CARRYOVER DRAFTING: RECYCLING REJECTED STATES FOR SPECULATIVE DECODING

Anonymous authors
Paper under double-blind review

ABSTRACT

Speculative decoding accelerates LLM inference by verifying multiple drafted tokens in parallel, allowing a single target forward pass to generate several tokens. By construction, verification computes representations for both accepted and rejected tokens, but conventional drafters retain only the representations of accepted tokens, leaving the substantial verifier computation spent on rejected tokens effectively wasted. We show that these discarded hidden states retain useful information about future tokens that can improve subsequent drafts. However, realizing this opportunity poses two distinct challenges. At inference, reusing rejected states introduces additional projection and attention costs, which can increase drafting latency and offset the speedup gained from higher acceptance. During training, standard parallel training does not expose the drafter to rejected states encountered at inference, while generating such states through sequential rollouts would sacrifice parallelism across training positions. We introduce **Carryover Drafting**, which enables efficient reuse of rejected states and parallel inference-aligned training. Carryover recycles rejected target hidden states as temporary KV context for the next draft, allowing the drafter to selectively attend to the rejected states through its attention layers. A single learned embedding distinguishes rejected states from committed context, while the carried-over KV context is replaced each drafting round to bound the additional context by one proposal block. We further introduce parallel draft–verify–draft training to expose the drafter to inference-aligned rejected states without sacrificing parallelism across training positions. Across DFlash and a DSpark-derived semi-autoregressive drafter, Carryover improves average acceptance length by up to 14.7% and end-to-end serving speedup by up to 14.4% over the corresponding baselines, with gains reaching 28.8% on translation.

1 INTRODUCTION

Autoregressive LLMs generate one token per forward pass, imposing a sequential dependency across output tokens. Speculative decoding uses additional parallel computation to accelerate this process: a lightweight drafter proposes several tokens, which the target verifies in a single forward pass, allowing multiple tokens to be accepted at once (Leviathan et al., 2023; Chen et al., 2023). Better drafts allow more tokens to be accepted per target pass, making draft quality a central factor in acceleration. To improve draft quality, recent methods reuse hidden states already computed by the target. EAGLE uses target hidden states from a single intermediate layer, while EAGLE-3 combines hidden states from multiple target layers (Li et al., 2024; 2025). Block diffusion drafters such as DFlash and DSpark similarly use hidden states from multiple target layers, projecting them into the drafter's KV context to improve the quality of parallel block prediction (Chen et al., 2026a; Cheng et al., 2026).

However, these drafters retain hidden states from accepted tokens and discard those from rejected tokens. In speculative decoding, a draft is often only partially accepted, even though verification computes representations for both accepted and rejected tokens. The substantial verifier computation used to produce rejected-token representations is therefore effectively wasted, raising a natural question: can these rejected hidden states be carried over to improve subsequent drafts?

Nevertheless, realizing this opportunity poses two distinct challenges. At inference, speedup from speculative decoding depends jointly on acceptance length and drafting overhead. The cost of projecting and attending to additional hidden states can outweigh the benefit of accepting more

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.

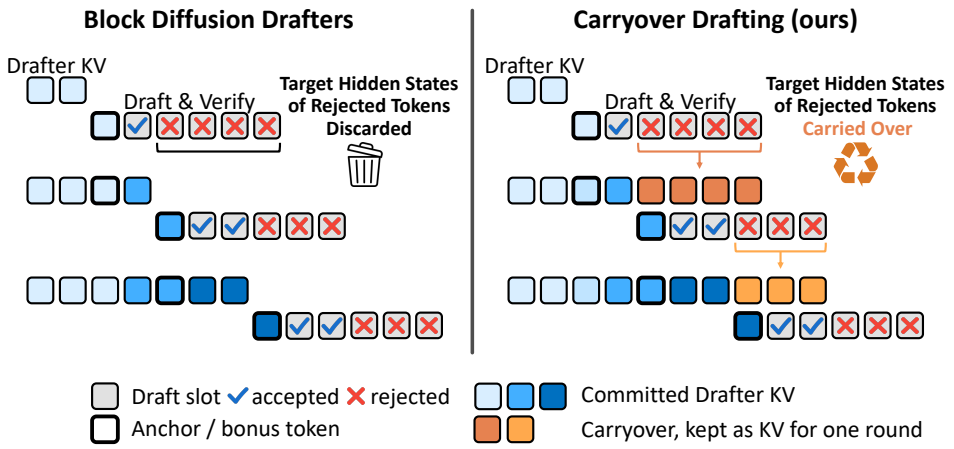

Figure 1: **Carryover Drafting.** Standard block diffusion drafting retains accepted target hidden states but discards rejected states. Carryover recycles rejected target hidden states as temporary KV context for the next proposal, enabling selective attention to rejected states.

tokens, while retaining rejected states across rounds would lead to an ever-growing KV context. During training, the drafter must learn to use the rejected hidden states that arise at inference time when the target verifies the drafter's own proposals. However, standard parallel drafter training on recorded target responses does not expose these states. A straightforward way to obtain them is through sequential rollouts, but this process would make later training positions depend on earlier draft-and-verify outcomes, sacrificing parallelism across positions.

We introduce **Carryover Drafting**, which addresses both challenges. Carryover recycles rejected target hidden states as temporary KV context for the next draft through the same projections used for accepted states as depicted in Figure 1. The drafter can then selectively attend to carried-over states through existing attention layers during the next draft. Without adding a dedicated gate or projection module, Carryover requires only a single new learned embedding to distinguish rejected states from accepted context, keeping the recycling interface lightweight. We replace the carried-over states each drafting round, bounding the additional KV context by one proposal block regardless of how much context has accumulated. Carryover therefore requires only the addition and replacement of carried-over states within the existing drafting interface, simplifying integration with standard block diffusion drafters.

To obtain inference-aligned rejected states without sequential rollouts, we introduce parallel *draft–verify–draft* training as illustrated in Figure 2. At sampled positions within a recorded target response, the drafter first generates detached proposals, which a frozen target verifies in parallel. Verification determines the next drafting position and produces the rejected hidden states that would naturally be available there at inference time. These states are then carried over to a subsequent draft, which is trained against the original recorded continuation at the shifted position, with gradients flowing only through this final drafting stage. Because each draft–verify–draft transition is constructed independently, training remains parallel across positions.

Across different target models and diverse tasks, Carryover improves both DFlash and a DSpark-derived semi-autoregressive drafter, increasing average acceptance length by up to 14.7% and end-to-end vLLM speedup by up to 14.4% over the corresponding baselines. Surprisingly, we find speedup gains reaching up to 28.8% on translation. These results demonstrate that recycling rejected hidden states already computed during inference can substantially improve subsequent drafts through a simple carryover mechanism.

2 Carryover Drafting

Carryover adds rejected target hidden states to the KV context for the next draft. We define the inference update and then show how to construct its training inputs in parallel.

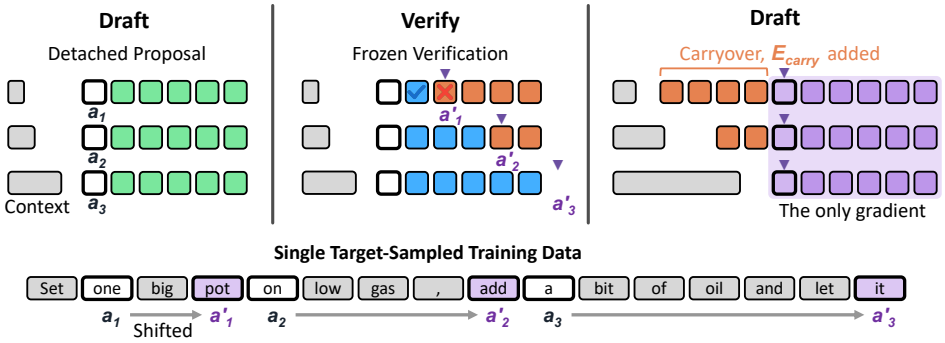

Figure 2: **Parallel draft–verify–draft training.** Randomly sampled anchors share a single recorded target response. Detached drafter proposals are verified by a frozen target; the second draft uses rejected states and recorded supervision at shifted anchors.

Preliminaries. In each round, the drafter proposes B tokens and the target accepts a prefix of length m. The target additionally supplies a token after the accepted prefix, which we refer to as the bonus token. Acceptance length τ includes that token. DFlash predicts a block from an anchor followed by masked positions (Chen et al., 2026a). Hidden states from selected target layers are concatenated, fused by a projection F, and passed to the key and value projections of every drafter layer. Draft positions attend to keys and values computed from these target hidden states by the learned drafter projections. Our DFlash and Markov baselines share this interface, which Carryover reuses for rejected states.

Acceptance and the cost of reuse. Following the per-token latency model in DFlash and DSpark (Chen et al., 2026a; Cheng et al., 2026), we account for the incremental cost of carryover. Latency L_{carry} and speedup S_{carry} are approximately

$$L_{\text{carry}} \approx \frac{t_{\text{draft}} + t_{\text{verify}} + t_{\text{reuse}}}{\tau}, \qquad S_{\text{carry}} \approx \frac{\tau\, t_{\text{AR}}}{t_{\text{draft}} + t_{\text{verify}} + t_{\text{reuse}}}.$$

Here t_{AR} is autoregressive latency per token. With $t_{\text{draft}} = T_{\text{draft}}^{\text{baseline}}$, we define $t_{\text{reuse}} = T_{\text{draft}}^{\text{Carryover}} - T_{\text{draft}}^{\text{baseline}}$ as the additional drafting latency caused by recycling rejected hidden states. Speedup therefore requires higher acceptance with low overhead. This motivates a lightweight carryover design: reusing existing projections and replacing rejected context every round keeps the added latency of recycling rejected hidden states minimal. In a matched-state profile of Qwen3-4B DFlash on an A100 GPU, Carryover adds 55 μs per drafting round, or 2.3% of baseline drafter latency (Appendix G).

2.1 One-round reuse through the existing interface

Let $H_r = (h_{r,1}, \ldots, h_{r,B})$ be the target hidden states at the proposed token positions in round r. Each $h_{r,j}$ concatenates the selected target-layer hidden states in the same order used by the baseline. If m_r proposals are accepted, define

$$H_r^{\text{acc}} = h_{r,1:m_r}, \quad H_r^{\text{rej}} = h_{r,m_r+1:B}, \quad \Phi_\ell(H) = \big(K_\ell(F(H)), V_\ell(F(H))\big), \tag{1}$$

where F is the existing projection and K_ℓ, V_ℓ are the existing projections in drafter layer ℓ. The equations show feature and KV projections; attention normalization and positional encoding are omitted. Let $C_{r+1,\ell}$ denote committed drafter KV context after the usual baseline update. Carryover constructs

$$U_{r+1,\ell} = \Phi_\ell(H_r^{\text{rej}} + E_{\text{carry}}), \qquad \text{ctx}_{r+1,\ell} = [C_{r+1,\ell}; U_{r+1,\ell}]. \tag{2}$$

A lightweight recycling interface. The embedding E_{carry} distinguishes rejected states from accepted context before projection by F. This embedding is the only module added to the baseline drafter; the drafter and its existing projections are also fine-tuned. A dedicated projector gives slightly higher acceptance in our interface probe (Section 4.3), but introduces another projection module on the drafting path.

Rejected states serve as additional read-only drafter context. They neither replace the hidden states of accepted tokens nor enter the target KV cache. Each verification replaces the previous carried-over context, and full acceptance leaves it empty. The added context is bounded by B rows per drafter layer, with storage reused from the verification window. For the full inference procedure and cache semantics, see Algorithm 1 in Appendix A.

2.2 Parallel training on recorded target responses

Following parallel block-drafter training in DFlash and DSpark, we sample random anchor positions a_i along each recorded target response y (Chen et al., 2026a; Cheng et al., 2026). Baseline and Carryover use identical target-sampled training data, with 512 anchors per response, and the same baseline-specific loss. This includes exponential position weighting, which prioritizes earlier draft positions, since later tokens can be accepted only if all preceding draft tokens are accepted. Appendix B.1 gives the loss components.

Parallel draft–verify–draft. For Carryover, we construct a local transition at each sampled anchor. The first drafter pass generates proposals without gradients. A frozen target verifies the branches in parallel using a block-diagonal causal mask: each branch reads its recorded prefix and its own proposed predecessors. Greedy verification returns accepted lengths m_i and detached rejected hidden states H_i^{rej}.

Each anchor shifts to $a_i' = a_i + m_i + 1$, the next correction or bonus position. The second drafter pass reads the recorded prefix through a_i', receives H_i^{rej} through Equation 2, and learns the recorded continuation. When all tokens are accepted, no rejected states are carried over, and the second drafter pass becomes equivalent to baseline training at the shifted anchor. We retain such cases in training.

Gradients flow only through the second drafter pass, including the shared projections and carryover embedding; the first pass and target verification are detached. Each sampled anchor supplies 1 supervised transition on the same recorded response. Training remains parallel across anchors without a full multi-round speculative rollout for every training example. For the full training procedure, see Algorithm 2 in Appendix A.

3 Experimental setup

Targets and drafters. We evaluate Qwen3-4B and Gemma 4 12B (Yang et al., 2025; Gemma Team, 2026) with 2 drafting regimes. **DFlash** predicts a block in parallel using target hidden states (Chen et al., 2026a). **Markov** adds the rank-256 first-order correction head introduced in DSpark to the parallel backbone (Cheng et al., 2026). It represents semi-autoregressive block drafting within the broader effort to strengthen within-block modeling, including Domino, xPress, and DFlash 2 (Huang et al., 2026; Wang et al., 2026; Inco AI, 2026). Comparing these regimes tests whether carryover remains beneficial with a stronger drafter that models dependencies between proposed tokens. Both use 5 drafter layers and 15 proposed tokens; confidence scheduling is disabled to keep verification length fixed.

Training configuration. All variants use the same 500K prompts from UltraChat-200K (Hugging Face H4, 2023; Ding et al., 2023) and Magpie-Llama-3.1-Pro-300K-Filtered (Magpie-Align, 2024; Xu et al., 2025), following the public speculator data recipe (Amazon Web Services, 2026). Each target regenerates the responses, which its Baseline and Carryover variants share. We train all baselines for 10 epochs following DSpark. This choice follows the convergence rationale reported in DSpark: all compared drafters, including DFlash, were trained for 10 epochs to ensure full convergence (Cheng et al., 2026). This exceeds the 6 epochs reported in the original DFlash recipe (Chen et al., 2026a). Carryover then trains for 3 epochs from the 10-epoch baseline checkpoint, using the same recorded responses. Appendix B.1 gives the shared training settings.

Evaluation. The Red Hat speculator benchmark contains 844 prompts: 164 for HumanEval, 200 for tool calling, and 80 each for Math, MT-Bench, QA, RAG, summarization, and translation (Red Hat AI, 2026). We evaluate at temperature $T = 0$ and $T = 1$ using vLLM on 1 H200 GPU (Kwon et al., 2023). Acceptance length includes bonus tokens from verification. Baseline–Carryover pairs share

Table 1: **Acceptance and speedup over autoregressive decoding.** Per-dataset speedup uses offline-vLLM, and Avg. speedup is a separate online-vLLM measurement. Avg. acceptance is micro average over 844 prompts.

			T=0		T=1	
			τ	Speedup	τ	Speedup
Target	Drafter	Dataset	Baseline | **+Carryover**	Baseline | **+Carryover**	Baseline | **+Carryover**	Baseline | **+Carryover**
Qwen3-4B	DFlash	HumanEval	4.89 | **5.33**(+9.0%)	4.74× | **5.17**×(+9.0%)	4.22 | **4.53**(+7.4%)	3.97× | **4.53**×(+14.0%)
		Math	5.96 | **6.51**(+9.2%)	5.58× | **6.10**×(+9.4%)	5.31 | **5.64**(+6.1%)	5.09× | **5.34**×(+4.9%)
		MT-Bench	3.45 | **3.77**(+9.1%)	3.40× | **3.72**×(+9.5%)	2.98 | **3.18**(+6.5%)	2.72× | **2.95**×(+8.6%)
		QA	3.03 | **3.26**(+7.7%)	2.96× | **3.15**×(+6.5%)	2.61 | **2.84**(+8.9%)	2.60× | **2.64**×(+1.5%)
		RAG	3.19 | **3.48**(+9.2%)	3.19× | **3.49**×(+9.5%)	2.96 | **2.99**(+1.1%)	2.75× | **2.83**×(+3.1%)
		Summ.	2.83 | **3.15**(+11.3%)	2.72× | **2.97**×(+9.3%)	2.52 | **2.72**(+8.1%)	2.63× | **2.80**×(+6.6%)
		Tool	3.50 | **3.84**(+10.0%)	3.39× | **3.78**×(+11.5%)	3.13 | **3.40**(+8.6%)	3.11× | **3.20**×(+2.8%)
		Transl.	2.57 | **3.14**(+22.1%)	2.12× | **2.41**×(+13.8%)	2.26 | **2.67**(+18.0%)	1.89× | **2.18**×(+15.3%)
		Avg.	3.78 | **4.14**(+9.4%)	3.34× | **3.60**×(+7.9%)	3.33 | **3.55**(+6.5%)	2.89× | **3.14**×(+8.6%)
	Markov	HumanEval	5.75 | **6.30**(+9.6%)	5.02× | **5.47**×(+9.0%)	5.36 | **5.69**(+6.2%)	4.72× | **4.87**×(+3.1%)
		Math	7.29 | **7.91**(+8.4%)	6.03× | **6.52**×(+8.1%)	6.63 | **7.42**(+11.8%)	5.54× | **6.15**×(+11.0%)
		MT-Bench	3.91 | **4.37**(+11.6%)	3.47× | **3.84**×(+10.7%)	3.68 | **4.08**(+10.8%)	3.09× | **3.67**×(+18.9%)
		QA	3.42 | **3.80**(+11.2%)	2.98× | **3.30**×(+10.8%)	3.32 | **3.45**(+4.0%)	2.75× | **2.98**×(+8.5%)
		RAG	3.64 | **4.07**(+11.9%)	3.12× | **3.47**×(+11.1%)	3.44 | **3.79**(+10.1%)	3.00× | **3.30**×(+9.9%)
		Summ.	3.14 | **3.63**(+15.4%)	2.66× | **3.06**×(+15.0%)	2.99 | **3.41**(+13.9%)	2.68× | **3.08**×(+14.8%)
		Tool	3.96 | **4.49**(+13.5%)	3.48× | **3.93**×(+12.7%)	3.70 | **4.22**(+13.8%)	3.23× | **3.56**×(+10.5%)
		Transl.	2.84 | **3.61**(+26.8%)	2.00× | **2.54**×(+27.3%)	2.65 | **3.41**(+28.6%)	1.94× | **2.50**×(+28.8%)
		Avg.	4.33 | **4.85**(+11.8%)	3.44× | **3.83**×(+11.5%)	4.06 | **4.48**(+10.3%)	3.20× | **3.54**×(+10.6%)
Gemma 4 12B	DFlash	HumanEval	4.86 | **5.28**(+8.7%)	3.65× | **4.02**×(+10.0%)	4.23 | **4.50**(+6.3%)	2.98× | **3.18**×(+6.7%)
		Math	5.44 | **6.03**(+10.8%)	3.97× | **4.35**×(+9.6%)	4.70 | **5.24**(+11.5%)	3.49× | **3.94**×(+13.1%)
		MT-Bench	3.25 | **3.57**(+10.1%)	2.49× | **2.66**×(+7.0%)	2.85 | **3.09**(+8.2%)	2.18× | **2.38**×(+9.5%)
		QA	2.87 | **3.15**(+9.8%)	2.23× | **2.44**×(+9.1%)	2.41 | **2.57**(+6.9%)	1.84× | **1.96**×(+6.8%)
		RAG	3.63 | **3.99**(+9.9%)	2.68× | **2.93**×(+9.7%)	3.27 | **3.62**(+10.8%)	2.41× | **2.73**×(+13.1%)
		Summ.	2.72 | **3.04**(+12.0%)	2.14× | **2.34**×(+9.4%)	2.47 | **2.70**(+9.3%)	1.96× | **2.07**×(+5.6%)
		Tool	2.94 | **3.24**(+10.3%)	2.16× | **2.37**×(+9.9%)	2.78 | **3.06**(+10.1%)	2.16× | **2.30**×(+6.7%)
		Transl.	3.14 | **3.60**(+14.6%)	2.44× | **2.75**×(+12.5%)	2.78 | **3.15**(+13.0%)	2.11× | **2.44**×(+15.7%)
		Avg.	3.65 | **4.03**(+10.3%)	2.58× | **2.79**×(+8.1%)	3.24 | **3.52**(+8.6%)	2.40× | **2.59**×(+7.9%)
	Markov	HumanEval	5.82 | **6.43**(+10.3%)	4.18× | **4.57**×(+9.2%)	5.46 | **5.98**(+9.5%)	3.74× | **4.03**×(+7.8%)
		Math	6.41 | **7.18**(+12.1%)	4.37× | **4.85**×(+10.9%)	6.06 | **6.87**(+13.2%)	4.34× | **4.64**×(+6.9%)
		MT-Bench	3.59 | **4.17**(+16.2%)	2.60× | **2.99**×(+15.2%)	3.44 | **3.92**(+14.0%)	2.46× | **2.87**×(+16.7%)
		QA	3.14 | **3.64**(+15.9%)	2.31× | **2.65**×(+15.0%)	2.81 | **3.40**(+21.0%)	2.06× | **2.22**×(+7.6%)
		RAG	4.16 | **4.73**(+13.7%)	2.89× | **3.22**×(+11.5%)	3.96 | **4.35**(+10.0%)	2.66× | **2.94**×(+10.7%)
		Summ.	3.01 | **3.51**(+16.6%)	2.22× | **2.54**×(+14.4%)	2.91 | **3.34**(+15.0%)	2.16× | **2.52**×(+16.8%)
		Tool	3.25 | **3.75**(+15.2%)	2.26× | **2.54**×(+12.5%)	3.29 | **3.69**(+12.2%)	2.34× | **2.63**×(+12.4%)
		Transl.	3.51 | **4.18**(+19.1%)	2.55× | **3.02**×(+18.2%)	3.22 | **3.96**(+23.2%)	2.30× | **2.78**×(+21.1%)
		Avg.	4.13 | **4.74**(+14.7%)	2.73× | **3.13**×(+14.4%)	3.92 | **4.49**(+14.5%)	2.71× | **3.09**×(+14.3%)

evaluation settings. Average speedup is measured on the combined benchmark workload, separately from dataset speedups; Appendix B.2 specifies workloads and aggregation.

4 RESULTS

4.1 ACCEPTANCE ACROSS TASKS AND DRAFTING REGIMES

We first test whether rejected hidden states improve both drafting regimes across targets. Table 1 shows acceptance gains in every evaluated dataset setting, with average acceptance-length improvements of 6.5–14.7%. The gains persist when moving from fully parallel DFlash to semi-autoregressive Markov. Carryover therefore complements the within-block dependency modeling of a stronger drafter.

The task breakdown reveals where recycling helps most. Translation has the largest relative acceptance gain in every target–drafter–temperature combination, reaching 28.6% for Qwen3-4B Markov at $T = 1$. This variation motivates the rejected-branch analysis in Section 5.2: does carryover help more when discarded states retain more recoverable information? Figure 3 compares acceptance length across datasets at $T = 0$; Appendix F reports $T = 1$.

4.2 END-TO-END SPEEDUP AND THROUGHPUT

Higher acceptance is useful only if it compensates for the cost of recycling. The average speedup measurements in Table 1 include projection and attention over carried-over states: Carryover improves

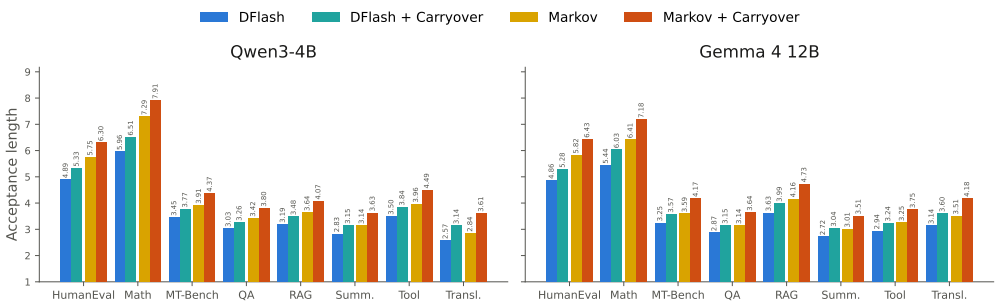

Figure 3: **Acceptance length at $T = 0$.** Baseline and Carryover for DFlash and Markov with Qwen3-4B and Gemma 4 12B across 8 datasets. The full measurements are in Table 1.

speedup over autoregressive decoding by 7.9–14.4% relative to the corresponding baseline. Thus the acceptance gains survive the added drafting overhead.

To evaluate speedup across serving loads, we vary concurrency from 1 to 32 (Table 2). As concurrency increases, both Baseline and Carryover provide smaller speedups over autoregressive decoding, but Carryover maintains relative gains over Baseline for both targets and drafting regimes. These experiments use fixed verification lengths. DSpark instead uses confidence scheduling to reduce verification cost and sustain speedup at higher concurrency (Cheng et al., 2026). Section 6 discusses how carryover could complement such adaptive scheduling.

4.3 Carryover interface ablation

What should be recycled? A rejected branch offers several forms of information for the next draft. Table 3(a) compares carrying over draft-emitted tokens, target-token predictions, and target hidden states. Token hints initialize the next proposal in place of `[MASK]` tokens. This Qwen3-4B Markov probe uses 6 training epochs on approximately 40K examples and evaluates HumanEval, Math, and MT-Bench. Draft-emitted token hints give inconsistent gains, while target-token hints yield modest improvements. Recycling target hidden states gives the largest gains among these designs. However, token hints enter through the initial proposal embeddings, whereas carried hidden states enter as KV context accessed by existing attention layers. Thus, this comparison changes both the information carried over and how it enters the drafter; it does not completely isolate their individual contributions.

How should rejected states be distinguished? Rejected hidden states describe an alternative continuation, motivating an explicit distinction from accepted context. Table 3(b) compares no explicit distinction, a separate projector, and the carryover embedding. Reusing rejected states without an explicit distinction from accepted context remains competitive, while the 19.7M-parameter projector gives slightly higher acceptance than the embedding. We choose the single embedding because it retains nearly the same acceptance without an additional projection module.

5 How do rejected states contribute to drafting?

In this section, we first test the effect of rejected states at matched decoding positions and then examine why gains vary across tasks.

5.1 Target-token recall at matched anchors

Controlling the decoding point. Even from the same context, different drafter rollouts can reach different verification boundaries. Therefore, we generate with Carryover ON and replay its draft starting points with 3 variants: Carryover ON, the same weights without carried-over states (OFF), and Baseline. All variants share committed tokens, accepted target hidden states, and target KV cache; only ON receives the recorded rejected states. Each proposal is scored under its own within-block prefix. We use the final Qwen3-4B Markov checkpoints on all 844 prompts at $T = 0$.

Recall@k measures how often the top target token appears among the top k draft candidates. We report Recall@1 and Recall@16 to assess the top prediction and coverage of the candidate set,

Table 2: **Throughput across concurrency.** Output tokens/s on the mixed 844-prompt workload at $T = 0$ on a single H200 GPU.

Target	c	Throughput (tok/s)				
		AR	DFlash		Markov	
			Baseline	**+Carryover**	Baseline	**+Carryover**
Qwen3-4B	1	251	837	**903**(+7.9%)	861	**961**(+11.5%)
	2	491	1,609	**1,739**(+8.1%)	1,652	**1,840**(+11.4%)
	4	946	3,015	**3,275**(+8.6%)	3,097	**3,443**(+11.2%)
	8	1,800	5,328	**5,741**(+7.7%)	5,337	**5,879**(+10.1%)
	16	3,181	7,243	**7,785**(+7.5%)	7,499	**8,269**(+10.3%)
	32	5,335	8,180	**8,804**(+7.6%)	8,608	**9,402**(+9.2%)
Gemma 4 12B	1	111	286	**309**(+8.2%)	304	**347**(+14.4%)
	2	216	553	**594**(+7.3%)	584	**671**(+14.9%)
	4	419	1,035	**1,109**(+7.2%)	1,097	**1,238**(+12.9%)
	8	785	1,821	**1,935**(+6.3%)	1,908	**2,145**(+12.4%)
	16	1,388	2,455	**2,650**(+8.0%)	2,573	**2,892**(+12.4%)
	32	2,297	2,789	**3,000**(+7.6%)	2,973	**3,315**(+11.5%)

Table 3: **Carryover-interface ablation.** Qwen3-4B Markov diagnostic probe at $T = 0$. Cells report τ (gain over baseline, %).

Variant	HumanEval	Math	MT-Bench
Baseline (nothing carried)	3.10	3.72	2.46
(a) Carried-over information			
Draft-emitted tokens	3.05 (-1.8%)	3.76 (+1.2%)	2.41 (-2.4%)
Target-emitted tokens	3.16 (+1.8%)	3.93 (+5.8%)	2.52 (+2.3%)
Target top-5 weighted	3.18 (+2.7%)	3.92 (+5.4%)	2.56 (+4.0%)
Target hidden states (ours)	**3.41** (+10.0%)	**4.27** (+14.8%)	**2.70** (+9.6%)
(b) Distinguishing carried-over hidden states			
No explicit discrimination	3.41 (+10.1%)	4.18 (+12.5%)	2.67 (+8.5%)
Separate projector	3.45 (+11.4%)	4.28 (+15.2%)	2.73 (+10.8%)
Carryover embedding (ours)	**3.41** (+10.0%)	**4.27** (+14.8%)	**2.70** (+9.6%)

following the analysis in DFlash 2 (Inco AI, 2026). All variants use matched rounds at each position (Appendix B.2).

Improved predictions from carried-over states. ON exceeds OFF at every displayed position by 1.28–5.60 Recall@1 points and 0.29–1.89 Recall@16 points (Figure 4). Supplying rejected states thus improves both the top prediction and coverage of the target token within the candidate set. Gains begin at position 0, where all variants share the same accepted anchor, and persist across the block.

ON–OFF indirectly tests the learned use of rejected states: OFF replays ON starting points rather than its own rollout and withholds context the model was trained to use. Appendix C complements this diagnostic by showing attention to carried states (Figure 6).

5.2 Why do gains differ across datasets?

Carryover improves acceptance length by different amounts across datasets, reaching 28.6% on translation at $T = 1$. To investigate this variation, we examine which properties of rejected states are associated with larger gains and how these properties differ across datasets. At $T = 0$, we compare rejected-state counts, consecutive and total proposal–verifier matches after rejection, and *re-convergence*. State count measures how much context is available; acceptance after rejection measures how often draft tokens are accepted along the rejected trajectory. Re-convergence measures whether predictions from the rejected trajectory match the eventual continuation. For clarity, we define the re-convergence count as follows. For a rejected suffix of length L, let z_q be the verifier's top-1 next-token prediction from its qth rejected state, and y_q the eventually committed token at the

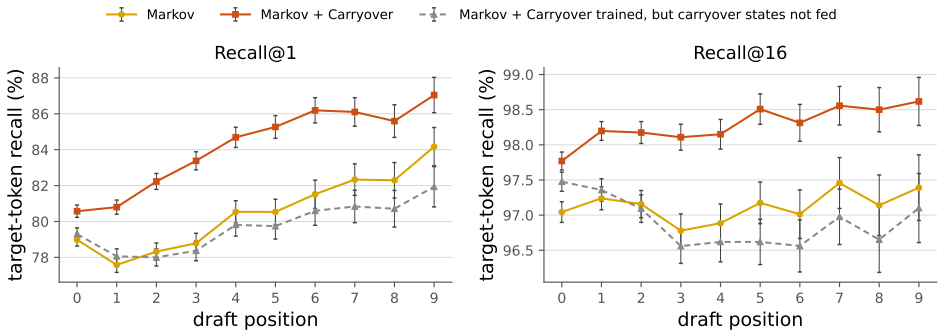

Figure 4: **Target-token recall from identical decoding anchors.** Qwen3-4B Markov on 844 prompts at $T = 0$: Baseline, Carryover with states (ON), and the same Carryover weights without them (OFF). Appendix B.2 details the sampling and error bars.

corresponding predicted position. We count

$$R = \sum_{q=0}^{L-1} \mathbf{1}[\exists\, \delta \in \{-1, 0, +1\} : z_{q+\delta} = y_q]\,, \tag{3}$$

ignoring out-of-range indices. The one-position tolerance allows a local insertion or deletion. R measures retrospective agreement with the eventual continuation; it is never used to select carried-over states or supervise training.

Figure 5 shows that Carryover gains are more closely associated with re-convergence than with rejected-state count or post-rejection acceptance (Appendix D). We distinguish post-rejection acceptance from re-convergence: a token may be accepted along the rejected branch yet differ from the token eventually committed by the target. Altogether, these results suggest that the usefulness of rejected states depends more on their predictiveness of the target's eventual continuation than on either post-rejection acceptance or rejected-suffix length.

Across drafters, we also find that Markov produces more re-converged positions than DFlash on all 8 datasets (2.76 versus 2.51 per rejected round) and obtains larger Carryover gains on 7 of 8 (Appendix E). Higher re-convergence thus offers one explanation for the larger gains of Markov. It also suggests that stronger drafters leave more useful rejected states, so the benefit of Carryover can grow as drafters improve.

6 Discussion

Confidence scheduling with Carryover. DSpark uses estimated acceptance probabilities together with hardware capacity to adapt the verification length (Cheng et al., 2026). Carryover is compatible in principle with adaptive verification, but the scheduling policy changes which rejected states are available to the next draft. A joint evaluation would therefore require training Carryover under the same scheduling policy used at inference, so that its rejected-state inputs match the deployment distribution. In addition, because Carryover changes the drafter's predictions, the confidence estimates and scheduling thresholds may require recalibration. We therefore do not evaluate Carryover with confidence scheduling in this work.

Tree decoding with Carryover. Tree verification methods such as Medusa and DDTree spend more compute on alternative paths to increase acceptance length (Cai et al., 2024; Ringel & Romano, 2026). We anticipate the combination of Carryover with tree verification to be especially interesting: the additional verification produces both more rejected states and different information across branches, which gives subsequent drafts more opportunity for reuse. It also introduces an additional design question: which branch states should be retained for the next draft? Applying Carryover to tree decoding would therefore require matching the tree-construction policy between training and inference and determining how states from multiple rejected branches should be selected or combined. We leave this extension to future work.

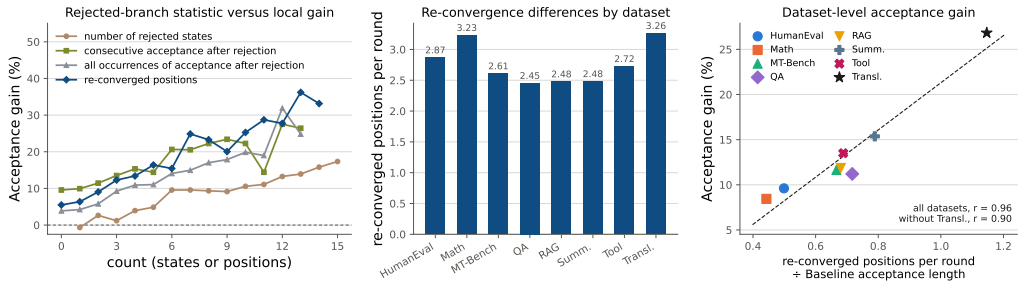

Figure 5: **Rejected-state statistics and Carryover gains.** Final Qwen3-4B Markov at $T = 0$. Left: local replay gain grouped by rejected-branch statistics. Center: re-converged positions per rejected round. Right: normalized re-convergence versus final dataset-level acceptance gain.

On-policy training with rollouts. Our parallel draft–verify–draft procedure is designed to obtain inference-aligned rejected states while retaining parallel training on recorded responses. When training already includes target-verified rollouts, however, rejected states arise naturally during verification. For example, Draft-OPD uses target-assisted rollouts and replays drafting at verification-exposed error positions (Lei et al., 2026). Carryover could reuse the hidden states produced by these verification steps without requiring an additional target verification stage. Such rollouts would also make it possible to train Carryover with retention beyond a single round, which our parallel training does not cover.

7 Related Work

Speculative decoding with target hidden states. EAGLE and EAGLE-3 use accepted target hidden states for autoregressive drafting, while block diffusion drafters such as DFlash use accepted target hidden states as projected KV context for parallel block prediction (Li et al., 2024; 2025; Chen et al., 2026a). To strengthen dependencies among tokens proposed in parallel, Domino, DSpark, xPress, DFlash 2, and LiLiCorr refine parallel drafts through causal correction, iterative refinement, or candidate reranking (Huang et al., 2026; Cheng et al., 2026; Wang et al., 2026; Inco AI, 2026; Rusanovsky et al., 2026). Carryover provides a complementary mechanism applicable to both DFlash and DSpark-derived drafters.

Recycling speculative computation. Drafting and verification expose intermediate computations that can be reused in subsequent rounds. Token Recycling caches target candidates, Lookahead reuses n-grams from Jacobi decoding, and Ouroboros reuses phrases to improve acceptance (Luo et al., 2025; Fu et al., 2024; Zhao et al., 2024). At the representation level, SD^2 learns a verifier-derived steering vector for an independently pretrained autoregressive drafter and injects it through MLPs (Berdoz et al., 2026), while Lyanna recycles drafter hidden states into token trees (Chen et al., 2026b). Carryover instead recycles rejected target hidden states as a temporary KV cache, allowing the drafter to selectively attend to them through its attention layers.

8 Conclusion

Carryover Drafting turns discarded verifier computation on rejected tokens into useful context for subsequent drafts. Rejected target hidden states are reused for one round through the existing KV interface with a single learned embedding, keeping the additional context bounded to one proposal block, while parallel draft–verify–draft training constructs the corresponding inputs without sequential rollouts. Across diverse settings, Carryover consistently improves acceptance length and end-to-end speedup, with larger gains when rejected branches remain predictive of the eventual continuation generated by the target model.

Together, these results suggest an opportunity to revisit computation typically discarded during speculative decoding. Carryover provides one concrete example: rejected verifier states, despite corresponding to rejected tokens, can remain useful for subsequent drafts. More broadly, this raises the question of whether other intermediate computations discarded during inference can also be selectively and efficiently reused.

AI USE STATEMENT

In this work, we used generative AI tools to provide feedback on research methodology and experiments, assist with method implementation, support qualitative and thematic data analysis, and assist in interpreting experimental results. We did not use generative AI tools to assist with translation or to clean and reformat datasets. The following required disclosure tasks were not applicable to this work: generating synthetic datasets, developing theoretical models or conceptual frameworks, formulating mathematical claims, providing critical ingredients for proving mathematical claims, assisting in the writing of proofs, and proposing or refining hypotheses.

Additionally, we used generative AI tools to draft parts of the manuscript and to edit the manuscript for readability. All AI-assisted research, implementation, analysis, and writing were reviewed by at least two authors. AI-assisted implementation was inspected and manually tested for correctness. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

REFERENCES

Amazon Web Services. GPT-OSS-120B P-EAGLE model card. Hugging Face model card, 2026. URL https://huggingface.co/amazon/gpt-oss-120b-p-eagle.

Frédéric Berdoz, Peer Rheinboldt, and Roger Wattenhofer. Steering pretrained drafters during speculative decoding. In *AAAI Conference on Artificial Intelligence*, pp. 30067–30075, 2026. doi: 10.1609/aaai.v40i36.40255.

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 *International Conference on Machine Learning*, pp. 5209–5235, 2024.

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

Jian Chen, Yesheng Liang, and Zhijian Liu. DFlash: Block diffusion for flash speculative decoding. In *International Conference on Machine Learning*, 2026a.

Yuetao Chen, Xuliang Wang, Xinzhou Zheng, Ming Li, Peng Wang, and Hong Xu. Make every draft count: Hidden state based speculative decoding. *arXiv preprint arXiv:2602.21224*, 2026b.

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

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 Houda Bouamor, Juan Pino, and Kalika Bali (eds.), *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pp. 3029–3051, Singapore, December 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.emnlp-main.183. URL https://aclanthology.org/2023.emnlp-main.183/.

Yichao Fu, Peter Bailis, Ion Stoica, and Hao Zhang. Break the sequential dependency of LLM inference using lookahead decoding. In *International Conference on Machine Learning*, 2024.

Gemma Team. Gemma 4 technical report. *arXiv preprint arXiv:2607.02770*, 2026.

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.

Hugging Face H4. HuggingFaceH4/ultrachat_200k. Hugging Face dataset, 2023. URL https://huggingface.co/datasets/HuggingFaceH4/ultrachat_200k. train_sft split.

Inco AI. DFlash 2: Keep drafting parallel. https://inco.ai/blog/dflash2/, August 2026.

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 SIGOPS Symposium on Operating Systems Principles*, 2023.

Haodi Lei, Yafu Li, Haoran Zhang, Shunkai Zhang, Qianjia Cheng, Xiaoye Qu, Ganqu Cui, Bowen Zhou, Ning Ding, Yun Luo, and Yu Cheng. Draft-OPD: On-policy distillation for speculative draft models. *arXiv preprint arXiv:2605.29343*, 2026. URL https://arxiv.org/abs/2605.29343.

Yaniv Leviathan, Matan Kalman, and Yossi Matias. Fast inference from transformers via speculative decoding. In *International Conference on Machine Learning*, 2023.

Yuhui Li, Fangyun Wei, Chao Zhang, and Hongyang Zhang. EAGLE: Speculative sampling requires rethinking feature uncertainty. In *International Conference on Machine Learning*, 2024.

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*, 2025.

Xianzhen Luo, Yixuan Wang, Qingfu Zhu, Zhiming Zhang, Xuanyu Zhang, Qing Yang, and Dongliang Xu. Turning trash into treasure: Accelerating inference of large language models with token recycling. In *Annual Meeting of the Association for Computational Linguistics*, pp. 6816–6831, 2025. doi: 10.18653/v1/2025.acl-long.338.

Magpie-Align. Magpie-Align/Magpie-Llama-3.1-Pro-300K-Filtered. Hugging Face dataset, 2024. URL https://huggingface.co/datasets/Magpie-Align/Magpie-Llama-3.1-Pro-300K-Filtered.

Red Hat AI. Speculator benchmarks. Hugging Face dataset, 2026. URL https://huggingface.co/datasets/RedHatAI/speculator_benchmarks. Revision 2ae86affa2cb97a972b7fc681dd51c04fbff083e.

Liran Ringel and Yaniv Romano. Accelerating speculative decoding with block diffusion draft trees. *arXiv preprint arXiv:2604.12989*, 2026.

Matan Rusanovsky, Yoav Miron, Roy Uziel, Omer Belhasin, Ran Zilberstein, Maor Ashkenazi, and Michael Elad. LiLiCorr: Lightweight likelihood correlation of parallel drafts for speculative decoding, 2026. URL https://arxiv.org/abs/2608.20530.

Zheng Wang, Davis Wertheimer, Yu Chin Fabian Lim, Mudhakar Srivatsa, Raghu K. Ganti, Minjia Zhang, and Naigang Wang. xPress: Parallel refinement for diffusion drafters in speculative decoding. *arXiv preprint arXiv:2608.02438*, 2026.

Zhangchen Xu, Fengqing Jiang, Luyao Niu, Yuntian Deng, Radha Poovendran, Yejin Choi, and Bill Yuchen Lin. Magpie: Alignment data synthesis from scratch by prompting aligned LLMs with nothing. In *International Conference on Learning Representations*, 2025. URL https://arxiv.org/abs/2406.08464.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, et al. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025.

Weilin Zhao, Yuxiang Huang, Xu Han, Wang Xu, Chaojun Xiao, Xinrong Zhang, Yewei Fang, Kaihuo Zhang, Zhiyuan Liu, and Maosong Sun. Ouroboros: Generating longer drafts phrase by phrase for faster speculative decoding. In *Empirical Methods in Natural Language Processing*, pp. 13378–13393, 2024. doi: 10.18653/v1/2024.emnlp-main.742.

A INFERENCE AND TRAINING ALGORITHMS

Algorithms 1 and 2 compare the baseline and Carryover directly. At inference, the change is to collect all proposal states, add the learned embedding only to the rejected portion, and apply the existing projections. At training time, the same fixed response supplies supervision at anchors shifted by a parallel draft–verify step. The added embedding is shown in orange.

A.1 INFERENCE UPDATE

Algorithm 1 Baseline and Carryover: the post-verification update

Given: target features $H_{1:B}$ at the proposed positions, accepted length m, and existing committed drafter KV C_ℓ. Both methods use the usual draft–verify procedure and commit the accepted prefix plus correction or bonus with the usual target KV update.

Baseline: collect accepted states

1. $H^{\text{keep}} \leftarrow H_{1:m}$
2. $Z \leftarrow F(H^{\text{keep}})$
3. $J_\ell \leftarrow (K_\ell(Z), V_\ell(Z))$
4. $C_\ell \leftarrow [C_\ell; J_\ell]$
5. $\text{ctx}_\ell \leftarrow C_\ell$

Carryover: collect all states

1. $H^{\text{keep}} \leftarrow H_{1:B}$
2. $H^{\text{keep}}_{m+1:B} \leftarrow H^{\text{keep}}_{m+1:B} + E_{\text{carry}}$
3. $Z \leftarrow F(H^{\text{keep}})$
4. $J_\ell \leftarrow (K_\ell(Z), V_\ell(Z))$
5. $C_\ell \leftarrow [C_\ell; J_{\ell,1:m}]$
6. $U_\ell \leftarrow J_{\ell,m+1:B}$ (Replace old U_ℓ)
7. $\text{ctx}_\ell \leftarrow [C_\ell; U_\ell]$

Apply the KV steps at every drafter layer ℓ. The next draft reads ctx_ℓ. Initialize $U_\ell = \varnothing$; full acceptance also leaves it empty.

Cache semantics. The two columns show the same feature-processing path, with a different choice of retained rows. F is the existing projection and K_ℓ, V_ℓ are the layer-specific key and value projections; J_ℓ denotes their output KV rows. Accepted rows extend committed drafter context C_ℓ. Rejected rows are exposed only as read-only drafter context U_ℓ for one next proposal. They are collected before the verification workspace is reused and never enter the target's committed KV cache. Stopping rules and output truncation follow the baseline. Each rejected state keeps the absolute position of the token at which it was computed, and RoPE is applied once, when the drafter projects it into keys.

Why verification remains valid. Conditioned on the decoding history and carried input, the drafter defines a proposal distribution q. Applying the usual target acceptance and residual correction with that actual q preserves the target distribution, just as for other history-dependent proposals. Carryover changes this proposal distribution but neither the target probabilities nor the acceptance rule. Rejected text remains an input to the drafter only; allowing it to overwrite committed target KV would violate that separation.

A.2 PARALLEL TRAINING UPDATE

Algorithm 2 Parallel training: shared setup and alternative update rules

Given: one fixed target-sampled response y per prompt and its recorded target features; frozen target P; drafter D_θ; proposal length B; baseline loss $\mathcal{L}_{\text{base}}$. The prompt is implicit in each prefix of y.
Existing interface: $\Phi_\ell(H) = (K_\ell(F(H)), V_\ell(F(H)))$ at every drafter layer ℓ. Only the additional embedding E_{carry} is shown in orange.
Shared setup

1. $\{a_i\} \leftarrow \text{RandomAnchors}(y, \text{count} = 512)$ (Shared random-anchor sampler)
2. Use recorded prefixes, target features, and continuation labels from the same y.

Apply either the Baseline update or the Carryover update below.
Baseline update With gradients; all anchors in parallel

1. $\widehat{y}_i \leftarrow \text{Draft}(D_\theta, y_{\le a_i})$
2. Update θ with $\mathcal{L}_{\text{base}}$ against recorded continuations $y_{>a_i}$.

Carryover update
Draft–verify: no gradients; all initial anchors in parallel

1. $d_i \leftarrow \text{GreedyDraft}(D_\theta, y_{\le a_i})$
2. $(m_i, H_i) \leftarrow \text{GreedyVerify}(P, y_{\le a_i}, d_i)$ (Frozen target; isolated branches)
3. $a'_i \leftarrow a_i + m_i + 1; \quad H_i^{\text{rej}} \leftarrow \text{sg}(H_{i,m_i+1:B})$
4. Keep every sampled anchor; if $m_i = B$, set $H_i^{\text{rej}} = \varnothing$ and use empty carried-over context.

Second draft: with gradients; all sampled anchors in parallel

5. $U_{i,\ell} \leftarrow \Phi_\ell(H_i^{\text{rej}} + E_{\text{carry}})$
6. $\widehat{y}_i \leftarrow \text{Draft}(D_\theta, y_{\le a'_i}, U_i)$ (Recorded prefix and features)
7. Update θ and E_{carry} with $\mathcal{L}_{\text{base}}$ against recorded continuations $y_{>a'_i}$.

One fixed response, shifted anchors. Both procedures reuse the same target-sampled response y and randomly sampled anchors, following parallel training in DFlash and DSpark (Chen et al., 2026a; Cheng et al., 2026). Our experiments sample 512 initial anchors per response. Here a_i denotes an anchor and m_i the number of accepted proposals. Carryover moves each anchor to $a'_i = a_i + m_i + 1$, the next correction or bonus position on that same response. The second draft reads the recorded prefix and recorded target features at a'_i and learns its recorded continuation. Verification supplies rejected-state inputs and the shift; it does not sample a new target response or replace the labels with a newly generated continuation. Each initial anchor supplies 1 supervised draft, including when the proposal is fully accepted.

Parallelism and gradients. The draft–verify–draft stages are sequential within each local transition, but all anchors are processed in parallel within a stage. A block-diagonal causal mask isolates target branches: each reads its recorded prefix and its own proposed predecessors. The target partial prefill may be chunked without making one anchor depend on another. The first drafter pass and target verification are detached. The second drafter pass, conditioned on carried-over states when present, trains the drafter, shared projections Φ_ℓ from Equation 1, and embedding using the corresponding baseline objective.

Fully accepted proposals. All sampled anchors contribute to the second-stage loss. When the first proposal is fully accepted, $H_i^{\text{rej}} = \varnothing$ and the second drafter pass uses only the recorded context at the shifted anchor. This is the corresponding baseline training procedure at the shifted anchor, so fully accepted proposals still provide supervision.

B TRAINING AND EVALUATION PROTOCOL

B.1 TRAINING DATA AND SETTINGS

Table 4: **Training settings.** Shared architecture and optimizer settings in the final release configuration; Carryover continues the corresponding Baseline-10 checkpoint. Target-layer IDs are zero-based transformer-block indices.

Setting	Qwen3-4B	Gemma 4 12B
Target feature layers	1, 9, 17, 25, 33	5, 17, 29, 41, 46
Drafter layers / proposed tokens	5 / 15	5 / 15
Markov correction rank	256	256
Initial anchors / response	512	512
Maximum training sequence length	4,096	4,096
Global batch size	512	512
Learning rate / warmup ratio	6×10^{-4} / 0.04	6×10^{-4} / 0.04
Precision / gradient clipping	bfloat16 / 1.0	bfloat16 / 1.0
Baseline / Carryover training epochs	10 / 3	10 / 3

Recorded supervision. The 500K-prompt mixture comprises UltraChat-200K and Magpie-Llama-3.1-Pro-300K-Filtered (Ding et al., 2023; Xu et al., 2025). Each target regenerates one response per prompt following the public speculator recipe (Amazon Web Services, 2026); its Baseline and Carryover variants share these responses.

Shared settings. Baseline and Carryover use the same architecture, target-feature layers, initial-anchor sampler (512 per response), and loss. The second drafter pass retains the objective of the corresponding baseline. As in DFlash and DSpark, loss weights decay exponentially with position inside the draft block, prioritizing early predictions because rejection stops acceptance of the suffix (Chen et al., 2026a; Cheng et al., 2026). DFlash uses position-weighted cross-entropy; the Markov baseline additionally uses distribution-matching and confidence objectives from the DSpark training recipe. Disabling confidence scheduling affects the inference policy, not these training objectives.

Continuation. Carryover continues the 10-epoch baseline checkpoint for 3 epochs (Table 4), with fresh optimizer and scheduler states. Its second-stage loss uses the shifted anchors in Algorithm 2. Detached proposal generation and frozen-target partial prefill add computation while preserving parallelism across anchors. The recorded target responses remain fixed.

B.2 EVALUATION PROTOCOLS

Table 5: **Evaluation protocols.** Offline and online measurements answer different questions and are not averaged together. All output caps count generated tokens.

	Dataset results	Average speedup	Concurrency sweep
Engine	Offline vLLM	Online vLLM	Online vLLM
Workload	Each dataset	Mixed 844 prompts	Mixed 844 prompts
Batch / client concurrency	Batch size 1	Concurrency 1	1, 2, 4, 8, 16, 32
Output cap	2,048	1,024	1,024
Temperature	0 and 1	0 and 1	0
Reported quantity	τ, speedup vs. AR	Speedup vs. AR	Output tokens/s
Hardware	1 H200	1 H200	1 H200

Benchmark and decoding. The Red Hat speculator benchmark comprises 844 prompts: 164 for HumanEval, 200 for tool calling, and 80 each for Math, MT-Bench, QA, RAG, summarization, and translation (Red Hat AI, 2026). Dataset-level and suite-speedup measurements cover greedy decoding ($T = 0$) and stochastic decoding ($T = 1$); the concurrency sweep uses $T = 0$.

Execution and output limits. All measurements use vLLM on 1 H200 GPU (Kwon et al., 2023). Dataset rows use offline inference at batch size 1 with a 2,048-token output cap. Average speedup is measured separately using online serving at client concurrency 1 on the mixed 844-prompt workload with a 1,024-token cap. The throughput sweep uses the same online workload and cap at concurrency 1, 2, 4, 8, 16, 32. All caps count generated tokens. Evaluation settings are identical within each baseline–Carryover pair. The Avg. speedup entries in Table 1 are these separate measurements on the combined workload, not arithmetic means of the offline dataset speedups.

Metric aggregation. Acceptance length includes bonus tokens from verification and is micro-averaged over all verification rounds in a dataset. For online throughput, total generated output tokens are divided by the post-warmup serving makespan. Average speedup uses the separately timed autoregressive baseline on the same online workload. Displayed values are rounded after aggregation, so recomputing percentage gains from the printed values can differ slightly from the reported percentages.

Matched-anchor recall protocol. At draft position j, all 3 variants are evaluated on the same rounds, namely those in which ON and Baseline match the target through position $j - 1$; OFF is included even if its earlier prediction differs. The set of rounds therefore changes with position. Recall is micro-averaged over these rounds, and error bars use a normal approximation.

C ATTENTION TO CARRIED STATES

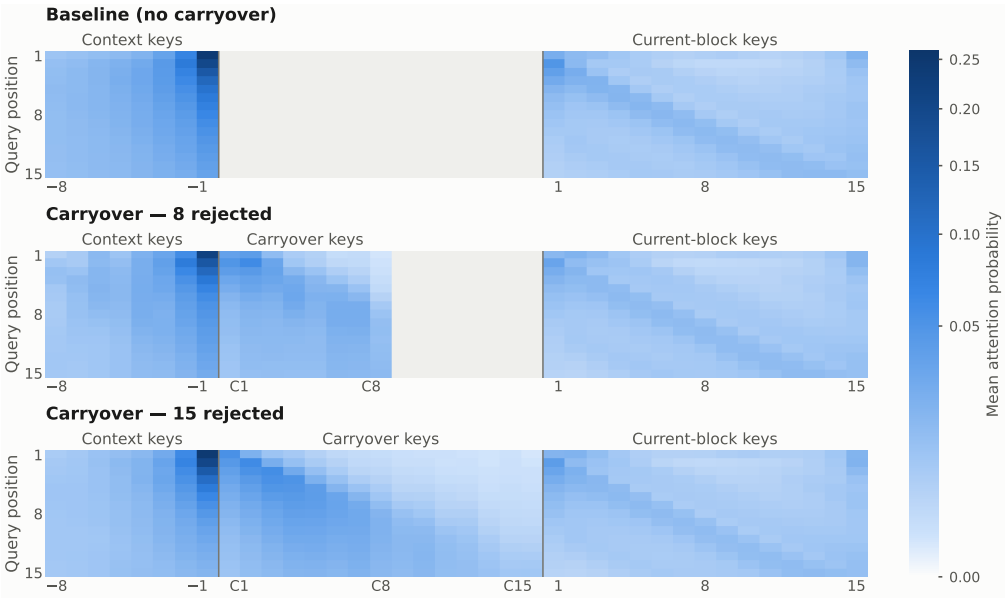

Figure 6: **Attention diagnostic.** Visualized attention for DFlash checkpoints on MT-Bench: baseline, and Carryover rounds with 8 or 15 rejected states. Probabilities are averaged across heads, drafter layers, and rounds in each subset. The display crops context positions without renormalizing the probabilities.

Figure 6 shows how much attention the drafter places on carried rows. 19.78% of full-softmax attention falls on carried rows. The 15-rejected subset assigns 23.40% to those rows. The diagnostic establishes that the carryover interface is read and that attention varies with position.

D RE-CONVERGENCE AND ACCEPTANCE AFTER REJECTION

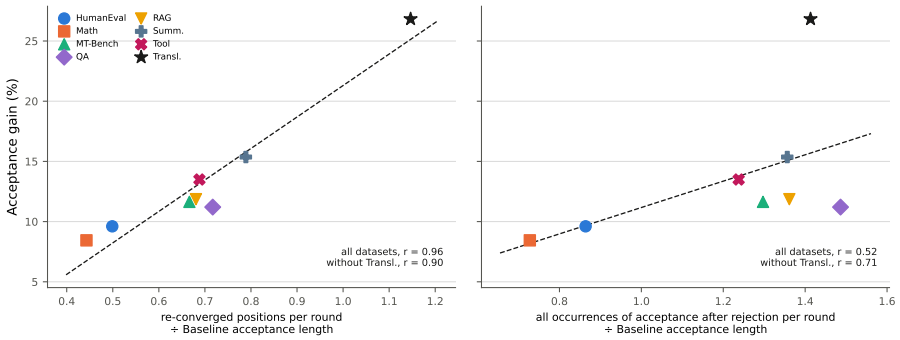

Figure 7: **Agreement with the eventual continuation versus acceptance after rejection.** 8-dataset associations for final Qwen3-4B Markov. The left statistic counts re-converged positions; the right counts all local proposal–verifier matches after rejection. Both are normalized by the dataset's Baseline acceptance length.

A draft token may be accepted later within its own rejected branch without matching the eventual continuation. Figure 7 distinguishes these notions. Normalized re-convergence correlates with final dataset-level gain at $r = 0.96$ (0.90 without Translation), compared with 0.52 (0.71) for all acceptance after rejection. For comparison, rejected-state count normalized by Baseline acceptance gives correlations of $r = 0.74$ across all datasets and 0.84 without Translation.

E RE-CONVERGENCE ACROSS DRAFTER TYPES

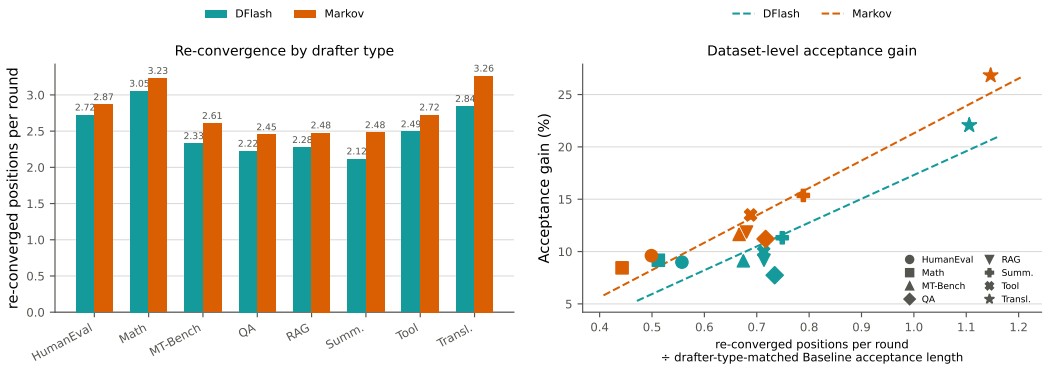

Figure 8: **Rejected-branch structure across drafters.** Final Qwen3-4B DFlash and Markov pairs at $T = 0$. Left: re-converged positions per rejected round. Right: the normalized statistic and final acceptance gain for each dataset and drafter.

Markov averages 2.76 re-converged positions versus 2.51 for DFlash, a 10.2% difference. It has higher re-convergence and baseline acceptance on every dataset, and higher Carryover gain on 7 of 8 datasets.

F ACCEPTANCE ACROSS TASKS AND TEMPERATURES

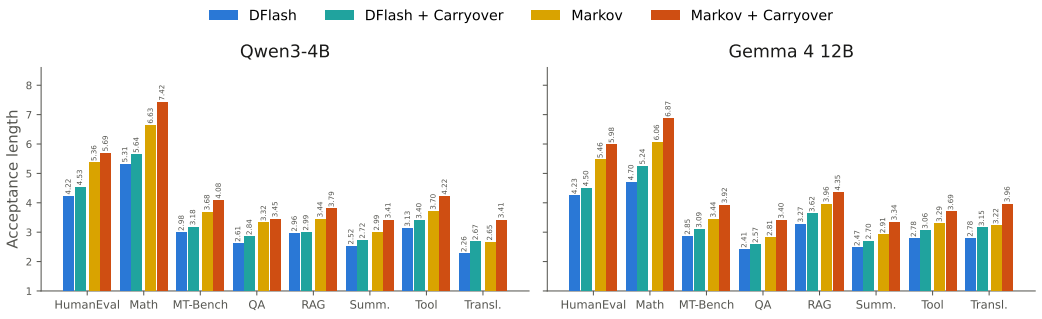

Figure 9: **Acceptance length at $T = 1$.** Baseline and Carryover for both targets and drafting regimes. These bars complement the greedy results in Figure 3; the numerical results are in Table 1.

G CARRYOVER LATENCY

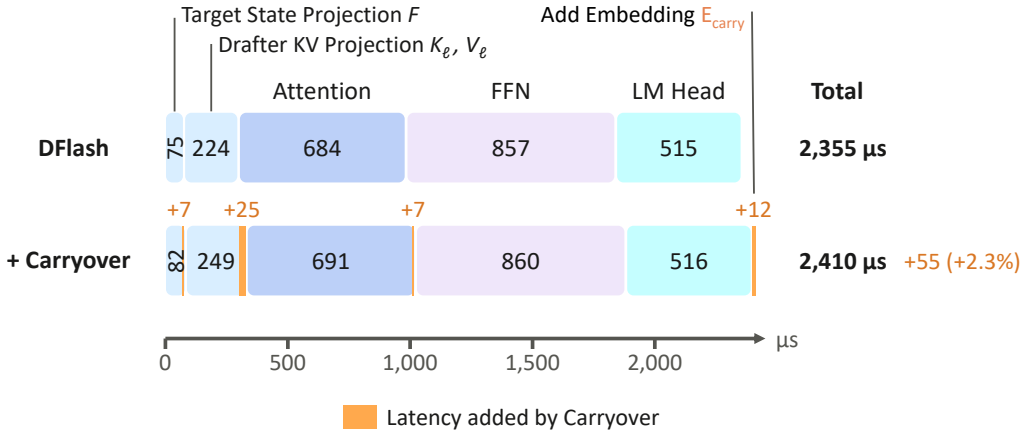

Figure 10: **Drafter latency per drafting round.** Qwen3-4B DFlash on one A100 GPU. Orange color marks the latency added by Carryover.

Section 2 defines $t_{\text{reuse}} = T_{\text{draft}}^{\text{Carryover}} - T_{\text{draft}}^{\text{baseline}}$ as the drafting latency added by recycling rejected states. Figure 10 measures it for Qwen3-4B DFlash.

Setup. To measure drafter latency, we use the 844-prompt benchmark at $T = 0$ with one NVIDIA A100 80GB GPU. A round carries 12.3 rejected states on average. For each round we restore the same committed drafter KV, anchor and positions, and run the drafter once without and once with the carried states.

The speedup and throughput results in Tables 1 and 2 are measured end to end in vLLM and therefore already include this cost. vLLM, however, runs the drafter as a captured CUDA graph without per-operation timing, and inserting timers would change the graph. We therefore time the PyTorch scaled-dot-product attention (SDPA) reference implementation of the same drafter. Absolute timing may differ from the serving engine, while the operations that Carryover changes are the same.

Carryover overhead. Carryover adds 55 μs per drafting round, from 2,355 to 2,410 μs (+2.3%). Figure 10 attributes most of this increase to (i) adding E_{carry} to rejected states and packing them with accepted states (+12 μs); (ii) projecting all B proposal states through F and the drafter's K_ℓ, V_ℓ, rather than only the m accepted states (+7 and +25 μs, respectively); and (iii) attending to at most B additional KV rows in the next draft (+7 μs). Carried-over states are read only as keys and values and never become queries, so the FFN and LM head process the same operation.