rethinking feature uncertainty. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 28935–28948. PMLR, 2024a. URL https://proceedings.mlr.press/v235/li24bt.html.

Yuhui Li, Fangyun Wei, Chao Zhang, and Hongyang Zhang. EAGLE-2: Faster inference of language models with dynamic draft trees. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pp. 7421–7432. Association for Computational Linguistics, 2024b. doi: 10.18653/v1/2024.emnlp-main.422. URL https://aclanthology.org/2024.emnlp-main.422/.

Yuhui Li, Fangyun Wei, Chao Zhang, and Hongyang Zhang. EAGLE-3: Scaling up inference acceleration of large language models via training-time test. In 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. doi: 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 *The Twelfth International Conference on Learning Representations*, 2024. URL https://arxiv.org/abs/2305.20050.

Xiaoxuan Liu, Lanxiang Hu, Peter Bailis, Alvin Cheung, Zhijie Deng, Ion Stoica, and Hao Zhang. Online speculative decoding. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 31131–31146. PMLR, 2024. URL https://proceedings.mlr.press/v235/liu24y.html.

Yu-Yang Qian, Hao-Cong Wu, Chen Chen, Jiacheng Sun, Zhenhua Dong, Peng Zhao, and Zhi-Hua Zhou. AdaFlash: Adaptive speculative decoding via on-policy distilled diffusion drafters. *arXiv preprint arXiv:2607.19223*, 2026a. doi: 10.48550/arXiv.2607.19223. URL https://arxiv.org/abs/2607.19223.

Yu-Yang Qian, Hao-Cong Wu, Yichao Fu, Hao Zhang, and Peng Zhao. When drafts evolve: Speculative decoding meets online learning. *arXiv preprint arXiv:2603.12617*, 2026b. doi: 10.48550/arXiv.2603.12617. URL https://arxiv.org/abs/2603.12617.

Zelei Shao, Vikranth Srivatsa, Sanjana Srivastava, Qingyang Wu, Alpay Ariyak, Ameen Patel, Jue Wang, Percy Liang, Tri Dao, Ce Zhang, et al. Beat the long tail: Distribution-aware speculative decoding for rl training. *Proceedings of Machine Learning and Systems*, 8:1626–1638, 2026.

Yu Sun, Xiaolong Wang, Zhuang Liu, John Miller, Alexei A. Efros, and Moritz Hardt. Test-time training with self-supervision for generalization under distribution shifts. In *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pp. 9229–9248. PMLR, 2020. URL https://proceedings.mlr.press/v119/sun20b.html.

Yu Sun, Xinhao Li, Karan Dalal, Jiarui Xu, Arjun Vikram, Genghan Zhang, Yann Dubois, Xinlei Chen, Xiaolong Wang, Sanmi Koyejo, Tatsunori Hashimoto, and Carlos Guestrin. Learning to (Learn at test time): RNNs with expressive hidden states. In Aarti Singh, Maryam Fazel, Daniel Hsu, Simon Lacoste-Julien, Felix Berkenkamp, Tegan Maharaj, Kiri Wagstaff, and Jerry Zhu

(eds.), *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 57503–57522. PMLR, 13–19 Jul 2025. URL https://proceedings.mlr.press/v267/sun25h.html.

Junxiong Wang, Fengxiang Bie, Jisen Li, Zhongzhu Zhou, Zelei Shao, Yubo Wang, Yinghui Liu, Qingyang Wu, Avner May, Sri Yanamandra, Yineng Zhang, Ce Zhang, Tri Dao, Percy Liang, Ben Athiwaratkun, Shuaiwen Leon Song, Chenfeng Xu, and Xiaoxia Wu. When RL meets adaptive speculative training: A unified training-serving system. *arXiv preprint arXiv:2602.06932*, 2026. doi: 10.48550/arXiv.2602.06932. URL https://arxiv.org/abs/2602.06932.

Tengyu Xu, Eryk Helenowski, Karthik Abinav Sankararaman, Di Jin, Kaiyan Peng, Eric Han, Shaoliang Nie, Chen Zhu, Hejia Zhang, Wenxuan Zhou, Zhouhao Zeng, Yun He, Karishma Mandyam, Arya Talabzadeh, Madian Khabsa, Gabriel Cohen, Yuandong Tian, Hao Ma, Sinong Wang, and Han Fang. The perfect blend: Redefining RLHF with mixture of judges. *arXiv preprint arXiv:2409.20370*, 2024. URL https://arxiv.org/abs/2409.20370.

A Complete SODA Procedure

The complete procedure uses a frozen target p_θ, a pretrained drafter q_ϕ, an offline set $\mathcal{D}_{\text{off}}$, selected layers $\mathcal{S}$, rank r, update interval U, and learning rate η.

Offline stage. Jointly optimize rank-r factors (A, B) on $\mathcal{D}_{\text{off}}$ using Eq. (4), set $(A_0, B_0) \leftarrow (A, B)$, and freeze A_0.

Online stage. For each request stream, initialize B_t from B_0, reset the AdamW states, and set $\mathcal{R} \leftarrow \varnothing$. At each speculation round, draft with $W_t = W + A_0 B_t$, verify with p_θ, commit the output, and append the verified block to $\mathcal{R}$. Whenever $(t+1) \bmod U = 0$, update B_{t+1} from $\mathcal{R}$ using Eq. (5) and clear the buffer; otherwise, set $B_{t+1} = B_t$.

B Additional Experimental Details

The main results use Qwen3-8B and Gemma-4-12B-IT, each with its own drafter and offline basis. The Qwen3-8B DFlash drafter is trained with SpecForge on PerfectBlend and OpenCodeInstruct. Gemma-4-12B-IT uses an 11.96B-parameter target and a 727.5M-parameter drafter. The online coefficient states contain 65.5K and 61.4K parameters for Qwen3-8B and Gemma-4-12B-IT, respectively. The matched fixed-drafter baseline uses the same target, drafter, prompts, and decoding budget but disables online updates.

Both configurations use Ascend 910C accelerators with HCCS communication and BF16 decoding. The context window is 4096 tokens, the maximum generation length is 1024 tokens, and each 16-position speculative block contains one target anchor and $K = 15$ draft candidates. The adapter is rank-16 FC LoRA with $\alpha = 32$, applied to the target-feature projection layer. Each update replays the blocks accumulated over the preceding 64 verification rounds and performs eight AdamW steps with learning rate 5×10^{-4} and micro-batch size 4. The asynchronous worker uses a single-slot latest-only queue for pending update requests and applies each returned B_t before subsequent drafting rounds.

E2E latency follows the runner's serving-path timing. It excludes dedicated worker prestart under the assumption of a persistent worker but includes model loading, all prompts, prompt-boundary operations, and run finalization for both the fixed drafter and SODA. For one-shot deployments, worker cold start should be counted separately; prior measurements show that it can erase the gain on the shorter MBPP workload.

C Canonical Cross-Domain Orders

We also evaluate the uninterrupted cross-domain streams using the canonical within-benchmark request orders. The forward stream obtains MAT values of 4.45, 6.03, 6.35, and 3.87 on MBPP, Math500, GSM8K, and LiveCodeBench, respectively. The reverse stream obtains 3.87, 6.27, 6.19,

Table 5: Measured update-worker occupancy and projected accelerator requirement for Qwen3-8B. Capacity includes a 10% allowance for scheduling and per-process training-state swapping.

Dataset	t_{adapt} (s)	$t_{\text{adapt}}/t_{\text{infer}}$	Inference processes / update accelerator	Update accelerators / inference process	E2E reduction
GSM8K	32.8	6.3%	14.3	0.07	15.7%
LiveCodeBench	79.4	4.8%	18.8	0.05	10.2%

and 4.52 in its execution order from LiveCodeBench to MBPP. Relative to the independently reset Qwen3-8B runs in Table 1, per-benchmark MAT changes by at most 0.20, with no systematic degradation for benchmarks appearing later in either tested stream.

D Hardware Allocation Analysis

Asynchronous execution removes adaptation from the inference path but requires separate update-worker capacity. We estimate this requirement for a deployment in which one update accelerator serially processes adaptation requests from multiple independent inference processes. Let t_{adapt} denote the cumulative adaptation time associated with one inference process and t_{infer} its corresponding inference time.

Table 5 reports occupancy ratios of 6.3% on GSM8K and 4.8% on LiveCodeBench. Under the serial scheduling model, these measurements project to 14.3–18.8 inference processes per update accelerator, or 0.05–0.07 update accelerators per inference process. These values are capacity estimates rather than measurements from a multi-process deployment.

E Model-Dependent Effectiveness

The frozen subspace can become a bottleneck when its rank or layer coverage is insufficient. On an OpenSeeker slim stream with 500 prompts and a 256-token generation limit, the FC rank-16 B-only variant leaves MAT unchanged at 2.44 to two decimal places and increases E2E latency from 538.32s to 560.02s.

Stronger adaptation variants show that this failure is not caused solely by the online-update mechanism. A q/v rank-4 A+B reference improves MAT to 4.20 and reduces E2E latency to 429.74s, while a q/v rank-64 B-only variant reaches MAT 4.16 and 436.15s, recovering most of the A+B gain while retaining a frozen basis. Within FC layers, increasing the B-only rank from 16 to 256 improves MAT to 3.87 and reduces E2E latency to 482.77s; FC rank-16 A+B reaches MAT 4.06 but increases E2E latency to 646.02s because of its update cost. These comparisons indicate that limited gains can result from insufficient subspace capacity or inappropriate adapter placement rather than from B-only adaptation alone.

F Additional Results

Learning-rate diagnostics on a 64-prompt Math500 subset show that synchronous online updates yield only small MAT gains while increasing latency because adaptation remains on the critical path. Low learning rates give small improvements, whereas larger learning rates degrade acceptance. This diagnostic motivates asynchronous execution but is not used as evidence for the main effectiveness claims.