RECIPROCAL GUIDANCE: ORCHESTRATING DRAFT AND VERIFY BUDGETS FOR ADVANCING THE DIFFUSION–AR SELF-SPECULATION FRONTIER

Anonymous authors
Paper under double-blind review

ABSTRACT

Diffusion drafting with autoregressive (AR) verification has emerged as a promising paradigm for efficient speculative decoding. Recent self-speculation models, represented by Nemotron-Labs-Diffusion, further simplify the speculative pipeline by unifying drafting and verification within a shared backbone, while enabling longer acceptance lengths. However, the Pareto frontier between aggregate and per-request throughput remains underexplored. At low concurrency, sequential draft–verify execution requires two model forward passes per round, limiting the effective tokens per forward (TPF). By contrast, at high concurrency, longer drafts incur increasingly expensive computation, forcing individual requests to operate under constrained speculation budgets and preventing full exploitation of the full-backbone drafter. Our key observation indicates that drafting and verification exhibit reciprocal predictability. Draft logits can anticipate likely verification mismatches, while recent verification outcomes predict future drafting utility and suitable block sizes. Building on this observation, we introduce **Reciprocal Guidance** (**RecGuide**), a runtime draft–verify orchestration framework that adapts speculative decoding to varying serving loads. RecGuide exploits spare compute capacity through verification-overlapped drafting at low concurrency, while dynamically allocating request-specific draft block sizes as the workload becomes increasingly compute-intensive. Experiments across a wide range of concurrency levels demonstrate consistent throughput improvements over vanilla self-speculation, achieving up to 1.8× speedup.

1 INTRODUCTION

Speculative decoding (Chen et al., 2023; Leviathan et al., 2023) is widely used to accelerate large language model (LLM) inference. It employs a lightweight draft model to autoregressively propose multiple tokens for parallel verification and potential acceptance in a single forward pass. Recently, inspired by diffusion language models (dLLMs) (Nie et al., 2026; Cheng et al., 2026a), diffusion-based speculative decoding (Chen et al., 2026; Cheng et al., 2026b) generates draft tokens in parallel with bidirectional attention before autoregressive verification. By reducing sequential drafting overhead while maintaining high acceptance rates, this paradigm has attracted growing attention.

Self-speculation methods such as LinearSS in Nemotron-Labs-Diffusion (Fu et al., 2026; Liu et al., 2026b) further share a single backbone between diffusion drafting and autoregressive verification by switching between bidirectional and causal attention. By reusing the full target backbone as the drafter (Zhang et al., 2024), this design simplifies the speculative pipeline (Zhang et al., 2026b) while offering stronger task generalization and longer-acceptance potential than lightweight drafters.

However, concurrency varies dynamically in real-world serving workloads (Chen et al., 2024), while current self-speculation typically relies on fixed draft lengths (referred to as block sizes hereafter) tailored to different concurrency regimes. Such coarse-grained allocation is insufficient (Sadhukhan et al., 2025). **At low concurrency**, sequential draft-then-verify execution requires two expensive forwards per round, reducing effective tokens per forward (TPF) to roughly half the acceptance length and limiting the benefit of the full-backbone drafter. **At high concurrency**, by contrast, large blocks become costly as GPU execution turns compute-bound and forward latency rises sharply. Yet

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

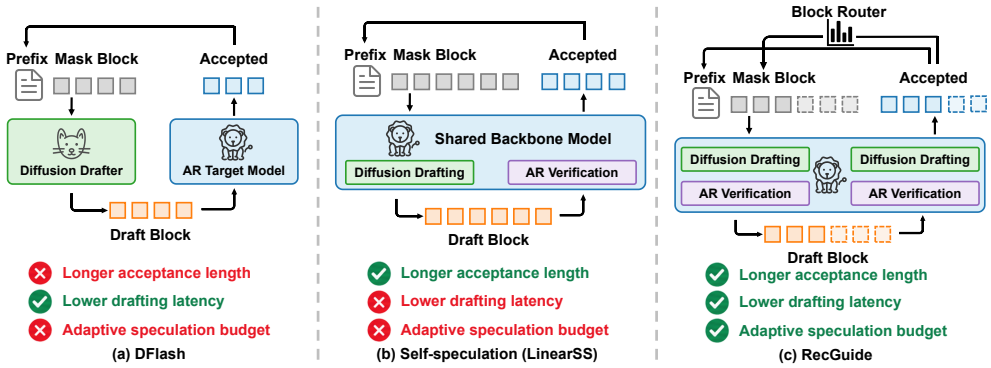


Figure 1: Comparison between RecGuide and existing diffusion-based speculative decoding.

small blocks underutilize self-speculation's potential for longer acceptance lengths compared with conventional speculative decoding paradigms. This motivates the following question:

Can we orchestrate draft and verify budgets to advance the aggregate and per-request throughput frontier, thereby making self-speculation a competitive paradigm for speculative decoding?

To address this challenge, we propose **Reciprocal Guidance** (**RecGuide**), grounded in our core observation that drafting and verification on the shared backbone are not independent, but exhibit strong reciprocal predictability. On the one hand, draft logits provide informative signals for anticipating likely mismatch positions in subsequent AR verification, as well as a likely correction token. On the other hand, recent verification outcomes reflect the local drafting difficulty of each request and provide an effective signal for selecting an appropriate block size in the next round.

Building on these observations, RecGuide orchestrates drafting and verification with complementary strategies across varying concurrency levels, as illustrated in Figure 1. At low concurrency, RecGuide predicts likely verification outcomes from draft probabilities and co-batches candidate next-round drafts with AR verification. Shared weights make this overlap natural for self-speculation, avoiding model switching or separate devices while exploiting spare compute in the memory-bound regime (Liu et al., 2025; Kumar et al., 2026). As workload increases, RecGuide instead uses verification feedback to estimate drafting utility and assign request-specific block sizes. Together with bucketed serving based on block size, this strategy balances drafting capability against available GPU compute.

We evaluate RecGuide under diverse serving regimes. Our method consistently improves throughput over vanilla self-speculation, enhancing speculative efficiency under light workloads and serving efficiency under heavy workloads. Our main contributions can be summarized as follows:

- We characterize the efficiency bottlenecks and pipeline challenges of self-speculation, and uncover reciprocal predictability between drafting and verification.
- We propose RecGuide, which orchestrates draft and verify budgets across varying serving loads through predictive draft overlap and adaptive block sizing.
- Extensive experiments demonstrate that RecGuide achieves consistent throughput improvements over vanilla self-speculation, with up to a 1.8× speedup.

2 Related Work

Drafting Paradigms for Speculative Decoding. Speculative decoding (Chen et al., 2023; Leviathan et al., 2023; Cai et al., 2024; Zhong et al., 2024) is widely used to accelerate large language model inference. It uses a lightweight drafter to propose multiple future tokens, which are verified in parallel by the target model, enabling multiple tokens to be accepted per target-model forward without altering the generation distribution. The EAGLE series (Li et al., 2024b;a; 2026b) improves drafting efficiency by exploiting target model representations. Nevertheless, autoregressive drafting still incurs latency that grows with draft length. Inspired by diffusion (Ho et al., 2020; Yan et al., 2026; 2025) and diffusion language models (Nie et al., 2026; Ye et al., 2025a; Bie et al., 2025; Wei et al., 2026), recent approaches (Chen et al., 2026; Huang et al., 2026; Cheng et al., 2026b) employ diffusion drafters

with bidirectional attention to generate a draft block in a single forward pass, reducing sequential drafting overhead while achieving higher acceptance rates. Meanwhile, self-speculation methods (Fu et al., 2026; Liu et al., 2026b; Zhang et al., 2026b) unify diffusion drafting and autoregressive verification within a shared model, offering stronger drafting capability and longer acceptance lengths at higher drafting cost. Our work focuses on the runtime efficiency of this paradigm.

Runtime Orchestration for Speculative Decoding. Beyond drafter design, recent work has explored parallel execution and adaptive runtime strategies (Zhang et al., 2026c; Lin et al., 2026), breaking the sequential draft-then-verify dependency and accommodating diverse serving conditions. PEARL (Liu et al., 2025) overlaps drafting and verification and adapts draft length to mitigate mutual waiting between a separate drafter and verifier. SSD (Kumar et al., 2026) further predicts likely verification outcomes before completion and asynchronously prepares future speculation. In contrast, our proposed RecGuide targets self-speculation where both stages share the same backbone, allowing future drafts to be co-batched with ongoing verification without model switching or additional devices. This turns the efficiency bottleneck into a limited-compute allocation problem. At the drafting stage, prior work (Sadhukhan et al., 2025; Ning et al., 2026; Zhang et al., 2026a) dynamically adapts the amount and configuration of speculation to improve throughput. At the verification stage, another line of work (Liu et al., 2026c; Cheng et al., 2026b; Liu et al., 2026a) selectively allocates verification effort based on confidence signals or hardware-aware runtime models. These two stages, however, are typically optimized separately. RecGuide instead treats them as mutually informative and coupled, using bidirectional signals to jointly orchestrate their budgets across concurrency regimes.

3 Preliminary and Bottleneck

3.1 Self-Speculation with a Shared Backbone

Unlike conventional speculative decoding with a separate lightweight drafter, shared-backbone self-speculation employs a single backbone to support both diffusion drafting and autoregressive verification, as illustrated in Figure 1(b). Given a generated prefix, the model first predicts multiple future tokens in parallel under diffusion mode and then switches to autoregressive mode for causal verification, accepting the draft prefix up to the first mismatch.

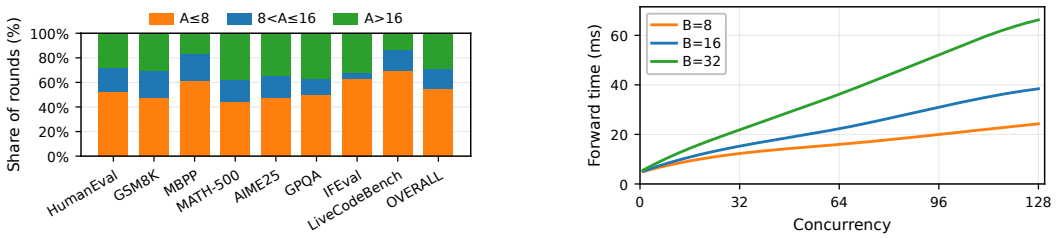
The two modes can be switched simply by changing the attention pattern. Let the current prefix length be P and the draft block size be B. We denote prefix positions by $\mathcal{P} = \{1, \dots, P\}$ and draft positions by $\mathcal{D} = \{P+1, \dots, P+B\}$. Under diffusion drafting, the prefix remains causal, while tokens in $\mathcal{D}$ attend to both the prefix and the entire draft region through bidirectional attention. Autoregressive verification instead enforces strict causal attention, where position i can only attend to positions $j \le i$. The corresponding attention masks are as follows:

$$M_{ij}^{\text{Diff}} = \begin{cases} 0, & i \in \mathcal{P},\ j \in \mathcal{P},\ j \le i, \\ 0, & i \in \mathcal{D},\ j \in \mathcal{P} \cup \mathcal{D}, \\ -\infty, & \text{otherwise}, \end{cases} \qquad M_{ij}^{\text{AR}} = \begin{cases} 0, & j \le i, \\ -\infty, & j > i. \end{cases} \tag{1}$$

This switching allows the full backbone itself to serve as the drafter without deploying a separate draft model. However, sharing the backbone also directly couples drafting and verification at runtime, giving rise to efficiency characteristics distinct from conventional speculative decoding. We next examine the resulting bottlenecks under different serving conditions.

3.2 Bottlenecks across Serving Loads

To characterize the efficiency of self-speculation across serving loads, we profile Nemotron-Labs-Diffusion 8B in terms of acceptance behavior and forward cost. As shown in Figure 2(a), with block size 32, we report the fraction of rounds whose acceptance length A falls into $A \le 8$, $8 < A \le 16$, and $A > 16$. Nearly 30% of the rounds accept more than 16 tokens, revealing strong potential for long drafts, while over half fail within the first 8 positions, indicating substantial variation in useful drafting length. We further examine forward time under fixed block sizes of 8, 16, and 32 as concurrency increases on a single B200 GPU, as illustrated in Figure 2(b). Forward time grows with concurrency, with larger block sizes incurring increasingly higher latency under heavier workloads.



(a) (b)

Figure 2: Acceptance variability and block-dependent forward cost of self-speculation. (a) Acceptance length distribution with $B = 32$. (b) Forward time across concurrency levels for $B \in \{8, 16, 32\}$.

Low-concurrency Inefficiency. At low concurrency, inefficiency comes from both pipeline serialization and wasted drafting. Sequential drafting and verification cause mutual waiting, often negligible with lightweight drafters but costly when both stages invoke the full backbone. Moreover, Figure 2(a) shows that large blocks can waste substantial drafting computation when mismatches occur early. Fortunately, execution remains largely memory-bound at low concurrency, leaving spare compute that can be used to overlap drafting with verification and proactively correct likely mismatches.

High-concurrency Inefficiency. As concurrency increases, Figure 2(b) shows that forward time becomes increasingly sensitive to block size, favoring smaller blocks under heavier workloads. However, uniformly using a small block remains suboptimal, since it sacrifices the long-acceptance potential observed in Figure 2(a). Since GPU latency typically grows sublinearly with the number of processed tokens, additional drafting remains beneficial when the acceptance gain outweighs the corresponding latency overhead. This motivates request-specific block-size selection rather than a fixed global budget. The resulting divergence in per-request computation lengths will be handled through bucketed serving, as discussed in Section 5.3.

4 Reciprocal Predictability between Drafting and Verification

The preceding bottlenecks call for adaptive runtime orchestration. We next establish **reciprocal predictability** between drafting and verification as the key signal for such adaptation.

4.1 Draft Distributions Predict Verification Failures

We first examine whether the draft distribution contains predictive cues about subsequent verification. Under greedy decoding, we define the confidence of the drafted token at position i as $C_i = P_{1\text{st}}(i)$. Figure 3 compares the cumulative distribution functions (CDFs) of C_i at verifier-passed and first-rejected positions across math, code, and QA benchmarks. Passed tokens are heavily concentrated at high confidence values, whereas first-rejected positions exhibit a substantially broader distribution shifted toward lower confidence. This clear separation reveals that draft confidence can anticipate likely verification failures before the verifier is executed.

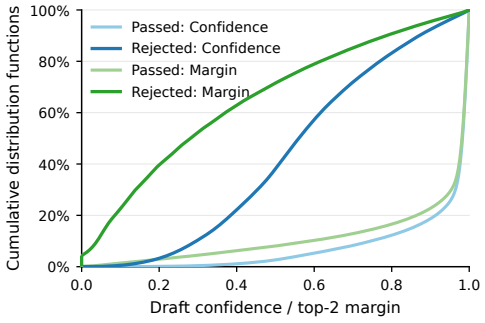


Figure 3: Cumulative distribution functions (CDFs) of draft confidence and top-2 probability margin during drafting rounds.

We further consider the probability margin between the two most likely draft candidates, $M_i = P_{1\text{st}}(i) - P_{2\text{nd}}(i)$. Its CDF exhibits a similar separation between passed and first-rejected positions. A small margin not only reflects uncertainty in the submitted draft token, but also shows that the second-ranked candidate remains competitive. Although this candidate is not necessarily correct, it offers a natural alternative at likely mismatch positions. These observations imply that likely verification failures can be anticipated and prepared for before causal verification completes (Appendix A.2).

4.2 Verification History Predicts Future Drafting Utility

Conversely, we examine whether completed verification is predictive of how far the next round should draft. Without introducing a specific utility metric, we analyze the temporal correlation of acceptance lengths across adjacent rounds, as shown in Table 1. To expose the model's full acceptance potential, the block size is fixed at 32, and the acceptance length at round t is denoted by A_t. We divide A_t into three intervals: short acceptance with $A_t \leq 8$, medium acceptance with $8 < A_t \leq 16$, and long acceptance with $A_t > 16$. These intervals align with the candidate block sizes 8, 16, and 32. Table 1 reports the conditional distribution of A_{t+1} over the three intervals. Both short and long acceptance exhibit clear temporal persistence, as the next round is most likely to remain in the same interval. This pattern suggests that local generation difficulty tends to persist across adjacent rounds, making verification history a useful prior for future drafting budgets.

Table 1: Transition probabilities of acceptance intervals between adjacent decoding rounds.

$A_t \backslash A_{t+1}$	$[1, 8]$	$(8, 16]$	$(16, 32]$
$[1, 8]$	**70.30%**	18.11%	11.60%
$(8, 16]$	58.39%	**22.42%**	19.19%
$(16, 32]$	20.83%	12.51%	**66.66%**

This persistence, however, is not static. Table 1 also shows frequent transitions between adjacent rounds. In particular, only 22.42% of medium acceptance rounds remain in the same interval, primarily because short acceptance rounds dominate all decoding rounds. Thus, each response exhibits both local persistence and regime transitions. A fixed block size therefore cannot consistently match the local drafting utility, calling for adaptive block size allocation under limited compute.

5 RecGuide: Runtime Draft-Verify Orchestration

RecGuide converts reciprocal predictability into runtime decisions, as illustrated in Figure 4. Under light workloads, spare compute prepares likely future drafts alongside verification. Once compute becomes limiting, verification history instead guides each request's drafting budget.

5.1 Predictive Draft-Verify Overlap

As discussed before, the probability margin $M_i = P_{1\text{st}}(i) - P_{2\text{nd}}(i)$ reflects draft uncertainty and the competitiveness of the second-ranked token. RecGuide marks positions with $M_i < \delta$ as potential mismatches and forms a correction branch by replacing the draft token with its second candidate.

Because later positions in a parallel draft are generally less reliable and less valuable for correcting the first mismatch, RecGuide retains only the earliest two risky positions. When at most two such positions exist, it additionally prepares a continuation branch that assumes full acceptance of the current block. Together with the verifier, a round therefore contains at most four logical rows, bounding the extra full-backbone computation. More details are in Appendix A.1.

These branches prepare drafts for the next round in parallel with current verification. A correction branch is reused only if the verifier rejects at the predicted position and produces the same replacement token, whereas retaining the continuation branch requires full acceptance of the current block and a match between its first token and the verifier bonus token. Unmatched branches are discarded. A confirmed draft can immediately repeat the procedure in the next round, enabling recursive overlap.

5.2 Verification Guided Adaptive Drafting

At higher concurrency, costly prospective branches shift the bottleneck to balancing computation and acceptance length. RecGuide uses completed verification history to select each request's next block size $B_t \in \{8, 16, 32\}$. Let e_t average verifier entropy over causally valid positions, excluding those after the first mismatch. We combine two-round certainty with the previous block outcome z_t:

$$s_t = -\frac{e_{t-1} + e_{t-2}}{2}, \qquad z_t = (B_{t-1}, F_{t-1}), \tag{2}$$

where F_{t-1} indicates whether the previous block was fully accepted. While s_t mainly reflects whether a long block should shrink under high uncertainty, z_t further encourages a larger drafting budget after

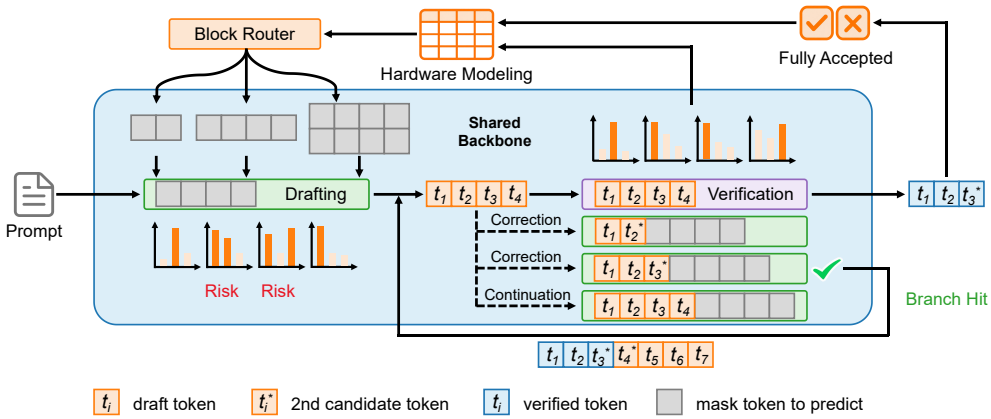


Figure 4: Overview of RecGuide runtime orchestration. Draft uncertainty guides predictive draft-verify overlap through correction and continuation branches, while verification feedback and hardware profiles guide request-specific block size allocation.

full acceptance, since the true acceptance potential may extend beyond the current block boundary. Given this, RecGuide estimates the expected accepted progress $V_B(s_t, z_t)$ from calibration data for each candidate B. Following the hardware-aware cost modeling of DSpark, an offline profile provides the execution cost $T_C(B)$ under serving load C. RecGuide then selects

$$B_t^* = \arg \max_{B \in \{8,16,32\}} \left[V_B(s_t, z_t) - \rho_C T_C(B) \right], \tag{3}$$

where ρ_C weights the profiled execution cost at load C. A larger block is chosen only when its expected acceptance gain justifies the additional computation. The decision is resolved through lightweight table lookup at runtime, introducing negligible overhead (Appendix B.2). The table and all coefficients are frozen before testing.

5.3 System Implementation

Algorithmic optimization alone does not directly translate into end-to-end gains. For draft-verify overlap, naively batching the verifier and prospective branches causes redundant prefix KV reads and padding due to unequal sequence lengths. We therefore extend SGLang with shared-prefix KV access and padding-free execution, allowing all branches to reuse the canonical prefix while processing only valid query tokens. We also redesign the mapping of mixed attention workloads onto GPU SMs.

Dynamic block sizing faces a similar issue: mixing different block sizes within a batch forces shorter requests to pad to the largest block, offsetting their computational savings. RecGuide therefore adopts block-size bucketed serving. After each verification round, requests are grouped by their planned block size, and the scheduler forms batches only from requests in the same bucket, preserving the efficiency of adaptive block sizing while keeping request states and canonical KV independent.

6 Experiments

6.1 Experimental Setup

Models and Benchmarks. We evaluate RecGuide on Nemotron-Labs-Diffusion (NLD) 8B and 14B (Fu et al., 2026), a representative state-of-the-art diffusion–AR self-speculation family. Fixing the backbone enables fair decoding-paradigm comparisons with LinearSS and its quadratic variant TiDAR (Liu et al., 2026b), isolating runtime gains from architectural differences. Our evaluation covers eight benchmarks spanning three categories: **Math** (GSM8K (Cobbe et al., 2021), MATH-500 (Lightman et al., 2024), AIME25 (Zhang & Math-AI, 2025)); **Code** (HumanEval (Chen et al., 2021), MBPP (Austin et al., 2021), LiveCodeBench (Jain et al., 2025)); **QA & instruction following** (GPQA (Rein et al., 2023), IFEval (Zhou et al., 2023)). We compare against vanilla LinearSS, DFlash,

Table 2: Average acceptance length (τ) per two-forward round and decoding speedup over autoregressive decoding on NLD-8B/14B at $C = 1$, $B = 16$. RecGuide can verify two blocks in this window.

Model	Method	Metric	MATH			CODE			QA & IF		OVERALL
			GSM8K	MATH-500	AIME25	HumanEval	MBPP	LCB	GPQA	IFEval	*Avg.*
NLD-8B	LinearSS	τ	9.94	11.21	10.49	9.71	8.15	9.78	10.62	9.57	9.93
		Speedup	3.03×	2.95×	2.42×	2.91×	2.67×	2.36×	2.58×	2.53×	2.68×
	DFlash	τ	4.26	4.31	4.90	4.28	3.78	4.64	5.11	3.90	4.40
		Speedup	2.80×	2.87×	3.13×	2.83×	2.47×	3.09×	3.39×	2.57×	2.89×
	TiDAR	τ	13.44	15.80	14.54	12.92	9.18	13.48	15.18	12.52	13.38
		Speedup	2.62×	2.87×	2.34×	2.52×	1.88×	2.32×	2.57×	2.26×	2.42×
	RecGuide	τ	13.95	16.08	15.68	13.64	11.08	13.54	14.84	13.33	14.02
		Speedup	4.05×	4.66×	4.38×	4.07×	3.22×	3.91×	4.27×	3.94×	4.06×
NLD-14B	LinearSS	τ	9.74	10.50	9.35	11.99	10.81	9.27	10.79	8.35	10.10
		Speedup	2.84×	3.06×	2.36×	4.15×	3.41×	2.50×	2.82×	2.43×	2.95×
	DFlash	τ	4.78	4.37	3.81	4.93	5.49	4.14	5.28	3.80	4.58
		Speedup	3.13×	2.94×	2.50×	3.30×	3.43×	2.80×	3.54×	2.57×	3.03×
	TiDAR	τ	13.10	16.24	13.90	17.72	15.80	13.26	17.24	12.20	14.94
		Speedup	2.74×	3.09×	2.47×	3.78×	3.13×	2.48×	3.08×	2.36×	2.89×
	RecGuide	τ	13.61	15.43	13.41	18.19	12.56	13.27	16.21	10.97	14.21
		Speedup	4.22×	4.82×	4.21×	5.49×	3.74×	4.17×	5.03×	3.55×	4.40×

Table 3: Branch outcomes in draft-verify overlap. Correction and Continuation report coverage / reuse rate (%); No Hit denotes rounds unmatched by any prospective branch.

Model	Branch Outcome	MATH	CODE	QA & IF	OVERALL
NLD-8B	Correction	32.46 / 21.46	41.63 / 25.81	33.31 / 17.25	36.11 / 22.04
	Continuation	43.85 / 42.92	27.89 / 27.14	48.55 / 47.88	39.04 / 38.24
	No Hit	23.68	30.48	18.14	24.85
NLD-14B	Correction	32.37 / 21.98	30.99 / 20.38	35.44 / 18.61	32.62 / 20.54
	Continuation	37.12 / 35.95	43.81 / 36.53	39.22 / 38.88	40.15 / 36.90
	No Hit	30.51	25.20	25.33	27.22

and TiDAR. Due to training budget constraints, DFlash is trained on ShareGPT (RyokoAI, 2023) following the SpecForge recipe (Li et al., 2026a), while other configurations follow official settings.

Evaluation Settings and Metrics. All experiments are conducted on a single NVIDIA B200 GPU with SGLang (Zheng et al., 2024) engine and FlashInfer (Ye et al., 2025b) backend. We evaluate our two RecGuide strategies under their corresponding serving regimes. Following NLD's recipe, all methods use greedy decoding. We set concurrency $C = 1$ and block size $B = 16$ for low-concurrency experiments. Potential mismatch positions are identified by $M_i < \delta$, with $\delta = 0.5$ throughout all experiments. For higher-concurrency serving, we use a fixed-seed 5%/95% request split: 180 calibration and 3403 held-out requests, and V_B and ρ_C are calibrated only on calibration requests (Appendix B.1). We vary concurrency over $C \in \{2, 4, 8, 16, 32, 64, 128\}$ and compare RecGuide against fixed $B \in \{8, 16, 32\}$ using both aggregate and per-request throughput.

6.2 LOW-CONCURRENCY PERFORMANCE

Table 2 compares RecGuide with representative diffusion-AR speculative decoding paradigms under the single-request setting, reporting average acceptance length τ and end-to-end speedup over autoregressive decoding. Although LinearSS accepts substantially more tokens per round than DFlash, its two full-backbone forwards limit end-to-end speedup, allowing DFlash with a lightweight drafter to remain faster. RecGuide, by contrast, achieves the highest speedup, averaging $4.06\times$ on NLD-8B and $4.40\times$ on NLD-14B over autoregressive decoding, and delivering up to $1.8\times$ speedup over vanilla LinearSS on AIME25. Under the same two-forward accounting, RecGuide raises the effective acceptance length to more than 14 tokens, approaching the $B = 16$ limit of serial execution.

Longer acceptance, however, does not necessarily imply faster decoding even after removing the two-forward dependency of linear self-speculation, as TiDAR illustrates. Its quadratic speculative layout prepares a future draft for every possible acceptance position, incurring $O(B^2)$ speculative

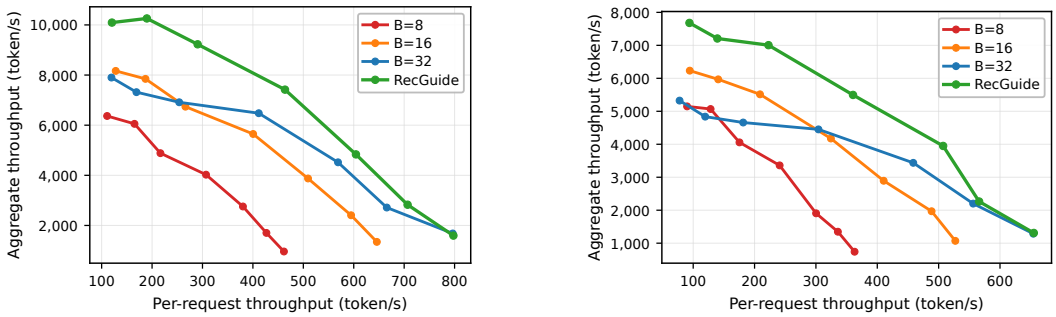


Figure 5: Aggregate and per-request throughput trade-offs under different concurrency levels on (a) NLD-8B and (b) NLD-14B. We compare fixed block sizes $B \in \{8, 16, 32\}$ with RecGuide.

Table 4: Dynamic block-size distributions across serving loads. Each entry reports the percentage of requests selecting $B = 8/16/32$, respectively; Overall is the macro average over eight benchmarks.

Model	Concurrency	MATH	CODE	QA & IF	OVERALL
NLD-8B	2	3.61 / 1.28 / 95.11	4.63 / 1.51 / 93.86	24.97 / 5.97 / 69.07	9.33 / 2.54 / 88.13
	8	0.63 / 26.91 / 72.46	0.66 / 39.18 / 60.16	12.85 / 48.73 / 38.43	3.70 / 36.96 / 59.34
	32	28.83 / 52.95 / 18.22	40.45 / 47.70 / 11.86	60.76 / 22.35 / 16.89	41.17 / 43.33 / 15.50
	128	72.46 / 17.25 / 10.30	82.03 / 11.11 / 6.87	81.12 / 5.65 / 13.23	78.21 / 12.04 / 9.75
NLD-14B	2	1.40 / 3.11 / 95.49	0.32 / 1.92 / 97.76	12.14 / 12.83 / 75.03	3.68 / 5.09 / 91.23
	8	0.56 / 42.26 / 57.18	0.04 / 37.14 / 62.82	6.27 / 58.77 / 34.97	1.79 / 44.47 / 53.74
	32	28.31 / 63.75 / 7.94	20.26 / 69.51 / 10.23	56.77 / 31.81 / 11.43	32.40 / 57.92 / 9.67
	128	84.07 / 11.56 / 4.37	83.80 / 10.21 / 5.99	83.41 / 5.94 / 10.66	83.81 / 9.65 / 6.55

computation. Consequently, its longer acceptance does not translate into higher end-to-end speedup because of the increased forward cost, leaving TiDAR behind LinearSS on average. Moreover, TiDAR drafts from hypothesized acceptance positions without explicitly supplying a correction token at likely mismatches. The repeated draft may still be incorrect; even after a successful correction, later tokens in the same parallel draft cannot condition on it, potentially degrading the draft quality.

RecGuide instead assigns the second-ranked candidate to selected risky positions, providing an explicit corrected prefix once verified. Despite using far fewer prospective branches than TiDAR, RecGuide retains high mismatch coverage, as shown in Table 3. Correction and continuation branches cover most decoding rounds, leaving only 24.85% and 27.22% of rounds on NLD-8B and NLD-14B, respectively, unmatched by any branch. Furthermore, among matched correction branches, the second-ranked candidate matches the verifier correction in over 60% of cases, supporting explicit correction as a reliable drafting signal. Continuation branches are even more reusable: after full-block acceptance, the first draft token almost always matches the additional next token produced by the verifier. RecGuide therefore preserves acceptance lengths comparable to TiDAR with far less speculative computation, ultimately achieving the highest decoding efficiency.

6.3 HIGH-CONCURRENCY PERFORMANCE

Figure 5 evaluates RecGuide under higher serving loads, showing the trade-off between aggregate and per-request throughput across $C \in \{2, 4, 8, 16, 32, 64, 128\}$. Increasing concurrency raises aggregate throughput at the expense of per-request throughput. Compared with fixed $B \in \{8, 16, 32\}$, RecGuide favors larger blocks when compute remains available and shifts toward smaller blocks as execution becomes compute limited. This adaptive allocation consistently improves aggregate throughput on both NLD-8B and NLD-14B, advancing the overall serving frontier.

Table 4 makes this load adaptation explicit. On NLD-8B, the fraction of $B = 32$ decreases from 88.13% at $C = 2$ to 9.75% at $C = 128$, while $B = 8$ increases from 9.33% to 78.21%; NLD-14B exhibits the same transition. This confirms that RecGuide adjusts drafting budgets with serving load rather than relying on a single fixed configuration. Block-size distributions also vary across task

categories at the same concurrency. In particular, QA & IF shifts toward smaller blocks earlier than MATH and CODE, consistent with higher verification uncertainty and demonstrating request-level adaptation beyond load awareness alone.

These distributions also explain the per-request throughput trends in Figure 5. At $C = 2$ and $C = 128$, RecGuide predominantly selects $B = 32$ and $B = 8$, respectively, so its per-request throughput approaches the corresponding fixed-block baselines, whereas the more diverse assignments at intermediate concurrency provide greater gains from request-specific adaptation. Notably, fixed $B = 8$ remains inferior to $B = 16$ in aggregate throughput at the highest concurrency, since shorter blocks require more decoding rounds and amplify progress imbalance across concurrent requests. RecGuide avoids this drawback by selecting short blocks only when beneficial and regrouping requests by their planned block size at each round, allowing aggressive use of $B = 8$ without inheriting the inefficiency of a globally fixed short-block policy.

6.4 Ablation Study

We further decompose RecGuide by progressively adding the key components of Predictive Draft-Verify Overlap and Verification Guided Adaptive Drafting to vanilla LinearSS under their respective load regimes. We report aggregate throughput in tokens per second (TPS) on NLD-8B / 14B.

Predictive Draft-Verify Overlap. Table 5 ablates the prospective branches at $C = 1$. Starting with the first correction branch, we add the second correction and continuation branches to isolate the benefits of broader mismatch coverage and full-block continuation. The first correction branch alone improves throughput by 34.29% and 28.72% on NLD-8B / 14B, respectively. The second correction branch yields a smaller gain, which we attribute to later risky positions being less causally grounded and harder to predict, supporting our decision to bound the number of correction branches. Adding the continuation branch further raises the improvement to 51.76% and 49.36%. These results show that three prospective branches capture most of the overlap benefit while keeping computation bounded.

Table 5: Ablation of prospective branches in Predictive Draft-Verify Overlap at $C = 1$. TPS is averaged over all benchmarks.

Method	NLD-8B	NLD-14B
LinearSS	595	470
+ 1st Correction	799 (+34.29%)	605 (+28.72%)
+ 2nd Correction	811 (+36.30%)	612 (+30.21%)
+ Continuation	903 (+51.76%)	702 (+49.36%)

Verification Guided Adaptive Drafting. We next ablate the adaptive block policy over $C \in \{2, 4, 8, 16, 32, 64, 128\}$, as shown in Table 6. Fixed Block follows NLD's official recipe, using a concurrency-specific block size shared by all requests. Adding two-round entropy s_t improves TPS by 10.49% / 7.60% on NLD-8B / 14B. The previous block size and full-acceptance state z_t further guide budget expansion. Request-specific block sizes introduce unequal draft lengths within batches, so padding to the largest block can offset these computational savings. Block-size bucketing groups requests by their selected size, translating compute savings into serving throughput gains. Together, adaptive selection and bucketed execution improve TPS by 39.84% and 32.80%, respectively.

Table 6: Ablation of verification guided adaptive drafting. TPS is averaged over all benchmarks and concurrency levels.

Method	NLD-8B	NLD-14B
Fixed Block	4725	3756
+ s_t	5221 (+10.49%)	4041 (+7.60%)
+ z_t	5254 (+11.19%)	4071 (+8.40%)
+ Bucketing	6608 (+39.84%)	4987 (+32.80%)

7 Conclusion

This work presents Reciprocal Guidance (RecGuide), a runtime draft-verify orchestration framework for diffusion-AR self-speculation. RecGuide leverages reciprocal predictability between drafting and verification to overlap the two stages at low concurrency and adapt request-specific draft budgets under heavier workloads. Across a wide range of concurrency levels, RecGuide consistently improves throughput over vanilla LinearSS, achieving up to 1.8× speedup. These results highlight self-speculation that reuses the full target backbone as the drafter as a promising alternative for efficient speculative decoding across diverse serving loads and concurrency regimes.

AI USE STATEMENT

Generative AI tools were used to assist with software development tasks, including code completion, debugging, and resolving implementation issues based on algorithmic designs and implementation specifications developed by the authors. They were also used for language polishing, readability improvements, and identifying potential typographical or formatting issues in the manuscript. The research idea, methodological design, algorithmic formulation, implementation architecture, and experimental design were independently developed by the authors. Generative AI tools were not used for research ideation, methodological or experimental design, drafting substantive sections of the paper, generating synthetic datasets, proving mathematical claims, interpreting experimental results, or other substantive research activities. All AI-assisted code and text were manually reviewed, tested, and verified by the authors. We take full responsibility for the final content of this work.

ETHICS STATEMENT

This work focuses on improving the inference efficiency of large language models. All models and datasets used in our experiments are publicly available or open-source resources and are used solely for research and evaluation purposes. The study does not involve human subjects, personally identifiable information, private user data, or the collection of sensitive data. To the best of our knowledge, the methodology and experiments presented in this work do not introduce specific concerns regarding privacy, fairness, discrimination, or other ethical issues covered by the ICLR Code of Ethics.

REPRODUCIBILITY STATEMENT

We have taken several steps to facilitate the reproducibility of this work. Section 3 describes the problem formulation and the inference bottlenecks motivating our study, while Section 4 presents the observations underlying reciprocal predictability between drafting and verification. Section 5 details the design and execution procedure of RecGuide, and Section 6 specifies the experimental methodology, including the models, datasets, baselines, and evaluation settings. Additional implementation details and experimental results are provided in the appendix.

REFERENCES

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, 2021. URL https://arxiv.org/abs/2108.07732.

Tiwei Bie, Maosong Cao, Kun Chen, Lun Du, Mingliang Gong, Zhuochen Gong, Yanmei Gu, Jiaqi Hu, Zenan Huang, Zhenzhong Lan, Chengxi Li, Chongxuan Li, Jianguo Li, Zehuan Li, Huabin Liu, Ling Liu, Guoshan Lu, Xiaocheng Lu, Yuxin Ma, Jianfeng Tan, Lanning Wei, Ji-Rong Wen, Yipeng Xing, Xiaolu Zhang, Junbo Zhao, Da Zheng, Jun Zhou, Junlin Zhou, Zhanchao Zhou, Liwang Zhu, and Yihong Zhuang. Llada2.0: Scaling up diffusion language models to 100b, 2025. URL https://arxiv.org/abs/2512.15745.

Tianle Cai, Yuhong Li, Zhengyang Geng, Hongwu Peng, Jason D. Lee, Deming Chen, and Tri Dao. Medusa: Simple LLM inference acceleration framework with multiple decoding heads. In *Proceedings of the 41st International Conference on Machine Learning*, pp. 5209–5235. 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, 2023. URL https://arxiv.org/abs/2302.01318.

Jian Chen, Yesheng Liang, and Zhijian Liu. Dflash: Block diffusion for flash speculative decoding. *arXiv preprint arXiv:2602.06036*, 2026.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray,

Nick Ryder, Mikhail Pavlov, Alethea Power, Lukasz Kaiser, Mohammad Bavarian, Clemens Winter, Philippe Tillet, Felipe Petroski Such, Dave Cummings, Matthias Plappert, Fotios Chantzis, Elizabeth Barnes, Ariel Herbert-Voss, William Hebgen Guss, Alex Nichol, Alex Paino, Nikolas Tezak, Jie Tang, Igor Babuschkin, Suchir Balaji, Shantanu Jain, William Saunders, Christopher Hesse, Andrew N. Carr, Jan Leike, Josh Achiam, Vedant Misra, Evan Morikawa, Alec Radford, Matthew Knight, Miles Brundage, Mira Murati, Katie Mayer, Peter Welinder, Bob McGrew, Dario Amodei, Sam McCandlish, Ilya Sutskever, and Wojciech Zaremba. Evaluating large language models trained on code, 2021. URL https://arxiv.org/abs/2107.03374.

Zhuoming Chen, Avner May, Ruslan Svirschevski, Yuhsun Huang, Max Ryabinin, Zhihao Jia, and Beidi Chen. Sequoia: Scalable, robust, and hardware-aware speculative decoding. *arXiv preprint arXiv:2402.12374*, 2024.

Shuang Cheng, Yihan Bian, Dawei Liu, Yuhua Jiang, Yihao Liu, Linfeng Zhang, Qian Yao, Zhongbo Tian, Wenhai Wang, Qipeng Guo, et al. Sdar: A synergistic diffusion-autoregression paradigm for scalable sequence generation. In *Findings of the Association for Computational Linguistics: ACL 2026*, pp. 22058–22075, 2026a.

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

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, 2021. URL https://arxiv.org/abs/2110.14168.

Yonggan Fu, Lexington Whalen, Abhinav Garg, Chengyue Wu, Maksim Khadkevich, Nicolai Oswald, Enze Xie, Daniel Egert, Sharath Turuvekere Sreenivas, Shizhe Diao, Chenhan Yu, Ye Yu, Weijia Chen, Sajad Norouzi, Jingyu Liu, Shiyi Lan, Ligeng Zhu, Jin Wang, Jindong Jiang, Morteza Mardani, Mehran Maghoumi, Song Han, Ante Jukic, Nima Tajbakhsh, Jan Kautz, and Pavlo Molchanov. Nemotron-labs-diffusion: A tri-mode language model unifying autoregressive, diffusion, and self-speculation decoding. Technical report, NVIDIA, 2026.

Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models. *Advances in neural information processing systems*, 33:6840–6851, 2020.

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. doi: 10.48550/arXiv.2605.29707. URL https://arxiv.org/abs/2605.29707.

Naman Jain, 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*, volume 2025, pp. 58791–58831, 2025.

Tanishq Kumar, Tri Dao, and Avner May. Speculative speculative decoding, 2026. URL https://arxiv.org/abs/2603.03251.

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

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, 2026a. URL https://arxiv.org/abs/2603.18567.

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, 2024a.

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

Yuhui Li, Fangyun Wei, Chao Zhang, and Hongyang Zhang. Eagle-3: Scaling up inference acceleration of large language models via training-time test. *Advances in Neural Information Processing Systems*, 38:136737–136756, 2026b.

Hunter Lightman, Vineet Kosaraju, Yuri Burda, Harrison Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. Let's verify step by step. In *International Conference on Learning Representations*, volume 2024, pp. 39578–39601, 2024.

Luxi Lin, Zhanpeng Zeng, Shuang Peng, Songwei Liu, and Rongrong Ji. Retrace: Rejected-trajectory conditioning for speculative decoding, 2026. URL https://arxiv.org/abs/2608.29748.

Hong Liu, Rui Cen, Junhan Shi, Guangshuo Qin, Jiebin Zhang, Tianyu Liu, Runzhi Fan, Guoliang Zhao, Ruobing Xie, Kai Zhang, Song Liu, Guanghua Yu, and Jianchen Zhu. Angelspec: Towards real-world high performance inference with speculative decoding, 2026a. URL https://arxiv.org/abs/2607.25852.

Jingyu Liu, Xin Dong, Zhifan Ye, Rishabh Mehta, Yonggan Fu, Vartika Singh, Ce Zhang, and Pavlo Molchanov. Tidar: Think in diffusion, talk in autoregression. *Proceedings of Machine Learning and Systems*, 8:748–762, 2026b.

Tianyu Liu, Yun Li, Qitan Lv, Kai Liu, Jianchen Zhu, Winston Hu, and Xiao Sun. Pearl: Parallel speculative decoding with adaptive draft length. In *International Conference on Learning Representations*, volume 2025, pp. 1085–1104, 2025.

Tianyu Liu, Yuhao Shen, Rui Cen, Junhan Shi, Jiebin Zhang, Guangshuo Qin, Hong Liu, Song Liu, Guanghua Yu, and Jianchen Zhu. D-cut: Adaptive verification depth pruning for batched speculative decoding, 2026c. URL https://arxiv.org/abs/2607.14647.

Shen Nie, Fengqi Zhu, Zebin You, Xiaolu Zhang, Jingyang Ou, Jun Hu, Jun Zhou, Yankai Lin, Ji-Rong Wen, and Chongxuan Li. Large language diffusion models. *Advances in Neural Information Processing Systems*, 38:50608–50646, 2026.

Zhiyuan Ning, Jiawei Shao, Ruge Xu, Xinfei Guo, Jun Zhang, Chi Zhang, and Xuelong Li. Cas-spec: Cascade adaptive self-speculative decoding for on-the-fly lossless inference acceleration of llms. *Advances in Neural Information Processing Systems*, 38:65145–65167, 2026.

David Rein, Betty Li Hou, Asa Cooper Stickland, Jackson Petty, Richard Yuanzhe Pang, Julien Dirani, Julian Michael, and Samuel R. Bowman. Gpqa: A graduate-level google-proof q&a benchmark, 2023. URL https://arxiv.org/abs/2311.12022.

RyokoAI. Sharegpt52k: A collection of chatgpt conversations. https://huggingface.co/datasets/RyokoAI/ShareGPT52K, 2023.

Ranajoy Sadhukhan, Jian Chen, Zhuoming Chen, Vashisth Tiwari, Ruihang Lai, Jinyuan Shi, Ian Yen, Avner May, Tianqi Chen, and Beidi Chen. Magicdec: Breaking the latency-throughput tradeoff for long context generation with speculative decoding. In *International Conference on Learning Representations*, volume 2025, pp. 6835–6850, 2025.

Linye Wei, Wenjue Chen, Pingzhi Tang, Xiaotian Guo, Le Ye, Runsheng Wang, and Meng Li. Orchestrating dual-boundaries: An arithmetic intensity inspired acceleration framework for diffusion language models, 2026. URL https://arxiv.org/abs/2511.21759.

RenYe Yan, Jikang Cheng, Yaozhong Gan, Shikun Sun, You Wu, Yunfan Yang, Liang Ling, Jinlong Lin, Yeshuang Zhu, Jie Zhou, et al. Entropy-adaptive diffusion policy optimization with dynamic step alignment. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 1924–1934, 2025.

Renye Yan, Jikang Cheng, Shikun Sun, Yi Sun, You Wu, Wei Peng, Zongwei Wang, Ling Liang, Junliang Xing, and Yimao Cai. Do less, achieve more: Do we need every-step optimization for rl fine-tuning of diffusion models? *arXiv preprint arXiv:2605.15855*, 2026.

Jiacheng Ye, Zhihui Xie, Lin Zheng, Jiahui Gao, Zirui Wu, Xin Jiang, Zhenguo Li, and Lingpeng Kong. Dream 7b: Diffusion large language models, 2025a. URL https://arxiv.org/abs/2508.15487.

Zihao Ye, Lequn Chen, Ruihang Lai, Wuwei Lin, Yineng Zhang, Stephanie Wang, Tianqi Chen, Baris Kasikci, Vinod Grover, Arvind Krishnamurthy, et al. Flashinfer: Efficient and customizable attention engine for llm inference serving. *Proceedings of Machine Learning and Systems*, 7, 2025b.

Jiebin Zhang, Zhenghan Yu, Liang Wang, Nan Yang, Eugene Yu, Zheng Li, Yifan Song, Dawei Zhu, Xingxing Zhang, Furu Wei, et al. Learning to draft: Adaptive speculative decoding with reinforcement learning. In *International Conference on Learning Representations*, volume 2026, pp. 30798–30819, 2026a.

Jun Zhang, Jue Wang, Huan Li, Lidan Shou, Ke Chen, Gang Chen, and Sharad Mehrotra. Draft& verify: Lossless large language model acceleration via self-speculative decoding. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 11263–11282, 2024.

Yaojie Zhang, Jianuo Huang, Junlong Ke, Yuhang Han, Yongji Long, Tianchen Zhao, Biqing Qi, and Linfeng Zhang. Flexdraft: Flexible speculative decoding via attention tuning and bonus-guided calibration, 2026b. URL https://arxiv.org/abs/2605.20022.

Yaojie Zhang, Linfeng Zhang, Bin Cui, and Xupeng Miao. Dflow: Enabling verifier information flow in block diffusion speculative decoding, 2026c. URL https://arxiv.org/abs/2609.06498.

Yifan Zhang and Team Math-AI. American invitational mathematics examination (AIME) 2025, 2025. URL https://huggingface.co/datasets/math-ai/aime25.

Lianmin Zheng, Liangsheng Yin, Zhiqiang Xie, Chuyue Sun, Jeff Huang, Cody H Yu, Shiyi Cao, Christos Kozyrakis, Ion Stoica, Joseph E Gonzalez, et al. Sglang: Efficient execution of structured language model programs. *Advances in neural information processing systems*, 37:62557–62583, 2024.

Shuzhang Zhong, Zebin Yang, Ruihao Gong, Runsheng Wang, Ru Huang, and Meng Li. Propd: Dynamic token tree pruning and generation for llm parallel decoding. In *Proceedings of the 43rd IEEE/ACM International Conference on Computer-Aided Design*, pp. 1–8, 2024.

Jeffrey Zhou, Tianjian Lu, Swaroop Mishra, Siddhartha Brahma, Sujoy Basu, Yi Luan, Denny Zhou, and Le Hou. Instruction-following evaluation for large language models, 2023. URL https://arxiv.org/abs/2311.07911.

A Predictive Draft–Verify Overlap: Further Analysis

This appendix connects the draft-distribution evidence in Section 4.1 to the overlap mechanism in Section 5.1. We describe predictive branch construction and reuse, compare draft-based first-failure predictors, and relate their signals to measured correction and continuation utility.

A.1 Predictive Branch Construction and Reuse

For draft position i, let $P_i(v)$ denote the token distribution normalized over the vocabulary excluding the mask token. Under greedy drafting, the selected token has confidence $C_i = P_{1\mathrm{st}}(i)$. RecGuide uses the margin $M_i = P_{1\mathrm{st}}(i) - P_{2\mathrm{nd}}(i)$ to identify positions where the leading token faces a competitive alternative. With $\delta = 0.5$, the candidate set is $\mathcal{K} = \{i : M_i < \delta\}$. These scores are available before verification, enabling prospective drafting alongside verification of the current draft.

RecGuide prepares a correction branch for each of the earliest two positions in $\mathcal{K}$, or for all candidates if fewer than two exist. Each branch uses the second-ranked token at its selected position to form a candidate corrected prefix for subsequent drafting. When $|\mathcal{K}| \leq 2$, RecGuide also prepares a continuation branch assuming full acceptance of the current block. The condition uses the candidate set before truncation. Thus, each round contains at most two correction rows, one continuation row, and the verifier row. This bounded allocation targets early mismatches and full-block continuation within the parallel computation available at low concurrency.

A correction branch becomes reusable when both its predicted first-rejection position and replacement token match the causal verifier. A continuation branch becomes reusable when the current block is fully accepted and its first prospective token matches the verifier's bonus token. Unmatched branches are discarded; when no branch is reusable, the next draft is generated from the verifier-confirmed prefix. A reusable branch supplies the next draft, which remains subject to causal verification and can initiate another round of predictive overlap. Causal verification authorizes all output commitment, while successful predictions remove the intervening serial draft forward.

A.2 Comparison of Draft-Based Failure Predictors

We evaluate first-failure prediction on NLD-8B with greedy drafting and block size $B = 16$, using request-disjoint held-out evaluation across benchmarks. Each predictor is evaluated at its frozen operating point. Ratios are first computed within each benchmark and then macro-averaged with equal benchmark weights. This analysis measures the predictive information available before verification and complements the distributional separation shown in Figure 3.

Evaluation quantities. Let q_r be the first rejected draft position in round r, with $q_r = \perp$ for full acceptance. A risk score $S_r(i)$ and threshold θ define $\mathcal{K}_r = \{i : S_r(i) > \theta\}$; let $p_r = \min \mathcal{K}_r$, or $\perp$ if the set is empty. Let $\mathcal{G}_r$ contain the verifier-passed positions preceding q_r, or all draft positions for a fully accepted block. We compute three complementary quantities within each benchmark:

$$\begin{aligned} \mathrm{SetR} &= \frac{\sum_r \mathbf{1}[q_r \in \mathcal{K}_r]}{\sum_r \mathbf{1}[q_r \neq \perp]}, \\ \mathrm{TokFPR} &= \frac{\sum_r |\mathcal{K}_r \cap \mathcal{G}_r|}{\sum_r |\mathcal{G}_r|}, \\ \mathrm{FirstP} &= \frac{\sum_r \mathbf{1}[p_r = q_r \neq \perp]}{\sum_r \mathbf{1}[p_r \neq \perp]}. \end{aligned} \tag{4}$$

Candidate-set first-failure recall (SetR) measures whether the candidate set contains the true first rejection. Passed-token false-positive rate (TokFPR) measures candidate markings among causally verified passed tokens; positions after the first rejection are excluded. First-crossing exact precision (FirstP) measures whether the earliest reported candidate is exactly the first rejection. A reported candidate in a fully accepted block counts as a miss for FirstP. SetR and TokFPR use the complete candidate set, while the deployed branch budget and its realized outcomes are assessed by Table 3.

Predictor families. Table 7 specifies the nine predictor families and their operating points. Confidence-drop scores compare C_i with a prefix, local, or historical reference. Margin risk instead compares the two leading token alternatives at the same position, while entropy uses the full token distribution. The prefix and local scores apply from the second position onward, where a preceding-token reference is available for the comparison.

Table 7: Draft-based risk scores and frozen operating points. A position is marked when $S_i > \theta$. Entropy uses natural logarithms; margin risk at $\theta = 0.5$ is equivalent to the deployed rule $M_i < 0.5$.

Predictor	Risk score S_i	θ
Prefix-mean drop	$1 - C_i / \operatorname{mean}_{j<i} C_j$	0.25
Prefix-median drop	$1 - C_i / \operatorname{median}_{j<i} C_j$	0.25
Local drop	$1 - C_i / C_{i-1}$	0.25
Absolute confidence risk	$1 - C_i$	0.35
Top-two margin risk	$1 - M_i$	0.50
Token-distribution entropy	$-\sum_v P_i(v) \log P_i(v)$	1.00
Historical-good drop	$1 - C_i / G$	0.25
Historical-error drop	$1 - C_i / E + 0.2 R_i$	0.00
Historical good/error separator	$(G - C_i)/(G - E) + 0.2 R_i$	0.75

Here, G and E are historical confidence references for verifier-passed and first-rejected tokens, respectively. The historical-good operating point uses the mean over the previous four rounds. The last two predictors also use a location prior $R_i = \operatorname{clip}(1 - |i - Q|/N, 0, 1)$, where Q is the mean first-rejection position over the previous four rounds and N is the recorded position count. When the required history is unavailable, these history-based predictors fall back to prefix-mean drop with threshold 0.15. Historical references use completed verification outcomes, making the scores available before causal verification of the current draft block begins.

Table 8: Held-out first-failure prediction across benchmarks on NLD-8B with $B = 16$. Entries are benchmark-macro-averaged percentages at the operating points in Table 7. Best values are bold.

Predictor	SetR ↑	TokFPR ↓	FirstP ↑
Prefix-mean drop	61.72	4.88	35.52
Prefix-median drop	63.93	5.23	35.43
Local drop	51.70	**4.23**	35.57
Absolute confidence risk	65.00	4.74	**38.53**
Top-two margin risk	**71.34**	5.81	38.00
Token-distribution entropy	56.69	4.67	33.34
Historical-good drop	68.41	5.38	37.69
Historical-error drop	64.19	7.96	33.10
Historical good/error separator	71.24	6.88	35.69

Evidence for margin-guided prediction. Across the reported operating points, candidate sets recover 51.70%–71.34% of first failures while marking 4.23%–7.96% of verifier-passed tokens. Predictive information is therefore present across multiple draft statistics. Margin risk attains the highest SetR, 71.34%, with a TokFPR of 5.81% and a FirstP of 38.00%. Compared with prefix-mean drop, it increases first-failure recall by 9.62 percentage points for a 0.93-point increase in passed-token false positives, providing broader coverage for prospective correction.

Margin risk also exceeds the historical good/error separator on all three reported measures: 71.34% versus 71.24% SetR, 5.81% versus 6.88% TokFPR, and 38.00% versus 35.69% FirstP. Its FirstP is within 0.53 percentage points of the highest value, obtained by absolute confidence risk, while its SetR is 6.34 points higher. These results support margin as a current-draft signal combining strong first-failure coverage with competitive first-location precision. Moreover, the score directly measures competition from the second-ranked token used by RecGuide's correction branch, linking the prediction criterion to the proposed replacement prefix.

A.3 Branch Utility and Decoding Efficiency

The branch outcomes in Table 3 connect position prediction to usable prospective drafts. On NLD-8B, correction and continuation branches have reported coverage / reuse rates of 36.11% / 22.04% and 39.04% / 38.24%, respectively; the corresponding NLD-14B values are 32.62% / 20.54% and 40.15% / 36.90%. Correction branches prepare explicitly amended prefixes at predicted mismatches, while continuation branches exploit full-block acceptance. Their complementary roles allow the same bounded overlap mechanism to benefit both rejected and fully accepted blocks.

Table 5 measures the resulting throughput gains over LinearSS at $C = 1$. The first correction branch raises throughput by 34.29% on NLD-8B and 28.72% on NLD-14B. Adding the second correction branch brings these gains to 36.30% and 30.21%; adding continuation further increases them to 51.76% and 49.36%. This progression shows that early correction supplies substantial benefit and full-block continuation adds a complementary source of reusable drafts, supporting the combination of both branch types within the fixed computation budget.

The end-to-end comparison in Table 2 places this design in the broader decoding setting. Whereas TiDAR prepares drafts across possible acceptance positions using a quadratic speculative layout, RecGuide allocates at most three prospective branches to selected corrections and full-block continuation. With the shared-prefix and padding-free execution described in Section 5.3, RecGuide achieves average speedups of $4.06\times$ and $4.40\times$ over autoregressive decoding on NLD-8B and NLD-14B, respectively, compared with TiDAR's $2.42\times$ and $2.89\times$. These results and the branch ablation support targeted predictive overlap for exploiting spare compute at low concurrency.

B Calibration and Execution of Adaptive Drafting

This appendix specifies the offline calibration and online execution of the adaptive drafting policy in Section 5.2. Calibration produces a shared conditional progress table and one time-cost coefficient for each serving width. These quantities are fixed during evaluation; each online round uses completed verification history to select a single block size for each request.

B.1 Calibration Protocol and Verifier State

We partition the eight benchmarks by request identity with a fixed seed, allocating approximately 5% of each benchmark to calibration and the remaining 95% to serving evaluation. Table 9 lists the resulting integer counts. The 180 calibration requests are used for state selection, entropy binning, conditional progress estimation, and time-cost coefficient selection. The 3,403 evaluation requests are excluded from these steps. After calibration, we freeze the complete policy and run it on the held-out requests in SGLang to measure serving performance.

State selection. Calibration considers historical progress, acceptance fraction, and full acceptance; verifier top-1 confidence, top-1/top-2 margin, and entropy at passed and first-rejected positions; and the previous block size together with its full-acceptance indicator. We consider history windows of 1, 2, 4, and 8 rounds. This comparison is confined to calibration requests. The deployed policy uses the two-round entropy summary and previous block/full state defined below, with the same state representation across all eight evaluation benchmarks.

Verifier state. For a completed round r, a verifier position is causally valid if all preceding draft tokens have passed verification. We use passed positions and, when a rejection occurs, the first-rejected position. Logits after the first rejection are excluded. Let $p_{r,j}$ be the verifier distribution at a valid decision position. Its entropy is $H(p_{r,j}) = -\sum_v p_{r,j}(v) \log p_{r,j}(v)$. We average over up to the first seven valid decisions to obtain the round-level entropy e_r.

Before drafting round t, the continuous state is the negative mean entropy of the two most recent completed rounds, as in Equation 2. With only one completed round, we use that round's entropy. A request without verification history uses the frozen cold-start block for its serving width. Thus, larger s_t indicates greater recent verifier certainty; the next round uses only completed history.

The discrete state $z_t = (B_{t-1}, F_{t-1})$ has six possible values: partial or full acceptance for each previous block size in $\{8, 16, 32\}$. This state preserves the distinction between a block boundary and

Table 9: Request counts for adaptive-drafting calibration and held-out serving evaluation. The nominal 5% calibration allocation is rounded within each benchmark.

Benchmark	Total	Calibration	Serving evaluation
GSM8K	1319	66	1253
MATH-500	500	25	475
AIME25	30	2	28
HumanEval	164	8	156
MBPP	378	19	359
LiveCodeBench	454	23	431
GPQA	197	10	187
IFEval	541	27	514
Total	3583	180	3403

an observed rejection. In particular, full acceptance of a short block leaves open the possibility of useful progress beyond that boundary. Both state components are updated only from the executed branch, using the verification results associated with its committed trajectory.

B.2 Progress and Cost Calibration

One-step block outcomes. On calibration requests, the collector follows a fixed behavior policy. At each visited state, it saves the committed prefix and completed verification history, and evaluates the three candidate blocks from this same state. Each candidate is checked by causal verification. Only the behavior-policy action advances the canonical prefix and history; the other outcomes are retained solely for calibration. The collector therefore records three local outcomes per visited state along one committed trajectory. Under the 5% split, the collector visits approximately 29,700 states and records 89,100 one-step outcomes for the three candidate block sizes.

Let $A_t(B)$ denote the number of new output tokens committed by a verified round using block size B. It counts each emitted token once, including a verifier correction or bonus token when emitted, and respects the request's stopping condition. This progress count is used consistently for all candidate actions. The conditional expected progress for candidate block size B is:

$$V_B(s, z) = \mathbb{E}[A_t(B) \mid s_t = s, z_t = z]. \tag{5}$$

For integer $A_t(B)$, let $S_{B,k}(s, z) = \Pr(A_t(B) \geq k \mid s_t = s, z_t = z)$. The tail-sum identity yields:

$$V_B(s, z) = \sum_{k \geq 1} S_{B,k}(s, z), \tag{6}$$

where the sum ranges over the finite support of committed progress.

Shared lookup table. We discretize s using weighted-quantile boundaries computed on calibration requests. Within each entropy bin and block/full state, we estimate the survival probabilities for each candidate block. Isotonic calibration regularizes these estimates to be nondecreasing with verifier certainty, while survival probabilities are constrained to be nonincreasing in k. Summing the calibrated survival probabilities produces the three conditional progress values. Sparsely supported block/full states fall back to an entropy-conditioned table pooled across z on the same calibration requests. The entropy boundaries, value tables, and fallback rules are frozen together.

Calibration weights. We give equal total weight to each benchmark, equal weight to requests within a benchmark, and equal weight to visited rounds within a request. Specifically, a row from round t of calibration request n in benchmark d receives weight $w_{d,n,t} = 1/(DN_dR_{d,n})$, where D is the number of benchmarks, N_d their respective calibration request counts, and $R_{d,n}$ the number of visited rounds for that request. These weights are used for quantile boundaries and conditional estimates. Each model has one shared table; routing does not use benchmark identity.

Matched execution cost. For the fixed model, precision, backend, and device, $T_C(B)$ is the profiled time of one diffusion draft forward plus one causal verification forward with a homogeneous block size B and full execution width C. Profiling and bucketed serving use matched full widths. The profile aggregates repeated measurements over representative decode contexts, with aligned context samples across candidate blocks. It models the block-dependent draft–verify computation; the final serving measurement is obtained from actual execution of the frozen policy.

A shared time-cost coefficient. Equation 3 compares expected token progress with the profiled time cost. If time is measured in milliseconds, V_B has units of tokens and ρ_C has units of tokens per millisecond. A larger block is selected when its predicted additional progress compensates for its additional time cost. For a fixed width, the homogeneous batch cost can equivalently be amortized as $T_C(B)/C$, with the constant scale absorbed into the width-specific coefficient.

We select one ρ_C for each model, device configuration, and serving width, shared by all tasks. Candidate coefficients are assessed on calibration trajectories: predicted progress is summed and divided by summed profiled time within each request; request efficiencies are then averaged within each benchmark and equally across benchmarks. This criterion selects the operating point of the routing rule in Equation 3. Final throughput comes from held-out execution of the frozen policy.

Finite breakpoint selection. For a fixed table entry, two candidate blocks have equal utility at:

$$\rho = \frac{V_{B_i}(s,z) - V_{B_j}(s,z)}{T_C(B_i) - T_C(B_j)}, \qquad T_C(B_i) \neq T_C(B_j). \tag{7}$$

The finite lookup table therefore induces a finite set of action-switching points. Between adjacent points, the selected actions are unchanged. We evaluate the resulting distinct assignments on calibration trajectories and retain a coefficient from the best operating interval. Equal-cost actions are compared by progress, and exact utility ties favor the smaller block. The selected coefficient is frozen before serving evaluation, together with the value table and initialization rules.

B.3 Offline Calibration and Online Execution

Offline calibration begins with the fixed-seed, benchmark-stratified request split in Appendix B.1. On calibration requests, the collector records candidate verifier-history statistics and the one-step outcomes of all three block sizes, while only the behavior-policy action advances the committed trajectory. These records support state selection and construction of the weighted entropy bins and monotone conditional progress tables. Homogeneous draft–verify profiles supply the execution costs for each serving width and candidate block size; the utility breakpoints then determine the candidate operating intervals for coefficient selection under the calibration efficiency criterion. Before held-out serving begins, we freeze the state representation, entropy-bin boundaries, progress and cost tables, width-specific coefficients, cold-start blocks, and fallback rules as a single policy configuration.

Online inference initializes each request with the frozen cold-start block for its serving width. Each round executes the selected block and causal verification, commits the resulting output tokens, and updates the verifier-entropy history and previous block/full state. The router then reads the three candidate progress values, compares their utilities using Equation 3, and places the request in the bucket for its next block size. Requests within a bucket execute homogeneous blocks at the profiled width, and this cycle continues until the stopping condition is met. Routing requires only three table lookups and utility comparisons; each request executes a single selected block, with alternative candidate forwards confined to offline calibration. Held-out evaluation measures the resulting block choices, verification outcomes, and execution times on actual serving trajectories, including the history changes induced by those choices. The policy remains fixed throughout evaluation, with no updates to its state representation, value tables, or time-cost coefficients.