SERVAL: EFFICIENT IN-FLIGHT DRAFT ADAPTATION FOR LLM SERVING

Anonymous authors
Paper under double-blind review

ABSTRACT

Speculative decoding accelerates large language model inference: a draft model proposes tokens and the target model verifies them. Its speedup depends strongly on the acceptance length, the number of tokens each verification step accepts, which often falls when a fixed draft encounters out-of-distribution traffic. We introduce SERVAL (SERVing-time Adaptive LoRA), a speculative-decoding serving system that adapts the draft (e.g., DFlash) to the traffic it serves using persistent per-domain LoRA adapters. At each decode step, the server sends the target model's hidden states and token-level acceptance outcomes to a trainer on separate GPUs, which updates the adapters asynchronously. The adapters are applied only to the query and output projections of a DFlash draft, whose prefix key–value (KV) cache is computed from the target's hidden states, so running requests switch to new adapter versions at the next decode step after they arrive, without recomputing that cache. Starting from a general adapter it trained earlier, SERVAL has a higher acceptance length and a lower mean time per output token than the frozen DFlash draft for three models on every out-of-distribution stream and benchmark we evaluate. On a mixed stream of code, finance, and medical requests, it raises the acceptance length by 12–57 % and lowers the time per output token by 12–51 %. The server's overhead for sending feedback and loading new adapter versions is only 0.6–2.1 % of the decode step. The per-domain adapters it learns can be saved for reuse: served frozen by another server, they keep a higher acceptance length than DFlash and the general adapter on Qwen3-8B.

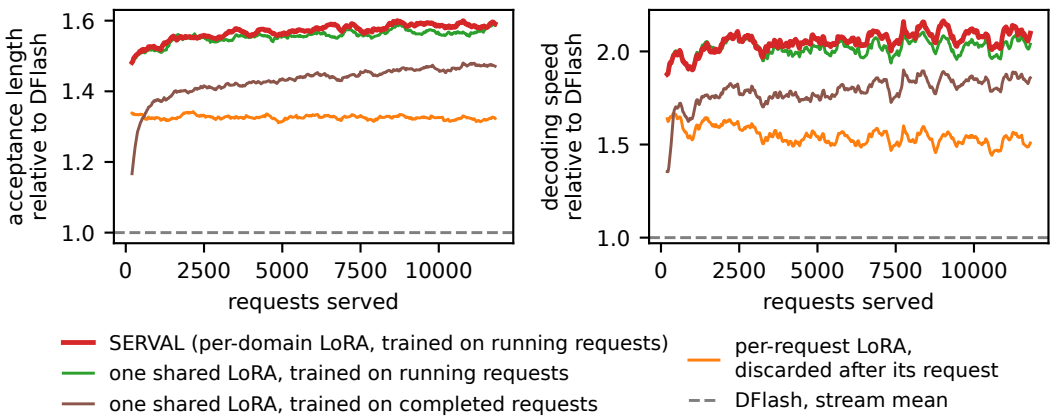


Figure 1: Qwen3-8B on the mixed code+finance+medical stream, adapted drafts started from DFlash. Acceptance length (left) and per-request decoding speed (right, the inverse of the mean time per output token), relative to the stream mean of DFlash, the released draft served frozen (dashed). SERVAL and the shared LoRA trained on running requests stay above the other two adapted drafts on both metrics.

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.

1 INTRODUCTION

Speculative decoding accelerates large language model (LLM) generation: a draft model proposes tokens and the target model verifies them, which preserves the target's output distribution (Leviathan et al., 2023; Chen et al., 2023). The number of tokens a verification step accepts is the acceptance length, and the speedup grows with it. Draft models are usually trained offline on a fixed corpus before deployment. On traffic outside the distribution of that corpus, the acceptance length falls, and so does the speedup (Liu et al., 2024; Xia et al., 2024). This out-of-distribution (OOD) problem has three sources in serving: the overall traffic, individual domains, and individual requests. The first is between the pretrained draft and the serving traffic: the draft is trained once and frozen, so its training distribution stays where it was while the traffic keeps changing; retraining it offline on collected traffic yields another frozen draft, which is mismatched again once the traffic changes. The second is between domains: the traffic mixes several domains with different distributions, and for any one domain, a draft trained on the mixture is partly trained on the other domains (Liu et al., 2024; Hong et al., 2025). The third is between requests: even within a familiar domain, a request can induce continuation patterns poorly represented in the draft's training distribution, through its specific notation, variable bindings, and reasoning trajectory. Verification provides feedback on these patterns while the request is running, but updating only after the request ends comes too late to benefit that request (Kumar et al., 2026; Sun et al., 2025). These observations motivate adapting the draft both across requests and while they are running.

We add low-rank adapters (LoRA; Hu et al., 2022) to the draft model. Each of the three sources of OOD corresponds to one design choice. First, persistence across requests: the draft is trained while it serves, and what it learns is kept across requests. Second, specialization by domain: each domain has its own adapter, trained only on its own requests. Third, updates within a request: feedback from a request updates its adapter while the request is still running, so that its own later tokens benefit. Controlled comparisons on three models and two streams of mixed-domain requests test each choice against its alternative (Figure 1, Table 1, Section 4.3). On every model and stream, one adapter shared by all requests reaches a higher acceptance length than per-request adapters discarded when their request ends. Per-domain adapters exceed the shared adapter in most settings and match it in the rest. On every model and stream, updating the shared adapter continuously from the feedback of running requests reaches a higher acceptance length than updating it by the rules of online speculative decoding (OSD; Liu et al., 2024), which use only completed requests.

Supporting these three choices together naturally requires three things of the serving system: (1) the draft is trained while the system serves, and sending feedback and loading new adapter versions cost less time than the gain in acceptance length saves; (2) requests with different adapters share one decode batch; (3) a running request switches to a new adapter version without recomputing its prefix key–value (KV) cache. We build on SGLang (Zheng et al., 2024) with DFlash drafts (Chen et al., 2026); in the SGLang release we use, the draft runs with one static set of weights for the whole batch. We present SERVAL (SERVing-time Adaptive LoRA), a speculative-decoding serving system that instead trains the draft on the traffic it serves: the requests of one decode batch use their own continuously updated per-domain LoRA adapters, new versions take effect in running requests without recomputing any prefix KV cache, and the per-domain adapters it learns can be saved for other servers to reuse. Section 5 contrasts its design with OSD, EvoSpec (Zhang et al., 2026), Test-Time Speculation (Kumar et al., 2026), and Aurora (Wang et al., 2026a).

SERVAL has two parts: a server that runs the target model and the draft model with its LoRA adapters, and a trainer on other GPUs of the same node that updates the adapters during serving. Each request is assigned an adapter by its domain label, and the draft model processes a whole batch in one forward pass, each request with its prefix KV cache and adapter. The adapters modify only the draft's query and output projections, and DFlash computes the prefix keys and values from the target's hidden states with the key and value projections, so a request's prefix KV cache does not depend on the adapter version. At every decode step the server sends the target's hidden states and token-level acceptance outcomes to the trainer as feedback; the trainer recomputes the step's draft forward pass from them, keeps one optimizer per adapter, and sends new adapter versions back. The server loads a version at a decode-step boundary, and running requests assigned to the version's adapter use it from the next step on. Starting from a general adapter it trained earlier (Section 4.1), SERVAL has a higher acceptance length and a lower mean time per output token (TPOT) than DFlash on three models, two out-of-distribution mixed streams, and benchmarks at fixed concurrency. On the

code+finance+medical stream, its acceptance length is 57 %, 21 %, and 12 % higher and its TPOT 51 %, 12 %, and 17 % lower on Qwen3-8B, gpt-oss-120b, and MiniMax-M2.7 (Table 1). Sending feedback and loading versions take only 0.6–2.1 % of the decode step.

This paper makes three contributions.

- We use controlled comparisons to guide three design choices: retaining adapters across requests, specializing them by domain, and updating them while requests run.
- We design and implement SERVAL, a serving system that trains the draft's adapters on the traffic it serves. Requests with different adapters share one forward pass of the draft, and running requests switch to new adapter versions without recomputing their prefix KV caches.
- We evaluate SERVAL on three models, two out-of-distribution mixed streams, and benchmarks at fixed concurrency. Its acceptance length is higher and its TPOT lower than DFlash's in every setting.

2 Background

2.1 Speculative Decoding and Acceptance Length

At each decode step, a draft model proposes the next K tokens of a request as one block, taking the most likely token at each position, and the target model T verifies the block in one forward pass (Stern et al., 2018) by speculative sampling (Leviathan et al., 2023; Chen et al., 2023). Since this proposal is deterministic, a drafted token is accepted with the probability the target assigns to it; at a rejection, the target samples from its own distribution renormalized without the rejected token and drops the rest of the block, so the output distribution is the target's. A step accepts $\tau \in [1, K+1]$ tokens, including the target's own; τ is the acceptance length (Li et al., 2024b; Chen et al., 2026). If s_t is the wall time of step t, a request's time per output token is $\mathrm{TPOT} = \sum_t s_t / \sum_t \tau_t = \bar{s}/\bar{\tau}$; TPOT falls when the relative gain in $\bar{\tau}$ exceeds the relative increase in $\bar{s}$ (Section 4.4).

We build on DFlash (Chen et al., 2026), which reads the target's hidden states as the EAGLE drafts (Li et al., 2024b; 2025) do. It fuses the hidden states of a few target layers into context features and projects them once into the keys and values of every draft layer at the prefix positions, so the draft's prefix KV cache is a projection of the target's states rather than an output of its own layers.

At every drafted position j, the verification pass computes the target's next-token distribution $p_{T,j}$ given the prefix and the drafted tokens before it. These distributions are computed at every step whether or not they are used, and distillation trains a draft toward them, offline (Zhou et al., 2024) or while it serves (Liu et al., 2024).

2.2 The Out-of-Distribution Problem in Speculative Decoding

A released draft is trained offline on responses the target generated for a fixed corpus (Chen et al., 2026; Hong et al., 2025); DFlash trains its drafts on sequences of at most 4,096 tokens. Traffic is out of distribution for a draft when its requests differ from these responses in domain or in length. Of the three sources of OOD, the mismatch between the pretrained draft and the serving traffic rules out any fixed draft, so the online systems of Section 5 train the draft while it serves, supervised by the target model. The other two determine how that training is organized and when its updates take effect. A single draft trained while serving must fit all domains at once, and OSD (Liu et al., 2024) reports a higher token acceptance rate for drafts adapted per domain than for one shared draft. A request's later tokens benefit only from updates that take effect before the request ends.

2.3 LLM Serving System

SERVAL is built into SGLang (Zheng et al., 2024). Its scheduler admits requests into a continuous-batching loop (Yu et al., 2022), every request's KV cache is stored in a paged pool (Kwon et al., 2023), and speculative decoding runs inside the loop, with one proposal and one verification per decode step for the whole batch. Any change the server makes between two decode steps therefore takes effect in running requests from the next step on.

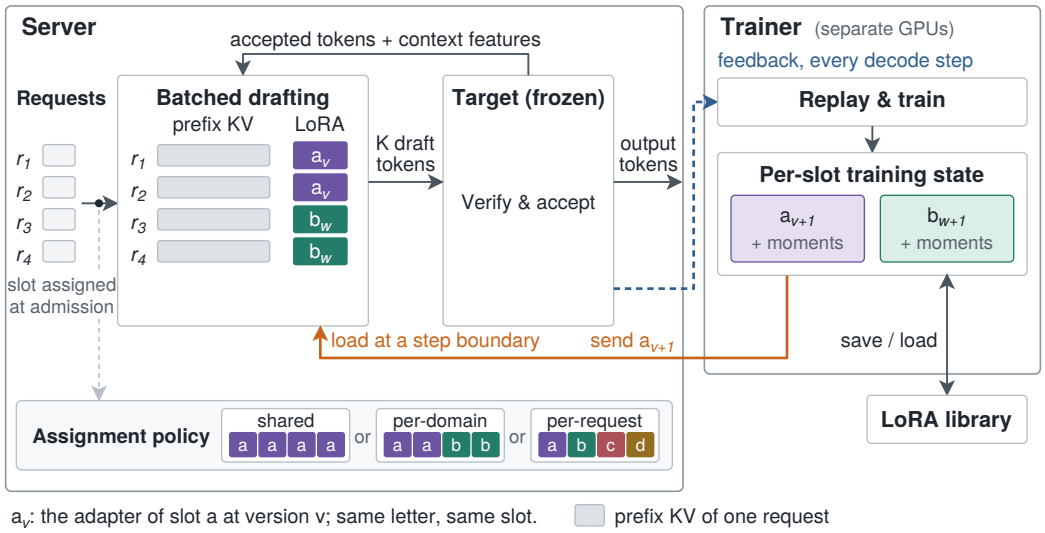


Figure 2: Overview of SERVAL. The assignment policy maps each request to a slot, one entry of the server's adapter pool; the request uses the slot's adapter. At every decode step the target's hidden states are sent to the trainer as feedback (dashed blue). The trainer sends new versions, loaded at step boundaries (orange).

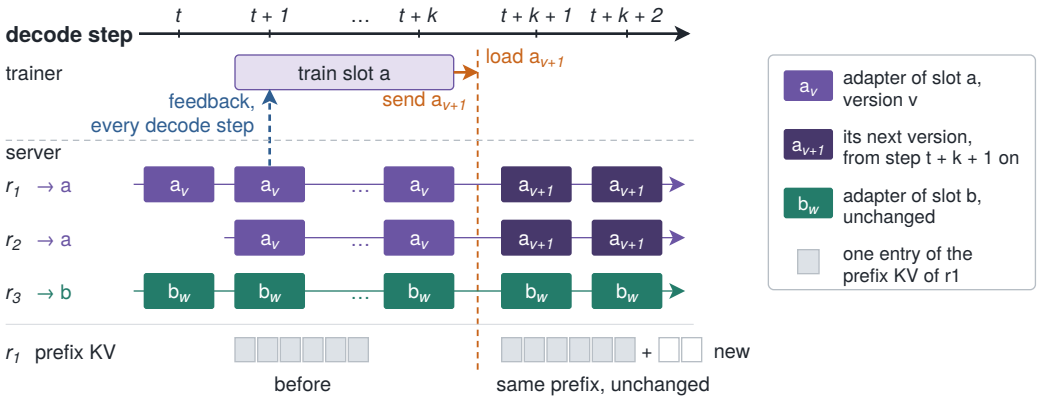


Figure 3: Loading a new version. After decode step $t{+}k$, a_{v+1} replaces a_v in slot a; the slot's requests use it from step $t{+}k{+}1$ on, with their prefix KV caches unchanged, and slot b stays at b_w.

3 SERVAL

SERVAL has two parts (Figure 2): a server that runs the target and draft models, and a trainer on separate GPUs. A request is assigned to an adapter slot at admission. At every decode step the server sends the trainer the target's hidden states and token-level acceptance outcomes. The trainer replays the step, updates the slot's adapter, and sends new versions, which the server loads between two steps without recomputing any prefix KV cache. Adapters can be saved to and loaded from a LoRA library.

3.1 Adapter Design

The adapter is a rank-32 LoRA (Hu et al., 2022) on the query and output projections of every draft layer. The keys and values a draft layer reads at prefix positions are computed once by projections the adapter does not modify (Section 2.1); a request's prefix KV cache therefore stays valid when

the request switches to a new adapter version. If each draft layer computed its prefix keys and values from the layer below, a new version would change them in every layer but the first.

3.2 Batched Multi-Adapter Draft

The server keeps adapters in a pool of slots: for each adapted projection a stacked tensor holds the low-rank factors of every slot, and a request carries the index of the slot it is assigned to. Each adapted projection has two fused Triton kernels, which run inside the decode CUDA graph. They gather each request's factors by its slot index and add the low-rank product to the projection's output. Each launch covers the whole batch, however many adapters its requests use. Slot indices are set by one of three assignment policies: one slot shared by all requests, one slot per domain, or one slot per request. The per-domain policy selects the slot by the domain label sent with the request, and a per-request slot is freed when its request finishes. Each new adapter starts from a saved adapter or from the zero adapter, whose low-rank product is zero.

3.3 Feedback Path and Training

After the verification pass of a step, the server sends the trainer one message per request in the batch: the block's input tokens and positions, the number of drafted tokens the step accepted, the context features of the accepted tokens, and the target's final hidden states at the block positions. The trainer has its own copy of the draft and of the target's LM head. It keeps a copy of each request's prefix, the context features sent at admission. To replay the step, it reruns the draft under the slot's current adapter on the block input given that prefix, recovers the step's distributions $p_{T,j}$ from the final hidden states, and then appends the accepted tokens' context features to the copy. To keep that copy consistent with the request, every message must reach the trainer. The server does not wait for the trainer to process a message, but when the queue of messages waiting for the trainer is full, the scheduler waits for the trainer to take one step's messages before starting the next step. This wait is part of the step overhead, the server's scheduler-thread time per decode step for sending feedback and loading versions.

For each message, the trainer computes the forward KL divergence between the target's and the draft's distributions at every drafted position j, with the slot's adapter θ_a and the block input x,

$$\mathcal{L}(\theta_a) = \sum_{j=1}^{K} w_j \, \mathrm{KL}\big(p_{T,j} \,\|\, q_{\theta_a}(\cdot \mid x, j)\big), \tag{1}$$

where the weights w_j favor the first rejected position and decay past it, as detailed in Appendix A. The trainer accumulates the message's gradient for its slot. Once a slot has accumulated N messages, the trainer takes one step of Adam (Kingma & Ba, 2015), modified as shown in Table 6, on their mean gradient with the slot's own optimizer state and sends the server the new version, labeled with the slot and the adapter it was trained for.

3.4 Loading New Versions

A version sent by the trainer enters a receive queue in the server. Between two decode steps, the server copies into each slot the newest version trained for the adapter currently in that slot. The copy precedes the next forward pass on the same CUDA stream, so a step reads either the previous version or the new one in full. From the step after the copy, every request assigned to the slot decodes with the new version, including requests admitted long before it was sent (Figure 3): a request is assigned to a slot, not to a version. Section 4.4 reports the update lag, the number of decode steps between the newest step whose feedback a version could reflect and the step that loads it.

3.5 Per-Domain Assignment and the LoRA Library

A per-domain adapter is trained on all requests of its domain and persists as long as the domain's traffic does. Requests without a domain label can be assigned to slots by a router inside the server. A slot's adapter can be saved at any time to the LoRA library. A server can load adapters from the library into its slots to continue training them, as with the general adapter of Section 4.1, or to serve them frozen, as in Section 4.3. Implementation details are in Appendix A.

4 EXPERIMENTS

4.1 SETUP

Models and hardware. Qwen3-8B (Yang et al., 2025), gpt-oss-120b (OpenAI, 2025), and MiniMax-M2.7 (MiniMax, 2026) with their released DFlash drafts, on H200 nodes with tensor parallelism across 2, 4, and 4 GPUs. The drafts propose $K = 15$, 9, and 7 tokens per step. The methods with a trainer run it on as many other GPUs as the server uses.

Traffic. Online serving: two mixed streams, finance+math and code+finance+medical, with a concurrency limit of 32 and Poisson arrivals at 0.85 of DFlash's capacity, the rate at which DFlash completes requests at that limit. Fixed concurrency of 32: four benchmarks, AIME 2022–25, LiveCodeBench-hard (LCB-hard; Jain et al., 2025a), code_contests (Li et al., 2022), and the FinQA test split (Chen et al., 2021). Requests are decoded by sampling, with thinking mode enabled and at most 24,576 output tokens. Each configuration runs three times; Appendix A describes the traffic in detail.

Compared methods. The baseline, DFlash, serves each model's released draft without adapters or training. The other methods add rank-32 adapters to the draft and start from the model's general adapter, which SERVAL trained earlier on 11,200 requests of general chat, mathematics, code, and agentic traffic. The adapters are trained at learning rate 6×10^{-4}, with one optimizer step per $N = 128$ messages of a slot. The runs of Figure 1 and of the full-parameter comparison in Section 4.3 instead start from DFlash itself; their adapters start from the zero adapter. General-frozen serves the general adapter without updates, separating its benefit from that of adaptation during serving. TTS-style, after Test-Time Speculation (TTS; Kumar et al., 2026), assigns a new slot to each request and discards it when the request ends; SERVAL-shared keeps one slot for the whole stream; SERVAL keeps one slot per domain, selected by the request's domain label. OSD-analog, after OSD (Liu et al., 2024), differs from SERVAL-shared only in OSD's update and loss rules: it buffers the messages of completed requests and takes one optimizer step over the buffer after every 32 completions, the concurrency limit; its loss covers only the first rejected position of each block. Table 6 in Appendix A lists all settings.

Metrics. Acceptance length is pooled over the verification steps of all requests of a run, then averaged over runs; TPOT is measured at the client per request and averaged over requests, then over runs. Under online serving at a fixed request rate, the TPOT reflects both the acceptance length and the number of running requests; the fixed-concurrency benchmarks hold the load at 32. Answer accuracy on AIME 2022–25 does not distinguish the methods, as shown in Appendix B.

4.2 MAIN RESULTS

In Table 1, starting from the general adapter, SERVAL raises the acceptance length over DFlash by 57 %, 21 %, and 12 % on Qwen3-8B, gpt-oss-120b, and MiniMax-M2.7 on the code+finance+medical stream, and by 41 %, 8 %, and 15 % on finance+math. Its TPOT is the lowest of all methods on every model and stream: 51 %, 12 %, and 17 % below DFlash on the code+finance+medical stream, and 53 %, 3 %, and 17 % on finance+math. On Qwen3-8B, SERVAL does almost as well without the general adapter. Started from DFlash as in Figure 1, it is well above DFlash from the first 400 requests on and reaches an acceptance length of 4.03 over the whole code+finance+medical stream, versus 4.04 from the general adapter.

On the fixed-concurrency benchmarks of Table 2, SERVAL has the highest acceptance length and the lowest TPOT in all twelve rows; relative to DFlash, its acceptance length is 7–74 % higher and its TPOT 0.5–43 % lower. FinQA's domain is outside the general adapter's training data; there, adaptation during serving raises the acceptance length by 3–11 % over the general adapter and lowers the TPOT by 2.6–8 %. On gpt-oss-120b, whose general adapter does not improve on DFlash for FinQA, adaptation during serving provides all of SERVAL's gain.

4.3 ABLATION STUDY

Each of the three design choices is tested below against its alternative. *Persistence across requests:* TTS-style uses per-request adapters. It has a lower acceptance length than SERVAL-shared on every

Table 1: Online serving: Poisson arrivals at 0.85 of DFlash's capacity, concurrency 32, 8,000 requests per run on finance+math and 12,000 on code+finance+medical. Acceptance length (AL) and time per output token (TPOT), mean over three runs; bold marks the best value of each metric in a row. Range over runs: Table 10.

Stream	AL ↑ / TPOT (ms) ↓					
	DFlash	general-frozen	TTS-style	OSD-analog	SERVAL-shared	SERVAL
Qwen3-8B						
finance+math	3.38 / 3.95	4.42 / 2.11	4.47 / 2.20	4.54 / 2.07	4.74 / 1.90	**4.78 / 1.87**
code+finance+medical	2.57 / 4.03	3.68 / 2.24	3.76 / 2.28	3.83 / 2.17	4.00 / 2.04	**4.04 / 1.99**
gpt-oss-120b						
finance+math	4.16 / 3.59	4.35 / 3.60	4.35 / 4.36	4.40 / 4.29	**4.51** / 3.49	**4.51 / 3.48**
code+finance+medical	2.88 / 4.14	3.32 / 3.83	3.36 / 3.98	3.39 / 3.83	3.47 / 3.66	**3.48 / 3.65**
MiniMax-M2.7						
finance+math	2.90 / 10.04	3.27 / 8.49	3.27 / 8.64	3.29 / 8.49	3.32 / 8.36	**3.33 / 8.30**
code+finance+medical	2.41 / 11.75	2.65 / 10.09	2.66 / 10.16	2.67 / 10.06	2.69 / 9.91	**2.70 / 9.80**

Table 2: Fixed concurrency of 32; one domain per benchmark, so SERVAL-shared coincides with SERVAL. Acceptance length (AL) and time per output token (TPOT), mean over three runs; bold as in Table 1. Range over runs: Table 11.

Model	Benchmark	AL ↑ / TPOT (ms) ↓				
		DFlash	general-frozen	TTS-style	OSD-analog	SERVAL
Qwen3-8B	AIME 2022–25	2.80 / 6.40	4.34 / 4.23	4.43 / 4.19	4.34 / 4.31	**4.55 / 4.07**
	LCB-hard	2.27 / 8.85	3.66 / 5.37	3.79 / 5.28	3.66 / 5.50	**3.91 / 5.08**
	code_contests	2.27 / 9.12	3.65 / 5.59	3.81 / 5.43	3.69 / 5.88	**3.94 / 5.23**
	FinQA test	3.82 / 3.95	4.28 / 3.60	4.30 / 3.78	4.41 / 4.05	**4.75 / 3.31**
gpt-oss-120b	AIME 2022–25	3.25 / 4.47	3.89 / 4.20	3.92 / 4.35	3.92 / 4.28	**3.97 / 4.19**
	LCB-hard	2.57 / 5.98	3.23 / **5.20**	3.26 / 5.25	3.23 / 5.28	**3.31 / 5.20**
	code_contests	2.46 / 6.82	3.07 / 5.75	3.14 / 5.79	3.11 / 5.79	**3.21 / 5.61**
	FinQA test	4.53 / 4.14	4.51 / 4.27	4.57 / 4.77	4.61 / 4.24	**4.84 / 4.12**
MiniMax-M2.7	AIME 2022–25	2.78 / 10.97	3.08 / 9.98	3.11 / 9.92	3.08 / 9.91	**3.12 / 9.87**
	LCB-hard	2.37 / 13.93	2.59 / 12.76	2.61 / 12.80	2.60 / 12.82	**2.62 / 12.68**
	code_contests	2.29 / 15.08	2.48 / 13.97	2.50 / 13.91	2.48 / 13.95	**2.51 / 13.85**
	FinQA test	2.96 / 9.73	3.42 / 8.56	3.44 / 8.81	3.48 / 8.50	**3.52 / 8.34**

model and stream. Relative to general-frozen, its adapters raise the acceptance length by at most 2 % but raise the TPOT by 1–21 %. The 21 % occurs on the finance+math stream of gpt-oss-120b, where its trainer falls behind the server and its step overhead is 0.90 ms, versus 0.19 ms for SERVAL (Tables 5 and 9). *Specialization by domain:* one adapter per domain reaches the highest acceptance length of all methods on every model and stream. On the code+finance+medical stream the per-domain adapters exceed the shared adapter in six of the nine model–domain pairs of Table 3 and match it in the other three; the six leads hold in every run, with the widest margin on finance. *Updates within a request:* OSD-analog trains on a request's messages only once the request completes. It has a lower acceptance length and a higher TPOT than SERVAL-shared on every model and stream. When its trainer falls behind, the server waits for it (Section 3.3): on the finance+math stream of gpt-oss-120b the step overhead is 1.8 ms, versus 0.18 ms for SERVAL-shared (Table 9).

New versions in running requests: updates within a request benefit that request only if it switches to new versions while it runs. SERVAL-shared (version-pinned) is SERVAL-shared except that each request uses the adapter version loaded at its admission until it completes. For Qwen3-8B on the code+finance+medical stream, its acceptance length is 3.96 versus SERVAL-shared's 4.00 starting from the general adapter, and 3.95 versus 4.00 starting from DFlash, lower in every run. The gap is largest over the first 250 requests, at 2.5 % and 5.3 % for the two starts; over the rest of the stream it stays at 0.8–1.6 %.

Table 3: Per-domain acceptance length on the code+finance+medical stream, mean over three runs. All methods: Table 12.

	Qwen3-8B			gpt-oss-120b			MiniMax-M2.7		
Method	code	finance	medical	code	finance	medical	code	finance	medical
DFlash	2.42	3.81	2.78	2.78	4.50	2.94	2.38	2.96	2.31
general-frozen	3.71	4.28	3.01	3.31	4.46	2.91	2.60	3.43	2.57
SERVAL-shared	4.04	4.74	3.19	3.45	4.84	3.05	2.64	3.53	2.61
SERVAL	4.07	4.93	3.25	3.45	4.95	3.05	2.64	3.58	2.63

Table 4: Qwen3-8B, acceptance length, mean over three runs. (a) Adapters frozen at the end of the code+finance+medical stream of Table 1, served on unseen prompts of the same domains. (b) Rank-32 adapter versus full parameters, trained at fixed concurrency on single-domain streams, first 1,500 requests (whole stream: Table 14). Params are the trained parameters over both GPUs; the LoRA share is the q/o LoRA's gain over DFlash divided by the gain of full training with K/V frozen.

(a) Frozen reuse

Method	code	finance	medical
DFlash	2.45	3.82	2.79
general-frozen	3.72	4.25	3.00
per-domain frozen	4.08	5.06	3.28
shared frozen	4.05	4.77	3.19

(b) LoRA versus full parameters

Method	params	finance	math
DFlash	–	3.82	3.27
LoRA (r32, q/o)	3.9M	4.64	4.66
LoRA (r32, q only)	2.0M	4.35	4.43
LoRA (r32, o only)	2.0M	4.63	4.50
full (K/V frozen)	922.8M	4.81	4.75
LoRA share of the full gain		83 %	94 %
full q/o	167.8M	4.56	–
full incl. K/V	964.7M	4.77	–

LoRA versus full parameters: the full-parameter variant trains every draft weight except those that compute the prefix keys and values. With 0.4 % as many trained parameters, a rank-32 adapter obtains 83 % of the full-parameter acceptance-length gain on finance and 94 % on mathematics over the first 1,500 requests of single-domain streams on Qwen3-8B (Table 4b). Training the key and value projections as well does not add to the gain: they change only the block's keys and values, and the prefix KV cache keeps the initial ones. *Reuse:* the per-domain adapters of Qwen3-8B are frozen at the end of the code+finance+medical stream and served by another server on unseen prompts. On every domain, they have a higher acceptance length than DFlash, the general adapter, and the frozen shared adapter (Table 4a). Their margin over the shared adapter is widest on finance.

4.4 Serving Overhead and Training Cost

Table 5 reports a step overhead of 0.15–0.19 ms per decode step, 0.6–2.1 % of the step. Table 8 in Appendix C profiles the serving process on Qwen3-8B: in-flight adaptation takes 2.8 % of the step in GPU time and 2.0 % in scheduler-thread time. Most of this is the hidden-state transfer; the fused multi-adapter LoRA computation takes 1.2 % of the step, and version loading almost none. Each server GPU sends its trainer GPU 68–176 KB per message, mostly the target's hidden states at the $K + 1$ positions of the verification pass; this is 21–42 KB per generated token, less where the acceptance length is higher. The update lag is 2–8 decode steps at the median and 3–110 at the 90th percentile; a slot receives a new version every 13–82 steps on average. Requests decode for a median of 140–710 steps, so a version trained on a request's early steps serves its later ones.

Training takes additional GPUs, as many as the server uses. With them, SERVAL has a higher acceptance length and a lower TPOT than DFlash on the same serving GPUs.

5 Related Work

Online speculative decoding (Liu et al., 2024) adapts the draft in a loop: it records the target's distributions, distills the draft toward them in a separate training pass, and periodically loads the

Table 5: Cost of in-flight adaptation in SERVAL, mean over three runs. Step overhead is the scheduler-thread time per decode step for sending feedback and loading versions, in ms and as a percentage of the step. Transfer is the size of the hidden states one server GPU sends to the trainer per message. The update lag (Section 3.4) is counted in decode steps.

	Qwen3-8B		gpt-oss-120b		MiniMax-M2.7	
	fin+math	code+fin+med	fin+math	code+fin+med	fin+math	code+fin+med
Step overhead, ms (%)	0.152 (1.8)	0.157 (2.1)	0.186 (1.1)	0.189 (1.4)	0.174 (0.6)	0.171 (0.6)
Transfer (KB/message)	176	168	94	83	73	68
Update lag p50 / p90	7 / 41	8 / 110	3 / 16	4 / 26	2 / 3	2 / 3

updated weights. It updates one draft or one per domain, which a router selects for each request. OnlineSPEC (Qian et al., 2026b) analyzes this loop as online learning. Later systems keep the loop. Two of them adapt a draft on device or in a single stream (Ramakrishnan et al., 2025; Bhansali & Heck, 2025). The rest differ from OSD and from one another in three respects. The first is whether the learned state persists across requests or is reset when a request ends: OSD, EvoSpec (Zhang et al., 2026), AdaFlash (Qian et al., 2026a), TIDE (Park et al., 2026), and Aurora (Wang et al., 2026a) keep it, and Test-Time Speculation (Kumar et al., 2026) resets it. The second is whether feedback reaches the trainer only after a request completes or during generation: OSD, AdaFlash, and Aurora wait for completion, and EvoSpec, TIDE, and Test-Time Speculation do not. The third is whether a decode batch uses one learned state or several. Only AdaFlash, TIDE, and Aurora report concurrent serving, and each uses one learned state per batch. Whether the updated weights reach the requests already running depends on when the server loads a new version, not on when feedback is collected. Test-Time Speculation updates a request's own draft while the request runs. The released implementations of AdaFlash and Aurora load new weights between requests, and EvoSpec and TIDE do not specify when they load them. SERVAL combines feedback during generation, persistent per-domain adapters whose new versions reach the requests already running, and several such adapters within one decode batch.

Multi-LoRA serving systems (Sheng et al., 2024; Chen et al., 2024; Wu et al., 2024; Predibase, 2023) apply many static adapters in one batched forward pass, and each request keeps one adapter for its lifetime. SERVAL batches adapters the same way but updates them while they serve. Systems that co-serve fine-tuning and inference (Oliaro et al., 2026; Zhang et al., 2025) share GPUs between the two in one runtime. In their evaluated configurations, the adapters under training do not serve requests. SERVAL's running requests use the adapters under training, and their prefix KV caches stay valid across versions. Test-time training (Sun et al., 2020; 2025; Yang et al., 2026; Wang et al., 2026b) gives each input its own mutable weights and thereby changes the model's output. SERVAL adapts only the draft, so the output distribution remains the target's.

6 Conclusion

We present SERVAL, a serving system that trains the draft model while it serves. The draft learns from the target's distributions that each verification pass computes. SERVAL's LoRA adapters modify only the query and output projections of a DFlash draft, so running requests switch to new versions without recomputing their prefix KV caches. Requests with different adapters share one decode batch. Starting from a general adapter, SERVAL raises the acceptance length by 12–57 % and lowers the TPOT by 12–51 % relative to DFlash for three models on the code+finance+medical stream. The step overhead is only 0.6–2.1 % of the decode step. On every model and stream, one adapter kept across requests has a higher acceptance length than per-request adapters, and updating it from running requests reaches a higher acceptance length than updating it by OSD's rules, which use only completed requests. One adapter per domain reaches the highest acceptance length of all methods. We plan to test the server's router on unlabeled traffic, evaluate a trainer shared by several servers, and apply SERVAL to other draft families. A draft is suitable if its prefix keys and values do not depend on the adapter, as in single-layer drafts (Li et al., 2024b; 2025).

AI USE STATEMENT

In this work, we used generative AI tools for three purposes. For research ideation and execution, the tools gave feedback on the design of the experiments and were used to implement parts of the serving and training code; to write the experiment drivers, the analysis scripts, and the scripts that clean and reformat the datasets; and to interpret measurement logs. For writing assistance, the tools were used to draft and edit parts of the text and to generate the tables from the experiment records. For retrieval and discovery, the tools were used to find and summarize related work. We have not used generative AI tools for the other tasks with required disclosure. We have reviewed all AI-assisted work: equivalence tests checked the code that produces the reported numbers against reference implementations, the reported numbers were re-read from the experiment records, cited work was read in the original, and we reviewed the text. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

ETHICS STATEMENT

SERVAL changes only the draft model, and the verification rule keeps the output distribution of the target model, so adaptation does not change the capabilities of the served model. The experiments use publicly released models and datasets, listed in Section 4.1 and Appendix A; we collected no new data and conducted no studies with human subjects. Because the adapters are trained on users' requests and can be saved and served by other servers, a deployment should apply the same privacy and access policies to them as to the requests themselves.

REPRODUCIBILITY STATEMENT

We provide the source code of SERVAL in the supplementary material, with a README that explains how to run it. The supplementary material includes the server and the trainer, the tests that check them against reference implementations, the scripts that run the experiments of Section 4 and build their streams and benchmarks from public datasets, and the general adapter of each model. Section 4.1 and Appendix A describe the experimental setup, Table 6 lists all hyperparameters, and the README lists the checkpoint revisions of the models and their DFlash drafts.

REFERENCES

Shrenik Bhansali and Larry Heck. Draft, verify, and improve: Toward training-aware speculative decoding. *arXiv preprint arXiv:2510.05421*, 2025. URL https://arxiv.org/abs/2510.05421v1.

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 *International Conference on Machine Learning (ICML)*, 2026. URL https://openreview.net/forum?id=Oz335dV48X.

Lequn Chen, Zihao Ye, Yongji Wu, Danyang Zhuo, Luis Ceze, and Arvind Krishnamurthy. Punica: Multi-tenant LoRA serving. In *Conference on Machine Learning and Systems (MLSys)*, 2024. URL https://arxiv.org/abs/2310.18547.

Zhiyu Chen, Wenhu Chen, Charese Smiley, Sameena Shah, Iana Borova, Dylan Langdon, Reema Moussa, Matt Beane, Ting-Hao Huang, Bryan Routledge, and William Yang Wang. FinQA: A dataset of numerical reasoning over financial data. In *Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 3697–3711, 2021. URL https://arxiv.org/abs/2109.00122.

Ning Ding, Yulin Chen, Bokai Xu, Yujia Qin, Shengding Hu, Zhiyuan Liu, Maosong Sun, and Bowen Zhou. Enhancing chat language models by scaling high-quality instructional conversations. In *Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 3029–3051, 2023. URL https://arxiv.org/abs/2305.14233.

Dan Hendrycks, Steven Basart, Saurav Kadavath, Mantas Mazeika, Akul Arora, Ethan Guo, Collin Burns, Samir Puranik, Horace He, Dawn Song, and Jacob Steinhardt. Measuring coding challenge competence with APPS. In *Advances in Neural Information Processing Systems (NeurIPS), Datasets and Benchmarks Track*, 2021a. URL https://arxiv.org/abs/2105.09938.

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 *Advances in Neural Information Processing Systems (NeurIPS), Datasets and Benchmarks Track*, 2021b. 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 *ICLR Workshop on Scalable Optimization for Efficient and Adaptive Foundation Models (SCOPE)*, 2025. URL https://arxiv.org/abs/2503.07807v2.

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. In *International Conference on Learning Representations (ICLR)*, 2022. URL https://arxiv.org/abs/2106.09685.

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 *International Conference on Learning Representations (ICLR)*, 2025a. URL https://arxiv.org/abs/2403.07974.

Naman Jain, Jaskirat Singh, Manish Shetty, Tianjun Zhang, Liang Zheng, Koushik Sen, and Ion Stoica. R2E-Gym: Procedural environment generation and hybrid verifiers for scaling open-weights SWE agents. In *Conference on Language Modeling (COLM)*, 2025b. URL https://arxiv.org/abs/2504.07164.

Di Jin, Eileen Pan, Nassim Oufattole, Wei-Hung Weng, Hanyi Fang, and Peter Szolovits. What disease does this patient have? A large-scale open domain question answering dataset from medical exams. *Applied Sciences*, 11(14):6421, 2021. URL https://arxiv.org/abs/2009.13081.

Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *International Conference on Learning Representations (ICLR)*, 2015. URL https://arxiv.org/abs/1412.6980.

Avinash Kumar, Sujay Sanghavi, and Poulami Das. Test-time speculation. *arXiv preprint arXiv:2605.09329*, 2026. URL https://arxiv.org/abs/2605.09329v2.

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 *ACM Symposium on Operating Systems Principles (SOSP)*, 2023. URL https://arxiv.org/abs/2309.06180.

Yaniv Leviathan, Matan Kalman, and Yossi Matias. Fast inference from transformers via speculative decoding. In *International Conference on Machine Learning (ICML)*, 2023. URL https://arxiv.org/abs/2211.17192.

Jia Li, Edward Beeching, Lewis Tunstall, Ben Lipkin, Roman Soletskyi, Shengyi Costa Huang, Kashif Rasul, Longhui Yu, Albert Jiang, Ziju Shen, Zihan Qin, Bin Dong, Li Zhou, Yann Fleureau, Guillaume Lample, and Stanislas Polu. NuminaMath, 2024a. URL https://huggingface.co/datasets/AI-MO/NuminaMath-1.5.

Yuhui Li, Fangyun Wei, Chao Zhang, and Hongyang Zhang. EAGLE: Speculative sampling requires rethinking feature uncertainty. In *International Conference on Machine Learning (ICML)*, 2024b. URL https://arxiv.org/abs/2401.15077.

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 (NeurIPS)*, 2025. URL https://arxiv.org/abs/2503.01840.

Yujia Li, David Choi, Junyoung Chung, Nate Kushman, Julian Schrittwieser, Rémi Leblond, Tom Eccles, James Keeling, Felix Gimeno, Agustin Dal Lago, Thomas Hubert, Peter Choy, Cyprien de Masson d'Autume, Igor Babuschkin, Xinyun Chen, Po-Sen Huang, Johannes Welbl, Sven Gowal, Alexey Cherepanov, James Molloy, Daniel J. Mankowitz, Esme Sutherland Robson, Pushmeet Kohli, Nando de Freitas, Koray Kavukcuoglu, and Oriol Vinyals. Competition-level code generation with AlphaCode. *Science*, 378(6624):1092–1097, 2022. URL https://arxiv.org/abs/2203.07814.

Xiaoxuan Liu, Lanxiang Hu, Peter Bailis, Alvin Cheung, Zhijie Deng, Ion Stoica, and Hao Zhang. Online speculative decoding. In *International Conference on Machine Learning (ICML)*, 2024. URL https://arxiv.org/abs/2310.07177v4.

MiniMax. MiniMax-M2.7, 2026. URL https://huggingface.co/MiniMaxAI/MiniMax-M2.7.

NVIDIA. MiniMax-M2.7-DFlash, 2026. URL https://huggingface.co/nvidia/MiniMax-M2.7-DFlash.

Gabriele Oliaro, Xupeng Miao, Xinhao Cheng, Vineeth Kada, Mengdi Wu, Ruohan Gao, Yingyi Huang, Remi Delacourt, April Yang, Yingcheng Wang, Colin Unger, and Zhihao Jia. FlexLLM: Token-level co-serving of LLM inference and finetuning with SLO guarantees. In *USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2026. URL https://www.usenix.org/conference/nsdi26/presentation/oliaro.

OpenAI. gpt-oss-120b & gpt-oss-20b model card. *arXiv preprint arXiv:2508.10925*, 2025. URL https://arxiv.org/abs/2508.10925.

Jiayi Pan, Xingyao Wang, Graham Neubig, Navdeep Jaitly, Heng Ji, Alane Suhr, and Yizhe Zhang. Training software engineering agents and verifiers with SWE-Gym. In *International Conference on Machine Learning (ICML)*, 2025. URL https://arxiv.org/abs/2412.21139.

Jiyoung Park, Hankyu Jang, Changseok Song, and Wookeun Jung. TIDE: Temporal incremental draft engine for self-improving LLM inference. In *International Conference for High Performance Computing, Networking, Storage and Analysis (SC)*, 2026. URL https://arxiv.org/abs/2602.05145v2.

Predibase. LoRAX: Multi-LoRA inference server, 2023. URL https://github.com/predibase/lorax.

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. In *COLM Workshop on Efficient Reasoning*, 2026a. 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. In *International Conference on Machine Learning (ICML)*, 2026b. URL https://arxiv.org/abs/2603.12617v2.

Ramchalam Kinattinkara Ramakrishnan, Zhaocong Yuan, Shaojie Zhuo, Chen Feng, Yicheng Lin, Chenzheng Su, and Xiaopeng Zhang. OmniDraft: A cross-vocabulary, online adaptive drafter for on-device speculative decoding. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2025. URL https://arxiv.org/abs/2507.02659v3.

Ying Sheng, Shiyi Cao, Dacheng Li, Coleman Hooper, Nicholas Lee, Shuo Yang, Christopher Chou, Banghua Zhu, Lianmin Zheng, Kurt Keutzer, Joseph E. Gonzalez, and Ion Stoica. S-LoRA: Serving thousands of concurrent LoRA adapters. In *Conference on Machine Learning and Systems (MLSys)*, 2024. URL https://arxiv.org/abs/2311.03285.

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

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 *International Conference on Machine Learning (ICML)*, 2020. URL https://arxiv.org/abs/1909.13231.

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 *International Conference on Machine Learning (ICML)*, pp. 57503–57522, 2025. URL https://arxiv.org/abs/2407.04620.

Junxiong Wang, Fengxiang Bie, Jisen Li, Zelei Shao, Qingyang Wu, Yinghui Liu, Yubo Wang, Avner May, Ben Athiwaratkun, Yineng Zhang, Shuaiwen Leon Song, Zhongzhu Zhou, Chenfeng Xu, and Xiaoxia Wu. When RL meets adaptive speculative training: A unified training-serving system. In *International Conference on Machine Learning (ICML)*, 2026a. URL https://openreview.net/forum?id=1U0lhpW0VL.

Yanbo Wang, Jinhua Hao, Yuze Shi, Kun Yuan, and Ming Sun. No time like the present: Agentic test-time training for LLM agents. *arXiv preprint arXiv:2607.03441*, 2026b. URL https://arxiv.org/abs/2607.03441v1.

Yuxiang Wei, Zhe Wang, Jiawei Liu, Yifeng Ding, and Lingming Zhang. Magicoder: Empowering code generation with OSS-Instruct. In *International Conference on Machine Learning (ICML)*, 2024. URL https://arxiv.org/abs/2312.02120.

Bingyang Wu, Ruidong Zhu, Zili Zhang, Peng Sun, Xuanzhe Liu, and Xin Jin. dLoRA: Dynamically orchestrating requests and adapters for LoRA LLM serving. In *USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, 2024. URL https://www.usenix.org/conference/osdi24/presentation/wu-bingyang.

Heming Xia, Zhe Yang, Qingxiu Dong, Peiyi Wang, Yongqi Li, Tao Ge, Tianyu Liu, Wenjie Li, and Zhifang Sui. Unlocking efficiency in large language model inference: A comprehensive survey of speculative decoding. In *Findings of the Association for Computational Linguistics: ACL 2024*, 2024. URL https://arxiv.org/abs/2401.07851.

Zhangchen Xu, Yang Liu, Yueqin Yin, Mingyuan Zhou, and Radha Poovendran. KodCode: A diverse, challenging, and verifiable synthetic dataset for coding. In *Findings of the Association for Computational Linguistics: ACL 2025*, pp. 6980–7008, 2025. URL https://arxiv.org/abs/2503.02951.

An Yang et al. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025. URL https://arxiv.org/abs/2505.09388.

Jian Yang, Zhizhuo Kou, Yao Tian, Hao Zhang, Han Chen, Sirui Han, and Yike Guo. RW-TTT: Batched serving for request-owned test-time training state. *arXiv preprint arXiv:2605.28053*, 2026. URL https://arxiv.org/abs/2605.28053v2.

Gyeong-In Yu, Joo Seong Jeong, Geon-Woo Kim, Soojeong Kim, and Byung-Gon Chun. Orca: A distributed serving system for transformer-based generative models. In *USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, 2022. URL https://www.usenix.org/conference/osdi22/presentation/yu.

Qiying Yu, Zheng Zhang, Ruofei Zhu, Yufeng Yuan, Xiaochen Zuo, Yu Yue, Weinan Dai, Tiantian Fan, Gaohong Liu, Juncai Liu, Lingjun Liu, Xin Liu, Haibin Lin, Zhiqi Lin, Bole Ma, Guangming Sheng, Yuxuan Tong, Chi Zhang, Mofan Zhang, Ru Zhang, Wang Zhang, Hang Zhu, Jinhua Zhu, Jiaze Chen, Jiangjie Chen, Chengyi Wang, Hongli Yu, Yuxuan Song, Xiangpeng Wei, Hao Zhou, Jingjing Liu, Wei-Ying Ma, Ya-Qin Zhang, Lin Yan, Yonghui Wu, and Mingxuan Wang. DAPO: An open-source LLM reinforcement learning system at scale. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2025. URL https://arxiv.org/abs/2503.14476.

Shuyu Zhang, Lingfeng Pan, Qicheng Wang, Yaqi Shi, Yueyang Tan, Ruyu Yan, Jiaqi Chen, Lixing Du, and Lu Wang. EvoSpec: Evolving speculative decoding via real-time vocabulary and parameter adaptation. *arXiv preprint arXiv:2605.27390*, 2026. URL https://arxiv.org/abs/2605.27390v3.

Yifan Zhang. StackMathQA: A curated collection of 2 million mathematical questions and answers sourced from Stack Exchange. Technical report, Math-AI, 2024. URL https://stackmathqa.github.io/StackMathQA.pdf.

Yuchen Zhang, Hanyue Du, Chun Cao, and Jingwei Xu. Loquetier: A virtualized multi-LoRA framework for unified LLM fine-tuning and serving. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2025. URL https://arxiv.org/abs/2511.00101.

Wenting Zhao, Xiang Ren, Jack Hessel, Claire Cardie, Yejin Choi, and Yuntian Deng. WildChat: 1M ChatGPT interaction logs in the wild. In *International Conference on Learning Representations (ICLR)*, 2024. URL https://arxiv.org/abs/2405.01470.

Lianmin Zheng, Liangsheng Yin, Zhiqiang Xie, Chuyue Sun, Jeff Huang, Cody Hao Yu, Shiyi Cao, Christos Kozyrakis, Ion Stoica, Joseph E. Gonzalez, Clark Barrett, and Ying Sheng. SGLang: Efficient execution of structured language model programs. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2024. URL https://arxiv.org/abs/2312.07104.

Yongchao Zhou, Kaifeng Lyu, Ankit Singh Rawat, Aditya Krishna Menon, Afshin Rostamizadeh, Sanjiv Kumar, Jean-François Kagy, and Rishabh Agarwal. DistillSpec: Improving speculative decoding via knowledge distillation. In *International Conference on Learning Representations (ICLR)*, 2024. URL https://arxiv.org/abs/2310.08461.

A Implementation and Setup Details

SERVAL is implemented in SGLang; the trainer is a separate process built from the same model code and connected to the server over ZMQ. The server's messages to the trainer are BIND (a request is assigned to a slot), PREFIX (the request's prefix context features), COLLECT (the feedback of one step), and FINISH (the request ended); the trainer builds its copy of the draft's key–value pool from the context features they carry and checks the prefix length of every message against it. Under tensor parallelism the adapter pools are sharded across ranks like the projections they attach to, and each trainer rank trains its shard.

Traffic. The finance+math stream draws on FinQA and mathematics, and the code+finance+medical stream on code_contests, FinQA, and MedQA. Each run serves 4,000 requests per domain in a seeded interleaved order. A domain's 4,000 prompts are the first 4,000 of a seeded sample of a deduplicated pool from the training splits of FinQA (Chen et al., 2021), the four-option USMLE questions of MedQA (Jin et al., 2021), code_contests (Li et al., 2022), and, for mathematics, levels 3–5 of MATH (Hendrycks et al., 2021b), the cn_k12 part of NuminaMath-1.5 (Li et al., 2024a), StackMathQA (Zhang, 2024), and DAPO-Math-17k (Yu et al., 2025) in equal parts. The next 1,000 prompts of the same sample are held out, and Section 4.3 serves 200 of them per domain. Arrivals are Poisson at 0.85 of DFlash's capacity, the rate at which DFlash completes requests at concurrency 32 over the first 1,000 requests of the stream (rates in Table 6). The runs of Figure 1 replay the arrival schedules of the Table 1 runs, whose DFlash results are the reference of the figure. The fixed-concurrency benchmarks serve every problem once per run: AIME 2022–25 (119 problems), the 80 hard problems of LiveCodeBench release 6, 100 code_contests problems rated 2200 or higher (88 distinct statements), and the FinQA test split (1,147 financial-report questions).

General adapter. Its 11,200 requests are 6,000 of general chat from UltraChat-200k (Ding et al., 2023) and WildChat-1M (Zhao et al., 2024); 2,400 of mathematics from MATH, NuminaMath-1.5 cn_k12, StackMathQA, and DAPO-Math-17k; 2,000 of code from APPS (Hendrycks et al., 2021a), KodCode (Xu et al., 2025), and Magicoder-Evol-Instruct (Wei et al., 2024); and 800 agentic tasks from SWE-Gym (Pan et al., 2025) and R2E-Gym (Jain et al., 2025b). They were served once in a seeded order at concurrency 32 with outputs capped at 8,192 tokens, and one shared adapter was trained on them with the settings of Table 6 except the 128-message accumulation. Qwen3-8B decoded greedily in that run. The prompts overlapping the code and mathematics pools of the streams were removed from those pools; the corpus has no FinQA or medical prompts, none of the LiveCodeBench, code_contests, or FinQA problems of the fixed-concurrency benchmarks, and 3 of the 119 AIME problems (through DAPO-Math-17k). The general adapters are included with the code.

Models. Qwen3-8B (bf16), gpt-oss-120b (MXFP4 experts), and MiniMax-M2.7 (FP8 weights) with the DFlash drafts (Chen et al., 2026) released for Qwen3-8B and gpt-oss-120b by the DFlash authors and for MiniMax-M2.7 by NVIDIA (NVIDIA, 2026), served by SGLang at upstream commit 694cea8 with our changes; the checkpoint revisions are listed with the code.

Training settings. The loss weights w_j of Equation 1 are 1 at accepted positions, 1.5 at the first rejected position, and $0.9 \cdot 0.7^{k-1}$ at the k-th position past it ($k \geq 1$), normalized to sum to one, so that a fully accepted block weights its positions equally. OSD-analog averages each step's gradient over the messages of the 32 requests completed since the previous step; its loss puts a block's whole weight on the first rejected position, so a fully accepted block adds no loss. The full-parameter variant of Section 4.3 is trained with the standard Adam. The LoRA and full-parameter variants are trained in the serving process on the finance and the mathematics requests of the finance+math stream, each domain served alone at concurrency 32, without the 128-message accumulation. Each variant's learning rate is the best of four on the first 1,500 finance requests: 6×10^{-4} for the adapter and 6×10^{-5} for the full parameters.

B Answer Accuracy Under Draft Adaptation

The verification rule leaves the output distribution unchanged; consistent with this, the one-standard-error intervals of the methods in Table 7 overlap.

Table 6: Settings in the experiments. Where a row gives three values, they are for Qwen3-8B / gpt-oss-120b / MiniMax-M2.7.

Setting	Value
Draft tokens per step	$K = 15$ / 9 / 7, in DFlash blocks of 16 / 10 / 8 positions that start with the last generated token
Tensor parallelism	2 / 4 / 4 for the server, matched by the trainer on as many GPUs
Concurrency; adapter slots	32 concurrent requests; 48 slots per server
Adapter	LoRA of rank 32, $\alpha = 32$, on the query and output projections of every draft layer; each tensor-parallel rank trains its own factors for its shard of a projection, 3.9M parameters in all on Qwen3-8B
Starting state	the general adapter (Section 4.1); in Figure 1 and Table 4b, DFlash (the zero adapter, Section 3.2)
Feedback	one message per request at every decode step
Optimizer	Adam with the first moment reset to the gradient where their signs differ, $\beta = (0.9, 0.999)$, no weight decay, learning rate 6×10^{-4}, one step per $N = 128$ messages of a slot
Loss	forward KL to the target's next-token distribution at temperature 1 (before the request's temperature, top-p, and top-k), positions weighted 1 (accepted), 1.5 (first rejected), $0.9 \cdot 0.7^{k-1}$ (k-th past it), normalized to sum to one
Versions	one sent per optimizer step of a slot (every $N = 128$ messages); loaded at the first step boundary after arrival
Request sampling	temperature 0.6 / 0.7 / 1.0, top-p 0.95, top-k 20 / 20 / 40, at most 24,576 output tokens
Arrivals (online serving)	Poisson at 0.85 of DFlash's capacity: 0.995 / 4.044 / 0.661 requests/s on finance+math, 0.501 / 1.492 / 0.284 on code+finance+medical
OSD-analog	OSD's rules: loss at the first rejected position only; one optimizer step per 32 completed requests

Table 7: Answer accuracy on AIME 2022–25 under each method at fixed concurrency, over the 357 attempts of three runs. Accuracy is in %, with $\pm$ one binomial standard error. The last column is the share of attempts that hit the 24,576-token cap, counted as wrong, from lowest to highest over the methods.

Model	DFlash	general-frozen	TTS-style	OSD-analog	SERVAL	capped
Qwen3-8B	65.3 ± 2.5	65.0 ± 2.5	68.9 ± 2.4	66.1 ± 2.5	67.5 ± 2.5	21–26 %
gpt-oss-120b	79.0 ± 2.2	79.3 ± 2.1	79.3 ± 2.1	78.7 ± 2.2	77.9 ± 2.2	$\leq$1 %
MiniMax-M2.7	74.2 ± 2.3	73.7 ± 2.3	75.9 ± 2.3	76.2 ± 2.3	77.3 ± 2.2	23–27 %

C Cost Breakdown

Table 8: Serving-side cost of SERVAL per decode step by component. GPU time is measured in one profiler window, and scheduler-thread time over the whole run.

Component (Qwen3-8B, finance+math)	GPU time		Scheduler thread	
	ms/step	% of step	ms/step	% of step
LoRA computation (5 draft layers, q and o)	0.096	1.2	–	–
Hidden-state transfer (7 device-to-host copies)	0.130	1.6	0.148	1.8
Adapter load (0.10 copies per step)	0.002	0.0	0.012	0.2
Total	0.228	2.8	0.160	2.0

Profile. The profile of Table 8 is one 15 s CUDA-activity window of the serving process (1,422 decode steps, a running batch of 13.9 requests on average) after 677 completed requests of a finance+math run on Qwen3-8B. GPU time is attributed to the LoRA kernels by name and to the hidden-state transfer and the adapter load by the profiler ranges that enclose their launches, and divided by the unprofiled steady-state step time of the same run (8.07 ms); the scheduler-thread columns are the same paths' time from the server's own timers over the whole run (94,609 steps).

Table 9 gives the costs per method behind Sections 4.3 and 4.4.

Table 9: Costs per method: every method with a trainer on the finance+math stream, SERVAL-shared and SERVAL on the code+finance+medical stream. Step overhead is the scheduler-thread time per decode step for sending feedback and loading versions. Transfer is the size of the hidden states one server GPU sends to the trainer per message. Versions are the adapter bytes one server GPU receives from the trainer per second. The update lag is the number of decode steps from the newest step the trainer had received when it sent a version to the step that loads it. OSD-analog sends a version once per 32 completed requests.

Model	Method	Step overhead ms (% of step)	Transfer KB/message	Versions MB/s	Update lag (steps) p50 / p90
finance+math					
Qwen3-8B	TTS-style	0.160 (1.8)	176	32.6	10 / 63
	OSD-analog	0.140 (1.6)	173	0.1	217 / 833
	SERVAL-shared	0.150 (1.8)	175	25.9	6 / 31
	SERVAL	0.152 (1.8)	176	28.3	7 / 41
gpt-oss-120b	TTS-style	0.902 (4.7)	104	32.0	295 / 1,860
	OSD-analog	1.832 (8.7)	93	0.5	1,689 / 2,661
	SERVAL-shared	0.183 (1.1)	94	34.1	3 / 15
	SERVAL	0.186 (1.1)	94	36.0	3 / 16
MiniMax-M2.7	TTS-style	0.175 (0.6)	75	14.8	2 / 3
	OSD-analog	0.153 (0.5)	73	0.1	30 / 175
	SERVAL-shared	0.177 (0.6)	73	13.2	2 / 3
	SERVAL	0.174 (0.6)	73	14.0	2 / 3
code+finance+medical					
Qwen3-8B	SERVAL-shared	0.153 (2.1)	168	17.7	8 / 97
	SERVAL	0.157 (2.1)	168	19.8	8 / 110
gpt-oss-120b	SERVAL-shared	0.173 (1.3)	83	25.8	4 / 21
	SERVAL	0.189 (1.4)	83	27.8	4 / 26
MiniMax-M2.7	SERVAL-shared	0.170 (0.6)	68	11.3	2 / 3
	SERVAL	0.171 (0.6)	68	12.0	2 / 3

D Additional Results

Tables 10 and 11 give the run-to-run ranges of the results in Tables 1 and 2, Tables 12 and 13 the per-domain acceptance length of all methods on the code+finance+medical and finance+math streams, and Table 14 the LoRA versus full-parameter comparison of Section 4.3 over the whole stream.

Table 10: Range over runs of the stream cells of Table 1.

Stream	AL range / TPOT range (ms)					
	DFlash	general-frozen	TTS-style	OSD-analog	SERVAL-shared	SERVAL
Qwen3-8B						
finance+math	0.01 / 0.12	0.01 / 0.01	0.01 / 0.01	0.01 / 0.03	0.00 / 0.03	0.01 / 0.01
code+finance+medical	0.01 / 0.02	0.00 / 0.01	0.00 / 0.01	0.00 / 0.02	0.01 / 0.01	0.00 / 0.01
gpt-oss-120b						
finance+math	0.06 / 0.12	0.02 / 0.03	0.02 / 0.26	0.01 / 1.12	0.03 / 0.01	0.03 / 0.02
code+finance+medical	0.01 / 0.02	0.00 / 0.01	0.00 / 0.00	0.01 / 0.08	0.01 / 0.01	0.00 / 0.02
MiniMax-M2.7						
finance+math	0.01 / 0.09	0.00 / 0.05	0.00 / 0.06	0.01 / 0.09	0.00 / 0.07	0.01 / 0.06
code+finance+medical	0.00 / 0.04	0.00 / 0.10	0.00 / 0.05	0.00 / 0.04	0.00 / 0.11	0.01 / 0.06

Table 11: Range over runs of the fixed-concurrency cells of Table 2.

Benchmark	AL range / TPOT range (ms)				
	DFlash	general-frozen	TTS-style	OSD-analog	SERVAL
Qwen3-8B					
AIME 2022–25	0.02 / 0.12	0.04 / 0.19	0.04 / 0.22	0.02 / 0.04	0.02 / 0.15
LCB-hard	0.03 / 0.04	0.07 / 0.20	0.00 / 0.22	0.05 / 0.20	0.04 / 0.13
code_contests	0.01 / 0.22	0.04 / 0.14	0.03 / 0.27	0.01 / 0.36	0.04 / 0.17
FinQA test	0.02 / 0.10	0.00 / 0.09	0.02 / 0.10	0.02 / 0.10	0.08 / 0.10
gpt-oss-120b					
AIME 2022–25	0.12 / 0.13	0.11 / 0.05	0.13 / 0.04	0.06 / 0.04	0.05 / 0.01
LCB-hard	0.02 / 0.12	0.02 / 0.04	0.04 / 0.03	0.02 / 0.07	0.03 / 0.04
code_contests	0.07 / 0.28	0.05 / 0.11	0.01 / 0.03	0.02 / 0.01	0.02 / 0.07
FinQA test	0.20 / 0.07	0.24 / 0.00	0.17 / 0.01	0.06 / 0.01	0.02 / 0.04
MiniMax-M2.7					
AIME 2022–25	0.03 / 0.18	0.04 / 0.30	0.02 / 0.15	0.07 / 0.44	0.03 / 0.18
LCB-hard	0.01 / 0.14	0.03 / 0.26	0.03 / 0.35	0.02 / 0.22	0.01 / 0.36
code_contests	0.00 / 0.36	0.03 / 0.43	0.02 / 0.24	0.01 / 0.16	0.03 / 0.31
FinQA test	0.05 / 0.06	0.04 / 0.02	0.05 / 0.08	0.07 / 0.10	0.03 / 0.09

Table 12: Per-domain acceptance length of every method on the code+finance+medical stream, mean over three runs.

Model	Domain	DFlash	general-frozen	TTS-style	OSD-analog	SERVAL-shared	SERVAL
Qwen3-8B	code	2.42	3.71	3.82	3.87	4.04	4.07
	finance	3.81	4.28	4.29	4.46	4.74	4.93
	medical	2.78	3.01	3.03	3.09	3.19	3.25
gpt-oss-120b	code	2.78	3.31	3.36	3.37	3.45	3.45
	finance	4.50	4.46	4.49	4.68	4.84	4.95
	medical	2.94	2.91	2.92	2.97	3.05	3.05
MiniMax-M2.7	code	2.38	2.60	2.62	2.62	2.64	2.64
	finance	2.96	3.43	3.42	3.48	3.53	3.58
	medical	2.31	2.57	2.57	2.58	2.61	2.63

Table 13: Per-domain acceptance length of every method on the finance+math stream, mean over three runs.

Model	Domain	DFlash	general-frozen	TTS-style	OSD-analog	SERVAL-shared	SERVAL
Qwen3-8B	finance	3.81	4.27	4.30	4.49	4.83	4.94
	math	3.28	4.47	4.52	4.55	4.72	4.74
gpt-oss-120b	finance	4.53	4.47	4.47	4.71	4.91	4.96
	math	4.08	4.33	4.33	4.33	4.42	4.42
MiniMax-M2.7	finance	2.96	3.41	3.43	3.45	3.58	3.59
	math	2.89	3.24	3.24	3.25	3.27	3.27

Table 14: Acceptance length of LoRA and full-parameter training on Qwen3-8B over all 4,000 requests of each single-domain stream, mean over three runs. Params are the trained parameters over both GPUs; the LoRA share is the LoRA gain over DFlash divided by the full-parameter gain.

Method	params	finance	math
DFlash	–	3.81	3.27
LoRA (r32, q/o)	3.9M	4.69	4.72
full (K/V frozen)	922.8M	5.00	4.88
LoRA share of the full gain		74 %	90 %
full q/o	167.8M	4.77	–
full incl. K/V	964.7M	4.97	–