WHEN TO TRUST THE PAST: RELIABILITY-CALIBRATED DRAFT-TREE ORCHESTRATION FOR SPECULATIVE DECODING

Anonymous authors
Paper under double-blind review

ABSTRACT

Speculative decoding accelerates autoregressive inference in large language models (LLMs) through a guess-then-verify paradigm, in which multiple candidate continuations are drafted and verified in parallel. Recent methods combine suffix-based retrieval with target-model logits recycled from previous verification steps to reuse historical continuations, thereby expanding candidate coverage. However, such reuse implicitly assumes that historical predictions remain reliable under the current prefix, while poorly transferable candidates occupy limited verification capacity and crowd out more promising alternatives when this assumption fails, an issue which is further exacerbated as history accumulates, since the same token recurs under increasingly diverse prefixes. Towards this end, we propose a novel framework termed Reliability-Calibrated Draft-Tree Orchestration for speculative decoding (**RECAP**). The core of **RECAP** lies in estimating prefix-dependent reliability of historical predictions and then using this calibrated evidence to jointly determine candidate composition and the allocation of the verification budget. In particular, we first construct a multi-context verification memory that stores target-model predictions together with their corresponding prefix contexts. Subsequently, we introduce a lightweight prefix-conditioned scorer that estimates the transferability of each memory entry to the current decoding state and aggregates reliable historical evidence. Based on the resulting evidence, **RECAP** combines retrieved continuations with logit-based candidates under a shared verification budget and expands the draft tree best-first along the paths of highest cumulative reliability. Together, these components transform draft-tree construction from a fixed heuristic into an orchestration process that adapts both candidate sources and tree shape to the available evidence. Extensive experiments show that **RECAP** consistently improves decoding efficiency over strong speculative-decoding baselines while preserving generation quality. The implementation of **RECAP** is available at https://anonymous.4open.science/r/RECAP7C2E.

1 INTRODUCTION

Large language models (LLMs) have achieved strong performance across reasoning (Snell et al., 2025), code generation (Ran et al., 2026), and interactive applications (Du et al., 2026). Yet their deployment remains constrained by autoregressive decoding, in which every new token requires a forward pass of the target model, so that latency grows with output length (Fu et al., 2026). To alleviate this bottleneck, speculative decoding (Leviathan et al., 2023) accelerates generation by proposing multiple future tokens and verifying them in parallel with the target model (Chen et al., 2023; Miao et al., 2024), but this also makes its effectiveness depend on whether a limited speculative budget goes to candidates that are actually likely to survive verification.

Accordingly, prior work has pursued more reliable drafting along two complementary directions (Huang et al., 2026; Do et al., 2026). Model-based approaches introduce auxiliary predictors (Cai et al., 2024), additional decoding heads (Li et al., 2024a), or target-model features to generate future tokens (Li et al., 2024b; 2025), which predict well at the cost of additional training and inference overhead. Model-free approaches instead reuse, without a draft model, information already available during generation (Luo et al., 2025), including repeated text patterns (He et al., 2024), suffix

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.

structures (Hu et al., 2025), and previously computed target-model logits (Liu et al., 2026c; Zhang et al., 2026). Furthermore, tree-based verification lets multiple continuations compete for one shared verification budget in each round (Miao et al., 2024; Wang et al., 2025a).

However, historical target-model predictions are prefix-conditioned rather than token-conditioned (Pandia & Ettinger, 2021): the same token may recur under distinct prefixes with different next-token distributions, so reuse based solely on the final token conflates *token identity* with *prediction state* (Bajaj et al., 2026; Khandelwal et al., 2025), and the mismatch grows as repeated tokens accumulate increasingly heterogeneous prefix states (Hou et al., 2025). As Figure 1 illustrates, model-free speculative decoding thus faces two coupled mismatches: an *evidence mismatch*, since evidence recorded under past prefixes must remain reliable under the current branch, which token matching or recency alone cannot ensure (Gur-Arieh et al., 2026; Wang et al., 2025b), and an *allocation mismatch*, since the tree budget is finite, yet weakly supported branches still consume exactly the same capacity as reliable ones (Chen et al., 2024).

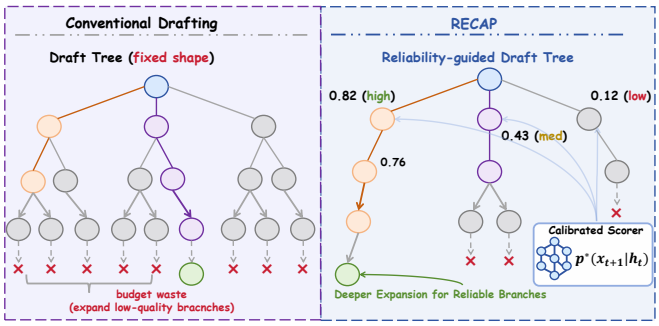


Figure 1: Paradigm comparison between conventional drafting and **RECAP**. While fixed-structure drafting allocates verification budget independently of branch reliability, **RECAP** calibrates contextual evidence and guides tree expansion toward reliable continuations, enabling adaptive budget use.

To address these mismatches, we propose **RECAP**, a reliability-calibrated framework for model-free speculative decoding. **RECAP** reformulates historical reuse as contextual evidence selection followed by budget-aware tree orchestration. First, *contextual evidence calibration* preserves multiple target-model prediction records associated with the same token and evaluates their relevance to the current speculative prefix. Evidence from compatible historical contexts is consolidated at the candidate level, while an explicit fallback state represents cases where the available history does not contain a reliable continuation. This prevents the decoder from becoming overconfident simply because cached candidates exist. Second, *reliability-guided draft orchestration* converts calibrated candidate confidence into path-level priorities and allocates a fixed tree budget through best-first expansion. Reliable branches naturally receive more speculative depth, whereas uncertain branches stop consuming verification capacity. Exact n-gram retrieval is retained as a lightweight structural anchor whenever sufficiently strong repeated patterns are readily available in the context.

The two mismatches are handled at different stages of speculation. Contextual calibration determines *which historical predictions remain transferable*, while reliability-guided orchestration determines *how much verification capacity they should receive*. The resulting draft tree is verified by the unchanged target model through standard tree attention, so **RECAP** modifies only how speculative capacity is allocated rather than the target decoding rule. This design allows richer decoding histories to provide useful evidence without assuming that all historical predictions are equally reusable. On the whole, the contributions of **RECAP** are summarized as follows:

❶ ***Novel Perspective.*** We identify an underexplored failure mode of history-based drafting: the same token can be reused across heterogeneous prefix contexts, making token-level historical evidence unreliable and wasting verification budget. This mismatch compounds with longer decoding histories, as recurring tokens accumulate evidence from increasingly diverse prefixes.

❷ ***New Framework with Theoretical Support.*** Training only a lightweight scorer, **RECAP** calibrates the transferability of historical predictions using cross-prefix evidence to guide draft-tree orchestration and budget allocation. Furthermore, we prove that when tree orchestration relies on these estimates rather than the true acceptance probabilities, the resulting loss in expected committed length is bounded by additive terms scaling with the free budget and squared tree depth.

❸ ***Empirical Evaluation.*** Across three target models and five benchmarks, **RECAP** consistently achieves the highest MAT and speedup, improving average MAT by 32-38% over the strongest baseline without compromising generation quality. Its advantage further widens markedly in persistent agent sessions, where the interaction history keeps accumulating across calls.

2 Related Work

Model-Free Speculative Decoding. Unlike model-based methods, which train auxiliary drafters (Cai et al., 2024; Li et al., 2025), model-free methods, including all our baselines, draft from what generation has already produced: PLD (Saxena, 2023) and SuffixDecoding (Oliaro et al., 2025) copy continuations from existing text, whereas Token Recycling (Luo et al., 2025) and LogitSpec (Liu et al., 2026c) reuse the predictions that the target has already computed, and RACER (Zhang et al., 2026) combines both through copy logits and rule-based breadth allocation. However, predictions attached to the same token may arise from different contexts and thus differ in relevance to the current branch (Ma et al., 2025; Chen et al., 2025a). By comparison, **RECAP** keeps context-annotated prediction records and calibrates their reliability with a lightweight scorer.

Draft-Tree Orchestration. Since the verification budget is limited, its allocation across branches matters as much as the candidates themselves (Miao et al., 2024; Wu et al., 2025). Sequoia (Chen et al., 2024) and OPT-Tree (Wang et al., 2025a) optimize the tree shape for the expected acceptance length, whereas EAGLE-2 (Li et al., 2024b) and SpecExec (Svirschevski et al., 2024) expand what the draft model deems most probable, and C2T (Huo et al., 2025) and Goose (Jin et al., 2026) adapt the tree to token reliability. Like Sequoia and OPT-Tree, **RECAP** maximizes the expected acceptance length under a fixed budget, but it estimates the acceptance of each branch from calibrated historical evidence rather than from a separate draft model and then expands the tree best-first.

3 The Proposed RECAP

Problem Formulation. Let p_θ be a frozen target language model and $X_t = (x_1, \ldots, x_t)$ the verified prefix. At each decoding step, we construct a prefix-closed draft tree $\mathcal{T}_t$ under a fixed verification budget $|\mathcal{T}_t| \le B$. Let A_v denote the event that the prefix path ending at node v is accepted by the target verifier. The draft-tree construction problem can be viewed as:

$$\max_{\mathcal{T}_t} \sum_{v \in \mathcal{T}_t} \Pr(A_v), \qquad \text{s.t. } \mathcal{T}_t \text{ is prefix-closed, } |\mathcal{T}_t| \le B. \tag{1}$$

On the whole, the objective of **RECAP** is to estimate which historical continuations remain reliable under the current prefix and allocate the limited tree budget accordingly.

3.1 Framework Overview

Within this budget, the draft tree should be filled with the continuations that the target model is most likely to accept, yet this acceptance is unknown before verification. To estimate it without an auxiliary drafter, **RECAP** organizes speculative decoding around contextual evidence calibration and reliability-guided tree orchestration. On the one hand, the target-model predictions recorded under earlier prefixes are weighed by how well they transfer from their source contexts to the current branch, which yields calibrated successor confidence together with an estimate of candidate coverage. On the other hand, this evidence is merged with retrieved continuations under a shared verification budget and turned into path priorities that grow the draft tree best-first. In this way, each round spends its budget on the continuations best supported by the available evidence. The detailed framework of **RECAP** is further illustrated schematically in Figure 2.

3.2 Reliability-Calibrated Historical Evidence

Since a token-indexed cache discards the prefix behind each prediction, we introduce a multi-context memory that keeps several prefix-tagged records per token and a scorer that learns which of them transfer to the current branch, weighting evidence by reliability, not by token identity.

Multi-Context Verification Memory. Since token-indexed history can conflate predictions produced under heterogeneous prefixes (Oh & Schuler, 2023), we construct a multi-context verification memory that stores, for every token, several target-model predictions together with the contexts that produced them. Formally, let s_u denote the prefix represented by draft-tree node u, i.e., the committed prefix followed by the draft path to u, and let v_u denote its final token. To keep the scoring cost independent of the prefix length, the scorer reads only the bounded context $c_u = \mathrm{suffix}_L(s_u)$ of the

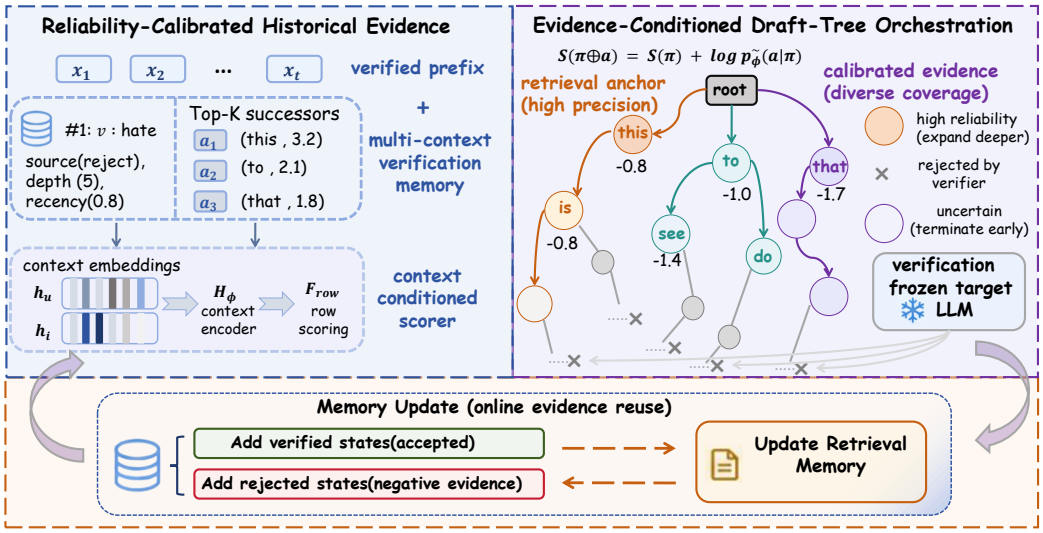


Figure 2: Overview of RECAP. Historical verifier states are calibrated against the current prefix and combined with retrieval to form a shared proposal space. A reliability-guided best-first orchestrator allocates the verification budget, while accepted and rejected states alike are recycled for later rounds.

last L tokens. For each token v, we then maintain a dedicated bank:

$$\mathcal{B}_t(v) = \{r_i\}_{i=1}^{m_v}, \qquad m_v \le M, \tag{2}$$

where the capacity M fixes both the memory and the number of records scored per node. Each such historical record:

$$r_i = \left(c_i, \{(a_{ij}, \ell_{ij})\}_{j=1}^{K}, m_i, \tau_i\right) \tag{3}$$

pairs the context c_i under which it was written with the Top-K successors a_{ij} and logits ℓ_{ij} returned by the target model, while the provenance m_i and the write index τ_i record where and when the state arose, so that the scorer can weigh evidence by origin and recency. Each round reads a frozen snapshot of this memory, and rejected states remain stored because the target has already paid for their predictions. In this way, the memory preserves the prefix information that token-indexed reuse discards, at a memory cost that stays strictly fixed per token.

Cross-Prefix Evidence Transfer. A lightweight evidence scorer then estimates how well each stored record transfers to the current branch: it compares the current context with every historical context in the retrieved bank and scores each stored candidate accordingly. Specifically, the scorer projects the frozen target embedding table to a compact space and encodes both of these contexts jointly with a single shared GRU (Cho et al., 2014),

$$\mathbf{q}_u = H_\phi(c_u), \qquad \mathbf{h}_i = H_\phi(c_i), \tag{4}$$

so that the two representations lie in the same space and can be compared directly. Moreover, since $\mathbf{h}_i$ is cached when its record is written, only the current context has to be encoded in each round. The scorer then compares the two representations through their difference and elementwise product and combines the result with the metadata $\boldsymbol{\eta}_i$, which describes where and when the record was written:

$$\boldsymbol{\rho}_{ui} = F_{\text{row}}\left([\mathbf{q}_u; \mathbf{h}_i; |\mathbf{q}_u - \mathbf{h}_i|; \mathbf{q}_u \odot \mathbf{h}_i; \boldsymbol{\eta}_i]\right). \tag{5}$$

For each occurrence (i, j), it further relates this row representation to the embedding $\mathbf{e}(a_{ij})$ of the candidate token and to the cached-logit features $\boldsymbol{\psi}_{ij}$, which summarize how confident the target model originally was in a_{ij} within its own source context:

$$\mathbf{o}_{uij} = F_{\text{occ}}\left([\boldsymbol{\rho}_{ui}; \mathbf{e}(a_{ij}); \mathbf{q}_u \odot \mathbf{e}(a_{ij}); \mathbf{h}_i \odot \mathbf{e}(a_{ij}); \boldsymbol{\psi}_{ij}]\right), \qquad s_{uij} = F_{\text{score}}(\mathbf{o}_{uij}). \tag{6}$$

Because the same successor can appear in several records, its evidence is deliberately merged only after the occurrence-level scoring step,

$$z_u(a) = \log \sum_{(i,j)\in\mathcal{O}_u(a)} \exp(s_{uij}), \qquad \mathcal{O}_u(a) = \{(i,j) : a_{ij} = a\}, \tag{7}$$

so that repeated observations reinforce one another instead of being treated as separate actions. Furthermore, since the stored candidates do not always cover the verifier's next token, we introduce an out-of-support state $\perp$, whose logit $z_u(\perp)$ is computed from a pooled summary of the retrieved rows and occurrences, which finally yields the overall scorer distribution:

$$\widetilde{p}_\phi(y \mid u) = \frac{\exp z_u(y)}{\exp z_u(\perp) + \sum_{a \in \mathcal{U}_u} \exp z_u(a)}, \qquad y \in \mathcal{U}_u \cup \{\perp\}, \tag{8}$$

where $\mathcal{U}_u$ is the set of unique historical successor tokens. Since $\perp$ is never inserted into the tree, a large $\perp$ probability lowers the mass of every executable proposal of u, so that the orchestrator spends less budget below u. We train the scorer with the frozen target model as the teacher, so that these estimates follow the verifier's actual decisions: for each training state u_i, the label Y_i is the verifier's Top-1 token when it belongs to $\mathcal{U}_{u_i}$ and $\perp$ otherwise, and the scorer minimizes:

$$\mathcal{L}_{\text{cal}} = -\sum_i w_i \log \widetilde{p}_\phi(Y_i \mid u_i). \tag{9}$$

Since the states that the orchestrator visits depend on the scorer itself, we first train on verifier trajectories collected in advance and then continue on-policy, on trajectories that the learned scorer collects on the same prompts, where accepted decisions that support longer verified continuations receive larger weights w_i. In this way, the scorer turns token-indexed history into a context-conditioned estimate of acceptance, while the target model remains frozen.

3.3 Evidence-Conditioned Draft-Tree Orchestration

Even with calibrated evidence, a fixed draft-tree template allocates the verification budget by a predefined shape regardless of how reliable each branch is, so that unreliable branches can occupy capacity that reliable ones could use (Chen et al., 2024). To resolve this allocation mismatch, **RECAP** lets the calibrated evidence decide both the composition of the proposals and the growth of the tree, first combining retrieval and history-based proposals under a shared budget and then expanding the tree best-first along the paths of highest cumulative reliability. In this way, the verification budget closely follows the evidence instead of a predefined tree shape.

Coverage-Adaptive Proposal Composition. Proposals come from two complementary sources: exact retrieval yields high-precision continuations whenever a matching n-gram exists but covers few positions (Chen et al., 2025b), whereas historical prediction covers far more positions but is reliable only when its source prefix transfers (Liu et al., 2026b). We therefore compose them under a shared verification budget rather than treating them as independent draft sources. Denoting by $\mathcal{T}_R$ the retrieval tree and by $\mathcal{T}_H$ the history-based tree, both rooted at the root token r, we can then write the resulting joint proposal set compactly as:

$$\mathcal{T} = \mathcal{T}_R \cup \mathcal{T}_H, \qquad |\mathcal{T}_R \setminus \{r\}| \le R, \qquad |\mathcal{T}| \le B, \tag{10}$$

where the reserve $R \ll B$ caps retrieval at R non-root nodes and releases any unused part to history-based expansion, so that retrieval acts as a small high-confidence anchor while history supplies the broad coverage that retrieval lacks, and a path proposed by both sources is verified only once.

Confidence-Adaptive Tree Budgeting. Within this proposal set, **RECAP** ranks complete speculative prefixes by their cumulative reliability and expands them best-first, so that the budget goes to the paths that are most likely to be accepted. Starting from the root path with $S(\pi_0) = 0$, the score of a path π extended by one further token a is defined recursively as:

$$S(\pi \oplus a) = S(\pi) + \log \widetilde{p}_\phi(a \mid \pi), \qquad \pi^\star = \arg\max_{\pi \in \mathcal{F}} S(\pi), \tag{11}$$

where $\mathcal{F}$ is the frontier of open paths. Scoring a path only pushes its eight best successors into $\mathcal{F}$, whereas a path consumes budget only when it is popped as the best open path, so that the popped paths form a prefix-closed tree of the highest-scoring candidates, since $S(\pi \oplus a) \le S(\pi)$, in which high-confidence branches grow deeper while uncertain ones terminate early. Expansion stops at $|\mathcal{T}| = B$, and RACER's fixed tree takes over when the root token has no stored record or fills the remaining budget entirely whenever the frontier happens to empty early.

3.4 Overall Pipeline

At the core of **RECAP** is a closed loop in which the verification memory supplies calibrated evidence for building each draft tree and every verification returns new evidence to the memory, as Algorithm 1 in Appendix B.1 summarizes. The prefill first writes one record of Eq. 3 per prompt token into the memory. Starting from this initialized memory, each decoding round takes the target's Top-1 token as the root and lets exact retrieval propose at most R nodes. On the memory frozen for the round, the remaining budget is filled by scoring candidate successors with Eqs. 4–8 and expanding the tree best-first by Eq. 11 until the proposal set of Eq. 10 holds B nodes. The target verifies the whole tree in one forward pass and commits the longest path that matches its greedy choices, after which every verified node, accepted or rejected, writes its Top-K predictions back into the memory, whose banks keep the M most recent records per token. In this way, every round both commits tokens and steadily enlarges the evidence available to all later rounds.

3.5 Theoretical Analysis

In this section, to examine whether the estimation errors of the scorer are amplified as the draft tree grows larger and deeper, we bound the expected committed length that an orchestration round loses because the orchestrator uses the estimated acceptance probabilities instead of the true ones, and show that this loss grows additively in the free budget and the squared tree depth, not in their product. All proofs and supporting lemmas are deferred to Appendix A.

Definition 3.1 *Let $\mathcal{C}$ be a finite candidate tree of maximum depth D that is rooted at the guaranteed root r and contains the retrieval subtree $\mathcal{T}_R$. A tree $\mathcal{T}$ is feasible if it is prefix-closed and satisfies $\mathcal{T}_R \subseteq \mathcal{T} \subseteq \mathcal{C}$ and $|\mathcal{T}| \le B$, and $n = B - |\mathcal{T}_R|$ denotes the remaining node capacity. For an edge $e = (u, v)$, where v also denotes the token that extends u, let $p_e = \Pr(A_v \mid A_u)$ be the true conditional acceptance probability, let $q_e = \widetilde{p}_\phi(v \mid u)$ be its estimate given by Eq. 8, and let $\mathcal{P}(v)$ be the set of edges on the path from r to v. The expected committed length, i.e., the objective of Eq. 1, and the surrogate optimized by the orchestrator are, respectively,:*

$$J(\mathcal{T}) = 1 + \sum_{v \in \mathcal{T} \setminus \{r\}} \prod_{e \in \mathcal{P}(v)} p_e, \qquad \widehat{J}(\mathcal{T}) = 1 + \sum_{v \in \mathcal{T} \setminus \{r\}} \prod_{e \in \mathcal{P}(v)} q_e, \tag{12}$$

and $\mathcal{T}^\star$ and $\widehat{\mathcal{T}}$ denote maximizers of J and $\widehat{J}$ over the feasible trees, respectively.

Lemma 3.1 *If $|p_e - q_e| \le \varepsilon$ holds for every candidate edge, then $0 \le J(\mathcal{T}^\star) - J(\widehat{\mathcal{T}}) \le 2\varepsilon n D$.*

Exploiting the normalization of the outgoing probabilities at each node, we obtain a bound in which the free budget and the tree depth enter only additively, instead of multiplicatively.

Theorem 3.1 *Let $|p_e - q_e| \le \varepsilon$ on every candidate edge, and $\sum_v p_{uv} \le 1$ and $\sum_v q_{uv} \le 1$ at every node u. Then it holds that*

$$0 \le J(\mathcal{T}^\star) - J(\widehat{\mathcal{T}}) \le \varepsilon\Big(2n + \tfrac{D(D-1)}{2}\Big). \tag{13}$$

Remark 3.1 *When $D \le n$, Theorem 3.1 sharpens the bound in Lemma 3.1 from $O(\varepsilon n D)$ to $O(\varepsilon(n + D^2))$. Both normalization conditions hold for **RECAP**: under greedy verification, at most one child per node can be accepted, while Eq. 8 assigns the residual probability mass to $\perp$.*

Since the bound depends only on where the two trees disagree, it can be sharpened further.

Theorem 3.2 *Let $\Delta = \mathcal{T}^\star \triangle \widehat{\mathcal{T}}$ with $m = |\Delta|$, let $d(v)$ denote the depth of node v, and let $K_\Delta = \max_\pi \sum_{v \in \Delta \cap \pi} (d(v) - 1)$, where π ranges over the root-to-leaf paths of $\mathcal{T}^\star \cup \widehat{\mathcal{T}}$. Under the conditions of Theorem 3.1, $0 \le J(\mathcal{T}^\star) - J(\widehat{\mathcal{T}}) \le \varepsilon\,(m + K_\Delta)$.*

Remark 3.2 *Since $m \le 2n$ and $K_\Delta \le D(D-1)/2$, Theorem 3.2 implies Theorem 3.1, and it is tighter when the two trees disagree on few nodes or only near the root.*

Table 1: Performance comparison across five benchmarks. The mean accepted tokens per verification round (MAT) and speedup over autoregressive decoding are reported in the table. **Bold** and underlined indicate the best and second-best results, respectively.

Backbone	Method	LiveBench		LongBench-Write		Spec-Bench		Alpaca		WildBench		Avg	
		MAT↑	Speedup↑	MAT↑	Speedup↑	MAT↑	Speedup↑	MAT↑	Speedup↑	MAT↑	Speedup↑	MAT↑	Speedup↑
Qwen3-8B	PLD	2.00	1.61	1.46	1.11	1.38	1.23	1.19	1.01	1.51	1.25	1.51	1.24
	TR	3.09	2.13	2.60	1.66	2.62	2.12	2.48	1.77	2.74	2.01	2.71	1.94
	LogitSpec	2.75	2.05	1.98	1.37	1.84	1.56	1.60	1.23	1.99	1.55	2.03	1.55
	RACER	3.27	2.40	2.54	1.78	2.45	2.04	2.16	1.66	2.63	2.03	2.61	1.98
	RECAP	**4.54**	**2.63**	**3.47**	**2.03**	**3.29**	**2.25**	**2.90**	**1.82**	**3.66**	**2.36**	**3.57**	**2.22**
Llama-3.1-8B	PLD	2.11	1.82	1.64	1.43	1.52	1.28	1.20	1.02	1.66	1.46	1.63	1.40
	TR	3.08	2.36	2.79	2.27	2.70	1.93	2.53	1.86	2.90	2.32	2.80	2.15
	LogitSpec	2.83	2.31	2.20	1.84	2.04	1.60	1.63	1.30	2.21	1.87	2.18	1.78
	RACER	3.33	2.68	2.71	2.24	2.62	2.03	2.23	1.76	2.84	2.36	2.75	2.21
	RECAP	**4.78**	**2.95**	**3.83**	**2.51**	**3.45**	**2.10**	**2.98**	**1.88**	**3.95**	**2.62**	**3.80**	**2.41**
Mistral-7B	PLD	1.94	1.49	1.40	1.11	1.51	1.22	1.17	0.98	1.59	1.23	1.52	1.21
	TR	2.62	1.78	2.41	1.69	2.45	1.71	2.33	1.70	2.58	1.75	2.48	1.73
	LogitSpec	2.54	1.85	1.83	1.37	1.96	1.48	1.59	1.24	2.09	1.52	2.00	1.49
	RACER	2.89	2.09	2.30	1.72	2.40	1.80	2.03	1.58	2.58	1.89	2.44	1.82
	RECAP	**4.11**	**2.33**	**3.24**	**1.94**	**3.33**	**1.96**	**2.79**	**1.75**	**3.70**	**2.17**	**3.43**	**2.03**

4 EXPERIMENTS

4.1 EXPERIMENTAL SETUP

Benchmark Datasets. To train the prefix-conditioned scorer, we incorporate 201 prompts drawn from 13 LongBench subsets (Bai et al., 2024), thereby exposing the scorer to heterogeneous training contexts and reducing reliance on domain-specific patterns. At test time, we evaluate **RECAP** on five complementary benchmarks: Spec-Bench (Xia et al., 2024), whose 80 multi-turn conversations come from MT-Bench (Zheng et al., 2023), for standard speculative decoding, LiveBench (White et al., 2025) for broad capability evaluation, LongBench-Write (Bai et al., 2025) for long-form writing, the 80-question Alpaca subset used by EAGLE (Taori et al., 2023; Li et al., 2024a) for short-form instructions, and WildBench (Lin et al., 2025) for real-world user queries. Beyond these benchmarks, we further use SWE-bench Verified (Jimenez et al., 2024) for persistent agent sessions. Additionally, detailed information about datasets is provided in Appendix B.2.

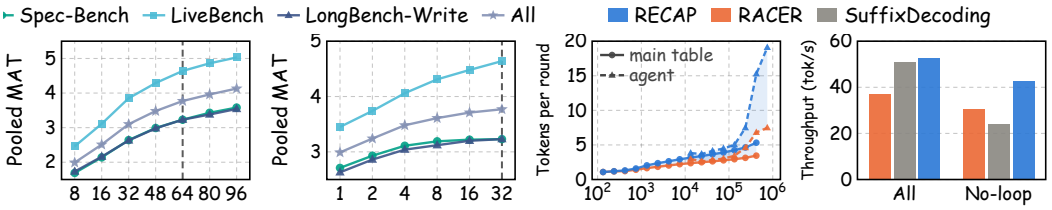
Evaluation Metrics. Following prior works (Leviathan et al., 2023; Li et al., 2024a;b), we report the mean number of tokens accepted per verification round (MAT) and the speedup in pooled throughput over autoregressive decoding to evaluate the performance of **RECAP**. To ensure comparable timing measurements, we run all methods within the same service epoch and on a single GPU. More details of the complete timing and measurement protocol can be found in Appendix B.3.

Compared Baselines. We compare the proposed **RECAP** against various baselines, i.e., autoregressive decoding (AR), PLD (Saxena, 2023), which retrieves continuations from the current context, and methods that reuse target-model predictions, including Token Recycling (TR) (Luo et al., 2025), LogitSpec (Liu et al., 2026c), and RACER (Zhang et al., 2026). Since EAGLE-3 (Li et al., 2025) has no public drafter for Mistral-7B, we compare with it separately on LiveBench for the targets whose drafters are public. All methods are verified by the same frozen target, so their outputs match AR up to FP16 near-ties. Appendix B.4 details the baselines, while Appendices C.2 and C.8 report the comparison with EAGLE-3 and the checks of output identity against AR.

Implementation Details. We adopt Qwen3-8B (Yang et al., 2025), Llama-3.1-8B-Instruct (Grattafiori et al., 2024), and Mistral-7B-Instruct-v0.2 (Jiang et al., 2023) as frozen target models, and additionally Qwen3-32B for the agent experiments. For the multi-context verification memory, we set the number of records per token M to 32, the number of candidates per record K to 8, and the context length L to 16. For draft-tree orchestration, we set the node budget B to 64 and the retrieval reserve R to 8. By default, we use greedy decoding in FP16 with batch size one and benchmark-specific output caps, and unless stated otherwise, every result applies a repetition rule that cuts an output reaching its cap inside a sustained loop. Further implementation details are given in Appendices B.5 and B.6, and the repetition rule in Appendix B.3.

4.2 PERFORMANCE COMPARISON

To demonstrate the effectiveness of **RECAP**, we compare it with various baselines across three target models on five benchmarks, and report MAT and speedup in Table 1. According to the results, we make the following observations: **Obs. ❶** ***RECAP*** *achieves the highest MAT and speedup in every*



(a) Node budget B (b) History capacity M (c) Evidence Φ (d) Agent sessions

Figure 3: Sensitivity Analysis. **Left (a,b):** MAT under different node budgets B and history capacities M on Qwen3-8B. **Middle (c):** Mean accepted tokens per verification round versus accumulated evidence Φ on the main benchmarks and agent sessions. **Right (d):** Pooled throughput of persistent agent sessions with Qwen3-32B on SWE-bench Verified, on all and on no-loop tasks.

setting, outperforming all baselines. Among them, TR benefits from a larger node budget, yet **RECAP** still outperforms it. **Obs. ❷** *The gain in speedup, however, is smaller than the gain in MAT.* This gap comes from our implementation: with the same verification cost, the drafting of **RECAP** runs many small dependent GPU kernels instead of a few large matrix products, which makes each round about a quarter longer than in RACER, so speedup also includes this implementation overhead. **Obs. ❸** *Measured by MAT, the gain over RACER grows with the length of the output.* Table 1 shows that for every target, the MAT gain is higher on LiveBench, LongBench-Write and WildBench than on Spec-Bench and Alpaca, whose outputs are capped at a lower length. This trend is consistent with the evidence mismatch: since RACER keeps only the most recent prediction of each token, whereas **RECAP** keeps several prefix-tagged records and selects among them, the advantage of **RECAP** grows as longer outputs let the same token recur under more distinct prefixes, which we examine further below with history that keeps accumulating across successive requests.

4.3 Ablation Study

To investigate the contribution of each component, we construct four variants of **RECAP**. V1 reduces the method to fixed-tree drafting using only the most recent record per token and no learned scorer, while V2 retains the scorer and multi-record memory but replaces best-first orchestration with a fixed tree. Additionally, V3 removes the retrieval reserve by setting $R = 0$, and V4 keeps only one record per token by setting $M = 1$. Concretely, we evaluate all variants with Qwen3-8B on three benchmarks, with results reported in Table 2. These results lead to the following insights: **Obs. ❹** *Each component contributes to the overall gain.* Removing any component consistently reduces MAT, with the only exception that V3 ties **RECAP** on Spec-Bench. Controlled comparisons on identical text further show that replacing the learned scorer with heuristic aggregation nearly eliminates the improvement over V1, indicating that the gain comes from learning which historical records transfer to the current prefix rather than merely enlarging the candidate pool. **Obs. ❺** *Multi-context memory plays the most important role for* ***RECAP****.* V4 causes the largest degradation among the component-wise ablations and even underperforms V2, which retains multiple records despite using fixed-tree drafting. Appendix C.3 further provides the heuristic-scorer variants and considerably more detailed per-benchmark comparisons.

Table 2: Ablation study of **RECAP** on Qwen3-8B. TPS denotes decoding throughput in tokens per second.

Variant	Spec-Bench		LiveBench		LongBench-Write		Avg	
	MAT	TPS	MAT	TPS	MAT	TPS	MAT	TPS
V1	2.45	102.9	3.27	132.4	2.54	99.2	2.75	111.5
V2	2.83	108.1	3.76	138.3	2.97	103.6	3.19	116.7
V3	3.28	**115.3**	4.38	142.2	3.46	113.9	3.71	123.8
V4	2.71	97.0	3.37	113.1	2.79	95.6	2.96	101.9
RECAP	**3.28**	113.9	**4.54**	**145.7**	**3.47**	**114.0**	**3.76**	**124.5**

4.4 Further Analysis

Sensitivity Analysis. We then vary the key hyperparameters of **RECAP**, namely the node budget B of the draft tree and the history capacity M. The results on Qwen3-8B are shown in Figure 3, from which we observe that: **Obs. ❻** ***RECAP*** *benefits from both a larger budget and a larger history.* On the one hand, MAT rises steadily with B on every corpus, and the gain over RACER widens with it, since best-first expansion always gives each added node to the most reliable open path. The reason why we still keep $B = 64$, although a larger budget would raise MAT further, is that this value matches the official setting of RACER and thus keeps the comparison fair. On the other hand, keeping more

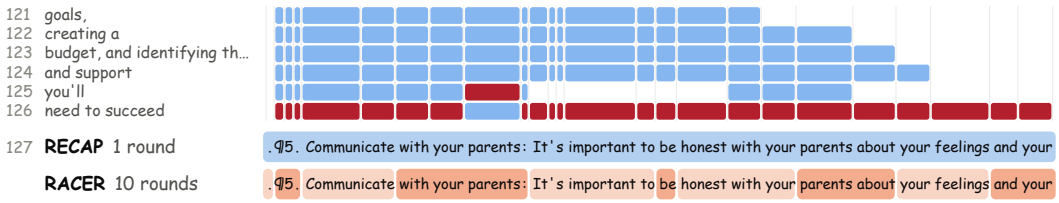


Figure 4: Case study of RECAP on WildBench. The bottom rows compare the passage committed by RECAP in round 127 and by RACER across ten rounds, while the upper rows show the six preceding rounds of RECAP. Light blue marks tokens previously appearing on rejected look-ahead branches, and red marks the highest-ranked historical record for each drafted token.

records per token also helps, but the returns shrink as M grows, and a candidate-availability oracle indicates that capacities beyond 32 would add little, so we fix M at 32. Appendix C.4 reports the complete sweeps for every corpus, together with the retrieval reserve R.

History Accumulation. Since every main-table conversation starts from an empty memory and lasts one request, or two turns for MT-Bench (Zheng et al., 2023), we further evaluate RECAP in a setting where the history keeps growing. Specifically, Qwen3-32B drives a software-engineering agent on SWE-bench Verified (Jimenez et al., 2024) with contexts up to 32K tokens, while all decoding state persists across calls. We additionally compare with SuffixDecoding (Oliaro et al., 2025), which is designed for agentic workloads. As shown in Figure 3(c), we conclude that **Obs. ❼** *The advantage of RECAP grows with accumulated history.* Both RECAP and RACER benefit from increasing evidence, but RECAP improves faster, yielding a widening gap in MAT and throughput. **Obs. ❽** *RECAP achieves the highest throughput in persistent sessions.* Figure 3(d) shows that RECAP consistently outperforms SuffixDecoding, with the margin widening substantially on non-looping tasks. Additional details of these sessions are provided in Appendix C.6.

Scorer Specialization. Since the scorer is the only learned module of RECAP, it enables lightweight domain specialization without modifying the target model. Therefore, we retrain the scorer on coding prompts with DeepSeek-Coder-6.7B and evaluate it on the coding, math, and reasoning subsets of LiveBench. We denote the coding-specialized variant as RECAP-C and retain RECAP for the original scorer. Table 3 reveals that **Obs. ❾** *Scorer specialization improves in-domain performance with limited degradation elsewhere.* On coding, MAT increases from 3.47 to 4.09, while a throughput deficit relative to RACER turns into a 14% advantage. These results highlight the potential of RECAP to generalize across broader application settings. Further detailed results are provided in Appendix C.7.

Table 3: Scorer specialization for RECAP with DeepSeek-Coder-6.7B-Instruct. RECAP employs the original scorer, whereas RECAP-C incorporates a new scorer retrained on coding prompts.

Method	Coding		Math		Reasoning		Avg	
	MAT	TPS	MAT	TPS	MAT	TPS	MAT	TPS
RACER	2.72	137.8	2.95	161.3	3.11	168.1	2.93	155.7
RECAP	3.47	121.8	3.75	146.3	**4.57**	**170.3**	3.93	146.1
RECAP-C	**4.09**	**157.4**	**4.08**	**165.3**	4.36	166.4	**4.18**	**163.0**

Case Study. We finally examine the round-level mechanism of RECAP through a WildBench case with Mistral-7B. As shown in Figure 4, RECAP commits a 25-token passage in a single verification round, whereas RACER requires ten. The key is that RECAP retains multiple records per token, rejected branches included: before reaching this passage, earlier rounds have already covered most of its tokens, and the scorer selects these records when drafting round 127. In contrast, only two drafted tokens have matching successors in the committed text RACER retrieves from. Appendix C.5 provides the full trace of round 127 and further examples from other questions.

5 Conclusion

We address a key challenge of history-based speculative decoding: it reuses stored predictions by token alone, ignoring that their reliability depends on the prefix that produced them. To this end, we propose RECAP, which calibrates how well prefix-tagged records transfer to the current branch and grows the draft tree best-first by cumulative reliability. Experiments on three target models and five benchmarks show that RECAP achieves the highest MAT and speedup in every setting, with ablations attributing the gain to the learned scorer and larger gains in persistent agent sessions.

AI USE STATEMENT

We used large language model (LLM) assistants in three ways during this work. First, for writing and literature search, LLMs helped translate the authors' drafts into English, edit wording and grammar, format tables and LaTeX, and search for related work, while all technical content and claims were written or checked by the authors and every citation was verified by the authors. Second, for engineering, LLM coding assistants helped implement and debug the serving, evaluation and analysis code, and helped launch and monitor experiment runs under the authors' supervision. Every reported number was traced back to the logged raw outputs and verified by the authors. Third, LLMs produced simulated reviews of our drafts, which we used only to find unclear passages and missing analyses. The research questions, the method and the interpretation of the results are the authors' own, and the authors take full responsibility for the entire content of this paper.

ETHICS STATEMENT

We have read and adhere to the ICLR Code of Ethics. This work accelerates the inference of existing language models without changing what they generate, except where floating-point rounding flips a near-tie between tokens, so it adds no capability to the target model and inherits its behavior and risks. The study involves no human subjects and collects no new data. All models and benchmarks are publicly available and are used under their licenses.

REPRODUCIBILITY STATEMENT

Section 4 and Appendix B describe the benchmarks, metrics, baselines, hyperparameters, hardware, timing protocol and repetition rule, and Appendix A gives complete proofs of the theoretical results. The anonymous repository linked in the abstract contains the implementation of `RECAP` and of the compared methods on a shared serving stack, the scorer-training pipeline, the evaluation drivers and baseline arms, and the repetition rule, together with the pooled measurements behind Tables 1 and 25 and a script that regenerates both tables from them. The benchmark question lists, the 201 training prompts, the drivers for Spec-Bench, Alpaca and WildBench, and a consolidated launcher for the serving stack will be added to the repository.

REFERENCES

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

Yushi Bai, Xin Lv, Jiajie Zhang, Hongchang Lyu, Jiankai Tang, Zhidian Huang, Zhengxiao Du, Xiao Liu, Aohan Zeng, Lei Hou, Yuxiao Dong, Jie Tang, and Juanzi Li. LongBench: A bilingual, multitask benchmark for long context understanding. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 3119–3137, Bangkok, Thailand, 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.acl-long.172. URL https://aclanthology.org/2024.acl-long.172/.

Yushi Bai, Jiajie Zhang, Xin Lv, Linzhi Zheng, Siqi Zhu, Lei Hou, Yuxiao Dong, Jie Tang, and Juanzi Li. LongWriter: Unleashing 10,000+ word generation from long context LLMs. In *International Conference on Learning Representations (ICLR)*, 2025.

Anooshka Bajaj, Deven Mahesh Mistry, Sahaj Singh Maini, Yash Aggarwal, and Zoran Tiganj. Beyond semantics: How temporal biases shapes retrieval in transformer and state-space models. In *Proceedings of the 19th Conference of the European Chapter of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 7562–7581, 2026.

Alina Beygelzimer, John Langford, Yury Lifshits, Gregory B. Sorkin, and Alexander L. Strehl. Conditional probability tree estimation analysis and algorithms. In *Proceedings of the Twenty-Fifth Conference on Uncertainty in Artificial Intelligence (UAI)*, pp. 51–58, 2009. URL https://arxiv.org/abs/0903.4217.

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

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

Guanzheng Chen, Qilong Feng, Jinjie Ni, Xin Li, and Michael Qizhe Shieh. RAPID: Long-context inference with retrieval-augmented speculative decoding. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 8093–8107. PMLR, 2025a. URL https://proceedings.mlr.press/v267/chen25s.html.

Jinglin Chen, Qiwei Li, Zuchao Li, Baoyuan Qi, Liu Guoming, Haojun Ai, Hai Zhao, and Ping Wang. Faster in-context learning for llms via n-gram trie speculative decoding. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 18040–18051, 2025b.

Zhuoming Chen, Avner May, Ruslan Svirschevski, Yuhsun Huang, Max Ryabinin, Zhihao Jia, and Beidi Chen. Sequoia: Scalable and robust speculative decoding. In *Advances in Neural Information Processing Systems 37*, 2024. doi: 10.52202/079017-4116. URL https://proceedings.neurips.cc/paper_files/paper/2024/hash/ea1f5f0878d43ff4fb8bf64ef4a2326c-Abstract-Conference.html.

Kyunghyun Cho, Bart van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using RNN encoder–decoder for statistical machine translation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 1724–1734, Doha, Qatar, 2014. Association for Computational Linguistics. doi: 10.3115/v1/D14-1179. URL https://aclanthology.org/D14-1179/.

Truong Dinh Do, Nguyen-Khang Le, and Minh Le Nguyen. Unispec: Training-free speculative decoding for robust llm acceleration across languages and hardware. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 6288–6310, 2026.

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. URL https://proceedings.mlr.press/v235/du24c.html.

Weihua Du, Hailei Gong, Zhan Ling, Kang Liu, Lingfeng Shen, Xuesong Yao, Yufei Xu, Dingyuan Shi, Yiming Yang, and Jiecao Chen. Generalizable end-to-end tool-use RL with synthetic CodeGym. In *International Conference on Learning Representations (ICLR)*, 2026. URL https://openreview.net/forum?id=QRSeFZfu8E.

Yichao Fu, Peter Bailis, Ion Stoica, and Hao Zhang. Break the sequential dependency of LLM inference using lookahead decoding. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 14060–14079. PMLR, 2024. URL https://proceedings.mlr.press/v235/fu24a.html.

Yichao Fu, Rui Ge, Zelei Shao, Zhijie Deng, and Hao Zhang. Scaling speculative decoding with lookahead reasoning. *Advances in Neural Information Processing Systems*, 38:172454–172486, 2026.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, et al. The Llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024. URL https://arxiv.org/abs/2407.21783.

Daya Guo, Qihao Zhu, Dejian Yang, Zhenda Xie, Kai Dong, Wentao Zhang, Guanting Chen, Xiao Bi, Y. Wu, Y. K. Li, Fuli Luo, Yingfei Xiong, and Wenfeng Liang. DeepSeek-Coder: When the large language model meets programming – the rise of code intelligence. *arXiv preprint arXiv:2401.14196*, 2024. URL https://arxiv.org/abs/2401.14196.

Yoav Gur-Arieh, Mor Geva, and Atticus Geiger. Mixing mechanisms: How language models retrieve bound entities in-context. In *International Conference on Learning Representations (ICLR)*, 2026.

Zhenyu He, Zexuan Zhong, Tianle Cai, Jason D. Lee, and Di He. REST: Retrieval-based speculative decoding. In *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pp. 1582–1595. Association for Computational Linguistics, 2024. URL https://aclanthology.org/2024.naacl-long.88/.

Ari Holtzman, Jan Buys, Li Du, Maxwell Forbes, and Yejin Choi. The curious case of neural text degeneration. In *International Conference on Learning Representations (ICLR)*, 2020. URL https://openreview.net/forum?id=rygGQyrFvH.

Yunlong Hou, Fengzhuo Zhang, Cunxiao Du, Xuan Zhang, Jiachun Pan, Tianyu Pang, Chao Du, Vincent YF Tan, and Zhuoran Yang. Banditspec: Adaptive speculative decoding via bandit algorithms. *arXiv preprint arXiv:2505.15141*, 2025.

Yuxuan Hu, Ke Wang, Xiaokang Zhang, Fanjin Zhang, Cuiping Li, Hong Chen, and Jing Zhang. SAM decoding: Speculative decoding via suffix automaton. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 12187–12204, Vienna, Austria, 2025. Association for Computational Linguistics. doi: 10.18653/v1/2025.acl-long.595. URL https://aclanthology.org/2025.acl-long.595/.

Haiduo Huang, Fuwei Yang, Zhenhua Liu, and Pengju Ren. Jakiro: Boosting speculative decoding via decoupled moe. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 10649–10668, 2026.

Feiye Huo, Jianchao Tan, Kefeng Zhang, Xunliang Cai, and Shengli Sun. C2T: A classifier-based tree construction method in speculative decoding. *arXiv preprint arXiv:2502.13652*, 2025. doi: 10.48550/arXiv.2502.13652. URL https://arxiv.org/abs/2502.13652.

Albert Q. Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, Lélio Renard Lavaud, Marie-Anne Lachaux, Pierre Stock, Teven Le Scao, Thibaut Lavril, Thomas Wang, Timothée Lacroix, and William El Sayed. Mistral 7B. *arXiv preprint arXiv:2310.06825*, 2023. URL https://arxiv.org/abs/2310.06825.

Carlos E. Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik Narasimhan. SWE-bench: Can language models resolve real-world GitHub issues? In *International Conference on Learning Representations (ICLR)*, 2024. URL https://arxiv.org/abs/2310.06770.

Tao Jin, Phuong Minh Nguyen, and Naoya Inoue. Goose: Anisotropic speculation trees for training-free speculative decoding. *arXiv preprint arXiv:2604.02047*, 2026. URL https://arxiv.org/abs/2604.02047.

Anant Khandelwal, Manish Gupta, and Puneet Agrawal. CoCoA: Confidence- and context-aware adaptive decoding for resolving knowledge conflicts in large language models. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 6835–6855, 2025. URL https://aclanthology.org/2025.emnlp-main.348/.

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

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

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

Yuhui Li, Fangyun Wei, Chao Zhang, and Hongyang Zhang. EAGLE-3: Scaling up inference acceleration of large language models via training-time test. In *Advances in Neural Information Processing Systems 38*, 2025. doi: 10.52202/085713-4562. URL https://proceedings.nips.cc/paper_files/paper/2025/hash/c7b5a35ea98b62512a869c19ea7b03cb-Abstract-Conference.html.

Bill Yuchen Lin, Yuntian Deng, Khyathi Chandu, Faeze Brahman, Abhilasha Ravichander, Valentina Pyatkin, Nouha Dziri, Ronan Le Bras, and Yejin Choi. WildBench: Benchmarking LLMs with challenging tasks from real users in the wild. In *International Conference on Learning Representations (ICLR)*, 2025.

Fangcheng Liu, Yehui Tang, Zhenhua Liu, Yunsheng Ni, Duyu Tang, Kai Han, and Yunhe Wang. Kangaroo: Lossless self-speculative decoding for accelerating LLMs via double early exiting. In *Advances in Neural Information Processing Systems 37*, 2024. URL https://proceedings.neurips.cc/paper_files/paper/2024/hash/16336d94a5ffca8de019087ab7fe403f-Abstract-Conference.html.

Runheng Liu, Jincheng Xie, Wen Hu, Xingchen Xiao, and Heyan Huang. AdaPLD: Adaptive retrieval and reuse for efficient model-free speculative decoding. *arXiv preprint arXiv:2606.05742*, 2026a. doi: 10.48550/arXiv.2606.05742. URL https://arxiv.org/abs/2606.05742.

Siran Liu, Yang Ye, Qianchao Zhu, Zane Cao, and Yongchao He. Heterospec: Leveraging contextual heterogeneity for efficient speculative decoding. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 12930–12947, 2026b.

Tianyu Liu, Qitan Lv, Hao Li, Xing Gao, Xiao Sun, and Xiaoyan Sun. LogitSpec: Accelerating retrieval-based speculative decoding via next next token speculation. In *Findings of the Association for Computational Linguistics: ACL 2026*, pp. 33070–33092, San Diego, California, United States, 2026c. Association for Computational Linguistics. doi: 10.18653/v1/2026.findings-acl.1655. URL https://aclanthology.org/2026.findings-acl.1655/.

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 *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 6816–6831, Vienna, Austria, 2025. Association for Computational Linguistics. doi: 10.18653/v1/2025.acl-long.338. URL https://aclanthology.org/2025.acl-long.338/.

Zhiyao Ma, In Gim, and Lin Zhong. Cacheback: Speculative decoding with nothing but cache. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 31079–31084, Suzhou, China, 2025. Association for Computational Linguistics. doi: 10.18653/v1/2025.emnlp-main.1581. URL https://aclanthology.org/2025.emnlp-main.1581/.

Ziyang Ma, Zihong Zhang, Zuchao Li, Lefei Zhang, Baoyuan Qi, Siqi Li, and Simin Yu. ECHO: Early-layer collaborative hierarchical orchestration with bonus logits in speculative decoding. *arXiv preprint arXiv:2609.17241*, 2026. URL https://arxiv.org/abs/2609.17241.

Xupeng Miao, Gabriele Oliaro, Zhihao Zhang, Xinhao Cheng, Zeyu Wang, Zhengxin Zhang, Rae Ying Yee Wong, Alan Zhu, Lijie Yang, Xiaoxiang Shi, Chunan Shi, Zhuoming Chen, Daiyaan Arfeen, Reyna Abhyankar, and Zhihao Jia. Specinfer: Accelerating large language model serving

with tree-based speculative inference and verification. In *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 3*, pp. 932–949. ACM, 2024. doi: 10.1145/3620666.3651335.

Byung-Doh Oh and William Schuler. Token-wise decomposition of autoregressive language model hidden states for analyzing model predictions. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 10105–10117, 2023.

Gabriele Oliaro, Zhihao Jia, Daniel Campos, and Aurick Qiao. SuffixDecoding: Extreme speculative decoding for emerging AI applications. In *Advances in Neural Information Processing Systems 38*, 2025.

Lalchand Pandia and Allyson Ettinger. Sorting through the noise: Testing robustness of information processing in pre-trained language models. In *Proceedings of the 2021 conference on empirical methods in natural language processing*, pp. 1583–1596, 2021.

Dezhi Ran, Yuan Cao, Mengzhou Wu, Simin Chen, Yuzhe Guo, Jun Ren, Zihe Song, Hao Yu, Jialei Wei, Linyi Li, et al. From assistant to independent developer—are GPTs ready for software development? In *International Conference on Learning Representations (ICLR)*, 2026. URL https://openreview.net/forum?id=XrP8dp1rCg.

Apoorv Saxena. Prompt lookup decoding, November 2023. URL https://github.com/apoorvumang/prompt-lookup-decoding/.

Charlie Snell, Jaehoon Lee, Kelvin Xu, and Aviral Kumar. Scaling LLM test-time compute optimally can be more effective than scaling parameters for reasoning. In *International Conference on Learning Representations (ICLR)*, 2025.

Ruslan Svirschevski, Avner May, Zhuoming Chen, Beidi Chen, Zhihao Jia, and Max Ryabinin. SpecExec: Massively parallel speculative decoding for interactive LLM inference on consumer devices. In *Advances in Neural Information Processing Systems 37*, 2024. URL https://proceedings.neurips.cc/paper_files/paper/2024/hash/1d91d5689e251d27993a3c2182dddcf7-Abstract-Conference.html.

Rohan Taori, Ishaan Gulrajani, Tianyi Zhang, Yann Dubois, Xuechen Li, Carlos Guestrin, Percy Liang, and Tatsunori B. Hashimoto. Stanford alpaca: An instruction-following LLaMA model. https://github.com/tatsu-lab/stanford_alpaca, 2023.

Jikai Wang, Yi Su, Juntao Li, Qingrong Xia, Zi Ye, Xinyu Duan, Zhefeng Wang, and Min Zhang. OPT-tree: Speculative decoding with adaptive draft tree structure. *Transactions of the Association for Computational Linguistics*, 13:188–199, 2025a. doi: 10.1162/tacl_a_00735. URL https://aclanthology.org/2025.tacl-1.8/.

Ziqi Wang, Hanlin Zhang, Xiner Li, Kuan-Hao Huang, Chi Han, Shuiwang Ji, Sham Kakade, Hao Peng, and Heng Ji. Eliminating position bias of language models: A mechanistic approach. In *International Conference on Learning Representations (ICLR)*, 2025b.

Sean Welleck, Ilia Kulikov, Stephen Roller, Emily Dinan, Kyunghyun Cho, and Jason Weston. Neural text generation with unlikelihood training. In *International Conference on Learning Representations (ICLR)*, 2020. URL https://openreview.net/forum?id=SJeYe0NtvH.

Colin White, Samuel Dooley, Manley Roberts, Arka Pal, Ben Feuer, Siddhartha Jain, Ravid Shwartz-Ziv, Neel Jain, Khalid Saifullah, Sreemanti Dey, Shubh Agrawal, Sandeep Singh Sandha, Siddartha Naidu, Chinmay Hegde, Yann LeCun, Tom Goldstein, Willie Neiswanger, and Micah Goldblum. LiveBench: A challenging, contamination-limited LLM benchmark. In *International Conference on Learning Representations (ICLR)*, 2025.

Zhaoxuan Wu, Zijian Zhou, Arun Verma, Alok Prakash, Daniela Rus, and Bryan Kian Hsiang Low. Tetris: Optimal draft token selection for batch speculative decoding. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 33329–33345, 2025.

Heming Xia, Zhe Yang, Qingxiu Dong, Peiyi Wang, Yongqi Li, Tao Ge, Tianyu Liu, Wenjie Li, and Zhifang Sui. Unlocking efficiency in large language model inference: A comprehensive survey of speculative decoding. In *Findings of the Association for Computational Linguistics: ACL 2024*, pp. 7655–7671, 2024.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, Chujie Zheng, Dayiheng Liu, Fan Zhou, Fei Huang, Feng Hu, Hao Ge, Haoran Wei, Huan Lin, Jialong Tang, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jing Zhou, Jingren Zhou, Junyang Lin, Kai Dang, Keqin Bao, Kexin Yang, Le Yu, Lianghao Deng, Mei Li, Mingfeng Xue, Mingze Li, Pei Zhang, Peng Wang, Qin Zhu, Rui Men, Ruize Gao, Shixuan Liu, Shuang Luo, Tianhao Li, Tianyi Tang, Wenbiao Yin, Xingzhang Ren, Xinyu Wang, Xinyu Zhang, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yinger Zhang, Yu Wan, Yuqiong Liu, Zekun Wang, Zeyu Cui, Zhenru Zhang, Zhipeng Zhou, and Zihan Qiu. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025. URL https://arxiv.org/abs/2505.09388.

John Yang, Carlos E. Jimenez, Alexander Wettig, Kilian Lieret, Shunyu Yao, Karthik Narasimhan, and Ofir Press. SWE-agent: Agent-computer interfaces enable automated software engineering. In *Advances in Neural Information Processing Systems 37*, 2024. URL https://proceedings.neurips.cc/paper_files/paper/2024/hash/5a7c947568c1b1328ccc5230172e1e7c-Abstract-Conference.html.

Nan Yang, Tao Ge, Liang Wang, Binxing Jiao, Daxin Jiang, Linjun Yang, Rangan Majumder, and Furu Wei. Inference with reference: Lossless acceleration of large language models. *arXiv preprint arXiv:2304.04487*, 2023. URL https://arxiv.org/abs/2304.04487.

Jun Zhang, Jue Wang, Huan Li, Lidan Shou, Ke Chen, Gang Chen, and Sharad Mehrotra. Draft & verify: Lossless large language model acceleration via self-speculative decoding. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 11263–11282, Bangkok, Thailand, 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.acl-long.607. URL https://aclanthology.org/2024.acl-long.607/.

Zihong Zhang, Zuchao Li, Lefei Zhang, Ping Wang, and Hai Zhao. RACER: Retrieval-augmented contextual rapid speculative decoding. In *Findings of the Association for Computational Linguistics: ACL 2026*, pp. 19962–19988, San Diego, California, United States, 2026. Association for Computational Linguistics. doi: 10.18653/v1/2026.findings-acl.998. URL https://aclanthology.org/2026.findings-acl.998/.

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 36 (Datasets and Benchmarks Track)*, 2023. URL https://proceedings.neurips.cc/paper_files/paper/2023/hash/91f18a1287b398d378ef22505bf41832-Abstract-Datasets_and_Benchmarks.html.

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

Appendix

A Proofs for the Theoretical Analysis **17**
A.1 Orchestration Model . . . 17
A.2 The Elementary Bound . . . 17
A.3 The Normalization-Aware Bound . . . 17
A.4 Scope of the Optimization Assumption . . . 18

B Additional Experimental Setup **19**
B.1 Decoding Procedure . . . 19
B.2 Benchmark Datasets . . . 19
B.3 Evaluation Metrics . . . 20
B.4 Baselines . . . 20
B.5 Scorer Architecture and Training . . . 21
B.6 Tree Orchestration and Serving . . . 23

C Additional Experimental Results **25**
C.1 Main-Table Details . . . 25
C.2 Comparison with EAGLE-3 and Concurrent Methods . . . 25
C.3 Ablation Details . . . 28
C.4 Sensitivity Details . . . 29
C.5 Mechanism and Case Studies . . . 32
C.6 Persistent Agent Sessions and Accumulated Evidence . . . 32
C.7 Scorer Specialization . . . 37
C.8 Output Quality . . . 38

D Additional Related Work **38**

A Proofs for the Theoretical Analysis

A.1 Orchestration Model

All probabilities below are conditional on a common information state $\mathcal{I}$, which fixes the candidate tree, the retrieval constraint and the scorer outputs of the round. The root is guaranteed, i.e., $\Pr(A_r \mid \mathcal{I}) = 1$, and distinct children of a node correspond to distinct next tokens. For a parent event of positive probability, we define $p_{uv} = \Pr(A_v \mid A_u, \mathcal{I})$; if the parent event has probability zero, p_{uv} may be chosen arbitrarily subject to subnormalization, which leaves every path probability unchanged. By the chain rule,

$$w_p(v) := \Pr(A_v \mid \mathcal{I}) = \prod_{e \in \mathcal{P}(v)} p_e, \qquad w_q(v) := \prod_{e \in \mathcal{P}(v)} q_e. \tag{14}$$

No independence assumption on the acceptance events is required. These probabilities refer to verifier-consistent continuations under the information available to the orchestrator rather than to sampling from the target distribution: conditioned on the complete target state, greedy verification is deterministic, and nondegenerate probabilities arise because the orchestrator observes only a summary of that state. Before output clipping, a prefix-closed tree commits $L(\mathcal{T}) = 1 + \sum_{v \in \mathcal{T} \setminus \{r\}} \mathbf{1}_{A_v}$ tokens, and taking conditional expectations yields Eq. 12. The analysis concerns a single round with a fixed candidate tree and does not account for future evidence, scoring cost or verification latency.

A.2 The Elementary Bound

Lemma 3.1 states that, if $|p_e - q_e| \le \varepsilon$ holds for every candidate edge, then $0 \le J(\mathcal{T}^\star) - J(\widehat{\mathcal{T}}) \le 2\varepsilon n D$.

Proof of Lemma 3.1. For a path of depth d, a telescoping expansion gives

$$\prod_{j=1}^{d} p_j - \prod_{j=1}^{d} q_j = \sum_{j=1}^{d} \Big(\prod_{i<j} p_i\Big)(p_j - q_j)\Big(\prod_{i>j} q_i\Big).$$

Since all factors lie in $[0,1]$, it follows that $|w_p(v) - w_q(v)| \le \sum_{e \in \mathcal{P}(v)} |p_e - q_e| \le \varepsilon d(v)$, where $d(v) = |\mathcal{P}(v)|$. This inequality is Lemma 2 of Beygelzimer et al. (2009) for conditional probability trees, whose tighter form retains the factor $\prod_{j \ne i} \max\{p_j, q_j\}$ in each term. Let $\Delta = \mathcal{T}^\star \triangle \widehat{\mathcal{T}}$. By the optimality of $\widehat{\mathcal{T}}$ for the surrogate objective, $\widehat{J}(\mathcal{T}^\star) - \widehat{J}(\widehat{\mathcal{T}}) \le 0$. Cancelling the nodes shared by the two trees gives

$$\begin{aligned} J(\mathcal{T}^\star) - J(\widehat{\mathcal{T}}) &= \widehat{J}(\mathcal{T}^\star) - \widehat{J}(\widehat{\mathcal{T}}) + \sum_{v \in \mathcal{T}^\star \setminus \widehat{\mathcal{T}}} \big[w_p(v) - w_q(v)\big] - \sum_{v \in \widehat{\mathcal{T}} \setminus \mathcal{T}^\star} \big[w_p(v) - w_q(v)\big] \\ &\le \sum_{v \in \Delta} |w_p(v) - w_q(v)| \le \varepsilon \sum_{v \in \Delta} d(v). \end{aligned}$$

Since both trees contain $\mathcal{T}_R$, we have $|\Delta| \le 2n$, and $d(v) \le D$ yields the upper bound $2\varepsilon n D$. Nonnegativity follows from the optimality of $\mathcal{T}^\star$ for J. □

A.3 The Normalization-Aware Bound

Let $m = |\Delta|$. For a root-to-leaf path π in $\mathcal{T}^\star \cup \widehat{\mathcal{T}}$, regarded as a set of vertices, define

$$K_\Delta = \max_{\pi} \sum_{v \in \Delta \cap \pi} \big(d(v) - 1\big), \qquad h = \max_{\pi} |\Delta \cap \pi|, \tag{15}$$

both of which are zero if Δ is empty. We prove the structure-dependent inequality

$$J(\mathcal{T}^\star) - J(\widehat{\mathcal{T}}) \le \varepsilon\,(m + K_\Delta) \le \varepsilon\Big[m + \tfrac{h(2D-h-1)}{2}\Big], \tag{16}$$

whose first inequality is Theorem 3.2 and from which Theorem 3.1 follows by $m \le 2n$ and $h \le D$.

Proof of Theorems 3.1 and 3.2. For $v \in \Delta$, let $\sigma_v = 1$ if $v \in \mathcal{T}^\star \setminus \widehat{\mathcal{T}}$ and $\sigma_v = -1$ otherwise, and define the path polynomial $F(x) = \sum_{v\in\Delta} \sigma_v \prod_{e\in\mathcal{P}(v)} x_e$. Then $F(p) = J(\mathcal{T}^\star) - J(\widehat{\mathcal{T}})$ and $F(q) \le 0$. Since both endpoints lie in the product of the outgoing probability simplices, so does the segment $x(t) = (1-t)q + tp$, and the fundamental theorem of calculus gives

$$J(\mathcal{T}^\star) - J(\widehat{\mathcal{T}}) \le F(p) - F(q) = \int_0^1 \big\langle p - q, \nabla F(x(t)) \big\rangle \, dt \le \varepsilon \sup_{t\in[0,1]} \|\nabla F(x(t))\|_1. \tag{17}$$

It remains to bound the gradient. For each node u, independently draw a vector Z_u that assigns one to the child edge (u, v) with probability x_{uv} and zero to all other child edges, and assigns zero to all child edges with the remaining probability $1 - \sum_v x_{uv}$. Then $\mathbb{E}Z_u = x_u$, and every realization has at most one edge of value one below each node. Since every path uses at most one edge below each node, each coordinate of ∇F is affine in each block x_u, and the independence of the blocks implies $\nabla F(x) = \mathbb{E}[\nabla F(Z)]$. Consequently,

$$\|\nabla F(x)\|_1 \le \mathbb{E}\|\nabla F(Z)\|_1 \le \max_z \|\nabla F(z)\|_1, \tag{18}$$

where the maximum is taken over the deterministic configurations z. The auxiliary randomization serves only to bound the gradient and imposes no assumption on the acceptance events. In a configuration z, the edges of value one that are reachable from the root form a single path π_z. Let $g_v(x) = \prod_{e\in\mathcal{P}(v)} x_e$, so that $\partial g_v / \partial x_e(z) = \prod_{f\in\mathcal{P}(v)\setminus\{e\}} z_f$. Hence $\|\nabla g_v(z)\|_1$ equals $d(v)$ if all edges of $\mathcal{P}(v)$ have value one, equals one if exactly one edge has value zero, and vanishes otherwise. Therefore,

$$\|\nabla F(z)\|_1 \le \sum_{v\in\Delta} \|\nabla g_v(z)\|_1 \le m + \sum_{v\in\Delta\cap\pi_z} \big(d(v) - 1\big) \le m + K_\Delta,$$

where π_z is restricted to $\mathcal{T}^\star \cup \widehat{\mathcal{T}}$; if π_z terminates early, extending it within the union can only add nonnegative terms. Combined with Eqs. 17 and 18, this proves the first inequality of Eq. 16. Finally, a path contains at most h disagreement nodes at distinct depths between one and D, hence $K_\Delta \le (D-1) + \cdots + (D-h) = h(2D-h-1)/2 \le D(D-1)/2$, and together with $m \le 2n$ this yields Eq. 13. □

Remark (a budget-only form). By prefix closure, a path that leaves one of the two trees cannot re-enter it below the missing node, so all disagreement nodes on a root-to-leaf path belong to the same side of the symmetric difference. Therefore $h \le n$ in addition to $h \le D$, and with $s = \min\{n, D\}$, Eq. 16 implies

$$J(\mathcal{T}^\star) - J(\widehat{\mathcal{T}}) \le \varepsilon \Big[2n + \tfrac{s(2D-s-1)}{2} \Big]. \tag{19}$$

This form also covers the case $n < D$, vanishes for $n = 0$, and coincides with Eq. 13 when $D \le n$. Moreover, for every realized continuation, the committed lengths of the two trees differ by a signed count of disagreement nodes on a single path, whose magnitude is at most h; consequently, each of the above bounds can be capped at h, and in particular at s.

A.4 Scope of the Optimization Assumption

Theorems 3.1 and 3.2 assume that $\widehat{\mathcal{T}}$ maximizes $\widehat{J}$ over the feasible trees. Exact best-first selection on a fixed, fully scored candidate tree satisfies this assumption: starting from $\mathcal{T}_R$, it exposes the children of the root and of every node it selects and repeatedly adds a frontier node with the largest path score $w_q(v)$, which equals $\exp S(\pi)$ for the path π to v in Eq. 11. In the implementation (Appendix B.6), the candidate tree $\mathcal{C}$ is accordingly the union of $\mathcal{T}_R$ with the paths reachable from the root through the eight highest-scored successors of each exposed node, so that the optimality holds relative to this restricted candidate space. Since $w_q(v) \le w_q(\mathrm{pa}(v))$, the first unselected ancestor of any unexposed node lies on the frontier and has no smaller score; hence a maximum-score unselected node is always available on the frontier (up to ties), and adding such nodes until the budget or the candidates are exhausted maximizes the additive surrogate. Mandatory nodes contribute equally to every feasible tree, and overlapping paths are counted once. This argument requires the candidates to be available with fixed scores during the round and does not cover the root fallback or the fixed-tree fill described in Appendix B.6. If the orchestrator only guarantees $\widehat{J}(\widehat{\mathcal{T}}) \ge \max_{\mathcal{T}} \widehat{J}(\mathcal{T}) - \eta$, then $F(q) \le \eta$ replaces $F(q) \le 0$, and η is added to each error-based

Algorithm 1 `RECAP`: Reliability-Calibrated Draft-Tree Orchestration

Input: Frozen target model p_θ; prompt X; scorer $\widetilde{p}_\phi$; tree budget B; retrieval reserve R
Output: Generated sequence X

Prefill X; initialize the KV cache, the historical bank and the retrieval memory
while EOS is not reached **do**
    `// Reliability-calibrated historical evidence (Sec. 3.2)`
    Set the target-model Top-1 token as the tree root and freeze the historical-memory snapshot
    Retrieve exact continuations $\mathcal{T}_R$ with at most R non-root nodes `// high-precision anchor`
    `// Evidence-conditioned tree orchestration (Sec. 3.3)`
    Admit $\mathcal{T}_R$ into $\mathcal{T}$ and initialize the frontier $\mathcal{F}$ with the scored successors of the root
    while $|\mathcal{T}| < B$ and $\mathcal{F} \neq \emptyset$ **do**
        Pop the highest-scoring path $\pi^\star$ from $\mathcal{F}$ by Eq. 11 and admit it into $\mathcal{T}$ `// consumes budget`
        Score the successors of $\pi^\star$ on $\mathcal{B}_t(v_{\pi^\star})$ by Eq. 8 and push the top eight into $\mathcal{F}$ `// exposure only`
    end while `// best-first expansion under the shared budget`
    `// Target verification and memory update`
    Verify $\mathcal{T}$ with one target-model forward pass; commit the longest path that matches the target's greedy choices
    Store verified states as accepted or rejected evidence; update the retrieval memory with committed tokens `// recycle every verified state`
end while
return X

bound, e.g., $J(\mathcal{T}^\star) - J(\widehat{\mathcal{T}}) \leq \eta + \varepsilon\big(2n + D(D-1)/2\big)$, whereas the cap at h holds regardless of surrogate optimality. The results do not assert that training attains a particular value of ε, nor that a smaller bound implies a monotone improvement in realized speed.

B Additional Experimental Setup

B.1 Decoding Procedure

To make the decoding procedure of Section 3.4 easy to reimplement, we summarize it step by step in Algorithm 1, in which each round first orchestrates a draft tree from exact retrieval and calibrated historical evidence and then verifies it with one forward pass of the target, whose verified states are written back into the memory for later rounds.

B.2 Benchmark Datasets

We evaluate on the corpora summarized in Table 4, and the results of the main table are expanded cell by cell in Appendix C.1.

Table 4: Evaluation corpora: questions used, generated requests per method, output cap in tokens and input format. LongBench-Write sets the cap by the requested length, in four tiers of 2,048, 6,144, 12,288 and 16,384 tokens. WildBench is the v2 test split. One SWE-bench task is dropped for every method because its agent session crashes in the agent framework's timeout handling; greedy decoding without speculation ran on the first 20 tasks only.

Corpus	Questions	Requests	Output cap	Input
Spec-Bench	480	560	1,024	80 two-turn MT-Bench conversations, 400 single-turn questions
LiveBench	500	500	4,096	one user turn, six categories
LongBench-Write	120	120	2,048–16,384	59 English and 61 Chinese prompts
Alpaca subset	80	80	1,024	one user turn
WildBench	1,024	1,024	4,096	748 single messages, 276 conversation histories; next answer only
SWE-bench Verified	99 of 100	one session	1,024 per turn	live agent sessions with Qwen3-32B

B.3 Evaluation Metrics

Every method is verified by the same frozen target under greedy decoding, so its output can differ from that of greedy autoregressive decoding (AR) only where FP16 rounding flips a near-tie (Appendix C.8), and we therefore compare the methods by speed, measured both in wall-clock time and in tokens accepted per verification round, rather than by task scores. In wall-clock time, we report latency, the mean end-to-end time that the model service spends on a generated request, prefill included, and pooled throughput (TPS), which divides the retained output tokens of a corpus by the summed end-to-end time of its requests; dividing the pooled throughput of a method by that of AR on the same questions and the same GPU then gives its speedup.

In tokens per verification round, we report MAT, the mean number of tokens accepted per round, which divides the retained output tokens of a corpus by its verification rounds and is thus, like TPS, a pooled ratio rather than a mean over questions; moreover, since the root token of each round counts as accepted, AR has a MAT of one. Whereas these quantities all come from timed runs, the reference MAT used for the sensitivity sweeps and for the reference ablation of Appendix C.3 comes from the untimed reference implementation of the same algorithm and carries no timing.

Greedy decoding can fall into sustained repetition once an output runs long (Holtzman et al., 2020; Welleck et al., 2020). History-based drafters predict such loops almost perfectly, and keeping them in the measurement would inflate speculative gains. We therefore adopt a repetition rule, under which an output that reaches the token cap inside sustained 8-gram repetition is cut where that repetition begins and re-measured on the kept prefix, while outputs ending with EOS are left as they are. This rule applies to every method, each cut on its own output, and, except where a result is marked FULL, to every result in the paper. After a cut, the kept prefix is re-measured by replaying the request up to the cut on the same GPU, except where a table states that the cut timing is estimated.

Before settling on this rule, we considered a stricter one that cut only exactly periodic endings (the last 240 tokens periodic with a period of at most 80 tokens), but rejected it, because it misses paragraph-level loops with periods of 116 to 320 tokens as well as near-verbatim loops, whereas on exact loops both rules cut at the same position. In the main table, the share of outputs that the rule cuts ranges from none on Alpaca to 17–19% on LiveBench with Llama-3.1-8B (Table 11), and within each model–benchmark cell the six methods differ in this share by less than two percentage points, except for the TR entry on WildBench.

B.4 Baselines

To compare the methods on equal terms, all of them run in the same model service with the same frozen target and the same timing scope, while each drafter keeps the settings released by its authors (Table 5), so that no baseline is weakened by our own tuning. The draft budgets follow these settings as well: PLD drafts a single chain and TR keeps its larger official tree, whereas LogitSpec, RACER, SuffixDecoding and **RECAP** fill trees of the same size. Moreover, two baselines keep state beyond a single request and therefore see more history than **RECAP**, which, like RACER, starts every main-table conversation from an empty memory: TR keeps its adjacency matrix over all requests of a run, whereas SuffixDecoding keeps one suffix tree per machine over all earlier responses.

On WildBench, however, two baselines could not be measured under exactly the same protocol as the other methods, and to ensure that these exceptions never favor **RECAP**, we resolve each of them in the way that is more favorable to the baseline concerned. The first exception concerns TR, which keeps its adjacency matrix across requests: since the cut outputs of a batch are replayed only after the whole batch has been served, the replay of TR runs on a matrix that has already seen the reply itself, and for 32, 14 and 7 requests of Qwen3-8B, Llama-3.1-8B and Mistral-7B it no longer reproduces the original reply. For these requests we therefore report the FULL output of TR, i.e., its output without the repetition-rule cut described above, whose repetitive tail is exactly where TR accepts the most tokens, thereby raising its speedup by at most 1.4% without changing any ranking.

The second exception concerns LogitSpec, whose released code performs CPU work in every round and was therefore slowed down, unlike every other method, by a CPU-thread setting of our first WildBench service epoch; we thus re-time it in a later epoch on the same GPU for 984, 932 and 949 requests, against the AR run of the first epoch, and report as FULL the one Mistral request for which

Table 5: Drafters and their configurations. All methods run in the same model service with the same frozen target, greedy verification and timing scope; SuffixDecoding appears only in the agent experiment and EAGLE-3 only in Appendix C.2.

Method	Draft source	Configuration
AR	none	one token per verification
PLD (Saxena, 2023)	n-gram match in the context	n-grams up to 3; at most 10 draft tokens
Token Recycling (Luo et al., 2025)	adjacency matrix of candidate tokens	official tree template 2.2.2 (80 draft nodes plus the root), top-8; matrix kept over the requests of a run and not updated from prompt logits
LogitSpec (Liu et al., 2026c)	next-next-token logits and retrieval	n-grams up to 3; 20 draft tokens; 64 nodes; continuation length 4, 3 or 1 by candidate rank (thresholds 8 and 32); pool rebuilt from the whole conversation at each turn
RACER (Zhang et al., 2026)	retrieval automaton and copied logits	official code; fixed tree of 64 nodes
SuffixDecoding (Oliaro et al., 2025)	suffix tree over the prompt and earlier responses	arctic-inference 0.3.0; tree speculation; depth 64, $\alpha = 1.0$, minimum probability 0.1; 64-node budget; one tree per machine for the whole run
EAGLE-3 (Li et al., 2025)	trained draft head	public drafters (Appendix C.2)
RECAP	calibrated history and retrieval	$B = 64$, $R = 8$, $M = 32$, $K = 8$, $L = 16$, $T = 1$; memory emptied for every main-table conversation

no cut re-measurement exists, which changes its speedup by less than 0.1%. In this way, neither exception can work in favor of **RECAP**.

B.5 Scorer Architecture and Training

Since the scorer is evaluated on every draft node of every round, we keep it small, so that its arithmetic remains negligible against the verification pass. To this end, the scorer works in a $d = 128$-dimensional space: one linear layer projects the frozen target embedding of a token to d dimensions, and since the projected table is computed once at inference, a token costs only a lookup. On top of these token embeddings, a single-layer GRU with d hidden units reads the 16-token context of the query node, whereas each stored row keeps the GRU state of its own context beside it, so that only the query context has to be encoded in a round.

Besides its context, each record carries the metadata $\boldsymbol{\eta}_i$ of Eq. 5, which encodes the source, depth, age and recency of the record, and each of its stored predictions carries the cached-logit features $\boldsymbol{\psi}_{ij}$ of Eq. 6, namely the centered cached logit, its normalized rank, the Top-1/Top-2 margin and the logit range. These features are encoded by two linear layers with GELU each, applied to the $4d + 7$ row features of Eq. 5 in the row encoder and to the $4d + 4$ occurrence features of Eq. 6 in the occurrence encoder, after which the score head, a layer norm followed by $d \to 64 \to 1$, scores each occurrence.

Alongside the score head, the $\perp$ head, which gives the logit $z_u(\perp)$ of the out-of-support state in Eq. 8, reads an attention pool over the rows (three $d \times d$ maps), the mean of the occurrence states and three count features. Altogether, with a target embedding width of 4096, the scorer has 0.96M parameters, 0.52M of them in the input projection; since the projection is the only part that depends on the target, the width 5120 of Qwen3-32B raises the total to 1.09M.

With this architecture, the scorer performs about 29M multiply-accumulates per scored node, of which 1.6M fall in the GRU, 2.7M in the row encoder, 23M in the occurrence encoder and score head for the $M \times K = 256$ occurrences, and 1.2M in the pools and heads. Since best-first expansion scores at most $B = 64$ nodes in a round, a round costs the scorer at most 3.7 GFLOPs, whatever the target. By comparison, the target's verification pass over the 64 tree tokens costs about 2×64 FLOPs per parameter, that is 0.9–1.1 TFLOPs for the three 7–8B targets and 4.2 TFLOPs for Qwen3-32B, so the scorer's arithmetic is at most about 0.4% of the target's for the main-table models and below 0.1% for the agent model.

The measured cost is larger, however: a **RECAP** round takes about a quarter longer than a RACER round (Table 6). To locate this overhead, we profile the rounds of both methods kernel by kernel on Qwen3-8B (Table 7). The verification pass costs the same 16.2 ms in both methods, whereas the drafting of **RECAP** adds 6.05 ms of GPU time, since it runs as about 2,700 small kernels in some 26 CUDA-graph replays per round rather than as a few large matrix products. As **RECAP** accepts more tokens per round, it still spends less time per token than RACER, and fusing these kernels is the natural next engineering step.

Table 6: Latency on Qwen3-8B: mean end-to-end time per request (s) and mean time per verification round (ms), the latter as latency divided by the number of rounds. **RECAP** finishes requests sooner although each of its rounds costs more.

	Latency (s)		Round time (ms)	
Corpus	RACER	**RECAP**	RACER	**RECAP**
LiveBench	5.41	4.90	25.1	31.7
LongBench-Write	18.25	16.69	26.1	31.3
Spec-Bench	2.74	2.48	23.6	28.7
Alpaca	2.88	2.61	21.7	26.7
WildBench	12.77	11.00	24.8	29.8

Table 7: Time of one verification round of RACER and **RECAP**, and of one decoding step of AR, split by GPU kernel on Qwen3-8B (FP16, one RTX 5090), with all rows except MAT in milliseconds. The kernel times are measured with the PyTorch profiler on eight LiveBench outputs per method, namely the first 256 tokens for one prompt of each category and two complete outputs of 795 to 1,075 tokens, and are pooled over their rounds, whereas the wall-clock time is measured on the same outputs without the profiler; in the rest of a round the GPU waits for the host, which issues the kernels one at a time and runs the CPU part of the drafting, all of it for RACER. The AR column profiles the first 64 tokens of one prompt.

Stage	Kernels	AR	RACER	**RECAP**
Drafting	scorer GRU (cuDNN)	–	–	1.89
	fused scorer layers (Triton)	–	–	1.66
	FP32 matrix products	–	–	1.17
	masks, top-k, reductions, copies	–	0.01	1.33
Verification	linear layers (FP16 GEMM)	9.01	9.87	9.87
	attention	0.49	1.99	1.98
	RMSNorm	1.34	1.93	1.92
	RoPE, KV-cache writes, masks	1.09	1.33	1.34
	LM head	0.75	0.76	0.76
	activation, residuals, posterior	0.16	0.27	0.31
Commit	KV compaction, tree top-k, state update	–	0.09	0.10
GPU kernels		12.84	16.23	22.32
GPU idle (host-side work)		8.44	7.88	8.77
Wall-clock time per round		21.28	24.12	31.09
Tokens per round (MAT)		1.00	3.14	4.31
Time per token		21.28	7.68	7.22

Each target has its own scorer, trained in two stages with the settings of Table 8 on 201 prompts from 13 subsets of LongBench (Bai et al., 2024), split into 149 for training, 26 for validation and 26 held out; none of them comes from an evaluation corpus. To keep the held-out estimate unbiased, the 26 held-out prompts are scored only once after training and play no part in checkpoint selection. In the base stage the target decodes the training and validation prompts with the reference implementation, and every verified draft node gives one training decision: the memory the orchestrator saw for that node, labelled with the verifier's Top-1 token when it is among the stored candidates and with $\perp$ otherwise (Section 3.2).

The on-policy stage starts from the base checkpoint and lets the base scorer build the trees itself on the same prompts, so that the decisions come from states the orchestrator actually visits. It samples accepted and other decisions one to one within each question, balances the weights over source datasets and questions, and gives an accepted decision the weight $1 + \min(n_{\text{acc}}, 4)$, where n_{acc} counts the tokens accepted after it in its round, so that decisions leading to long accepted runs count more. Although their training traces were recorded in BF16, the scorers of the main-table targets were not retrained when serving moved to FP16.

Table 8: Scorer training settings. Both stages train only the scorer, with the target model and its embedding table frozen; the scorer width (128), context length ($L = 16$), candidates per record ($K = 8$) and memory size ($M = 32$) are the same in every run. Traces are capped at 512 generated tokens. Epoch limits are upper bounds, and early stopping sets the actual number. n_{acc} is the number of tokens accepted after a decision in its round.

Setting	Base stage	On-policy stage
Data	verifications on 149 + 26 prompts	trees built by the base scorer on the same prompts
Initialization	random	base checkpoint
Optimizer	AdamW	AdamW
Learning rate	10^{-3}	5×10^{-5}
Weight decay	10^{-4}	10^{-4}
Batch size	128	256
Maximum epochs	35	20
Early stopping (patience, min. change)	6, 10^{-5}	5, 10^{-5} on a moving average (0.8)
Decision weights	uniform	balanced over source datasets and questions
Sampled decisions	all	accepted and other, 1:1 within each question
Accepted-depth weight	none	$1 + \min(n_{\text{acc}}, 4)$
Checkpoint selection	validation objective	weighted validation objective
Seed	20260829	20260902

B.6 Tree Orchestration and Serving

In the orchestration step of Algorithm 1, each round starts from the target model's Top-1 token as the root, and exact retrieval contributes $\mathcal{T}_R$, at most R non-root nodes, which enter the tree directly as the anchor without being scored. The frontier $\mathcal{F}$ starts from the successors of the root: whenever a path is scored, the evidence scorer reads the memory rows of its last token, and the eight highest-scored successors are pushed into $\mathcal{F}$ with the cumulative scores of Eq. 11, which exposes them as candidates but consumes no budget. A path enters the tree only when it is popped as the highest-scoring path in $\mathcal{F}$, one path at a time, so that admission follows the exact best-first order of Appendix A until the tree holds B nodes or $\mathcal{F}$ is empty; a popped path that already lies in $\mathcal{T}_R$ is counted only once, and retrieval nodes are extended only where the scorer proposes the same path. Here the scores are the scorer's log-probabilities at the orchestrator temperature $T = 1$ (Appendix C.4). To reduce latency, the scorer is called in batches: before a pop, the highest-scoring paths in $\mathcal{F}$, up to eight, are scored together and their results are cached without being admitted, so that the admission order is exactly that of one-at-a-time scoring, since every score depends only on its own path and on the memory frozen for the round.

Two situations, however, leave the best-first tree short of B nodes, and in both of them RACER's fixed tree fills in, so that the verification budget is never left unused. The former arises when the memory holds no row for the root token, so that best-first expansion cannot start: the round then uses RACER's fixed tree over the retrieval automaton with the same budget B, in which the scorer still orders the children of deeper nodes where rows exist (variant V2 of Appendix C.3), while the retrieval memory is still advanced so that later rounds proceed normally. The latter arises when the frontier empties before the budget is spent, because the stored predictions along every open path are used up: the same fixed tree is then built, and its paths fill the remaining slots while the paths already

selected are kept, whereas a round that selects exactly B paths builds no fixed tree. Both events are counted per round in the service logs, and on the sessions of the timed ablation (Appendix C.3) that end with EOS, the former affects 15.2% of the rounds of **RECAP**.

On the serving side, all target models except the additional ones that appear only in the comparisons of Appendix C.2 are listed with their pinned revisions in Table 9, and the inference settings in Table 10. Under these settings, unless a table states otherwise, all targets are served in FP16 without quantization and decode greedily at batch size one. In addition, Qwen3's thinking mode is off and the date that Llama-3.1's chat template inserts is fixed, so that every run sees the same prompt tokens. Beyond identical prompts, every speedup also compares methods under the same hardware state, since for each model and corpus all methods run in one service epoch on one GPU and each question is timed for every method on that GPU; the WildBench exceptions are listed in Appendix B.4. The timing itself covers the service's handling of a request, prefill included, but excludes model loading, self-tests and one warm-up request per service.

Table 9: Target models, their Hugging Face repositories and pinned revisions, and where they are used. Weights come from ModelScope mirrors whose files match these revisions (for Llama-3.1-8B-Instruct the mirror revision is `72956b202e77350459703a6bd652025126e6f8ab`). Every target has its own scorer checkpoint, frozen during evaluation.

Model	Repository and revision	Used in
Qwen3-8B (Yang et al., 2025)	`Qwen/Qwen3-8B` `b968826d9c46dd6066d109eabc6255188de91218`	main table, ablation, sensitivity
Llama-3.1-8B-Instruct (Grattafiori et al., 2024)	`meta-llama/Llama-3.1-8B-Instruct` `0e9e39f249a16976918f6564b8830bc894c89659`	main table
Mistral-7B-Instruct-v0.2 (Jiang et al., 2023)	`mistralai/Mistral-7B-Instruct-v0.2` `63a8b081895390a26e140280378bc85ec8bce07a`	main table
Qwen3-32B (Yang et al., 2025)	`Qwen/Qwen3-32B` `9216db5781bf21249d130ec9da846c4624c16137`	agent sessions
DeepSeek-Coder-6.7B-Instruct (Guo et al., 2024)	`deepseek-ai/deepseek-coder-6.7b-instruct` `e5d64addd26a6a1db0f9b863abf6ee3141936807`	scorer specialization

Table 10: Inference, hardware and software settings shared by all runs unless a table states otherwise. Llama-3.1-8B on WildBench sets the SDPA attention kernel explicitly. Software versions are those recorded on the reference machine; every run also records hashes of its inputs, source code and scorer checkpoint.

Setting	Value
Precision	FP16 weights and activations, no quantization
Decoding	greedy, batch size 1; Qwen3 thinking mode off
Context / generation limit	32,768 / 16,384 tokens; per-corpus caps in Table 4
RECAP tree	$B = 64$ nodes including the root; retrieval reserve $R = 8$
RECAP memory	$M = 32$ records per token; $K = 8$ candidates; $L = 16$ context tokens
Scorer	width 128; orchestrator temperature $T = 1$
Retrieval automaton	capacity 40,000 nodes
EOS tokens	Qwen3: 151645, 151643; Llama-3.1: 128001, 128008, 128009; Mistral: 2
Llama-3.1 chat-template date	fixed to 04 Sep 2026
Timing	model service time per request, prefill included; loading, self-tests and one warm-up request excluded
GPUs	RTX 5090; RTX PRO 6000 for Qwen3-32B
Software	Python 3.12.3, PyTorch 2.8.0 with CUDA 12.8, Transformers 4.52.3

Table 11: Throughput (TPS), mean retained output length over the configured cap, and the share of generated requests cut by the repetition rule, for every cell of Table 1. Requests per method: 500 LiveBench, 120 LongBench-Write, 560 Spec-Bench, 80 Alpaca, 1,024 WildBench; LongBench-Write caps are per question (2,048 to 16,384 tokens). On WildBench, the TR and LogitSpec rows are the exceptions described in Appendix B.4.

		Qwen3-8B Efficiency			Llama-3.1-8B Efficiency			Mistral-7B Efficiency		
Benchmark	Method	TPS ↑	$\bar{\ell}/\ell_{\max}$	Cut (%)	TPS ↑	$\bar{\ell}/\ell_{\max}$	Cut (%)	TPS ↑	$\bar{\ell}/\ell_{\max}$	Cut (%)
LiveBench (500 questions)	AR	54.46	698.9 / 4096	7.00	56.27	458.0 / 4096	17.20	70.88	403.3 / 4096	0.80
	PLD	87.65	708.8 / 4096	6.60	102.52	450.6 / 4096	19.00	105.95	408.5 / 4096	1.20
	TR	115.80	721.9 / 4096	6.60	132.80	452.5 / 4096	18.80	125.85	405.2 / 4096	0.80
	LogitSpec	111.87	716.2 / 4096	6.80	129.77	459.9 / 4096	17.20	131.23	408.2 / 4096	2.20
	RACER	130.53	705.7 / 4096	6.80	151.03	450.4 / 4096	18.00	147.86	403.1 / 4096	1.20
	`RECAP`	143.08	701.3 / 4096	6.60	165.99	460.2 / 4096	18.20	165.20	403.6 / 4096	1.20
LongBench-Write (120 questions)	AR	54.60	1850.2 / *	5.83	52.49	988.4 / *	7.50	71.82	924.9 / *	4.17
	PLD	60.64	1899.9 / *	5.00	75.03	994.9 / *	8.33	80.07	952.4 / *	3.33
	TR	90.89	1854.9 / *	5.00	119.26	988.8 / *	7.50	121.06	930.5 / *	4.17
	LogitSpec	74.92	1908.8 / *	5.83	96.85	996.1 / *	9.17	98.38	936.1 / *	5.00
	RACER	97.15	1772.7 / *	5.83	117.72	961.1 / *	7.50	123.52	952.1 / *	3.33
	`RECAP`	110.97	1851.6 / *	5.00	131.73	969.9 / *	7.50	139.44	956.6 / *	4.17
Spec-Bench (480 questions)	AR	50.80	283.5 / 1024	1.25	65.56	221.3 / 1024	1.79	72.39	204.8 / 1024	0.00
	PLD	62.36	283.5 / 1024	0.89	83.86	221.1 / 1024	1.79	88.35	205.5 / 1024	0.00
	TR	107.48	282.9 / 1024	1.25	126.74	224.1 / 1024	1.43	123.98	204.7 / 1024	0.00
	LogitSpec	79.27	283.7 / 1024	1.07	104.65	221.5 / 1024	1.61	107.04	205.9 / 1024	0.00
	RACER	103.84	284.9 / 1024	1.07	132.97	221.6 / 1024	1.79	130.49	204.8 / 1024	0.00
	`RECAP`	114.41	283.3 / 1024	1.25	137.53	222.3 / 1024	1.43	141.95	204.7 / 1024	0.00
Alpaca (80 questions)	AR	59.94	281.3 / 1024	0.00	66.62	268.8 / 1024	0.00	73.44	264.2 / 1024	0.00
	PLD	60.51	287.0 / 1024	0.00	68.06	265.0 / 1024	0.00	72.11	262.4 / 1024	0.00
	TR	106.02	282.4 / 1024	0.00	123.73	266.5 / 1024	0.00	124.55	261.9 / 1024	0.00
	LogitSpec	73.70	282.9 / 1024	0.00	86.64	269.1 / 1024	0.00	90.98	261.4 / 1024	0.00
	RACER	99.54	286.5 / 1024	0.00	117.56	267.9 / 1024	0.00	116.25	260.6 / 1024	0.00
	`RECAP`	108.91	283.9 / 1024	0.00	125.32	267.9 / 1024	0.00	128.34	260.5 / 1024	0.00
WildBench (1,024 questions)	AR	52.09	1346.2 / 4096	5.37	53.56	819.1 / 4096	5.37	69.74	731.2 / 4096	1.27
	PLD	64.88	1346.5 / 4096	5.18	78.19	829.9 / 4096	4.98	85.85	730.5 / 4096	1.66
	TR	104.51	1425.6 / 4096	1.95	124.14	872.6 / 4096	3.52	122.11	752.5 / 4096	0.88
	LogitSpec	80.90	1350.0 / 4096	4.59	100.07	830.0 / 4096	4.79	105.72	727.3 / 4096	1.27
	RACER	105.99	1352.9 / 4096	4.59	126.51	831.3 / 4096	4.69	131.56	737.0 / 4096	1.46
	`RECAP`	122.71	1350.0 / 4096	4.69	140.33	820.2 / 4096	5.18	151.06	732.2 / 4096	1.27

C Additional Experimental Results

C.1 Main-Table Details

Since the main table reports only MAT and speedup, we list the throughput, the mean output length and the share of outputs cut by the repetition rule for each of its cells in Table 11, and add the mean latency per request and the AR rows in Table 12.

C.2 Comparison with EAGLE-3 and Concurrent Methods

To compare `RECAP` with a strong trained drafter, we use EAGLE-3 on the five targets for which a public drafter exists (Table 13), whereas we did not train drafters for the remaining targets ourselves. As these runs belong to the BF16 campaign on RTX 5090 GPUs that preceded the FP16 main table, they predate the repetition rule, which we apply afterwards: each method's own output is cut by the main-table detector, and since the tokens and verification rounds up to the cut are recorded, only the time before the cut has to be estimated, as the non-decoding time of the request plus its decoding time in proportion to the rounds kept.

For EAGLE-3, whose runner does not record the decoding time separately, the non-decoding time is taken from the AR request for the same question, and spreading the whole request time evenly over the rounds instead changes no speedup by more than 0.006. In this way, all methods are measured under the same rule as the main table, which here cuts between 31 and 89 of the 500 outputs of each method and target. To avoid the estimate altogether, we also compare on the questions where

Table 12: Table 1 with mean latency per request and the AR rows. Bold marks the best result in each block; on WildBench, the TR and LogitSpec rows are the exceptions described in Appendix B.4.

		Qwen3-8B			Llama-3.1-8B			Mistral-7B		
Benchmark	Method	Lat. (s) ↓	MAT ↑	Speedup ↑	Lat. (s) ↓	MAT ↑	Speedup ↑	Lat. (s) ↓	MAT ↑	Speedup ↑
LiveBench	AR	12.83	1.00	1.00×	8.14	1.00	1.00×	5.69	1.00	1.00×
	PLD	8.09	2.00	1.61×	4.40	2.11	1.82×	3.86	1.94	1.49×
	TR	6.23	3.09	2.13×	3.41	3.08	2.36×	3.22	2.62	1.78×
	LogitSpec	6.40	2.75	2.05×	3.54	2.83	2.31×	3.11	2.54	1.85×
	RACER	5.41	3.27	2.40×	2.98	3.33	2.68×	2.73	2.89	2.09×
	`RECAP`	**4.90**	**4.54**	**2.63**×	**2.77**	**4.78**	**2.95**×	**2.44**	**4.11**	**2.33**×
LongBench-Write	AR	33.89	1.00	1.00×	18.83	1.00	1.00×	12.88	1.00	1.00×
	PLD	31.33	1.46	1.11×	13.26	1.64	1.43×	11.89	1.40	1.11×
	TR	20.41	2.60	1.66×	8.29	2.79	2.27×	7.69	2.41	1.69×
	LogitSpec	25.48	1.98	1.37×	10.29	2.20	1.84×	9.51	1.83	1.37×
	RACER	18.25	2.54	1.78×	8.16	2.71	2.24×	7.71	2.30	1.72×
	`RECAP`	**16.69**	**3.47**	**2.03**×	**7.36**	**3.83**	**2.51**×	**6.86**	**3.24**	**1.94**×
Spec-Bench	AR	5.58	1.00	1.00×	3.38	1.00	1.00×	2.83	1.00	1.00×
	PLD	4.55	1.38	1.23×	2.64	1.52	1.28×	2.33	1.51	1.22×
	TR	2.63	2.62	2.12×	1.77	2.70	1.93×	1.65	2.45	1.71×
	LogitSpec	3.58	1.84	1.56×	2.12	2.04	1.60×	1.92	1.96	1.48×
	RACER	2.74	2.45	2.04×	1.67	2.62	2.03×	1.57	2.40	1.80×
	`RECAP`	**2.48**	**3.29**	**2.25**×	**1.62**	**3.45**	**2.10**×	**1.44**	**3.33**	**1.96**×
Alpaca	AR	4.69	1.00	1.00×	4.03	1.00	1.00×	3.60	1.00	1.00×
	PLD	4.74	1.19	1.01×	3.89	1.20	1.02×	3.64	1.17	0.98×
	TR	2.66	2.48	1.77×	2.15	2.53	1.86×	2.10	2.33	1.70×
	LogitSpec	3.84	1.60	1.23×	3.11	1.63	1.30×	2.87	1.59	1.24×
	RACER	2.88	2.16	1.66×	2.28	2.23	1.76×	2.24	2.03	1.58×
	`RECAP`	**2.61**	**2.90**	**1.82**×	**2.14**	**2.98**	**1.88**×	**2.03**	**2.79**	**1.75**×
WildBench	AR	25.85	1.00	1.00×	15.29	1.00	1.00×	10.49	1.00	1.00×
	PLD	20.75	1.51	1.25×	10.61	1.66	1.46×	8.51	1.59	1.23×
	TR	13.64	2.74	2.01×	7.03	2.90	2.32×	6.16	2.58	1.75×
	LogitSpec	16.69	1.99	1.55×	8.29	2.21	1.87×	6.88	2.09	1.52×
	RACER	12.77	2.63	2.03×	6.57	2.84	2.36×	5.60	2.58	1.89×
	`RECAP`	**11.00**	**3.66**	**2.36**×	**5.84**	**3.95**	**2.62**×	**4.85**	**3.70**	**2.17**×

every method ends with EOS, on which `RECAP` is faster on four targets and close to EAGLE-3 on Llama-3.1-8B (2.77× against 2.79×).

Of the methods that appeared while this work was under way, we compare with ECHO (Ma et al., 2026), which extends RACER's code base with bonus logits from an early exit of the target, because Goose (Jin et al., 2026) had no public implementation at the time of writing and we leave AdaPLD (Liu et al., 2026a) to future work. Since these bonus logits require target weights whose early layers were trained for early exit, we evaluate on LayerSkip-Llama3-8B, as the ECHO paper does, and follow the released protocol of ECHO (Table 14), under which almost all outputs of every method reach the cap of 512 new tokens and are kept in full, as in ECHO's evaluation code, instead of being cut by the repetition rule of Appendix B.3.

To keep the comparison faithful to ECHO, we run ECHO and its AR with the released code and the settings that its README gives for this model, whereas AR, RACER and `RECAP` run in our harness on identical prompt tokens and on the same RTX 5090; since the two AR runs decode at the same speed, every speedup is computed against the AR run of the same code base. Two differences remain, and both work in favor of ECHO: on the one hand, its evaluation keeps the retrieval automaton across questions, whereas our harness resets all state for every request; on the other hand, `RECAP` reuses the scorer of Llama-3.1-8B-Instruct from the main table without retraining, which the shared Llama 3 tokenizer permits.

Under this protocol, `RECAP` achieves the highest pooled MAT and speedup (Table 14) and, per category, the highest MAT everywhere and the highest speedup everywhere except summarization (Table 15), whereas ECHO accepts more tokens per round than RACER but still falls behind it in

Table 13: EAGLE-3 and RECAP on LiveBench500 in an earlier BF16 campaign on RTX 5090 GPUs, all 500 questions. EAGLE-3 was timed in its own runs, so its speedups are compared across runs. The Llama-3.1-8B drafter is the official release; the Qwen3 drafters (AngelSlim) and the Llama-3.2-3B drafter are third-party releases. No public drafter exists for Mistral-7B-Instruct-v0.2 or DeepSeek-Coder-6.7B-Instruct.

	MAT		Speedup over AR	
Target	EAGLE-3	RECAP	EAGLE-3	RECAP
Repetition rule applied to the recorded outputs (cut timing estimated)				
Qwen3-1.7B	4.06	**4.68**	2.15×	**2.92**×
Qwen3-4B	3.54	**4.46**	1.96×	**2.95**×
Qwen3-8B	3.57	**4.48**	2.08×	**2.73**×
Llama-3.2-3B	3.41	**4.75**	1.92×	**2.87**×
Llama-3.1-8B	4.48	**4.88**	2.57×	**3.03**×
FULL outputs, no repetition rule				
Qwen3-1.7B	4.37	**5.38**	2.31×	**3.34**×
Qwen3-4B	3.94	**5.80**	2.18×	**3.77**×
Qwen3-8B	3.89	**5.39**	2.24×	**3.25**×
Llama-3.2-3B	3.61	**7.46**	2.01×	**4.37**×
Llama-3.1-8B	4.16	**7.85**	2.39×	**4.86**×

speedup. These absolute speedups are not comparable with those reported in the ECHO paper, which were measured on an RTX 3090.

Table 14: Comparison with ECHO under its own evaluation protocol and released settings: LayerSkip-Llama3-8B, greedy decoding in FP16 on one RTX 5090, the 400 Spec-Bench questions without MT-Bench with the FastChat one-shot template, at most 512 new tokens and outputs reaching the cap kept (FULL), as in ECHO's evaluation. ECHO and its AR run the released code (commit 44ee06b) with the README settings for this model: exit layer 10, verification threshold 3 and at most 30,000 nodes. MAT is the mean accepted length per verification round, TPS the pooled throughput in tokens per second, and speedup is relative to the AR run of the same code base on the same GPU.

Method	Code base	MAT	TPS	Speedup
AR	unified harness	1.00	68.4	1.00×
RACER	unified harness	3.10	165.6	2.42×
RECAP	unified harness	**4.38**	**176.9**	**2.59**×
AR	ECHO release	1.00	67.9	1.00×
ECHO	ECHO release	3.26	157.9	2.32×

Table 15: Results of Table 14 per Spec-Bench category. Speedups are relative to the AR run of the same code base (RACER and RECAP: unified harness; ECHO: ECHO release); bold marks the highest value in each row and group.

		Speedup			MAT		
Category	n	RACER	RECAP	ECHO	RACER	RECAP	ECHO
Math reasoning	80	2.42×	**2.66**×	2.26×	3.07	**4.45**	3.09
Question answering	80	2.09×	**2.31**×	2.02×	2.63	**3.73**	2.86
Retrieval (RAG)	80	2.43×	**2.68**×	2.33×	3.18	**4.67**	3.34
Summarization	80	**2.89**×	2.58×	2.82×	3.80	**4.53**	3.97
Translation	80	2.40×	**2.74**×	2.32×	3.03	**4.66**	3.27
All	400	2.42×	**2.59**×	2.32×	3.10	**4.38**	3.26

Table 16: Timed ablation per corpus with Qwen3-8B on the questions of the main table: MAT, pooled throughput (TPS, tokens per second) and speedup over AR on Spec-Bench (560 requests), LiveBench (500) and LongBench-Write (120), and pooled over the three (All, 1,180). Every configuration runs on the RTX 5090 that also runs AR on the same questions (V2–V4 in their own service instances) and is scored on its own output, as in the main table; speedup divides by the AR throughput on the same questions and GPUs. Bold marks the best value in each column.

Variant	Spec-Bench			LiveBench			LongBench-Write			All		
	MAT	TPS	Speedup	MAT	TPS	Speedup	MAT	TPS	Speedup	MAT	TPS	Speedup
AR	1.00	51.1	1.00×	1.00	51.2	1.00×	1.00	50.6	1.00×	1.00	51.0	1.00×
V1: RACER	2.45	102.9	2.01×	3.27	132.4	2.59×	2.54	99.2	1.96×	2.82	114.0	2.24×
V2: + scorer, fixed tree	2.83	108.1	2.12×	3.76	138.3	2.70×	2.97	103.6	2.05×	3.26	119.0	2.34×
V3: + best-first, R=0	**3.28**	**115.3**	**2.26**×	4.38	142.2	2.78×	3.46	113.9	**2.25**×	3.79	126.0	2.47×
V4: single record (M=1)	2.71	97.0	1.90×	3.37	113.1	2.21×	2.79	95.6	1.89×	3.02	103.7	2.04×
`RECAP` (full)	**3.28**	113.9	2.23×	**4.54**	**145.7**	**2.85**×	**3.47**	**114.0**	**2.25**×	**3.86**	**127.2**	**2.50**×

C.3 Ablation Details

The ablation of Section 4.3 is timed on the production system with Qwen3-8B, on the questions of the main table (all 480 Spec-Bench questions with both turns of the multi-turn ones, 500 LiveBench and 120 LongBench-Write questions), split over six RTX 5090s so that all configurations of a question run on the same GPU. Since V2, V3 and V4 each alter one component of the compiled orchestrator, they run on a build of the production service that differs from it only by one switch read at start-up.

Of these, V2 builds RACER's fixed tree in every round, with the same shape and budget, but lets the scorer choose each node's children, namely the candidates of the stored rows ranked by the scorer for the current path, while RACER's most recent row is used only where the scorer has no row; this is also the code path that the production system takes in rounds whose root token has no stored row (Appendix B.6). V3, in contrast, keeps best-first orchestration but sets the retrieval reserve to $R = 0$, so that the frontier starts from the root alone, whereas V4 keeps only the most recent row of each token, both in storage and in what the scorer sees.

To keep the comparison fair, we first check that, with the switch unset, this build reproduces the outputs and the per-round acceptance of the production service token for token on 60 control questions (10 Spec-Bench first turns per GPU). Since each configuration is served by its own service instance, one after another on the same GPU, the same control questions also check the instances: AR, V1 and the full model share one instance, and on these questions a separate instance of the full model reached the pooled throughput of the shared one to within 1%, so the separate instances hardly favor V2–V4.

Beyond these checks, the repetition rule is applied to every configuration's own output, and each cut output is replayed up to its cut on the same instance; when the cut falls in the first turn of a two-turn question, the second turn is generated again on the cut history, as in the main table. The resulting MAT, throughput and speedup per corpus are reported in Table 16, from which the cost of a verification round follows as MAT divided by throughput: pooled over the three corpora, it is 24.8 ms for RACER, 27.4 ms once the scorer is added (V2) and 30.3 ms for the full model.

In Table 16, the RACER and full-model rows reproduce the MAT of the main table to within 0.2% on every corpus, and their speedups are within about 1% of the main table on Spec-Bench and 8–11% higher on LiveBench and LongBench-Write, where AR ran at 51 tokens per second on these GPUs against 55 in the main-table run. However, since every configuration verifies its own output and FP16 near-ties make long outputs drift apart, different configurations do not always produce the same text. To compare the configurations on the same text, we therefore restrict them to the 636 questions where every configuration reproduces the AR output, on which MAT rises from V1 to V2, V3 and the full model (2.92, 3.34, 3.72 and 3.81), while V4 falls back to 3.05. Likewise, on the 732 questions where both V3 and the full model reproduce the AR output, the retrieval reserve adds 2.0% MAT.

To remove text divergence altogether, we repeat the ablation on all 480 Spec-Bench first turns in the reference implementation, which scores every variant on AR's output and pairs the questions across variants, so that the intervals come from resampling questions (Table 17). The resulting ordering matches that of the same-text subset above: the full model first, V3 close behind with an overlapping interval, then V2 and V4. This ordering holds in every task category except the MT-Bench first turns, where V3 is marginally ahead. Among the categories, translation gains least overall, since there the scorer in the fixed tree helps about as much as elsewhere, but best-first orchestration adds almost nothing.

Since none of the variants above isolates the scorer, as V1 removes it together with the memory and the planner while V2–V4 keep it, we further isolate what the learned scorer itself contributes by keeping the full model unchanged except for its scorer, which we replace by one of two rules that require no training: V5 keeps only the records whose stored context shares the longest exact suffix with the current prefix and averages their cached distributions, whereas V6 averages the cached distributions of all M records regardless of their contexts, so that the former lets exact context matches decide which records count while the latter keeps every candidate but ignores the context. Neither rule recovers the gain of the full model (Table 17), since the exact-suffix rule barely improves on RACER and averaging all records even falls below it, which suggests that pooling predictions from heterogeneous prefixes without weighting them spreads the budget over candidates that the current prefix does not support. In this way, the gain of **RECAP** comes from learning which historical records transfer to the current prefix rather than from the larger candidate set alone.

Table 17: Ablation in the reference implementation on the 480 Spec-Bench first turns with Qwen3-8B: MAT per task category (questions in parentheses) and pooled (All), and the gain over RACER with a 95% interval from a paired bootstrap over questions (2,000 resamples). V5 and V6 keep the full model but replace its learned scorer by a rule that requires no training. Every variant is scored on AR's output, so all rows share the same text; the repetition rule cuts the same 5 of the 15 capped outputs in every row, at round boundaries, and keeps 97% of the tokens. Reference MAT carries no timing.

Variant	Math (80)	MT (80)	QA (80)	RAG (80)	Summ. (80)	Transl. (80)	All (480)	Gain (%)	95% interval
V1: RACER	2.74	2.31	2.20	2.36	2.52	1.69	2.37	–	–
V2: + scorer, fixed tree	3.17	2.67	2.62	2.83	2.84	1.96	2.77	+16.7	[16.2, 17.3]
V3: + best-first, R=0	3.69	**3.05**	**3.03**	3.34	3.23	**2.00**	3.19	+34.4	[33.4, 35.5]
V4: single record (M=1)	3.03	2.58	2.53	2.75	2.81	1.90	2.68	+13.0	[12.3, 13.7]
V5: hard suffix, no learning	2.82	2.27	2.24	2.46	2.53	1.60	2.39	+0.9	[0.0, 1.8]
V6: mean of M records, no learning	2.64	2.10	2.07	2.26	2.39	1.58	2.23	−6.1	[−6.7, −5.4]
RECAP (full)	**3.72**	**3.05**	**3.03**	**3.38**	**3.24**	**2.00**	**3.20**	+35.0	[33.9, 36.0]

C.4 Sensitivity Details

All sweeps of Section 4.2 run on three sensitivity populations, namely 113 Spec-Bench first turns, 60 LiveBench questions (10 per category) and 12 LongBench-Write questions (3 per length tier), and report the reference MAT of Appendix B.3. To check that the advantage of **RECAP** does not stem from its budget alone, we add the matched-budget comparison that Figure 3(a) omits, i.e., RACER and **RECAP** at the same node budget on the same questions (Table 18): the gain of **RECAP** over RACER grows with the budget on every corpus, from 16% at $B = 8$ to 38% at $B = 64$ and 45% at $B = 96$ on the pooled population.

Beyond the budget and the history of Section 4.2, the orchestrator reads the scorer through a temperature T: it divides every logit of Eq. 8, that of $\perp$ included, by T before normalizing, so that $T < 1$ concentrates the budget on the best-scored successors and grows deeper trees, whereas $T > 1$ spreads it over more branches. Since the scorer itself is trained at $T = 1$, we check whether its probabilities need rescaling (Table 19): $T = 1$ gives the highest MAT on every corpus, and a departure costs more the further it goes, so the trained probabilities can be used as they are.

The temperature sweep shows which scaling serves the orchestrator best, but not whether the probabilities of the scorer are calibrated. We therefore measure calibration on the 164,836 decisions of the 26 validation prompts, which serve only for early stopping: at $T = 1$, the deployed scorer is moderately overconfident, with an expected calibration error of 0.104 over 15 equal-width bins, a Brier score of 0.638 and a negative log-likelihood of 1.85, whereas a single temperature fitted on these

Table 18: Matched-budget comparison behind Figure 3(a): pooled reference MAT of RACER and **RECAP** at the same node budget B on Qwen3-8B, per corpus and pooled over the three (All). Budgets above 64 are reference-only, as the timed system is compiled for $B = 64$.

Corpus	Method	B=8	16	32	48	64	80	96
Spec-Bench	RACER	1.47	1.77	2.08	2.28	2.37	2.43	2.46
	RECAP	1.69	2.14	2.64	2.99	3.23	3.43	3.58
LiveBench	RACER	2.03	2.45	2.88	3.16	3.31	3.39	3.45
	RECAP	2.46	3.11	3.85	4.29	4.64	4.86	5.03
LongBench-Write	RACER	1.55	1.84	2.15	2.32	2.40	2.42	2.45
	RECAP	1.73	2.16	2.63	2.98	3.23	3.38	3.54
All	RACER	1.71	2.06	2.41	2.63	2.74	2.80	2.84
	RECAP	1.99	2.51	3.10	3.48	3.77	3.96	4.13

decisions, $T = 1.29$, lowers the calibration error to 0.023, or 0.022 under two-fold cross-fitting. Orchestrating at $T = 1.3$, however, leaves MAT unchanged, at 3.232 against 3.234 for $T = 1$ on the 113 Spec-Bench questions of the sensitivity analysis, with a paired difference of -0.06% and a 95% interval of $[-0.96, 0.84]$, so this overconfidence does not affect the orchestration.

Table 19: Reference MAT of **RECAP** on Qwen3-8B as the orchestrator temperature T varies ($B = 64$, $R = 8$, $M = 32$), on the sensitivity populations (113 Spec-Bench first turns, 60 LiveBench and 12 LongBench-Write questions) and pooled over the three (All, 185). The scorer is unchanged; only the probabilities the orchestrator derives from it are sharpened ($T < 1$) or flattened ($T > 1$). The shaded row is the default.

Setting	Spec-Bench	LiveBench	LongBench-Write	All
$T = 0.5$	3.09	4.18	3.06	3.51
$T = 0.75$	3.20	4.50	3.20	3.70
$T = 1$	3.23	4.64	3.23	3.77
$T = 1.5$	3.19	4.52	3.13	3.68
$T = 2$	3.06	4.18	2.99	3.48

To check whether any setting interacts with the output length, we split the Spec-Bench questions of the sensitivity analysis into three groups by the length of the reference output (Table 20). In this split, the gain over RACER grows with the budget in every group and, at every budget, with the output length, which is the within-corpus counterpart of Obs. ❸ in Section 4.2. Among the other settings, keeping only a few records per token costs most on long outputs, whereas the reserve shows no length dependence, and the temperature is the only setting whose best value depends on the length: on the shortest outputs a moderately flatter distribution does slightly better than the default, while on medium and long outputs every departure from $T = 1$ lowers MAT.

The reserve R bounds how many non-root nodes exact retrieval may claim before best-first expansion fills the rest of the budget (Eq. 10), and we vary it with the other settings at their defaults (Table 21). Without the reserve, MAT drops on every corpus, least on Spec-Bench, whereas above the default it changes little: only LiveBench keeps gaining slowly, so pooled MAT moves by about half a percent between $R = 8$ and $R = 32$. The default $R = 8$ itself was fixed early in development, before larger reserves were evaluated, and both this sweep and the timed ablation (Appendix C.3) put the value of the reserve at one to two percent of MAT.

In the history sweep of Figure 3(b), whose values are listed in Table 22, every corpus gains from each doubling of the history, LiveBench most, and between 16 and 32 records the gain is small on Spec-Bench and LongBench-Write but still clear on LiveBench. Since the system stores at most 32 records per token, larger histories can only be examined through the oracle of Table 23, which was computed during development with Qwen3-1.7B on 26 development questions and asks only whether the verifier's next token appears among the stored candidates of the last M records of the same token, thereby bounding what any scorer could find. In this oracle, availability rises steeply over the first

Table 20: Spec-Bench sensitivity questions grouped by the length of the reference output (Qwen3-8B, reference MAT). The groups are tertiles of the reference length, fixed in advance. Upper block: gain of **RECAP** over RACER at the same node budget B, with $R = B/8$. Lower block: change of **RECAP**'s MAT when one setting departs from its default ($B = 64$, $R = 8$, $T = 1$, $M = 32$).

	Short	Medium	Long	All
Questions	38	36	39	113
Reference length (tokens)	8–129	132–241	245–1,024	8–1,024
Mean reference length	53	179	582	276
*Gain of **RECAP** over RACER at the same budget (%)*				
$B = 16$	+11.9	+18.5	+22.6	+20.8
$B = 32$	+17.1	+25.7	+28.9	+27.2
$B = 48$	+22.1	+26.7	+34.3	+31.5
$B = 64$	+25.7	+31.8	+39.6	+36.6
*Change of **RECAP**'s MAT from its default (%)*				
$R = 0$	−0.3	−1.8	−0.1	−0.5
$R = 4$	−0.1	−1.6	−0.7	−0.9
$R = 16$	−0.5	−0.1	0.0	−0.1
$T = 0.5$	−3.1	−5.8	−4.1	−4.4
$T = 0.75$	−0.9	−1.0	−1.1	−1.1
$T = 1.5$	+1.4	−0.3	−2.0	−1.3
$T = 2$	−0.3	−4.1	−6.4	−5.4
$M = 1$	−10.8	−15.4	−17.2	−16.2
$M = 2$	−4.8	−8.6	−10.1	−9.3
$M = 4$	−1.7	−2.9	−4.2	−3.7
$M = 8$	+0.3	+0.3	−1.8	−1.2
$M = 16$	+0.5	−1.1	−0.1	−0.3

Table 21: Reference MAT of **RECAP** on Qwen3-8B as the retrieval reserve R varies ($B = 64$, $T = 1$, $M = 32$), on the sensitivity populations (113 Spec-Bench first turns, 60 LiveBench and 12 LongBench-Write questions) and pooled over the three (All, 185). The shaded row is the default.

Setting	Spec-Bench	LiveBench	LongBench-Write	All
$R = 0$	3.22	4.52	3.15	3.70
$R = 4$	3.20	4.56	3.18	3.72
$R = 8$	3.23	4.64	3.23	3.77
$R = 16$	3.23	4.68	3.23	3.78
$R = 32$	3.24	4.71	3.20	3.79

few records and then flattens, and 32 records already reach nearly everything an unbounded history would, at a fixed memory cost per token.

Table 22: Reference MAT of **RECAP** on Qwen3-8B when the orchestrator sees only the M most recent records of each token ($B = 64$, $R = 8$, $T = 1$; no retraining), on the sensitivity populations (113 Spec-Bench first turns, 60 LiveBench and 12 LongBench-Write questions) and pooled over the three (All, 185); these are the values plotted in Figure 3(b). The shaded row is the default.

Setting	Spec-Bench	LiveBench	LongBench-Write	All
$M = 1$	2.71	3.45	2.63	2.99
$M = 2$	2.93	3.74	2.86	3.24
$M = 4$	3.11	4.06	3.04	3.48
$M = 8$	3.19	4.31	3.12	3.61
$M = 16$	3.22	4.48	3.20	3.71
$M = 32$	3.23	4.64	3.23	3.77

C.5 Mechanism and Case Studies

To see which stored predictions the orchestrator actually uses, we traced 34 WildBench replies of Mistral-7B by re-running the benchmark service with read-only hooks: the four passages with the largest difference between the two methods, and 30 replies spaced evenly over those that **RECAP**, RACER and AR produce identically. These traced runs reproduce the benchmark runs token for token and round for round. For each draft token that **RECAP** accepted, we then record where its memory held that token before the round, and, for each round in which RACER stopped before the next token of the reply, why that token was missing from RACER's tree (Table 24).

On **RECAP**'s side, almost half of the accepted draft tokens were present only in records of rejected draft nodes. Most of the rest come from records of committed text, about a third of them from a candidate below the verifier's top-1. On RACER's side, about four in ten early stops happen because the needed prediction had been stored in an older record of the same token that RACER's single record per token had since overwritten, about one in six because the latest record held it but the fixed tree did not expand it, and nearly all others because no earlier record of that token had predicted it. Keeping the records of rejected nodes and keeping several records per token are therefore both used heavily. These shares are similar across categories, except for the two reasoning replies, which draw most of their tokens from committed text.

The case of Figure 4 in the main text is shown in full in Figure 5: it covers all rounds from 102 to 126 instead of the last six and keeps every layer of the trace, which its legend explains. This passage does not come from an unusually easy response, since the response that contains it even has a lower MAT than the average of **RECAP** on WildBench with this target.

The same mechanism also appears in code (Figure 6): in WildBench question 792, Mistral-7B writes a gamma-correction routine in Python. While the model is still emitting the digits of four numeric constants, several look-ahead branches already continue past them into the comment and the channel split that follow. The verifier rejects these branches in their own rounds, but **RECAP** keeps their verified states, and in round 72 it commits the 19-token passage in a single verification round, whereas RACER needs twelve rounds for the same tokens. Only a few of these tokens have an earlier n-gram precedent in the committed text, so exact retrieval could not have drafted the passage in one round.

C.6 Persistent Agent Sessions and Accumulated Evidence

To evaluate **RECAP** where the history persists, each method runs its own live trajectory on the same 99 SWE-bench Verified tasks with mini-swe-agent, the minimal agent of the SWE-agent project (Yang et al., 2024); since a lossless method reproduces greedy decoding, the trajectories differ only through FP16 near-ties. Among these tasks, the no-loop ones, 21 of the 99, are those on which every method keeps the repeat fraction of its assistant messages below 0.5. Besides RACER, we compare with SuffixDecoding in its official implementation (arctic-inference 0.3.0), with a suffix tree over the

Table 23: History-capacity oracle on the development corpus: for each capacity M, the share of the 245,782 verifier decisions of the 26-question development partition (Qwen3-1.7B, 512-token cap) whose Top-1 token appears in the latest M rows of the same-key history, and that share as a fraction of what the full history reaches. This is an occurrence diagnostic that motivates the bounded default $M = 32$; it is not a MAT or speed result.

Capacity M	Reachable decisions	Availability (%)	Full-history coverage (%)	Increment (pp)
1	123,419	50.21	63.86	–
2	142,648	58.04	73.81	+7.82
4	158,123	64.33	81.82	+6.30
8	169,929	69.14	87.93	+4.80
16	178,824	72.76	92.53	+3.62
32	**184,901**	**75.23**	**95.67**	**+2.47**
64	188,895	76.85	97.74	+1.63
128	191,411	77.88	99.04	+1.02
256	192,688	78.40	99.70	+0.52
Full history	193,260	78.63	100.00	+0.23

Table 24: Where **RECAP**'s accepted draft tokens come from, and why RACER's rounds end early, on 34 traced WildBench replies of Mistral-7B. **RECAP** columns: share of accepted draft tokens whose token was, before that round, the verifier's top-1 in a record of committed text (Top-1), among its candidates 2–8 in such a record (Rank 2–8), present only in records of rejected draft nodes (Rejected), or absent from the memory (None; these rounds all carry retrieval paths). Shares are in percent of the counts. RACER columns: for each round that ended before the next token of the reply, whether that token had been predicted in an older record of the same token that RACER's single record per token had overwritten (Overwritten), was in the latest record but not expanded by the tree (Not expanded), had never been predicted for that token (Never), or the round had reached RACER's nine-token limit (Cap). All replies end with EOS, so the repetition rule does not change them, and every traced run reproduces the benchmark run token for token and round for round. The first row holds the four passages with the largest difference between the methods; the evenly spaced replies are those that **RECAP**, RACER and AR produce identically, with 150 to 2,500 tokens.

		RECAP: accepted draft tokens					RACER: rounds ending early				
Replies	n	Count	Top-1	Rank 2–8	Rejected	None	Count	Overwritten	Not exp.	Never	Cap
Largest-gap passages	4	1,154	25.6	18.0	56.2	0.1	800	41.1	16.2	42.5	0.1
Evenly spaced replies	30	10,079	38.7	16.8	44.3	0.2	6,345	40.2	17.1	41.3	1.3
All	34	11,233	37.3	16.9	45.6	0.2	7,145	40.3	17.0	41.5	1.2
Coding & Debugging	8	3,095	44.8	16.5	38.4	0.2	1,671	43.6	19.9	34.2	2.3
Planning	6	2,338	35.0	18.3	46.7	0.0	1,493	42.4	18.6	38.9	0.1
Creative Writing	4	1,436	29.5	15.0	55.5	0.0	1,075	39.6	14.1	46.1	0.1
Information seeking	5	1,260	35.1	14.9	50.0	0.0	857	34.2	16.2	49.4	0.2
Advice seeking	4	1,240	24.8	18.8	55.9	0.5	858	42.1	15.7	41.0	1.2
Editing	2	723	39.3	15.4	45.1	0.3	518	42.5	14.1	42.9	0.6
Reasoning	2	549	61.4	19.5	18.2	0.9	190	32.6	22.6	30.0	14.7
Role playing	2	432	32.9	17.1	50.0	0.0	361	32.1	12.7	55.1	0.0
Data Analysis	1	160	31.2	21.2	47.5	0.0	122	35.2	14.8	50.0	0.0

prompt and all earlier responses of the run, tree speculation, default hyperparameters and a 64-node budget.

As in the main table, capped looping turns are cut by the repetition rule; the cut turns of **RECAP** and RACER are re-measured, whereas those of SuffixDecoding use the uniform-round estimate, which matched the measurement of the other two methods to within 0.05%. The MAT and throughput of these sessions, which Figure 3(d) plots, are listed in Table 25 and their per-turn latency in Table 26. Since these sessions use a larger target than the main table, we also run Spec-Bench on Qwen3-32B as a control (Table 27), where the MAT gain of **RECAP** over RACER, +31.2%, stays close to the

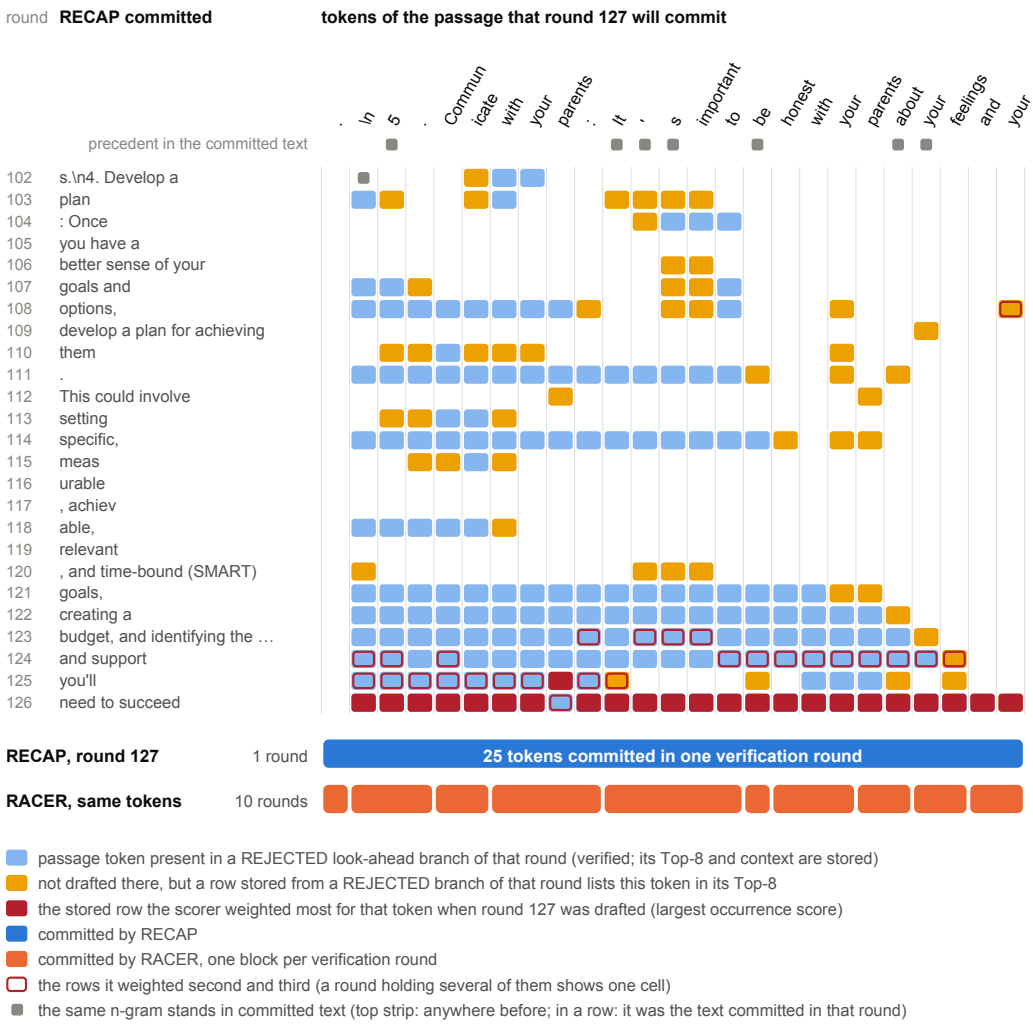


Figure 5: An advice-generation passage from WildBench with Mistral-7B. Fragments of the item-5 continuation appear in rejected draft branches while the model is still writing item 4; **RECAP** retains those prediction rows and commits the 25-token span in a single verification round, whereas RACER needs ten. Red marks the rows the scorer weighted most for each drafted token (filled: first; outlined: second and third), using recomputed scores that rebuild the 64-node draft tree of round 127 exactly.

+34.3% on Qwen3-8B and far below the +68.4% of the agent sessions, so the target size alone does not explain the larger gain.

Table 25: Persistent agent sessions with Qwen3-32B on SWE-bench Verified: pooled MAT and throughput (TPS) on all tasks and on the tasks where no method loops; Figure 3(d) plots the throughput.

	All tasks		No-loop tasks	
Method	MAT	TPS	MAT	TPS
RACER	5.16	36.9	3.46	30.6
SuffixDecoding	5.99	50.9	2.51	23.9
RECAP	**8.69**	**52.9**	**5.88**	**42.9**

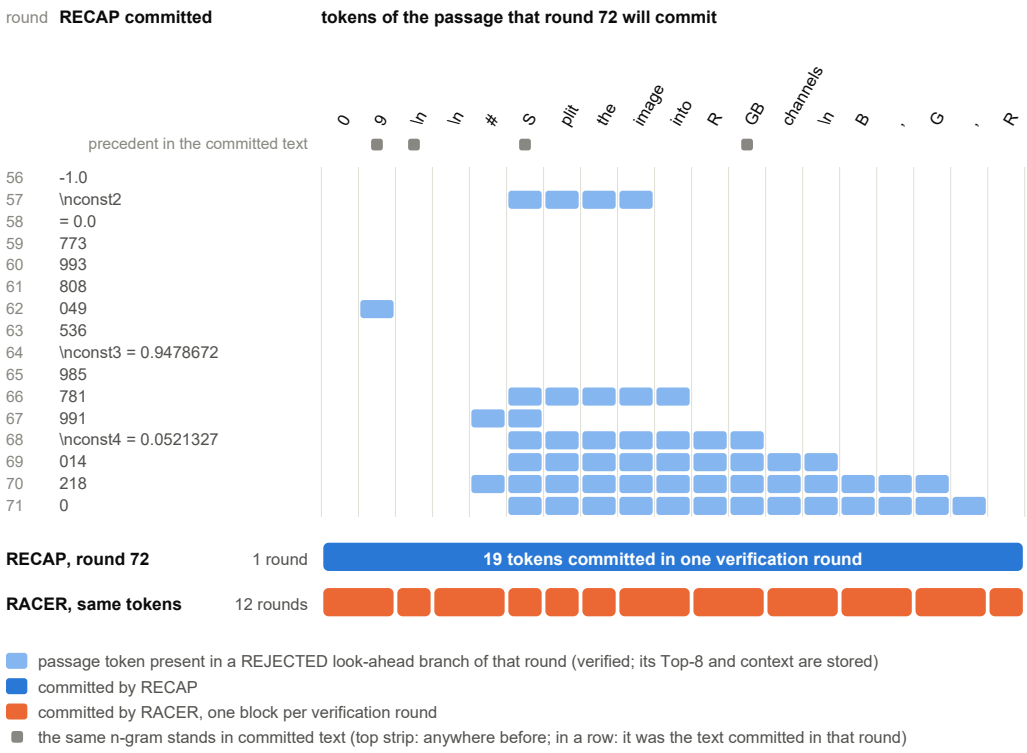


Figure 6: A code-generation passage from WildBench (question 792) with Mistral-7B. Light blue marks tokens of the passage that already appear in rejected look-ahead branches of earlier rounds; **RECAP** retains those states and commits the 19-token passage in round 72, whereas RACER needs twelve rounds for the same tokens.

Table 26: Per-turn latency behind the agent results of Table 25 (Qwen3-32B, all tasks): mean model-service latency per turn, pooled MAT and throughput over the 99 tasks that all three methods complete. Latency excludes tool, network and session-close time and depends on each method's own trajectory, so turn counts are given.

Method	Sessions	Turns	Latency (s) ↓	MAT ↑	TPS ↑
RACER	99	3,920	2.34	5.16	36.9
SuffixDecoding	99	4,147	1.79	5.99	50.9
RECAP	99	3,618	1.88	**8.69**	**52.9**

Speedups over AR, in contrast, exist only for the first 20 tasks, fixed before the runs, because greedy decoding without speculation was run only on them (Table 28). On them SuffixDecoding has the highest speedup, whereas on the six of them where no speculative method loops, **RECAP** has the highest, as on the 21 no-loop tasks of Table 25; as in Section 4.2, the advantage of SuffixDecoding thus stems from sessions that loop. Indeed, SuffixDecoding's sessions loop most often, 57 of the 99 against 45 for RACER and 42 for **RECAP**, which is consistent with its larger numbers of turns and of capped turns in the workload of Table 29.

To relate acceptance to the amount of accumulated evidence, we place each verification round at the evidence Φ accumulated before it (Figure 7), which counts every record written so far, including those that the cap of M records per token has since evicted. This evidence accumulates in two ways: every prompt token leaves one record of $K = 8$ target logits, whereas every verification round leaves one record for each verified node, at most $B = 64$; at the pooled MAT of **RECAP** over the main-table runs, 3.77, one generated token therefore corresponds to about $8 \cdot 64/3.77 \approx 136$ stored logits, so

Table 27: Spec-Bench control for the agent results at both target sizes, on all 480 questions. The Qwen3-32B control uses the agent scorer and service; within each question AR, RACER and **RECAP** run on the same GPU. The agent row uses the 99 tasks that all three methods complete.

		RACER		**RECAP**		Gain	
Workload	Target	MAT	TPS	MAT	TPS	MAT	TPS
Spec-Bench (480)	Qwen3-32B	2.38	34.16	3.13	41.23	+31.2%	+20.7%
Spec-Bench (480)	Qwen3-8B	2.45	103.84	3.29	114.41	+34.3%	+10.2%
SWE-bench agent (99)	Qwen3-32B	5.16	36.94	8.69	52.91	+68.4%	+43.3%

Table 28: Speedup over greedy decoding without speculation (AR) in the agent experiment with Qwen3-32B, on the first 20 tasks of the subset, fixed before the runs and the only ones on which AR ran, and on the 6 of them where no speculative method's session loops. Latency is the mean time per generated turn; speedup divides pooled throughput by AR's on the same tasks. Every method follows its own trajectory, so turn counts differ between methods; this is not a speedup of the agent's wall-clock time per task.

Method	Latency (s)	MAT	TPS	Speedup
20 AR tasks				
AR	9.69	1.00	10.35	1.00×
RACER	3.35	5.17	37.13	3.59×
SuffixDecoding	1.93	6.69	57.69	5.57×
RECAP	3.06	7.91	47.25	4.57×
6 of them without loops				
AR	7.48	1.00	13.79	1.00×
RACER	2.65	3.22	32.27	2.34×
SuffixDecoding	4.04	2.52	26.82	1.95×
RECAP	2.23	4.63	39.83	2.89×

Table 29: Workload behind the agent results with Qwen3-32B: generated turns, their mean retained length in tokens, turns that reach the 1,024-token cap, and turns the repetition rule cuts, on the 99 tasks of Table 25 and on the 20 tasks where AR also ran. Cut turns of **RECAP** and RACER are re-measured; the 22 of SuffixDecoding use the uniform-round estimate of Appendix C.6.

Method	Turns	Mean length	Capped	Cut (share)
99 tasks				
RACER	3,920	86.5	45	22 (0.56%)
SuffixDecoding	4,147	91.1	71	22 (0.53%)
RECAP	3,618	99.4	69	26 (0.72%)
20 AR tasks				
AR	516	100.3	2	0 (0.00%)
RACER	534	124.3	23	0 (0.00%)
SuffixDecoding	763	111.1	24	0 (0.00%)
RECAP	667	144.4	26	3 (0.45%)

that a round that follows P prompt tokens and G generated tokens is placed at

$$\Phi = 8P + 136\,G, \tag{20}$$

where P counts every prompt appended so far and thus grows with each tool result in agent sessions. We apply the same formula to **RECAP** and RACER, so that rounds at the same position of the same text share the same Φ, whereas counting the records each method actually writes would make the axis depend on the method, because RACER needs more rounds for the same text: on the main table its count is a median 1.54 times Eq. 20, against 1.03 for **RECAP**.

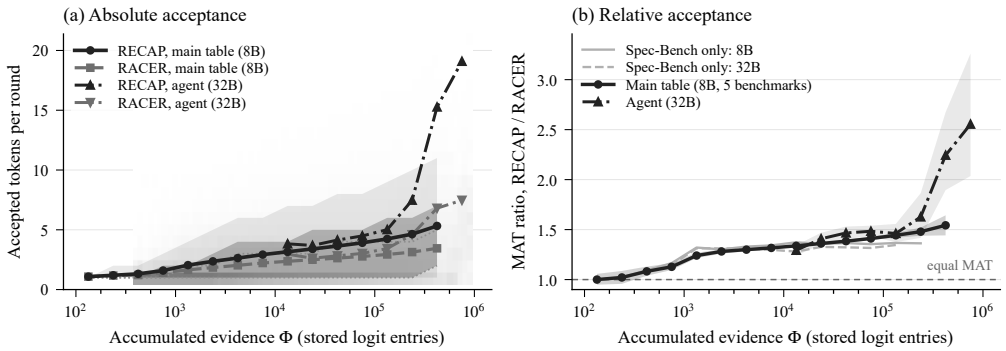


Figure 7: Accepted tokens per verification round against the evidence Φ accumulated before that round, estimated from the prompt and output lengths. (a) Absolute acceptance; shaded bands give the spread of **RECAP** rounds. (b) **RECAP**/RACER ratio with 95% bootstrap intervals over questions; the grey lines use Spec-Bench alone on each backbone.

Along this axis, rounds are grouped in bins of 0.25 in $\log_{10}\Phi$, and a bin is plotted when it holds rounds from at least 30 questions of each method, with intervals from 1,000 bootstrap resamples of questions. To separate the effect of the target from that of the workload, we also draw grey curves in panel (b) that use Spec-Bench alone, once for the three main-table targets pooled and once for Qwen3-32B from the control of Table 27, and these differ by at most 0.09. In panel (b), where the evidence is equal, the main-table and agent workloads give roughly the same **RECAP**/RACER ratio, but the agent sessions accumulate far more evidence and thus reveal how large the advantage of **RECAP** becomes in truly long contexts.

Moreover, acceptance rises within a question as well: among the main-table questions with at least 40 rounds, the last quarter of rounds accepts more tokens on average than the first quarter in 4,659 of 5,618 questions for **RECAP** and in 4,850 of 5,884 for RACER, so the trend in Figure 7 does not merely reflect differences between questions. At the low end of the axis, the leftmost main-table bin, which holds the first rounds of 196 requests with 13–22-token prompts, serves as a sanity check: the two methods accept the same number of tokens in every one of these rounds, and 92% of them accept only the verifier's own token.

C.7 Scorer Specialization

Since the scorer is the only learned component of **RECAP**, adapting it to a domain is cheap, and to test whether this pays off, we retrain the scorer of DeepSeek-Coder-6.7B-Instruct on coding prompts, which touches nothing else and took under four hours on one RTX 5090, and evaluate it on the coding, math and reasoning questions of LiveBench (Table 3). Specialization pays off in its own domain and carries over to a neighboring one: the gain is largest on coding, where a throughput deficit against RACER becomes a lead, and math gains in both measures. It does, however, cost a little where the original scorer was already strong: on reasoning, MAT and throughput drop slightly, so a specialized scorer complements the original rather than replacing it.

Like the EAGLE-3 runs, the coder experiment was recorded without cuts, so the repetition rule is applied afterwards in the same way (Appendix C.2). Here, no coding output reaches the token cap, whereas on math the rule cuts two outputs of the original scorer and five of the coding scorer, the three extra ones being outputs that loop until the cap, and on reasoning it cuts two of each. Since the coding scorer ran in a separate epoch on the same GPU, AR and RACER were run on the 64 coding questions again in that epoch and after switching back, and their throughput moved by at most 2.1%, well below the gains on coding and math.

With the original scorer, **RECAP** accepts more tokens per round than RACER on all three domains but is slower on coding and math, because the per-round overhead of Appendix B.5 weighs more on this smaller target with its shorter forward pass; the extra acceptance of the coding scorer is what turns this deficit into a lead. This overhead stems from implementation settings fixed when the system was frozen early in development rather than from the method, and reducing it is a natural next step.

C.8 Output Quality

RECAP changes which tokens are drafted, never which token is committed: every committed token is the target's greedy choice given the committed prefix, so in exact arithmetic every method in this paper returns the AR output. In floating point, however, a verification pass over a tree evaluates the same positions in a different order of operations than sequential decoding, and where the two highest logits are nearly tied the greedy choice can flip; from then on the output is a different greedy continuation of the target, which we checked in three ways.

The most direct check compares the first-turn replies of RACER, **RECAP** and LogitSpec with the AR reply of the same question under BF16 and FP16 serving (Table 30). In BF16, most replies depart from AR somewhere, whereas in FP16 most MT-Bench and LiveBench replies stay identical and the rate of departures per character falls by about an order of magnitude, although the LongBench-Write replies, five to ten times longer, depart more often. That rate is about the same for the three drafters in both precisions, so the departures belong to the arithmetic, not to the drafter; this comparison is why every target is served in FP16.

The departures themselves were audited in an earlier multi-turn BF16 run with Qwen3-8B, in which every turn that reached the token cap (250 turns over four decoding arms) and 45 randomly chosen turns at which a speculative arm first departs from AR were re-decoded with a fresh full-prefill pass of the frozen target on the arm's exact context. At every audited position, the token the arm committed is among the target's two highest logits, with a gap to the top logit of at most 0.25 logits in all cases but one (0.5), and more than half of the gaps are exactly zero; since the plain AR arm, which continues from a cached KV state, differs from the fresh pass in the same way and as often, the audit measures tie resolution between two prefill paths rather than an effect of drafting.

The third check is a self-test at every service start, which compares each method's multi-turn continuation with a full-history recompute of the same conversation, and in FP16 the continuations are identical over their whole length for every method (73 tokens on Qwen3-8B, 96 on Llama-3.1-8B and Mistral-7B). Since every method thus returns greedy continuations of the target, we compare speed rather than task scores.

D Additional Related Work

Speculative decoding obtains candidate continuations either from learned drafters or from reusable text and predictions. The former rely on auxiliary drafting modules or on the target's own layers (Cai et al., 2024; Ankner et al., 2024; Zhou et al., 2024; Du et al., 2024; Zhang et al., 2024; Liu et al., 2024; Li et al., 2025; Ma et al., 2026), which typically yields accurate drafts but requires training or modifying the model, and Appendix C.2 compares **RECAP** with EAGLE-3 and ECHO. The latter need no drafter training, since they either copy continuations from existing text (Saxena, 2023; Yang et al., 2023; He et al., 2024; Hu et al., 2025; Liu et al., 2026a; Oliaro et al., 2025) or reuse the predictions that the target has already computed (Fu et al., 2024; Luo et al., 2025; Liu et al., 2026c).

Table 30: Agreement of first-turn replies with the AR reply of the same question under FP16 and BF16 serving, from the comparison that preceded the main table (RTX 5090): replies that are token-identical to AR, and the number of positions per 1,000 characters of shared text at which a reply first departs from AR. MT-Bench is the 80-question first-turn set; the BF16 LongBench-Write runs of Llama-3.1-8B cover 78 questions, and LiveBench was compared in FP16 only.

			Identical to AR		Departures per 1,000 chars	
Corpus	Target	Method	FP16	BF16	FP16	BF16
MT-Bench	Qwen3-8B	RACER	64/80	19/80	0.136	1.724
		RECAP	54/80	18/80	0.249	1.837
		LogitSpec	54/80	21/80	0.258	1.572
	Llama-3.1-8B	RACER	64/80	19/80	0.133	1.306
		RECAP	67/80	21/80	0.100	1.499
		LogitSpec	66/80	24/80	0.118	1.079
	Mistral-7B	RACER	64/80	24/80	0.185	1.354
		RECAP	63/80	32/80	0.201	0.906
		LogitSpec	70/80	29/80	0.109	0.986
LongBench-Write	Qwen3-8B	RACER	43/120	3/120	0.245	2.929
		RECAP	43/120	5/120	0.246	2.961
		LogitSpec	40/120	3/120	0.252	2.277
	Llama-3.1-8B	RACER	65/120	4/78	0.158	1.919
		RECAP	59/120	1/78	0.140	2.192
		LogitSpec	65/120	2/78	0.119	2.155
	Mistral-7B	RACER	62/120	2/120	0.182	1.505
		RECAP	70/120	3/120	0.159	1.486
		LogitSpec	61/120	1/120	0.203	1.395
LiveBench	Qwen3-8B	RACER	382/500	–	–	–
		RECAP	374/500	–	–	–
		LogitSpec	387/500	–	–	–
	Llama-3.1-8B	RACER	413/500	–	–	–
		RECAP	408/500	–	–	–
		LogitSpec	422/500	–	–	–
	Mistral-7B	RACER	419/500	–	–	–
		RECAP	425/500	–	–	–
		LogitSpec	417/500	–	–	–