LOOPDRAFT: TARGET-GUIDED MULTI-STATE RECURRENCE FOR SPECULATIVE DECODING

Anonymous authors
Paper under double-blind review

ABSTRACT

Increasing a speculative drafter's physical depth can improve proposal quality, but also ties computation to independently parameterized layers. A drafter need not match the target model's full capability: it only needs to approximate short-horizon predictive distributions. This raises the question of whether the benefit of depth requires distinct layers or can instead be recovered through iterative refinement. We introduce LoopDraft, which reuses a shared computation block across an evolving latent state, recovering much of a deeper drafter's benefit with about one quarter of its draft parameters. However, further recurrence quickly encounters diminishing returns, motivating better use of a fixed refinement budget. LoopDraft therefore maintains multiple persistent states and uses verified-prefix target features to guide state access and retention while preserving shared-core recurrent computation. On Qwen3-8B, R5S4 reaches 4.55× speedup with a mean acceptance length of 6.35, improving over a local five-layer DFlash baseline by 6.4% while reducing draft parameters by 73.9%. On Qwen3.5-4B, the same design improves acceptance length across DFlash, DSpark, and DFlash2.

1 INTRODUCTION

Speculative decoding uses a cheaper drafter to propose several tokens for parallel target-model verification (Stern et al., 2018; Leviathan et al., 2023; Chen et al., 2023; Xia et al., 2023). Its benefit depends on both the tokens advanced per verification cycle and proposal latency. Better proposals help only when reduced target-model work outweighs added draft computation. A drafter need not reproduce the target model's full capability. Its role is to approximate the target's short-horizon predictive distributions well enough to produce proposals for efficient verification.

Target-aware drafters reuse internal target representations to improve proposals. EAGLE uses them for autoregressive drafting (Li et al., 2024a;b; 2025); DFlash conditions parallel block prediction on target features (Chen et al., 2026). DFlare uses layer-specific target fusion to exploit greater physical

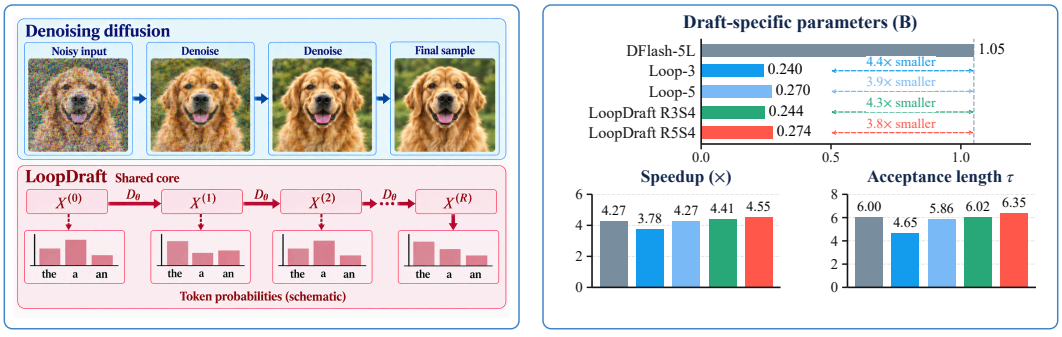


Figure 1: **Shared refinement and draft efficiency. Left:** Shared refinement with schematic token probabilities at one fixed position; only the final readout is used for drafting. **Right:** Qwen3-8B parameters, speedup, and acceptance length τ ($T = 0$, $K = 16$; five-task means). Parameters exclude shared embeddings and the LM head; τ includes the anchor.

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.

draft depth (Zhang et al., 2026). These approaches leave open whether improving proposal quality requires more independently parameterized layers, or whether shared computation can repeatedly refine the proposal representation. Iterative denoising in image and video diffusion provides a useful precedent for generation through repeated evaluations of a shared network (Ho et al., 2020; 2022). We explore this principle within a speculative drafter: a shared block repeatedly updates an evolving latent proposal representation, whose final readout approximates the target's short-horizon predictive distributions.

Our first observation is that much of physical depth's benefit does not require independently parameterized layers. On Qwen3-8B with $K = 16$ and greedy decoding, a one-layer core applied five times nearly matches our locally trained five-layer DFlash baseline: both speedups round to $4.27\times$, with acceptance length $\tau = 5.86$ versus 6.00, including the anchor. The recurrent model uses 0.27B rather than 1.05B draft-specific parameters, a 3.9-fold reduction (Figure 1). Both configurations perform five layer evaluations.

These results disentangle two resources normally coupled by physical depth: independent parameters and repeated computation. Once weights are shared, however, additional refinement still incurs latency. Increasing R from 5 to 15 raises τ only to 6.06 while lowering speedup to $3.58\times$. This shifts the remaining question from how to add more computation to how to increase the information carried by each refinement. We therefore test whether proposal quality can improve by expanding the representation carried across a fixed number of refinements. A single-state drafter carries all information between refinements through one evolving block representation. Multiple states can retain and update representations separately while sharing the computation that refines them. State width S thus offers another capacity resource: it expands what is carried between refinements without increasing how often the core runs.

Multiple states still feed one core input; therefore, the benefit of additional state capacity depends on whether the drafter can adaptively decide what information to read and retain. Static read maps and step-specific retention coefficients do not adapt to the input or block position; correspondingly, our static four-state model raises τ only from 5.86 to 5.92 while lowering speedup to $4.22\times$. Our target-conditioned setting provides a natural control signal: the latest verified-prefix target representation, which encodes the confirmed context and is already available without another target pass. LoopDraft uses this signal to guide state access and persistence. An access controller reweights low-rank state contributions before fusion; a persistence controller scales old-state retention during write-back. Both also use local state, block position, and recurrent step. The low-rank read/write projections remain learned and input-independent, connecting S states to one shared-core evaluation per step.

R5S4 improves proposals at the same five-step depth, reaching $4.55\times$ speedup and $\tau = 6.35$ with 0.274B draft parameters. R3S4 reaches $4.41\times$ and $\tau = 6.02$ with three core calls and 0.244B parameters: its acceptance length exceeds Loop-5's and reaches DFlash-5L's acceptance level, with lower draft latency than Loop-5. This lower-compute operating point shows that better recurrent-state utilization can partially substitute for additional refinement steps. Experiments on Qwen3.5-4B with DFlash, DSpark, and DFlash2 test whether these state-control benefits also hold when the underlying proposal mechanism changes.

Our contributions are:

- We distinguish physical depth, shared recurrent computation, and recurrent state as drafter capacity resources, and show that shared latent refinement recovers most of physical depth's benefit with roughly one quarter of its draft parameters.
- We introduce target-guided state access and persistence to use multiple recurrent states at a fixed core-call count. Controlled ablations test the roles of state width, adaptive control, and target context.
- We demonstrate improved five-step performance and a lower-compute three-step operating point on Qwen3-8B, showing that effective state control can improve proposals without simply increasing recurrent depth. Gains across three draft backbones and two regimes on Qwen3.5-4B support transfer.

2 Related Work

Target-conditioned drafting. Compact drafters often reuse representations already computed by the target. EAGLE uses target hidden features, with EAGLE-3 adding multi-layer fusion (Li et al., 2024a;b; 2025); GliDe accesses target KV states (Du et al., 2024). DFlash conditions parallel drafting on fused target features (Chen et al., 2026), while DFlare makes fusion layer-specific to support deeper drafters (Zhang et al., 2026). LoopDraft uses verified target context to control recurrent state as well as core computation. Distillation and online adaptation improve draft–target alignment through training (Zhou et al., 2024; Liu et al., 2024).

Shared computation and recurrent state. Universal Transformers and ALBERT share parameters across depth (Dehghani et al., 2019; Lan et al., 2020); recurrent-depth language models scale latent computation by repeating shared blocks (Geiping et al., 2025). In speculative decoding, ReDrafter's RNN generates successive candidates (Cheng et al., 2024), whereas concurrent work LoopSpec reads intermediate states of a looped target for training-free, pipelined self-speculation (Cho et al., 2026). Our separate drafter recurrently refines a fixed block without requiring a looped target. Hyper-Connections expand residual streams (Zhu et al., 2025), and mHC constrains their mixing to stabilize signal propagation (Xie et al., 2026). Hyperloop uses input-dependent stream reads, writes, and residual mixing at loop boundaries for language modeling (Zeitoun et al., 2026). Inspired by these multi-stream designs, LoopDraft uses input-independent low-rank read/write maps and verified-prefix target features to control scalar access and persistence within the drafter.

Dependencies across candidate positions. Block and multi-head proposals reduce sequential drafting (Cai et al., 2024; Liu et al., 2026; Hui et al., 2026), while Hydra connects successive draft heads (Ankner et al., 2024). Domino and DSpark add lightweight causal modules to parallel backbones (Huang et al., 2026; Cheng et al., 2026); DFlash2 uses predecessor-conditioned selection (Inco AI, 2026). These correction and selection mechanisms address dependencies across candidate positions. Our state control refines representations before token selection, allowing composition with DSpark and DFlash2 (Section 4.3).

3 Method

We describe drafter capacity by physical depth D, recurrent depth R, and recurrent state width S. LoopDraft reuses a shallow core across R refinements, carrying S persistent states (*streams*) between them. Target-guided access and persistence control how these states are read and retained (Figure 2). We describe the DFlash instantiation and its composition with other backbones.

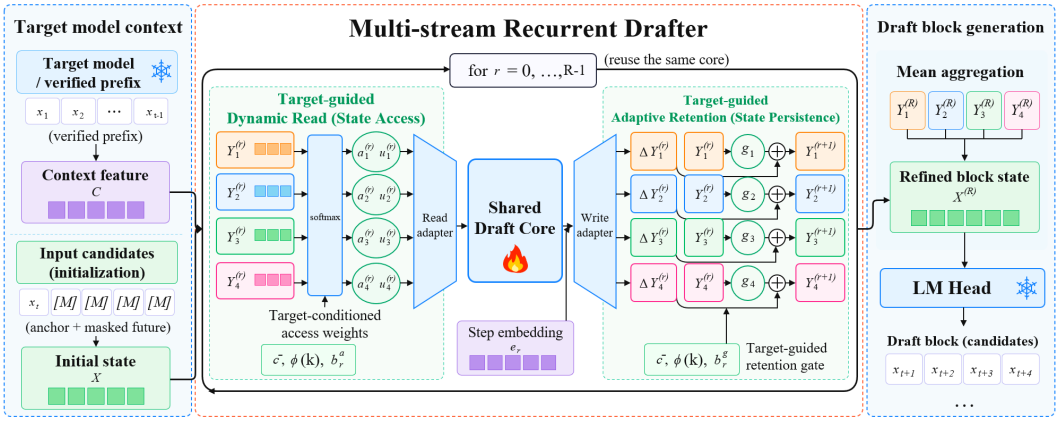


Figure 2: **Overview of LoopDraft.** Verified-prefix target features C condition the shared core; $\bar{c} = C_{:,-1}$ guides scalar access before fusion and persistence during write-back. Learned, input-independent low-rank read/write maps connect S states to one core call per step; each a_s or g_s represents K block-position-wise coefficients.

3.1 TARGET-CONDITIONED SPECULATIVE BLOCK DRAFTING

Let M be a frozen autoregressive target model and $x_{1:t}$ the confirmed token prefix at the start of a speculative round. The anchor x_t is the next token supplied by the preceding target pass: its identity is known, but the target has evaluated positions only through $t-1$. Thus no target hidden state at the anchor position is available when drafting begins. We use N for batch size, K for block length (the anchor plus $K-1$ future slots), and H for draft hidden width. From selected target layers $\mathcal{J}$, we concatenate the available token-wise features, project them to the draft width, and normalize:

$$C = \mathrm{RMSNorm}(W_c \,\mathrm{Concat}_{j\in\mathcal{J}}\, H_j)\,. \tag{1}$$

Here C contains features for the newly evaluated confirmed positions; older prefix features are represented in the draft KV cache. Its last entry corresponds to position $t-1$. The draft input is the anchor x_t followed by $K-1$ masks for positions $t+1,\ldots,t+K-1$, with embedding $X^{(0)} \in \mathbb{R}^{N\times K\times H}$. Following DFlash, the evolving draft state supplies queries, while target context and draft state jointly supply keys and values:

$$\mathbf{Q} = W_q X, \qquad \mathbf{K} = [W_k C; W_k X], \qquad \mathbf{V} = [W_v C; W_v X], \tag{2}$$

where older verified-prefix cache terms are omitted for clarity. The frozen target LM head maps the final states at the $K-1$ masked positions to proposal distributions. We define acceptance length τ as the mean number of tokens advanced per target verification cycle, including the anchor; it is not the count of accepted draft candidates alone. We retain the underlying draft backbone's masking protocol.

Training objective. We freeze the target model, shared token embeddings, and language-model head, while jointly optimizing all draft-side parameters, including the target-feature projection, recurrent draft core, low-rank stream adapters, normalization parameters, recurrent embeddings, retention priors, and state controllers. We apply the same position-weighted drafting objective as the underlying DFlash backbone only to the masked future positions of the final recurrent output $X^{(R)}$. Gradients propagate through the complete recurrent unroll, without intermediate recurrent supervision or state-control-specific auxiliary losses. For every training block, both the draft core and state controllers are conditioned only on target features available before its anchor.

3.2 SHARED RECURRENT DEPTH

We refine the proposal representation in latent space and decode its final state into proposal distributions. Let D_θ denote a core with D physical layers. The single-state drafter reuses its trainable parameters across R steps:

$$X^{(r+1)} = D_\theta\Big(X^{(r)}, C, \mathcal{K}_{\mathrm{prefix}}\Big)\,, \qquad r = 0,\ldots,R-1, \tag{3}$$

where $\mathcal{K}_{\mathrm{prefix}}$ is the draft KV cache associated with verified history. Recurrence refines the same K logical block positions; it does not extend sequence length. The number of physical block applications is DR, whereas the dominant core parameter count scales with D, not DR.

Cache semantics. Intermediate refinements read the verified-prefix draft cache without appending block KV states. Any final-step block entries, including the anchor, are discarded before verification; subsequent draft history uses target features of newly confirmed positions. The target cache retains only the accepted prefix, preventing repeated positions or rejected candidates from entering later attention contexts.

3.3 LOW-RANK MULTI-STREAM RECURRENT STATE

To expand the representation carried across refinements at a fixed core-call count, we maintain S states and fuse them into one core input. We initialize them by copying the block representation:

$$Y_s^{(0)} = X^{(0)}, \qquad Y^{(0)} \in \mathbb{R}^{N\times K\times S\times H}. \tag{4}$$

They persist across the R refinements of one proposal call, but not across speculative decoding cycles.

Input-independent low-rank read. A shared projection $P_{\text{down}} \in \mathbb{R}^{\rho \times H}$ compresses each state to rank $\rho \ll H$. The learned, input-independent multi-state baseline concatenates the resulting latents and maps them back to the core width:

$$u_s^{(r)} = P_{\text{down}} Y_s^{(r)}, \tag{5}$$

$$h_{\text{static}}^{(r)} = \text{RMSNorm}\Big(P_{\text{read}}[u_1^{(r)}; \ldots; u_S^{(r)}]\Big), \tag{6}$$

where $P_{\text{read}} \in \mathbb{R}^{H \times S\rho}$. Section 3.4 adds input-dependent, block-position-wise access coefficients while retaining the same projection. Its matrix blocks can weight streams differently. The core is evaluated once irrespective of S:

$$z^{(r)} = D_\theta\Big(h^{(r)}, C, \mathcal{K}_{\text{prefix}}\Big), \qquad \widetilde{z}^{(r)} = z^{(r)} + e_r. \tag{7}$$

The post-core embedding $e_r \in \mathbb{R}^H$ affects write-back and subsequent reads, not the current core evaluation.

Input-independent low-rank write. A shared learned map produces S rank-ρ slices, followed by stream-specific expansions:

$$[v_1^{(r)}; \ldots; v_S^{(r)}] = P_{\text{write}} \widetilde{z}^{(r)}, \tag{8}$$

$$\Delta Y_s^{(r)} = P_{\text{up},s} v_s^{(r)}, \tag{9}$$

$$Y_s^{(r+1)} = \sigma(\alpha_{r,s}) Y_s^{(r)} + \Delta Y_s^{(r)}. \tag{10}$$

The static retention prior $\alpha_{r,s}$ is initialized to zero. Stream-specific writes let identical initial states diverge. Their parameters are input-independent, while their content is target-conditioned through $z^{(r)}$. After R steps, we aggregate and normalize:

$$X^{(R)} = \text{RMSNorm}\left(\frac{1}{S}\sum_{s=1}^{S} Y_s^{(R)}\right). \tag{11}$$

The primary $S = 1$ baseline follows Equation 3 directly and bypasses the multi-stream adapters.

3.4 Target-Guided State Control

The core already uses target context to transform the fused state. We extend this conditioning to the state interface: access control determines the relative stream contributions before fusion, and persistence control sets old-state retention during write-back. Both adapt to the current input, block position, and recurrent step. The read remains low rank; control changes its inputs rather than removing its compression.

Both controllers use $\bar{c} = C_{:,-1}$ at verified position $t-1$. For block position $k \in \{0, \ldots, K-1\}$, with $k = 0$ the anchor, $\phi(k)$ is a four-dimensional relative-position code (Appendix F). We omit the batch index below.

Target-guided state access. Before the S low-rank states are compressed into one core input, we compute

$$q_{r,s,k}^a = A_u u_{r,s,k} + A_p \phi(k) + b_r^a + A_c \bar{c}, \tag{12}$$

$$\ell_{r,s,k} = w_a^\top \,\text{SiLU}\big(q_{r,s,k}^a / \sqrt{m_a}\big), \tag{13}$$

$$a_{r,s,k} = S\,\frac{\exp(\ell_{r,s,k})}{\sum_{j=1}^{S}\exp(\ell_{r,j,k})}, \tag{14}$$

$$h_{r,k} = \text{RMSNorm}(P_{\text{read}}[a_{r,1,k}u_{r,1,k}; \ldots; a_{r,S,k}u_{r,S,k}]). \tag{15}$$

Here $m_a = 4$ counts the additive feature branches, and b_r^a is a learned controller-step embedding, distinct from the post-core embedding e_r. The factor S preserves the scale of the static concatenated read. We zero-initialize w_a, making $a_{r,s,k} = 1$ initially. For the same stream-adapter parameters, this exactly recovers the static read at initialization. Removing $A_c\bar{c}$ and using $m_a = 3$ gives the local dynamic-access ablation.

Target-guided state persistence. After the core and learned, input-independent write maps produce $v_{r,s,k}$ and $\Delta Y_{r,s,k}$, a separate controller determines how much old state survives:

$$q^g_{r,s,k} = G_u u_{r,s,k} + G_v v_{r,s,k} + G_p \phi(k) + b^g_r + G_c \bar{c}, \tag{16}$$

$$\delta_{r,s,k} = w_g^\top \operatorname{SiLU}\left(q^g_{r,s,k}/\sqrt{m_g}\right), \tag{17}$$

$$g_{r,s,k} = \sigma(\alpha_{r,s} + \delta_{r,s,k}), \tag{18}$$

$$Y^{(r+1)}_{s,k} = g_{r,s,k} Y^{(r)}_{s,k} + \Delta Y^{(r)}_{s,k}. \tag{19}$$

Here $m_g = 5$, and b^g_r is a separate learned controller-step embedding. Removing $G_c\bar{c}$ and using $m_g = 4$ gives local dynamic retention; setting $\delta_{r,s,k} = 0$ recovers static retention. The gate scales only the old state, so the update is not a convex interpolation of old and new states. We zero-initialize w_g, so the adaptive controller initially reproduces the static prior. Together, the zero-initialized controllers reproduce the corresponding static multi-stream transition.

Each controller emits NKS scalars per step, totaling $2NKS$; neither generates read/write matrices or requires another target call. Initially identical streams yield $a_{0,s,k} = 1$ even after training. Differential access becomes possible after stream-specific write-back, so $R > 1$ is required to feed divergent streams into a later core evaluation.

Cost and parameter accounting. We count distinct draft-specific parameters, excluding the token embeddings and LM head shared with the target. Recurrent reuse does not multiply this count. At $H = 4096$, $\rho = 64$, $G_a = G_g = 32$, $R = 5$, and $S = 4$, the stream/control modules add 3.70M parameters. The core is evaluated R times, independent of S; additional read/write compute and state memory scale as $O(RNKSH\rho)$ and $O(NKSH)$, respectively. Appendix H gives the parameter formula, MAC counts, and training activation-memory scaling considerations.

Composition with other backbones. State control refines $X^{(R)}$ before token selection, allowing subsequent causal correction or predecessor-conditioned selection. Section 4.3 evaluates this composition with DSpark and DFlash2; Appendix I describes the interface.

4 Experiments

Our experiments test how much shared recurrence recovers from physical depth, where further computation saturates, and whether target-guided state control improves proposals at a fixed core-call budget. We first report decoding and transfer results, then isolate these effects through controlled comparisons.

4.1 Experimental setup

Evaluation setup and baselines. We evaluate Qwen3-8B (Yang et al., 2025) on GSM8K, MATH-500, HumanEval, MBPP, and MT-Bench (Cobbe et al., 2021; Hendrycks et al., 2021; Chen et al., 2021; Austin et al., 2021; Zheng et al., 2023). Evaluation uses NVIDIA A30 GPUs, bfloat16, and batch size 1 per worker, with block size $K = 16$ under greedy decoding ($T = 0$) and sampling ($T = 1$). Thinking is disabled at both temperatures. Our primary baseline is DFlash-5L trained locally from scratch; EAGLE-3 with inference tree budget 16 is an additional local reference (Chen et al., 2026; Li et al., 2025).

Training data. We use 1.86M conversations, counted before training-time filtering, with all assistant responses regenerated by Qwen3-8B. Appendix B describes the sources and preprocessing.

Training protocol and configuration. Standalone Loop and local DFlash drafters are trained independently from scratch. Loop-5 is trained for six epochs; all $R = 5$ state-width and state-control variants branch from this checkpoint and adapt for three epochs. Static, access-only, persistence-only, and full $S = 4$ variants share parameter shapes and this adaptation budget. R3S4 adapts from the independently trained Loop-3 for three epochs with $R_{\text{train}} = R_{\text{eval}} = 3$. Multi-state models use $H = 4096$, $\rho = 64$, and controller widths $G_a = G_g = 32$. Appendix C details the optimization settings, target-feature taps, high-R adaptation, and evaluation configuration.

Table 1: **Qwen3-8B decoding results at $T = 0$ and $T = 1$.** Parallel block drafters use $K = 16$; τ includes the anchor. Params are draft-specific billions excluding shared embeddings and the LM head; Avg. is the five-task mean. Bold marks the best value for each metric.

Method	Params (B)	MATH				CODE				CHAT			
		GSM8K		MATH-500		HumanEval		MBPP		MT-Bench		*Avg.*	
Temperature = 0		Speedup	τ	Speedup	τ	Speedup	τ	Speedup	τ	Speedup	τ	Speedup	τ
EAGLE-3	0.260	1.49×	3.68	1.53×	3.75	1.30×	3.46	1.25×	3.44	1.21×	3.19	1.36×	3.50
DFlash-5L	1.050	4.64×	6.61	5.71×	7.83	4.25×	5.64	4.01×	5.48	2.76×	4.43	4.27×	6.00
Loop-3	**0.240**	4.17×	5.00	4.82×	5.93	3.77×	4.50	3.51×	4.22	2.65×	3.58	3.78×	4.65
Loop-5	0.270	4.76×	6.51	5.62×	7.50	4.30×	5.58	3.87×	5.30	2.79×	4.40	4.27×	5.86
LoopDraft R3S4	0.244	4.92×	6.65	5.80×	7.65	4.36×	5.75	4.09×	5.42	2.87×	4.63	4.41×	6.02
LoopDraft R5S4	0.274	**5.08**×	**7.00**	**5.96**×	**8.05**	**4.49**×	**6.08**	**4.20**×	**5.72**	**3.01**×	**4.90**	**4.55**×	**6.35**
Temperature = 1		Speedup	τ	Speedup	τ	Speedup	τ	Speedup	τ	Speedup	τ	Speedup	τ
EAGLE-3	0.260	1.34×	3.35	1.22×	3.02	1.09×	2.94	1.08×	3.04	1.08×	2.86	1.16×	3.04
DFlash-5L	1.050	4.18×	6.06	4.51×	6.40	3.55×	4.77	3.50×	4.76	2.47×	3.99	3.64×	5.20
Loop-3	**0.240**	3.79×	4.55	3.83×	4.85	3.14×	3.84	3.08×	3.70	2.36×	3.20	3.24×	4.03
Loop-5	0.270	4.32×	5.99	4.46×	6.07	3.61×	4.71	3.33×	4.60	2.50×	3.95	3.64×	5.06
LoopDraft R3S4	0.244	4.50×	6.04	4.61×	6.26	3.65×	4.88	3.52×	4.79	2.60×	4.14	3.78×	5.22
LoopDraft R5S4	0.274	**4.65**×	**6.45**	**4.78**×	**6.50**	**3.74**×	**5.21**	**3.68**×	**5.02**	**2.70**×	**4.40**	**3.91**×	**5.52**

Metrics. We report speedup over target-only autoregressive decoding and acceptance length τ, the mean number of tokens advanced per verification cycle, including the anchor. Cross-benchmark averages are unweighted arithmetic means over the five tasks; relative gains use unrounded averages. Appendix A defines the per-request metrics and timing boundaries. For the four main configurations, Table 1 reports means over three complete runs at each temperature. Repeats measure runtime variation at $T = 0$ and both sampling and runtime variation with distinct seeds at $T = 1$ (Appendix G).

4.2 MAIN RESULTS

At the same five-step depth as Loop-5, R5S4 improves both metrics on every $T = 0$ benchmark, reaching 4.55× mean speedup and $\tau = 6.35$ (Table 1). These are gains of 6.4% and 5.9% over local DFlash-5L with 73.9% fewer draft parameters, and 6.6% and 8.4% over Loop-5 at the same recurrent depth.

With 0.244B parameters and three core calls, R3S4 reaches 4.41× and $\tau = 6.02$: mean gains of 3.3% in speedup and 2.8% in acceptance length over Loop-5, with improvements on every task and 40% fewer core calls.

Matching Loop-5's five-tap interface raises R3S4 to 4.44×/$\tau = 6.06$ at $T = 0$ and 3.83×/$\tau = 5.27$ at $T = 1$, preserving the three-step operating point (Appendix E).

Temperature-1 results. At $T = 1$, R5S4 reaches 3.91× and $\tau = 5.52$, while R3S4 reaches 3.78× and $\tau = 5.22$. Both improve on Loop-5 in both metrics on every task; R5S4 also exceeds local DFlash-5L (Table 1).

Evaluation variability. Across three runs, speedup standard deviations are at most 0.02× at both temperatures (Appendix G).

4.3 TRANSFER ACROSS TARGET MODELS AND DRAFT BACKBONES

We apply state control to local six-layer reproductions of DFlash, DSpark, and DFlash2 on Qwen3.5-4B (Qwen Team, 2026; Chen et al., 2026; Cheng et al., 2026; Inco AI, 2026). Recurrent variants retain each native proposal mechanism and reuse one core. Original uses eight target taps as a physical-depth reference; recurrent variants share five, and R5S4 adapts from the corresponding Loop-5 with the same core and feature interface. This gives a matched-interface comparison. We evaluate the same five subsets using vLLM at client concurrency 16, with separate target-regenerated corpora and checkpoints for thinking-enabled $T = 1$ sampling and thinking-disabled $T = 0$ greedy

Table 2: **Qwen3.5-4B transfer across draft backbones.** Five-task mean τ for thinking ($T = 1$ sampling) and non-thinking ($T = 0$ greedy). Original is our local six-layer reproduction; Param. $\downarrow$ is relative to it. Counts exclude shared embeddings and the LM head; bold marks the best mean per backbone and regime.

Backbone	Variant	D	R	S	Params (B)	Param. $\downarrow$	Avg. τ	
							Thinking ($T = 1$)	Non-thinking ($T = 0$)
DFlash	Original	6	1	1	0.630	–	4.84	5.69
	Loop-3	1	3	1	0.130	79.4%	4.02	5.09
	Loop-5	1	5	1	0.130	79.4%	4.72	5.56
	R5S4	1	5	4	0.132	79.0%	**5.08**	**5.96**
DSpark	Original	6	1	1	0.700	–	5.40	6.86
	Loop-3	1	3	1	0.200	71.4%	4.72	6.17
	Loop-5	1	5	1	0.200	71.4%	5.28	6.55
	R5S4	1	5	4	0.202	71.1%	**5.57**	**7.06**
DFlash2	Original	6	1	1	0.649	–	5.86	7.32
	Loop-3	1	3	1	0.135	79.2%	5.16	6.60
	Loop-5	1	5	1	0.135	79.2%	5.72	6.98
	R5S4	1	5	4	0.138	78.7%	**5.91**	**7.37**

decoding. Table 2 reports three-run means, with $T = 1$ seeds matched across variants; Appendix C.2 gives full details.

Loop-5 retains 97.5–97.8% of Original's mean acceptance length under thinking-enabled sampling and 95.4–97.7% under thinking-disabled greedy decoding, with 71.4–79.4% fewer parameters (Table 2). With recurrent taps fixed, Loop-5 improves over Loop-3 at unchanged parameter counts in both regimes.

For DFlash, DSpark, and DFlash2, R5S4 improves over Loop-5 by 7.6%, 5.5%, and 3.3% in the thinking regime, and 7.2%, 7.8%, and 5.6% in the non-thinking regime. In all six combinations, R5S4 exceeds the corresponding six-layer Original's mean acceptance length with 71.1–79.0% fewer draft parameters.

4.4 Physical depth and shared recurrent computation

The remaining analyses use Qwen3-8B at $T = 0$. With identical three-tap inputs, shared recurrence raises τ from a one-layer drafter's 3.19 to 4.65 in three steps, both at 0.24B. It approaches a three-layer drafter's $\tau = 4.76$ at 0.62B with the same three layer evaluations (Appendix Table 9). Shared refinement thus improves proposals at fixed model size. At five evaluations with matched taps, Loop-5 retains 97.7% of DFlash-5L's acceptance length and 99.9% of its speedup using 25.7% of its parameters (Appendix Table 12).

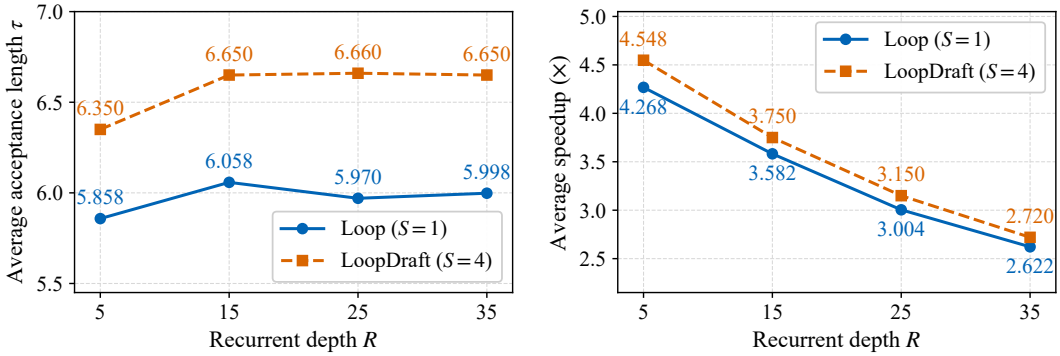


Figure 3: **Recurrent-depth adaptation on Qwen3-8B** ($D = 1$, $K = 16$, $T = 0$). Values are five-task means; $R > 5$ models adapt from Loop-5 for three epochs at their evaluation depth.

We next examine how far additional shared refinement remains useful. Under the depth-adaptation protocol in Figure 3, increasing single-state recurrence from $R = 5$ to 15 raises τ from 5.86 to 6.06 (+3.4%) but lowers speedup from 4.27× to 3.58× (-16.1%). At $R = 35$, $\tau = 6.00$ is only 2.4% above the five-step value, with 2.62× speedup. The plateau appears within each benchmark.

Four-state recurrence reaches $\tau = 6.65$ at $R = 15$ and remains near it at $R = 25, 35$, while speedup falls from 3.75× to 2.72×. Four states improve acceptance at every depth, but their gains also saturate with R, shifting the question to state use under a fixed recurrent budget.

4.5 Effects of access, persistence, and target context

Table 3: **Core state-control ablations at $R = 5$.** Target Ctrl. marks controller target input; cores remain target-conditioned. $S = 4$ rows share parameter shapes; the $S = 1$ baseline bypasses adapters.

Variant	Access	Persistence	Target Ctrl.	Speedup	τ
Loop-5 ($S = 1$)	–	–	–	4.27×	5.86
Static $S = 4$	×	×	×	4.22×	5.92
Persistence only	×	✓	✓	4.28×	6.05
Access only	✓	×	✓	4.34×	6.17
Full $S = 4$	✓	✓	✓	**4.55×**	**6.35**

Static widening barely changes acceptance length, indicating that state capacity alone is insufficient here. Under the matched 0.274B/three-epoch budget, access and persistence improve complementary parts of state utilization, while Full reaches $4.55\times/\tau = 6.35$. It improves speedup by 7.8% and acceptance length by 7.3% over Static, and outperforms either mechanism on every task (Appendix Table 7).

Removing controller target input while keeping the core conditioned yields $4.35\times/\tau = 6.17$. Parameter-matched $S = 1$ and continued-training controls remain near Loop-5, separating the gains from extra parameters or training (Appendix D).

4.6 Choosing recurrent depth and state width

At $R = 5$, $S = 4$ gives the highest speedup (4.55×); $S = 8$ raises τ to 6.44 but latency to 10.75 ms, reducing speedup to 4.48× (Appendix Table 13). With three calls, R3S4 cuts Loop-5 latency from 9.56 to 6.65 ms and reaches $4.41\times/\tau = 6.02$, exceeding Loop-5 and matching DFlash-5L's acceptance (6.00). Stronger proposals thus need not require deeper recurrence. R5S4 retains its gain with a 16-node tree verifier (Appendix K).

5 Limitations

Additional recurrent steps incur latency and diminishing returns; the plateau may reflect optimization or tied transitions. Recurrent-state memory and low-rank communication costs can make state-width selection hardware dependent. State control relies on verified-prefix target features; its effectiveness with restricted target access remains untested. Evaluation covers Qwen-family targets and several block drafting backbones; broader validation across target architectures remains open.

6 Conclusion

We study speculative drafting as short-horizon target-distribution approximation through shared latent refinement. Reusing a core across evolving proposal states recovers most of physical depth's benefit with far fewer draft parameters, while further recurrence yields diminishing returns. LoopDraft therefore shifts capacity toward persistent recurrent state and uses verified-prefix target features to control what is read and retained across refinements. On Qwen3-8B, R5S4 reaches 4.55× with $\tau = 6.35$, while three-call R3S4 reaches 4.41× with $\tau = 6.02$. Consistent gains across DFlash, DSpark, and DFlash2 on Qwen3.5-4B show that the same state-control principle transfers across proposal mechanisms.

AI Assistance Statement

Generative AI tools were used to assist with manuscript writing and polishing, literature retrieval and discovery, and research ideation. All AI-assisted content was reviewed, verified, and revised by the authors, who take full responsibility for the final manuscript and its contents.

Reproducibility statement

An anonymous implementation is available at `https://anonymous.4open.science/r/LoopDraft-DD11`. We document the data construction, training and evaluation protocols, model configurations, verification procedures, timing methodology, and controlled ablations throughout the main paper and Appendices A–K. The final version will release model weights, model configurations, training scripts, checkpoint hashes, benchmark indices, raw per-request timings, and plotting code through a public model repository to facilitate reproduction of the reported results.

References

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

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

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

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

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

Jian Chen, Yesheng Liang, and Zhijian Liu. DFlash: Block diffusion for flash speculative decoding. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2026. URL `https://z-lab.ai/projects/dflash/`.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, et al. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*, 2021.

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

Yunfei Cheng, Aonan Zhang, Xuanyu Zhang, Chong Wang, and Yi Wang. Recurrent drafter for fast speculative decoding in large language models. *arXiv preprint arXiv:2403.09919*, 2024.

SangLyul Cho, Langqing Cui, Sehoon Kim, Dongsu Han, and Insu Han. LoopSpec: Pipelined self-speculative decoding for looped transformers. *arXiv preprint arXiv:2609.17184*, 2026.

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

Mostafa Dehghani, Stephan Gouws, Oriol Vinyals, Jakob Uszkoreit, and Łukasz Kaiser. Universal transformers. In *International Conference on Learning Representations*, 2019.

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 *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pp. 3029–3051. Association for Computational Linguistics, 2023. doi: 10.18653/v1/2023.emnlp-main.183.

Cunxiao Du, Jing Jiang, Xu Yuanchen, Jiawei Wu, Sicheng Yu, Yongqi Li, Shenggui Li, Kai Xu, Liqiang Nie, Zhaopeng Tu, and Yang You. GliDe with a CaPE: A low-hassle method to accelerate speculative decoding. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 11704–11720. PMLR, 2024.

Jonas Geiping, Sean McLeish, Neel Jain, John Kirchenbauer, Siddharth Singh, Brian Bartoldson, Bhavya Kailkhura, Abhinav Bhatele, and Tom Goldstein. Scaling up test-time compute with latent reasoning: A recurrent depth approach. In *Advances in Neural Information Processing Systems*, volume 38, pp. 41340–41391, 2025. doi: 10.52202/085713-1380.

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. Measuring mathematical problem solving with the MATH dataset. In *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks*, 2021. URL https://datasets-benchmarks-proceedings.neurips.cc/paper_files/paper/2021/hash/be83ab3ecd0db773eb2dc1b0a17836a1-Abstract-round2.html.

Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models. In *Advances in Neural Information Processing Systems*, volume 33, pp. 6840–6851, 2020.

Jonathan Ho, Tim Salimans, Alexey Gritsenko, William Chan, Mohammad Norouzi, and David J. Fleet. Video diffusion models. In *Advances in Neural Information Processing Systems*, volume 35, pp. 8633–8646, 2022. doi: 10.52202/068431-0628.

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.

Mude Hui, Xin Huang, Jaime Campos Salas, Yue Sun, Nathan Pemberton, Xiang Song, Ashish Khetan, and George Karypis. P-EAGLE: Parallel-drafting EAGLE with scalable training. *arXiv preprint arXiv:2602.01469*, 2026.

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

Yoon Kim and Alexander M. Rush. Sequence-level knowledge distillation. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pp. 1317–1327. Association for Computational Linguistics, 2016. doi: 10.18653/v1/D16-1139.

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

Zhenzhong Lan, Mingda Chen, Sebastian Goodman, Kevin Gimpel, Piyush Sharma, and Radu Soricut. ALBERT: A lite BERT for self-supervised learning of language representations. In *International Conference on Learning Representations*, 2020.

Yaniv Leviathan, Matan Kalman, and Yossi Matias. Fast inference from transformers via speculative decoding. In *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pp. 19274–19286. PMLR, 2023.

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, 2024a.

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

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

Fuliang Liu, Xue Li, Ketai Zhao, Yinxi Gao, Ziyan Zhou, Zhonghui Zhang, Zhibin Wang, Wanchun Dou, Sheng Zhong, and Chen Tian. DART: Diffusion-inspired speculative decoding for fast LLM inference. *arXiv preprint arXiv:2601.19278*, 2026.

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

Dhruv Nathawani, Shuoyang Ding, Vitaly Lavrukhin, Igor Gitman, Somshubra Majumdar, Evelina Bakhturina, Boris Ginsburg, and Jane Polak Scowcroft. Nemotron-Post-Training-Dataset-v2. NVIDIA, version 2.0, `https://huggingface.co/datasets/nvidia/Nemotron-Post-Training-Dataset-v2`, 2025.

Qwen Team. Qwen3.5: Towards native multimodal agents, February 2026. URL `https://qwen.ai/blog?id=qwen3.5`.

ShareGPT. ShareGPT Vicuna Unfiltered. Dataset snapshot distributed by Aeala, `https://huggingface.co/datasets/Aeala/ShareGPT_Vicuna_unfiltered`, 2023.

Mitchell Stern, Noam Shazeer, and Jakob Uszkoreit. Blockwise parallel decoding for deep autoregressive models. In *Advances in Neural Information Processing Systems*, 2018.

Heming Xia, Tao Ge, Peiyi Wang, Si-Qing Chen, Furu Wei, and Zhifang Sui. Speculative decoding: Exploiting speculative execution for accelerating seq2seq generation. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, pp. 3909–3925. Association for Computational Linguistics, 2023. doi: 10.18653/v1/2023.findings-emnlp.257.

Zhenda Xie, Yixuan Wei, Huanqi Cao, et al. mHC: Manifold-constrained hyper-connections. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2026.

An Yang et al. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025.

Abbas Zeitoun, Lucas Torroba-Hennigen, and Yoon Kim. Hyperloop transformers. *arXiv preprint arXiv:2604.21254*, 2026.

Jiebin Zhang, Zhenghan Yu, Song Liu, Eugene J. Yu, Zheng Li, Dawei Zhu, Jiangshan Duo, Weimin Xiong, Yifan Song, Guanghua Yu, Jianchen Zhu, and Sujian Li. DFlare: Scaling up draft capacity for block diffusion speculative decoding. *arXiv preprint arXiv:2606.02091*, 2026.

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric P. Xing, Hao Zhang, Joseph E. Gonzalez, and Ion Stoica. Judging LLM-as-a-judge with MT-Bench and chatbot arena. In *Advances in Neural Information Processing Systems*, 2023.

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

Yongchao Zhou, Kaifeng Lyu, Ankit Singh Rawat, Aditya Krishna Menon, Afshin Rostamizadeh, Sanjiv Kumar, Jean-François Kagy, and Rishabh Agarwal. DistillSpec: Improving speculative decoding via knowledge distillation. In *International Conference on Learning Representations*, 2024.

Defa Zhu, Hongzhi Huang, Zihao Huang, Yutao Zeng, Yunyao Mao, Banggu Wu, Qiyang Min, and Xun Zhou. Hyper-connections. In *International Conference on Learning Representations*, 2025.

A Benchmark and Verification Protocol

Output agreement. We compare generated token sequences after removing each request's prompt. All standard-verification $T = 0$ outputs exactly match target-only decoding on the evaluated requests.

Metrics and aggregation. For each benchmark, speedup is the ratio of mean per-request time per output token for target-only and speculative decoding. Its τ averages the per-request mean acceptance length, including the anchor. Timing starts after prefill and the first target token and includes the first draft proposal. Cross-benchmark averages are unweighted arithmetic means of the five task-level values, computed before rounding; relative improvements use these unrounded averages.

Sampling and verification. The Qwen3-8B DFlash-based configurations use the same verification mechanism as the DFlash repository (Chen et al., 2026). Draft candidates are selected greedily. Target verification selects tokens by argmax at $T = 0$ and samples from the temperature-scaled target distribution at $T = 1$. We accept the longest candidate prefix matching the corresponding target samples and emit the target sample at the first mismatch. If all candidates match, we emit the additional target sample after the block. Later verification positions are retained only while all preceding candidate tokens match, so every retained target distribution is conditioned on the actual generated prefix.

The standard speculative verification used in the Qwen3-8B and Qwen3.5-4B experiments preserves the target decoding distribution. The approximate recoverable-tree experiment in Appendix K is reported separately.

B Training-data construction

The Qwen3-8B training corpus combines prompts and dialogue structure from CodeAlpaca, Nemotron-Post-Training-Dataset-v2, ShareGPT, and UltraChat (Chaudhary, 2023; Nathawani et al., 2025; ShareGPT, 2023; Ding et al., 2023). Regenerated responses provide sequence-level teacher supervision (Kim & Rush, 2016).

Table 4: **Composition before training-time filtering.** Every retained assistant response is regenerated by Qwen3-8B. Counts refer to conversation records, not tokens or sampled training blocks.

Prompt source	Retained conversations
CodeAlpaca + Nemotron (chat/code/math/STEM)	1,416,949
ShareGPT	234,940
UltraChat-200k, train_sft	207,865
Total	1,859,754

Offline response regeneration. For the CodeAlpaca/Nemotron component, the regeneration process removes the source assistant messages and generates replacement responses turn by turn with Qwen3-8B. Multi-turn conversations retain their user turns and condition each new response on the preceding regenerated dialogue. Records beginning with an assistant turn are discarded. Generation uses SGLang (Zheng et al., 2024), temperature 0, a maximum of 4,096 new tokens per response, and thinking disabled. The run uses two SGLang server instances with request concurrency set to 256 and retains 1,416,949 successful records; 260 failed records are stored separately and excluded.

The ShareGPT and UltraChat component also replaces assistant responses with Qwen3-8B generations using the same decoding configuration. Of 235,198 formatted ShareGPT records, 234,940 succeed and 258 fail. All 207,865 UltraChat train_sft records succeed. The successful files are concatenated and shuffled to form 442,805 records, then mixed with the CodeAlpaca/Nemotron component.

Training-time filtering. We apply the Qwen chat template and tokenize each conversation with a maximum sequence length of 2,048 tokens. The supervision mask covers assistant responses only. The mixed corpus contains 1,859,754 records before tokenization, truncation, and removal

of conversations with fewer than $2K = 32$ supervised tokens. The 4,096-token offline generation cap and 2,048-token training sequence limit apply at different pipeline stages. The corpus counts in Table 4 refer to records before training-time filtering. Shuffling uses seed 42 before partitioning over data-parallel workers.

C TRAINING AND EVALUATION DETAILS

This appendix records the primary Qwen3-8B configuration and the Qwen3.5-4B transfer setup.

C.1 QWEN3-8B STUDY

Configuration. Table 5 lists the training configuration. Decoding evaluation uses four NVIDIA A30 GPUs in bfloat16. EAGLE-3 uses a training-time-test length of 7 and an inference tree budget of 16; these are separate training and inference settings.

Table 5: **Qwen3-8B training configuration.** Epoch counts apply to the specified runs. The effective global batch is 8 GPUs $\times$ 8 sequences $\times$ 12 accumulation steps $=$ 768 sequences per optimizer update.

Setting	Recorded value
Hardware and precision	$8\times$ NVIDIA H100; bfloat16
Per-device micro-batch	8 sequences
Gradient accumulation	12 steps
Effective global batch	768 sequences per optimizer update
Sequence and block length	Maximum sequence length 2,048; $K = 16$
Main stream configuration	$H = 4096$, $\rho = 64$, $G_a = G_g = 32$
Frozen components	Target model, shared token embeddings, LM head
Trainable components	Draft core, target-feature projection, normalization, stream adapters, step embeddings, retention priors, controllers
Supervision	The same position-weighted DFlash drafting objective for all models; final recurrent output only, masked assistant-response positions, and no controller-specific auxiliary loss
Recurrent gradients	Full backpropagation through the recurrent unroll
Loop-5 base training	Six epochs
Adaptation	Three epochs for multi-state and high-R extensions
Optimizer	AdamW
Base-training learning rate	1×10^{-3}
Adaptation learning rate	5×10^{-4}
Adaptation schedule	Cosine decay with warmup
Adaptation optimizer state	Reinitialized for each adaptation run
Controller initialization	$w_a = w_g = 0$; new static retention priors $\alpha_{r,s} = 0$
Other new linear layers	Framework default initialization
Data shuffling	Seed 42 before data-parallel partitioning

Evaluation comparability. We evaluate 128 GSM8K, 128 MATH-500, 164 HumanEval, 128 MBPP, and 80 MT-Bench benchmark items. All controlled methods use the same selected items, evaluation harness, and timing boundaries. Each generation call is capped at 2,048 new tokens (`max_new_tokens=2048`). This output limit is separate from the 2,048-token training sequence limit. Qwen3-8B evaluation uses thinking-disabled generation at both $T = 0$ and $T = 1$. The sampling procedure is specified in Appendix A. Four independent workers each execute batch-1 decoding on one A30 GPU in bfloat16, processing disjoint requests with up to four requests in parallel. Speedup is computed from the per-request timings collected by these workers using the metric in Appendix A.

Checkpoint origins and adaptation. Standalone Loop-1, Loop-3, Loop-5, and local physical-depth baselines are trained independently from scratch. Each standalone Loop model trains one

shared core at its evaluation recurrent depth. Multi-state adaptation uses the same access–read–core–write–retain recurrence and (R, S) as inference. The six-epoch Loop-5 checkpoint initializes the $R = 5$ multi-state variants and the $R = 15, 25, 35$ extensions, each adapted for three epochs at its evaluation depth. The high-R models are therefore adapted separately. Static and adaptive four-state branches share the backbone initialization and adaptation budget. They each receive three more epochs than the standalone Loop-5 baseline. For each adaptation run, model weights are loaded from the corresponding Loop checkpoint and the optimizer state is reinitialized. Newly introduced controllers use $w_a = w_g = 0$ and static retention priors use $\alpha_{r,s} = 0$. All other newly introduced linear layers, including the read/write adapters, use the framework default initialization. An unchanged-architecture three-epoch continuation of Loop-5 is reported in Appendix Table 8.

R3S4 initialization. The primary R3S4 starts from the independently trained Loop-3 checkpoint and is adapted for three epochs with $R = 3, S = 4$. Both Loop-3 and primary R3S4 use the three target-feature taps $[1, 17, 32]$. Evaluation uses the same recurrent depth, $R_{\text{train}} = R_{\text{eval}} = 3$, and the same taps as adaptation. Loop-5 and the $R = 5$ multi-state variants use $[1, 9, 17, 25, 33]$. Thus the R3S4–Loop-5 comparison spans operating points with different target-feature selections. Appendix E reports both single-state Loop-3 tap sensitivity and a five-tap R3S4 control matched to Loop-5's target-feature interface. The latter uses the same training-epoch budget and evaluation protocol as primary R3S4 and gives similar results.

Training batch and optimization. Training uses AdamW on eight H100 GPUs in bfloat16 with a per-device micro-batch of eight sequences and 12 gradient-accumulation steps, giving an effective global batch of 768 sequences per optimizer update. Base training uses a learning rate of 1×10^{-3}. Stage-2 adaptation halves this to 5×10^{-4} with cosine decay and warmup. The state-control branches use the same batch settings and three-epoch adaptation schedule. All models retain the same position-weighted DFlash drafting objective, with no additional controller-specific loss.

Timing and sampling repeats. We evaluate DFlash-5L, Loop-5, R3S4, and R5S4 in three complete runs at each temperature. $T = 0$ repeats measure runtime variation; $T = 1$ runs use distinct sampling seeds matched across methods. Table 1 reports their three-run means, with speedup variability in Appendix G.

C.2 QWEN3.5-4B TRANSFER SETUP

Training corpus. We follow the prompt sources and data-construction pipeline of Appendix B. For each generation regime, we construct a separate corpus by regenerating all assistant responses with Qwen3.5-4B under the corresponding thinking configuration. Both corpora use temperature 0 and a maximum of 4,096 new tokens per response; the thinking flag is enabled or disabled to match the corresponding generation regime. Within each regime, all DFlash, DSpark, and DFlash2 variants use the same target-regenerated corpus.

Evaluation protocol. Table 6 summarizes the transfer configuration. Evaluation uses a vLLM serving backend (Kwon et al., 2023) with a client concurrency of 16. All variants use the same selected benchmark items and generation limit under two regimes: thinking-enabled sampling at $T = 1$ and thinking-disabled greedy decoding at $T = 0$. For each variant, the two regimes use different trained checkpoints. Within the sampling regime, the sampling configuration is matched across variants. We report the unweighted five-task mean of acceptance length, including the anchor, separately for each regime. Table 2 displays two decimal places; relative improvements are computed before rounding these means for display.

Physical depth and target-feature inputs. Original denotes our locally trained reproduction for each of DFlash, DSpark, and DFlash2; all three Originals have six independently parameterized draft-core layers. Loop-3, Loop-5, and R5S4 instead reuse one trainable core, with parameters tied across recurrent steps. Within each backbone and generation regime, all variants share the training corpus, target model, tokenizer, and block construction. Original uses the eight target-layer indices $[1, 5, 9, 13, 17, 21, 25, 29]$; Loop-3, Loop-5, and R5S4 use $[1, 8, 15, 22, 29]$ and identical feature-projection shapes within each backbone. Original thus has six draft layers and eight target taps. The Original–Loop comparison changes both depth and target-feature selection. The Loop-3–Loop-5

comparison changes recurrent computation with the same feature interface and parameter count at the displayed precision. Training and evaluation use the same recurrent depth.

Optimization and initialization. For each regime, Original, Loop-3, and Loop-5 are each trained for six epochs with the settings in Table 6. The recurrent baselines are trained independently at their respective depths. R5S4 starts from the six-epoch Loop-5 checkpoint for the same backbone and regime, and jointly adapts the drafter and new stream and control modules for three epochs with $R = 5, S = 4$. Each adaptation run loads the model weights and reinitializes the optimizer state. The new controller output projections use $w_a = w_g = 0$, retention priors use $\alpha_{r,s} = 0$, and all remaining newly introduced linear layers use the framework default initialization.

Table 6: **Qwen3.5-4B transfer configuration.** Settings are shared across the DFlash, DSpark, and DFlash2 comparisons.

Setting	Value
Target model	Qwen3.5-4B
Benchmarks	GSM8K / MATH-500 / HumanEval / MBPP / MT-Bench
Evaluation items	128 / 128 / 164 / 128 / 80, respectively
Block size	$K = 16$
Maximum new tokens	2,048
Evaluation backend	vLLM serving
Client concurrency	16 concurrent requests
Evaluation hardware	NVIDIA A30; bfloat16
Generation regimes	Thinking-enabled sampling ($T = 1$); thinking-disabled greedy decoding ($T = 0$)
Sampling configuration	Matched across variants at $T = 1$
Training hardware	$8\times$ NVIDIA H100; bfloat16
Per-device micro-batch	8 sequences
Gradient accumulation	12 steps
Effective global batch	768 sequences per optimizer update
Optimizer	AdamW
Base-training learning rate	1×10^{-3}
Adaptation learning rate	5×10^{-4}
Adaptation schedule	Cosine decay with warmup
Original / Loop-3 / Loop-5	Six training epochs each, per regime
R5S4 adaptation	Three epochs from Loop-5 of the same backbone and regime
Stream configuration	Rank $\rho = 64$; $S = 4$ for R5S4
Original target taps	$[1, 5, 9, 13, 17, 21, 25, 29]$
Recurrent target taps	$[1, 8, 15, 22, 29]$ for Loop-3, Loop-5, and R5S4

Recurrent adaptation of each native architecture. We construct each Loop and LoopDraft variant within its native DFlash, DSpark, or DFlash2 architecture. Original retains that architecture's six-layer draft core and proposal mechanism. Loop-3 and Loop-5 replace the draft-core stack with repeated applications of one shared core, together with the target-feature configuration specified above. R5S4 adds low-rank multi-state read/write interfaces and target-guided access and persistence. DSpark retains its causal-correction component, and DFlash2 retains its correction/selection mechanism throughout the comparison.

Parameter accounting and reporting. Draft parameter counts are obtained from model parameter tensors, excluding the token embeddings and LM head shared with the target. Distinct parameters are counted once regardless of trainability or recurrent reuse. Table 2 reports these counts rounded to 0.001B, with parameter reductions relative to the corresponding Original.

D Additional state-control results

Per-benchmark core ablations. Table 7 expands the five-task averages in Table 3. At $R = 5$, Full improves both speedup and acceptance length over Static, Persistence-only and Access-only on every benchmark.

Table 7: **Core ablations by benchmark at $T = 0$.** All rows use $R = 5$. Loop-5 has $S = 1$; the remaining rows have $S = 4$. Each task reports speedup and acceptance length, including the anchor. Avg. is the arithmetic mean of the five tasks. Bold marks the best value in each column.

Variant	GSM8K		MATH-500		HumanEval		MBPP		MT-Bench		*Avg.*	
	Speedup	τ	Speedup	τ	Speedup	τ	Speedup	τ	Speedup	τ	Speedup	τ
Loop-5	4.76×	6.51	5.62×	7.50	4.30×	5.58	3.87×	5.30	2.79×	4.40	4.27×	5.86
Static	4.72×	6.58	5.57×	7.58	4.25×	5.64	3.84×	5.36	2.72×	4.44	4.22×	5.92
Persistence only	4.78×	6.68	5.64×	7.70	4.31×	5.78	3.89×	5.50	2.78×	4.59	4.28×	6.05
Access only	4.84×	6.80	5.70×	7.86	4.37×	5.90	3.94×	5.60	2.85×	4.69	4.34×	6.17
Full	**5.08×**	**7.00**	**5.96×**	**8.05**	**4.49×**	**6.08**	**4.20×**	**5.72**	**3.01×**	**4.90**	**4.55×**	**6.35**

Table 8: **Additional state-control and training-budget comparisons at $T = 0$.** All rows use $R = 5$. Loop-5 uses 0.270B parameters; other rows use approximately 0.274B. Loop-5 and one-state adapters use $S = 1$; others use $S = 4$. Metrics are five-task arithmetic means.

Configuration	Speedup	τ
Loop-5 (6 epochs)	4.27×	5.86
Loop-5 (+3 epochs)	4.25×	5.90
Full target-guided control	4.55×	6.35
Local dynamic access + persistence	4.35×	6.17
Parameter-matched $S = 1$ adapters	4.21×	5.90
Full + dynamic write scale	4.56×	6.38

Continued-training control. We continue the six-epoch Loop-5 checkpoint for three epochs with the same optimizer reset and adaptation schedule as the multi-state runs, without changing the architecture. Speedup changes from 4.27× to 4.25× and τ from 5.86 to 5.90. Extra training thus yields little benefit under this schedule. Full reaches 4.55× and $\tau = 6.35$ with the same additional training budget. Its gains over this control are 7.1% in speedup and 7.6% in acceptance length.

Target context. Removing the target-feature branches from both controllers, as in the local dynamic variant, gives $\tau = 6.17$ and 4.35×. Full improves these averages by 0.18 tokens and approximately 0.20×.

Parameter control. An approximately parameter-matched one-state adapter control reaches $\tau = 5.90$ and 4.21×, compared with 6.35 and 4.55× for Full.

Adaptive write scaling. Adding a dynamic write scale to Full yields $\tau = 6.38$ and 4.56×, compared with $\tau = 6.35$ and 4.55× for Full. Given the small observed differences, we retain the input-independent write maps in our main model (Section 3).

E Additional shallow and recurrent baselines

Table 9: **Additional local baseline configurations at $T = 0$.** Parameters are in billions; speedup and τ average the five benchmarks.

Drafter	Target taps	Params (B)	Speedup	τ
1-layer DFlash	[18]	0.21	2.28×	2.56
1-layer DFlash	[33]	0.21	2.65×	3.08
1-layer DFlash	[1, 17, 32]	0.24	2.82×	3.19
3-step Loop	[1, 17, 32]	0.24	3.78×	4.65
3-layer DFlash	[1, 17, 32]	0.62	3.89×	4.76

The first two rows provide single-tap references. The final three fix target taps to $[1, 17, 32]$: the one-layer and three-step drafters compare refinement at the same parameter count, while the three-step and three-layer drafters compare shared and independent parameters at the same number of layer evaluations. The primary R3S4 also uses these three taps.

Target-feature tap sensitivity. We evaluate single-state Loop-3 with one target-feature tap $[32]$, three taps $[1, 17, 32]$, and five taps $[1, 9, 17, 25, 33]$ (Table 10). At $T = 0$, mean speedup ranges from $3.72\times$ to $3.84\times$ and τ from 4.56 to 4.73, showing modest sensitivity to tap selection in this single-state $R = 3$ baseline.

Matched-tap R3S4 control. The primary R3S4 uses $[1, 17, 32]$, whereas Loop-5 uses $[1, 9, 17, 25, 33]$. We evaluate an R3S4 control with Loop-5's five taps and the corresponding feature projection, keeping the training-epoch budget and evaluation protocol unchanged. At $T = 0$, this control reaches $4.44\times$ and $\tau = 6.06$, compared with $4.41\times$ and 6.02 for primary R3S4. At $T = 1$, the five- and three-tap variants reach $3.83\times/\tau = 5.27$ and $3.78\times/\tau = 5.22$, respectively. Matching Loop-5's target-feature interface thus preserves the lower-compute R3S4 operating point.

Table 10: **Target-tap sensitivity and matched-tap R3S4 control on Qwen3-8B.** All rows use $D = 1$, $R = 3$, $K = 16$, and $T = 0$; Loop-3 uses $S = 1$ and R3S4 uses $S = 4$. Metrics are five-task means; τ includes the anchor. Three-tap rows are the primary configurations. Bold marks the best values within each variant.

Variant	Target taps	Speedup	τ
Loop-3	$[32]$	$3.72\times$	4.56
Loop-3	$[1, 17, 32]$	$3.78\times$	4.65
Loop-3	$[1, 9, 17, 25, 33]$	**$3.84\times$**	**4.73**
R3S4	$[1, 17, 32]$	$4.41\times$	6.02
R3S4	$[1, 9, 17, 25, 33]$	**$4.44\times$**	**6.06**

F State-control position features

For block-position index $k \in \{0, \ldots, K-1\}$, where $k = 0$ is the anchor, let

$$q_k = \begin{cases} k/(K-1), & K > 1, \\ 0, & K = 1. \end{cases} \tag{20}$$

Both controllers use the block-length-independent position code

$$\phi(k) = \left[q_k,\ q_k^2,\ \sin(\pi q_k),\ \cos(\pi q_k)\right]^\top. \tag{21}$$

The access and persistence controllers project these features with A_p and G_p, respectively.

G Timing and sampling variability

Table 11 summarizes three complete Qwen3-8B runs per temperature for the four main configurations. Checkpoints, benchmark items, $K = 16$, the 2,048-token generation limit, and the four-worker bfloat16 A30 setup are fixed. Each run remeasures both target-only and speculative decoding using the timing boundaries in Appendix A. We average task-level speedups equally within each run, then average across runs for Table 1.

At $T = 0$, generated sequences are fixed and the repetitions measure runtime variation. At $T = 1$, each repetition uses a distinct sampling seed, matched across methods within that repetition. These runs capture sampling-induced sequence variation as well as runtime variation.

At either temperature, the largest sample standard deviation rounds to $0.02\times$. R3S4 and R5S4 improve mean speedup over Loop-5 by $0.14\times$ and $0.28\times$ at $T = 0$, and by $0.14\times$ and $0.27\times$ at $T = 1$. These mean gains exceed the observed run-to-run standard deviations.

Table 11: **Run-to-run speedup variability at $T = 0$ and $T = 1$.** Each temperature uses three complete evaluation runs. Mean and sample standard deviation ($n = 3$) are computed across run-level five-task averages. $T = 0$ measures runtime variation; distinct sampling seeds at $T = 1$ add sequence variation.

Method	$T = 0$				$T = 1$
	Run 1	Run 2	Run 3	Mean ± Std.	Mean ± Std.
DFlash-5L	4.26×	4.28×	4.27×	$(4.27 \pm 0.01)\times$	$(3.64 \pm 0.01)\times$
Loop-5	4.25×	4.28×	4.28×	$(4.27 \pm 0.02)\times$	$(3.64 \pm 0.02)\times$
LoopDraft R3S4	4.40×	4.42×	4.41×	$(4.41 \pm 0.01)\times$	$(3.78 \pm 0.01)\times$
LoopDraft R5S4	4.53×	4.56×	4.56×	$(4.55 \pm 0.02)\times$	$(3.91 \pm 0.02)\times$

H Parameter, compute, and memory cost

Reported draft parameter counts include all draft-specific parameters, excluding token embeddings and the language-model head shared with the target. Each distinct parameter is counted once regardless of trainability; reusing the core across recurrent steps does not multiply its parameter count.

Let G_a and G_g be the hidden widths of the access and persistence controllers. The additional parameter count is

$$P_{\text{extra}} = H\rho(1 + 3S) + H + R(H + S) + G_g(H + 2\rho + R + 5) + G_a(H + \rho + R + 5). \quad (22)$$

The first terms cover the learned, input-independent read/write maps, normalization, step embeddings, and static retention priors. For $H = 4096$, $\rho = 64$, $G_a = G_g = 32$, $R = 5$, and $S = 4$, the base multi-stream path adds 3.43M parameters, persistence adds 0.14M, and access adds 0.13M, for 3.70M in total. For fixed R and $S > 1$, access/retention mode ablations instantiate the same controller parameters and disable a mechanism through its mode, keeping model shape fixed. This statement does not apply to the plain $S = 1$ baseline.

Ignoring normalization and element-wise operations, the four read/write projections each require $NKSH\rho$ multiply–accumulate operations (MACs) per recurrent step. In particular, the shared down-projection is applied to all S streams. Thus

$$\text{MACs}_{\text{read/write}} \approx 4RNKSH\rho, \qquad \text{complexity} = O(RNKSH\rho). \quad (23)$$

Counting one MAC as two FLOPs gives approximately $8RNKSH\rho$ FLOPs. The two controllers add $O(RNKS\rho(G_a + 2G_g) + RNH(G_a + G_g))$. The attention–FFN core is evaluated R times, not RS times. Persistent recurrent-state memory scales as $O(NSKH)$; training activation memory additionally depends on the unrolled recurrence and checkpointing strategy. Thus D duplicates independently parameterized Transformer layers, R adds shared-core latency, and S adds low-rank communication and state memory without multiplying core evaluations.

For $K = 16$, $H = 4096$, and bf16 states, the recurrent-state tensor occupies 0.125, 0.50, and 1.00 MiB per active sequence at $S = 1, 4, 8$, respectively. These figures exclude temporary activations, KV-cache storage, and framework workspaces.

I Composition with backbone-specific correction

Recurrent state width and semi-autoregressive correction act at different points in the proposal path. LoopDraft refines latent states across recurrent depth before token selection; a DSpark-style head subsequently uses sampled predecessor tokens to propagate information across candidate positions. The causal head therefore composes with $X^{(R)}$ without changing the inner access–read–core–write–retain recurrence. Section 4.3 evaluates the resulting state-control strategy with DSpark and DFlash2 backbones, which incorporate causal correction and predecessor-conditioned candidate selection, respectively.

J PHYSICAL-DEPTH AND STATE-WIDTH MEASUREMENTS

The following tables provide the numerical comparisons discussed in the main Qwen3-8B depth and width experiments. Both use $T = 0$; parameter counts exclude token embeddings and the LM head shared with the target.

Table 12: **Physical and recurrent depth.** Target-feature taps are matched within each pair. All rows have $S = 1$. Parameters are in billions; speedup and τ are five-task arithmetic means.

Drafter	D	R	Target taps	Params (B)	Speedup	τ
DFlash-3L	3	1	$[1, 17, 32]$	0.62	3.89×	4.76
Loop-3	1	3	$[1, 17, 32]$	0.24	3.78×	4.65
DFlash-5L	5	1	$[1, 9, 17, 25, 33]$	1.05	4.27×	6.00
Loop-5	1	5	$[1, 9, 17, 25, 33]$	0.27	4.27×	5.86

Table 13: **State width and a lower-compute operating point.** Multi-state rows use full target-guided control; Loop-5 ($R = 5, S = 1$) is the single-state recurrent baseline. Draft latency is measured per speculative proposal round in milliseconds, including all R recurrent steps. Speedup and τ are arithmetic task averages, not reconstructed from averaged timings.

R	S	Params (B)	Draft (ms)	Speedup	τ
5	1	0.270	9.56	4.27×	5.86
5	2	0.272	9.80	4.43×	6.13
5	4	0.274	10.12	4.55×	6.35
5	8	0.277	10.75	4.48×	6.44
3	4	0.244	6.65	4.41×	6.02

K COMPOSITION WITH RECOVERABLE TREE VERIFICATION

We compose the fixed Loop-5 and R5S4 drafters with the same 16-node recoverable tree verifier. Block distributions and request-local path continuity define a prefix-closed tree; the target scores its 16 non-root nodes plus the anchor in one ancestor-masked pass. Exact following takes priority, after which an already-scored sibling may pass frequency, target-support, and stop-token gates. Recovery adds no target pass or drafter training. Because a recovered token may differ from the target's greedy choice, this auxiliary composability test uses approximate decoding.

Tree results use the same subsets, A30 hardware, target-only baseline, and timing boundaries as Table 1. Recovery memory is calibrated per drafter with the same held-out protocol and no evaluation-set tuning; verifier hyperparameters remain fixed. Here τ counts committed tokens, including the anchor and recovered siblings.

Table 14: **Composition with recoverable tree verification.** Speedup and τ are five-task arithmetic means on Qwen3-8B at $T = 0$ with thinking disabled. Standard-linear rows from Table 1 serve as references; the controlled comparison is Loop-5 versus R5S4 under the same 16-node tree verifier. The tree permits up to 17 current tokens including the anchor, versus 16 for standard linear verification.

Drafter	Verification	Speedup	τ
Loop-5	Standard linear	4.27×	5.86
LoopDraft R5S4	Standard linear	4.55×	6.35
Loop-5	Recoverable tree (16 nodes)	4.42×	7.01
LoopDraft R5S4	Recoverable tree (16 nodes)	**4.68×**	**7.46**

Under the identical recoverable tree verifier, replacing Loop-5 with R5S4 increases mean speedup from 4.42× to 4.68× (+5.9%) and τ from 7.01 to 7.46 (+6.4%). The R5S4 advantage therefore persists when its block distributions feed downstream tree construction and recovery.