TRAJECTORY-RETRIEVAL SPECULATIVE DECODING: WHEN DOES A MODEL'S OWN HISTORY HELP?

Anonymous authors
Paper under double-blind review

ABSTRACT

Long chain-of-thought reasoning increases sequential decoding cost while creating a growing history of potentially reusable continuations. We investigate when this history supplies useful drafts and complements an existing drafter. Controlled source comparisons reveal trajectory-specific reuse, motivating our method **T**rajectory-**L**ocal **A**daptive **R**etrieval (TLAR). TLAR retrieves approximately matched continuations from the current trajectory and uses recent verification outcomes to adapt retrieval activation and candidate width. TLAR combines retrieved continuations with model-generated drafts in a shared candidate tree, preserving the target model's output distribution through exact verification. Across code debugging, mathematics, and open-ended writing, our evaluation connects source reuse, incremental acceptance, and execution cost. Combining TLAR with strong retrieval baselines improves token acceptance under matched verification budgets and increases end-to-end throughput over the draft-model baseline. These findings support generated trajectories as runtime memory for adaptive inference. See implementation details at: https://anonymous.4open.science/r/tlar-5BEE.

1 INTRODUCTION

Large language models (LLMs) can improve their performance on difficult tasks by generating long chains of thought (CoT) before producing a final answer (Wei et al., 2022). Long-CoT reasoning is used in competition mathematics (Lightman et al., 2024; Brown et al., 2024; Muennighoff et al., 2025), program synthesis and debugging (Jimenez et al., 2024; Jain et al., 2025), and scientific reasoning (Rein et al., 2023). This benefit comes at a cost: a single response may contain thousands of reasoning tokens, as autoregressive models generate these tokens one at a time, each new token can be produced only after the preceding tokens are available (Leviathan et al., 2023). As reasoning traces grow longer, this sequential decoding cost accumulates, making efficient token generation increasingly important for practical long-CoT inference.

A practical goal is to accelerate this process without changing the target model's output distribution. Recently, speculative decoding does so by using a faster drafter to propose several future tokens, and the target model then verifies in parallel (Leviathan et al., 2023; Chen et al., 2023). Most methods use a smaller model or a lightweight prediction head trained to imitate the target model (Li et al., 2024b; Cai et al., 2024). Currently, some methods extend the same idea from individual tokens to complete reasoning steps (Pan et al., 2026; Wang et al., 2026; Shi et al., 2025). These methods can be effective, but they require an additional trained component. To avoid this requirement, retrieval-based methods construct drafts from existing text. They search the prompt, an external datastore, or tokens already generated by the target model, and use the continuations following a match as draft candidates (Somasundaram et al., 2025; He et al., 2024; Fu et al., 2024; Hu et al., 2025; Oliaro et al., 2026; Li et al., 2024a; Luo et al., 2025; Chen et al., 2025). Among these, we focus on the tokens generated earlier in the current trajectory. Similar contexts may appear more than once during long reasoning. When this happens, the tokens that followed an earlier context can be reused as a draft. However, using the current trajectory as a draft source raises three challenges.

Challenge 1: Separating trajectory reuse from common text. Retrieval may succeed because reasoning traces share common text. Different reasoning traces may share common words, symbols, and formatting patterns, allowing retrieval to succeed even when the current history is not used. High acceptance can also result from a model repeating the same text without making progress. A fair test must compare the current trajectory with other retrieval sources while keeping the task, target model, and matching rule unchanged.

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.

Challenge 2: Finding tokens beyond a small draft model. Even if the current trajectory provides useful drafts, a small draft model may already predict the same tokens. In that case, adding retrieval does not increase the number of accepted tokens. So we must compare the combined system with the same small draft model used alone.

Challenge 3: Controlling the cost of retrieval verification. Every retrieved candidate must be verified by the target model which adds computation, while reusable content does not appear at every position in a reasoning trace. An effective method should decide when to use retrieval and how many candidates to verify based only on results observed at earlier positions.

We address these challenges through source comparisons and **T**rajectory-**L**ocal **A**daptive **R**etrieval (TLAR). After defining acceptance and cost in Section 2, we compare current history with control sources and another rollout of the same problem in Section 3. These observations guide TLAR in Section 4: we find approximate context matches in generated history and combine their continuations with SmallDraft in a prefix-sharing tree. We retain the SmallDraft candidate and use earlier verification results to choose when to retrieve and how many candidates to include. Periodic probes check for new useful content, and the target model verifies every proposed token exactly. In Section 5, we test added acceptance over SmallDraft, added coverage under shared retrieval-node budgets, and the controller's gain–cost trade-off separately. We summarize our contributions as follows:

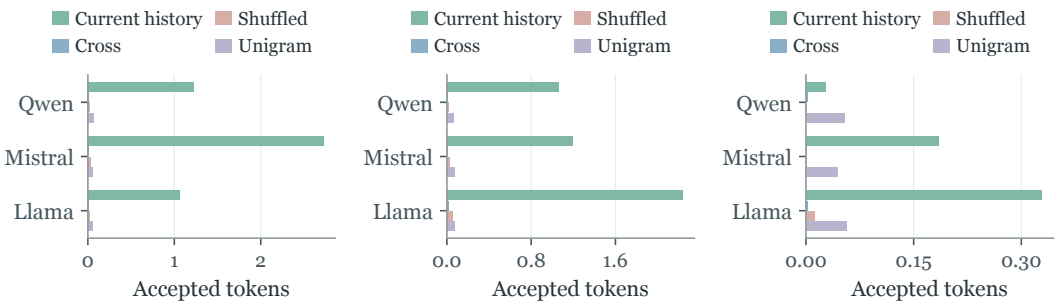
- **Uncovering Trajectory-Specific Reuse.** We test when the current history provides more accepted tokens than other sources, and use same-prompt controls to distinguish the effect of the trajectory from that of the problem. The advantage depends on model and task.
- **More Accepted Tokens Within the Same Budget.** We measure how many additional tokens the target model accepts when TLAR is combined with a draft model compared with using the same draft model alone. We then combine TLAR with the strongest retrieval baseline under shared verification-node budgets. The union yields more accepted tokens than either source alone, showing the value of trajectory-local candidates within the budget.
- **An Adaptive Retrieval Method.** We introduce TLAR, which searches the current trajectory and uses earlier verification results to decide when to retrieve and how many candidates to submit. Experiments across models, tasks, and verification budgets show when this approach is useful and how its gain changes with its cost.

2 PRELIMINARIES

Speculative Decoding and Token Verification. Given a prompt x, a target language model $\mathcal{M}$ generates a token sequence $\mathbf{y} = (y_1, \ldots, y_T)$. At position t, the tokens already generated form the history $h_t = (y_1, \ldots, y_{t-1})$. In speculative decoding, a drafter uses this history to propose a sequence of future tokens $\mathbf{c} = (c_1, \ldots, c_L)$. The target model verifies these tokens in one forward pass and accepts the longest prefix that matches $(y_t, \ldots, y_{t+L-1})$. We define the accepted length as $a(\mathbf{c}, t) = \max\{\ell : 0 \le \ell \le L,\ c_{1:\ell} = y_{t:t+\ell-1}\}$. Then we first generate one target sequence $\mathbf{y}$ for each prompt, and test every drafting method at the same positions in this sequence, so all methods are asked to predict the same next tokens.

Model-Based and Retrieval-Based Drafting. The candidate $\mathbf{c}$ defined above can be produced in two ways. A model-based drafter uses a smaller language model $\mathcal{M}_{\text{draft}}$ to predict the next tokens from the prompt and the current history (Li et al., 2024b; Cai et al., 2024). Let $\mathbf{d}_t$ denote its draft at position t. Its accepted length is $a_{\text{small}}(t) = a(\mathbf{d}_t, t)$. A retrieval-based drafter constructs candidates from text that is already available (Somasundaram et al., 2025; He et al., 2024; Hu et al., 2025; Oliaro et al., 2026). It searches a text source for a context similar to the current one and uses the tokens following that context as a draft. In this work, we focus on retrieval from the current history h_t. Let $\mathcal{R}_t$ denote the set of candidates retrieved at position t. Its accepted length is $a_{\text{retr}}(t) = \max_{\mathbf{r} \in \mathcal{R}_t} a(\mathbf{r}, t)$, with $a_{\text{retr}}(t) = 0$ when no candidate is retrieved.

Incremental Acceptance and Verification Cost. We now measure whether retrieval adds value beyond the small draft model. We first consider positions where the small draft model fails on its first token, so $a_{\text{small}}(t) = 0$. The recovery rate is $R = \Pr(a_{\text{retr}}(t) \ge 1 \mid a_{\text{small}}(t) = 0)$. It measures how often retrieval finds at least one correct token when the small draft model fails immediately. Since recovery does not cover positions where the small draft model predicts some correct tokens but retrieval provides a longer continuation, we define the added accepted tokens at position t as



(a) Code debugging (100 tasks) (b) Mathematics (24 tasks) (c) Open-ended writing (24 tasks)

Figure 1: Current history outperforms the source controls in code and mathematics, but not consistently in open-ended writing. We compare mean exactly accepted tokens, E_{exact}, from four sources. Current history (Within) uses the generated prefix. Cross uses another trajectory from the same model and dataset, limited to the same history length. Shuffled uses the current history with its blocks reordered. Unigram uses token frequencies without matching the current context. Panels use separate horizontal scales; Llama mathematics excludes repeated-copying cases (Appendix C). Exact values appear in Appendix Table 8.

$\delta(t) = \max\{a_{\text{small}}(t), a_{\text{retr}}(t)\} - a_{\text{small}}(t)$. The average gain is $\Delta = \frac{1}{T}\sum_{t=1}^{T}\delta(t)$. Retrieval receives credit only when it allows the target model to accept more tokens than the same small draft model used alone. We then measure verification cost by the number of retrieval nodes submitted to the target model. Let $N_{\text{retr}}(t)$ denote this number at position t, with $N_{\text{retr}}(t) = 0$ when retrieval is not used. The average cost is $C = \frac{1}{T}\sum_{t=1}^{T} N_{\text{retr}}(t)$.

3 An Exploratory Study

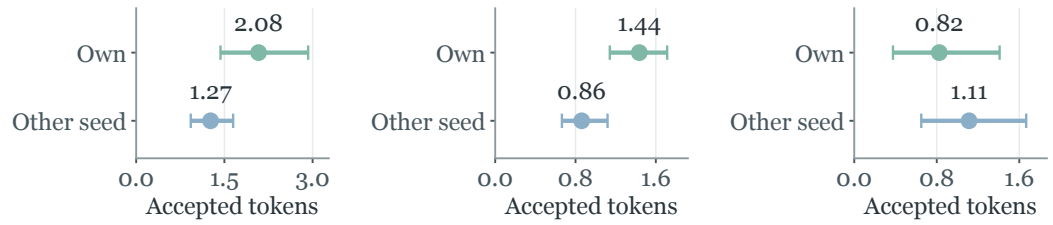
Before testing whether retrieval can improve a small draft model, we first ask whether the current trajectory is a better retrieval source than other available text. We then ask whether any advantage remains when the alternative trajectory comes from the same problem, and whether successful retrieval is spread evenly through generation.

Retrieval and controller settings. The default retrieval configuration uses context length $c = 4$, mismatch tolerance $\epsilon = 1$, history threshold $\tau = 512$, and continuation depth $d = 4$. Fixed-width experiments use $k = 4$. The adaptive controller uses $k_{\min} = 1$, $k_{\max} = 4$, $\rho_{\min} = 0.05$, half-life $H = 32$, and probe interval $P = 32$. Section 5.4 examines candidate configurations and controller components; Appendix E reports the half-life and probe-interval sweeps.

Evaluation Protocol. We keep the target trajectory and acceptance rule fixed and compare four draft sources. *Within* searches the current generated history. *Cross* searches another trajectory from the same target model and dataset, restricted to the same available history length. *Shuffled* searches a block-shuffled version of the current history. *Unigram* uses token frequencies without matching the current context. Within, Cross, and Shuffled use the same matcher. We report the mean exactly accepted length, E_{exact}, for Qwen3-8B, Mistral-Small-3.2-24B-Instruct, and Llama-3.1-8B-Instruct.

Finding 1: Current history helps more in code and mathematics. We find higher accepted lengths from Within than from each of the three controls for all three models in code debugging and mathematics (Figure 1). Changing the source changes acceptance even when the target tokens remain fixed. The effect is smaller in open-ended writing: Mistral's Within advantage becomes smaller, and Qwen does not exceed Unigram. We also retain the Llama repeated-copying audit, since long matches caused by repeated text do not by themselves show useful reuse. These differences limit the claim to the models and tasks in which the source controls support it.

Finding 2: The same problem need not provide the same drafts. Cross changes both the problem and the trajectory. To separate them, we compare two rollouts of each of 24 code-debugging problems with different random seeds. We show the paired source comparisons in Figure 2, comparing own and other history within each panel. Qwen and Mistral favor their own history over the other rollout, while Llama shows the opposite ordering. These paired comparisons support trajectory-specific reuse for Qwen and Mistral, whereas Llama's reusable drafts appear to depend more on the shared problem than on its own rollout.



(a) Qwen3-8B: exact context (b) Mistral-24B: exact context (c) Llama-3.1-8B: exact context

Figure 2: Own-history and same-prompt, different-seed retrieval. Points show mean accepted tokens and bars show 95% bootstrap intervals. Each comparison uses 24 paired problems.

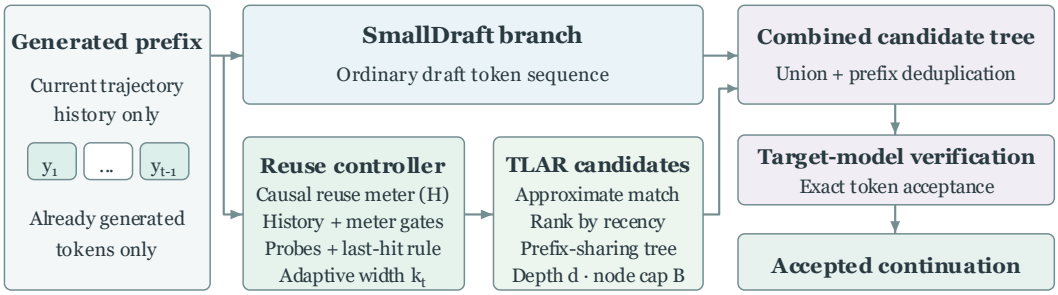


Figure 3: Trajectory-Local Adaptive Retrieval. TLAR uses earlier verification results to decide whether retrieval should run and how many candidates to retrieve from the current history. The retrieved candidates are combined with the SmallDraft candidate and verified by the target model.

Finding 3: Retrieval success is concentrated at nearby positions. Finally, successful retrieval is not evenly distributed. In the code-debugging retrieval measurements, a hit is more likely after another hit than at an arbitrary position for all three models. Thus, recent verification results provide information about nearby positions. This motivates the running score and periodic probes in Section 4. Whether using this information saves enough verification work is a separate question, we tested in Section 5.4. Together, these findings motivate searching current history and adapting retrieval during generation. We combine the two sources under target verification in Section 4, then measure their added acceptance and cost separately in Section 5.

4 Method: Trajectory-Local Adaptive Retrieval

TLAR adds a controlled history search to SmallDraft, as we show in Figure 3. We describe the method in four parts. First, we find approximate context matches in generated history, rank them by recency, and place their continuations in a prefix-sharing tree (Section 4.1). Next, we track earlier verified retrieval results in a running score (Section 4.2). We use this score, the last retrieval result, and periodic probes to choose when to retrieve and how many candidates to keep (Section 4.3). Finally, we combine these candidates with SmallDraft, remove duplicate prefixes, and verify the resulting tree with the target model (Section 4.4).

4.1 Retrieving Candidates from the Current History

At position t, TLAR searches only the generated history $h_t = (y_1, \ldots, y_{t-1})$. This choice follows from Section 3, where the current history provides better candidates than the control sources in the settings where retrieval is useful. Let $\mathbf{u}_t = (y_{t-c}, \ldots, y_{t-1})$ be the most recent c tokens. For an earlier position j, let $\mathbf{u}_j = (y_{j-c}, \ldots, y_{j-1})$ be the context that appeared before y_j. TLAR identifies earlier positions whose contexts match the current context within a Hamming distance of ϵ and whose next d tokens are fully contained in the generated history:

$$\mathcal{J}_t = \{j : c + 1 \le j \le t - d,\ d_{\mathrm{Ham}}(\mathbf{u}_t, \mathbf{u}_j) \le \epsilon\}\,. \tag{1}$$

where d_{Ham} counts the token differences between two contexts. The value ϵ is the largest number of differences allowed. The condition $j \le t - d$ ensures that the continuation $(y_j, \ldots, y_{j+d-1})$ is already contained in h_t. TLAR cannot use current or future target tokens when constructing a candidate. For every $j \in \mathcal{J}_t$, TLAR takes the next d historical tokens $\mathbf{r}_j = (y_j, \ldots, y_{j+d-1})$ as a

candidate. Allowing up to ϵ context differences prevents small changes in punctuation, formatting, or wording from removing an otherwise useful candidate. Approximate matching is used only to find candidate locations. The retrieved tokens are still accepted using the exact verification rule defined in Section 2. TLAR ranks the matching positions by recency and selects the first k_t. Their continuations are inserted into a prefix-sharing tree. When two candidates begin with the same tokens, their shared prefix appears only once in the tree. The target model does not need to verify duplicate prefixes. Section 4.3 explains how TLAR chooses k_t.

4.2 Tracking Recent Retrieval Success

Let $m_t \in \{0, 1\}$ indicate whether retrieval is attempted at position t. When $m_t = 1$, let $z_t = \mathbb{I}[a_{\text{retr}}(t) \geq 1]$ indicate whether the target model accepts at least one retrieved token. TLAR stores these results in a running score $\hat{\rho}_t$, initialized as $\hat{\rho}_1 = 0$ and updated after verification:

$$\hat{\rho}_{t+1} = \begin{cases} (1-\alpha)\hat{\rho}_t + \alpha z_t, & m_t = 1, \\ (1-\alpha)\hat{\rho}_t, & m_t = 0. \end{cases} \tag{2}$$

When retrieval is attempted, the first case adds the latest result to the score. When retrieval is not attempted, the second case gradually reduces the effect of older results. A larger $\hat{\rho}_t$ means that retrieval has worked more often at recent positions. The score uses only verification results observed before position t. It does not require another trained predictor, an external trajectory, or additional target-model outputs. We set $\alpha = 1 - 2^{-1/H}$, where H controls how quickly earlier results lose weight. We use $H = 32$ by default and test other values in Section 5.4.

4.3 Choosing When and How Much to Retrieve

TLAR uses recent verification feedback to allocate retrieval work along the generated trajectory. The controller first requires $|h_t| > \tau$, allowing historical contexts and their continuations to accumulate. Once the history passes this threshold, retrieval activation combines a running success score, the latest observed retrieval outcome, and periodic probes. These signals capture sustained reuse, respond to newly observed success, and discover reuse as generation enters new regions. Let $q_t \in \{0, 1\}$ indicate a probe position, and let $s(t) < t$ denote the most recent retrieval attempt, with $z_{s(t)} = 0$ before the first attempt. The activation rule is $g_t = \mathbb{I}[|h_t| > \tau]\,\mathbb{I}[\hat{\rho}_t \geq \rho_{\min} \vee z_{s(t)} = 1 \vee q_t = 1]$. The running score sustains retrieval across regions with repeated success. The latest outcome allows a successful attempt to trigger retrieval at the next decision even while the running score remains below the threshold. During inactive periods, the score decays and periodic probes collect fresh verification feedback. A successful probe can reactivate retrieval as useful continuations become available.

For each active decision, TLAR also allocates a candidate width. At an active non-probe position, it sets $k_t = \text{clip}(\lceil k_{\max}\hat{\rho}_t \rceil, k_{\min}, k_{\max})$; at a probe position, it sets $k_t = k_{\min}$. This rule ties the search breadth to accumulated evidence of reuse: lower scores admit fewer historical continuations, while higher scores expand candidate coverage. The selected continuations enter a prefix-sharing tree, so their verification cost also depends on how many prefixes they share. Candidate width controls the proposed work, and the distinct-node count measures the resulting tree size. Activation and width form a feedback loop with target verification. Activation determines when to collect retrieval evidence; width determines how many continuations contribute candidates at that decision. After verification, the observed retrieval outcome updates the score used for subsequent decisions. Section 5.4 evaluates these components through activation and width ablations, probe removal, and parameter sensitivity experiments.

4.4 Combining TLAR with SmallDraft

At each position, SmallDraft produces the candidate $\mathbf{d}_t$ defined in Section 2. If $g_t = 1$, TLAR also constructs a candidate tree from the current history. If $g_t = 0$, the TLAR candidate set is empty. The SmallDraft candidate and the TLAR candidates are inserted into one prefix-sharing tree. Duplicate prefixes are merged before verification, and the target model verifies the resulting tree. After verification, TLAR uses the retrieval result to update $\hat{\rho}_{t+1}$. We evaluate the combined system using the recovery rate R, added accepted tokens Δ, and verification cost C defined in Section 2. When TLAR is inactive, we set $a_{\text{retr}}(t) = 0$ and $N_{\text{retr}}(t) = 0$. The evaluation measures the complete method, including positions where its controller decides not to retrieve.

5 EXPERIMENTS: UTILITY, SPECIFICITY, AND COMPLEMENTARITY

Section 3 tests where reusable tokens come from. We now ask whether they add value beyond SmallDraft (**RQ1**), whether they improve coverage alongside another retrieval source under the same verification budget (**RQ2**), and how the controller changes gain and verification cost (**RQ3**). These questions distinguish a useful source from a useful addition to an existing drafter.

5.1 MODEL AND EVALUATION SETTINGS

We evaluate Qwen3-8B, Mistral-24B, and Llama-3.1-8B-Instruct on 100 code-debugging, 24 mathematical-reasoning, and 24 open-ended-writing problems. SmallDraft is Qwen3-0.6B, Ministral-8B-Instruct-2410, and Llama-3.2-1B-Instruct, respectively. For each target, the offline comparisons use the same SmallDraft predictions and fixed target trajectories. Code debugging is the main setting; the other domains test whether the observed gains extend beyond it. Keeping each target trajectory fixed lets us compare candidate acceptance without changing the target tokens between methods. We do not compare one method's generated response with a different response from another method. The source controls and same-prompt tests in Section 3 answer a different question and are not counted as additional tests of SmallDraft complementarity.

We report additional exactly accepted tokens per position, Δ, over SmallDraft alone. We first fix the history gate and candidate width, then test the adaptive controller separately. For RQ2, we compare TLAR, the strongest single-source retrieval baseline, and their budgeted union under shared caps of $B \in \{4, 8\}$ distinct retrieval nodes per position. Union candidates are merged into one prefix-sharing tree, deduplicated, and pruned to the common node budget. Acceptance comes from offline replay, and node counts describe retrieval-side work. We report 95% bootstrap intervals where available. Appendix B records the settings and cost measures, while Appendix F reports the completed causal-boundary and event-driven replay audits. Appendix G separately evaluates the complete adaptive union tree under matched end-to-end serving conditions. We keep the three comparisons separate: RQ1 adds retrieval to SmallDraft, RQ2 compares single-source retrieval with a multi-source union under the same verification-node budget, and RQ3 changes TLAR's controller. Each uses its own matched reference. In particular, a larger accepted-token increment does not by itself show that its verification work is worthwhile.

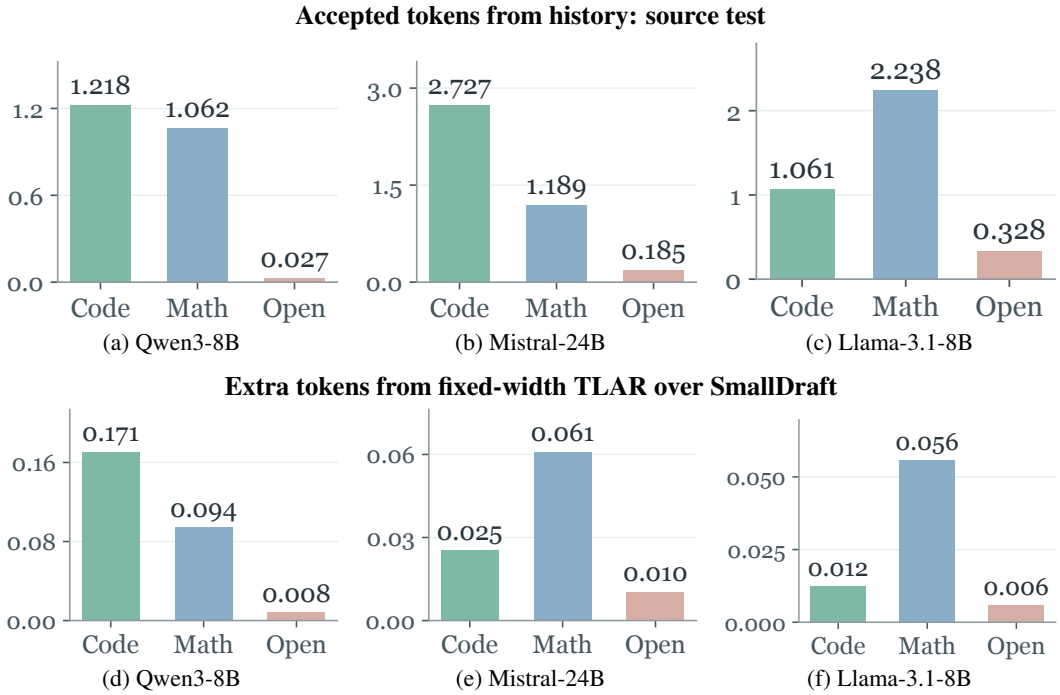
5.2 RQ1: DOES TRAJECTORY-LOCAL MEMORY COMPLEMENT SMALLDRAFT?

We first test whether retrieval supplies tokens that the same SmallDraft misses. We use 100 code-debugging tasks per model, a fixed 512-token history gate, and $k = d = 4$, without adaptive activation or width. We report added accepted length in Table 1; SmallDraft's own accepted length e is shown separately. We retain the full node-cost comparison in Appendix D.1. In Figure 4, we place the source measurements above TLAR's increment over SmallDraft for the same model and task. The top row asks how much the history supplies in the source test. The bottom row asks how much retrieval adds after SmallDraft's accepted tokens are already counted and its reference is SmallDraft alone.

Table 1: Added acceptance over SmallDraft on code debugging. Bold marks the union, which has the largest increment per model.

Method	Qwen	Mistral	Llama
SmallDraft (e)	2.383	3.148	2.823
SuffixDecoding (Δ)	0.128	0.026	0.013
STAND (Δ)	0.164	0.017	0.011
TLAR (fixed k; Δ)	0.171	0.025	0.018
STAND$\cup$TLAR (Δ)	**0.228**	**0.032**	**0.026**

TLAR adds accepted tokens for all three targets. Its single-method point estimate is highest for Qwen and Llama; SuffixDecoding is slightly higher for Mistral. The uncapped union in Table 1 has the largest accepted-token increment, motivating the matched-budget complementarity test in Section 5.3. The domain comparison shows why source reuse and added acceptance must be tested separately: Mistral gains more over SmallDraft in mathematics than in code, despite stronger source reuse in code. Open-ended increments remain small for all three targets. A useful history can overlap with what SmallDraft already predicts. The middle column makes the distinction concrete: Mistral has more accepted history tokens in code than in mathematics, but the added tokens over SmallDraft follow the opposite ordering. Thus, choosing where to use retrieval from source acceptance alone can be misleading. We use the matched SmallDraft comparison to measure that added value directly.



(a) Qwen3-8B (b) Mistral-24B (c) Llama-3.1-8B

(d) Qwen3-8B (e) Mistral-24B (f) Llama-3.1-8B

Figure 4: More reuse need not mean more added tokens. For Mistral (middle column), history supplies more accepted tokens in code, but retrieval adds more beyond SmallDraft in mathematics. Top: Within-source E_{exact} from Section 3. Bottom: fixed-gate, fixed-width TLAR's increment over the same SmallDraft, $\Delta = T^{-1} \sum_t [\max(a_{\text{small}}(t), a_{\text{retr}}(t)) - a_{\text{small}}(t)]$ on fixed target tokens. Code uses 100 tasks; Math and Open use 24 each. Each panel has its own scale.

5.3 RQ2: Does TLAR Improve Coverage Under a Matched Verification Budget?

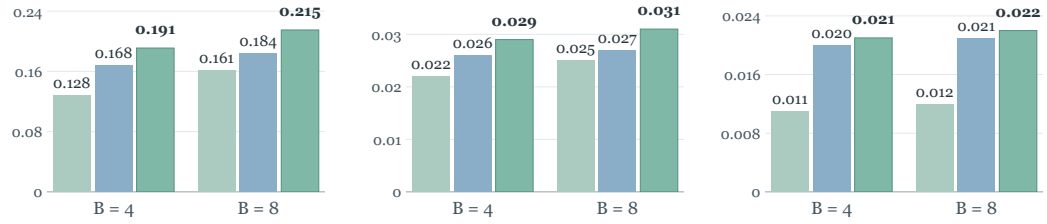
We next test whether trajectory-local candidates improve coverage when they compete with another retrieval source for the same verification budget. For each target, we compare TLAR, the strongest single-source baseline, and their budgeted union on code debugging. The strongest baseline is STAND for Qwen and Mistral and SuffixDecoding for Llama. All methods receive the same cap of $B \in \{4, 8\}$ distinct retrieval nodes per position. For the union, candidates from both sources enter one prefix-sharing tree. Duplicate prefixes are merged, and the tree is pruned to the shared node budget. Ranking uses source-local recency and verification outcomes observed at earlier decoding events. Candidates from both sources compete for the available tree nodes.

Figure 5 shows that the budgeted union achieves the largest accepted-token gain for all three targets at both budgets. At $B = 4$, its gains over the strongest single-source baseline are 13.7%, 11.5%, and 5.0% for Qwen, Mistral, and Llama, respectively. At $B = 8$, the corresponding gains are 16.8%, 14.8%, and 4.8%. Trajectory-local candidates improve the composition of the verification tree within the shared node cap. Appendix Table 18 provides the exact point estimates.

The relative improvement is largest for Qwen, followed by Mistral and Llama. This pattern is consistent with the same-prompt source comparison in Section 3: Qwen and Mistral favor their own trajectory, while Llama favors the other rollout. The source comparison and the budgeted union test connect trajectory-specific reuse to added acceptance in the hybrid drafter.

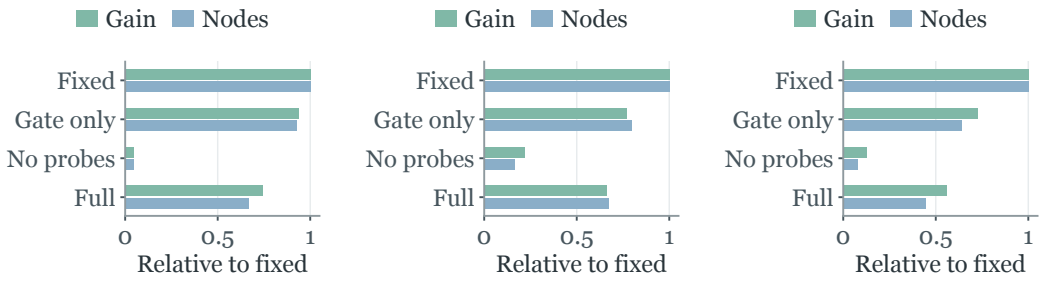
5.4 RQ3: How Does Adaptive Retrieval Change Gain and Cost?

We compare the full controller with a fixed gate and width, adaptive activation alone, and a controller without periodic probes, using the same replay and distinct-prefix node counter. We normalize both added acceptance over SmallDraft and nodes by the fixed configuration in Figure 6. Separately, we vary context length, mismatch tolerance, and candidate width over 27 settings per model in Figure 7. This grid measures recovery at SmallDraft first-token misses. For the controller comparison, both quantities equal one for the fixed configuration. A lower node value means less retrieval-side verification work; a lower gain value means fewer additional accepted tokens. Reading both is necessary: lower cost alone can also result from rarely retrieving. The grid instead holds the



(a) Qwen3-8B (b) Mistral-24B (c) Llama-3.1-8B

Figure 5: TLAR complements the strongest retrieval baseline under matched verification budgets on code debugging. In each budget group, the bars from left to right represent TLAR (light green), the strongest single-source retriever (blue), and their budgeted union (dark green). Bar heights show point estimates of additional accepted tokens per position over SmallDraft; labels give their values.



(a) Qwen3-8B (b) Mistral-24B (c) Llama-3.1-8B

Figure 6: Controller choices change both accepted gain and verification work. Gain is the increment over SmallDraft; nodes are distinct retrieval prefixes. Both are normalized by the fixed-gate, fixed-width replay, not the branch-count costs in Appendix D.1.

evaluation rule fixed and changes the candidates, so its highest-gain point is a setting of the same retrieval method. The controller reduces node cost but also gives up accepted tokens. Gain per node improves for Qwen and Llama. Removing probes lowers gain for every model, supporting their role in finding useful content after retrieval becomes inactive. Candidate settings offer a separate trade-off: the default uses fewer nodes than the highest-gain setting in each model's grid, but recovers fewer tokens. The grid changes available candidates; the controller changes when and how much retrieval is used. We provide exact values and sensitivity tests in Appendix B.

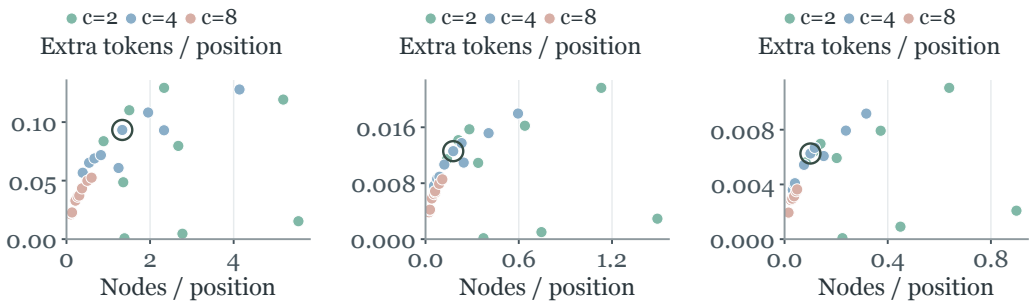
At matched verification-node costs, recent-hit feedback yields higher accepted-token increments than random or periodic activation across all three targets. We show further details in Appendix Table 23.

Table 2: Qwen/B200 throughput (tokens/s) across output lengths. Values are medians of five warmed runs; gains are relative to SmallDraft. At 512 tokens, retrieval is inactive and the difference captures controller/index overhead.

	Batch 1			Batch 32		
Tokens	SmallDraft	Union	Gain	SmallDraft	Union	Gain
512	151.80	150.89	−0.6%	2528.0	2510.3	−0.7%
1,024	143.20	155.09	+8.3%	2405.0	2587.8	+7.6%
2,048	132.60	146.52	+10.5%	2222.0	2422.0	+9.0%
4,096	119.40	134.21	+12.4%	1986.0	2200.5	+10.8%

End-to-end evaluation. We evaluate the complete adaptive union tree from Section 4 for Qwen3-8B/Qwen3-0.6B and Mistral-24B/Ministral-8B on one NVIDIA B200. End-to-end timing includes retrieval, tree construction, prefix deduplication, budgeted verification, controller updates, KV-cache updates, and sampling correction. Relative to the same draft-model baseline, throughput improves by 8.3% and 7.6% for Qwen and 4.1% and 3.5% for Mistral at batch sizes 1 and 32, respectively. Table 2 shows that Qwen's relative gains increase with output length, reaching 12.4% and 10.8% at 4,096 tokens. Longer generation provides more positions for active retrieval beyond the 512-token gate, helping amortize history-maintenance costs. Appendix G provides execution settings and repeated-run distributions.

From acceptance to execution efficiency. Throughput depends on both the tokens committed per decoding event and the elapsed time per event. Additional accepted tokens increase progress, while retrieval, tree construction, verification, and memory movement contribute to execution time. Appendix G.4 formalizes this relationship, calculates net time savings from the measured runtimes,



(a) Qwen3-8B (b) Mistral-24B (c) Llama-3.1-8B

Figure 7: Candidate settings change recovered gain and verification work. All 27 combinations of $c \in \{2, 4, 8\}$, $\epsilon \in \{0, 1, 2\}$, and $k \in \{2, 4, 8\}$ are shown. Color indicates context length; the circled point is $c = 4, \epsilon = 1, k = 4$.

and examines the implementation's resource demands and opportunities for overlap. It also develops a controlled component-evaluation framework linking candidate quality, execution cost, and serving performance, motivating retrieval policies that adapt to both recent reuse and system load.

6 Related Work

Model-based drafting. Speculative decoding uses a faster drafter to propose tokens and a target model to verify them (Leviathan et al., 2023; Chen et al., 2023). EAGLE and Medusa improve proposal quality through trained drafting components (Li et al., 2024b; Cai et al., 2024), while SpecInfer, EAGLE-2, Sequoia, and OPT-Tree improve tree construction and parallel verification (Miao et al., 2023; Li et al., 2024c; Chen et al., 2024; Wang et al., 2025a). These methods improve how model-generated candidates are produced or verified, but do not test whether the generated trajectory provides a distinct candidate source beyond the same draft model. TLAR addresses this question by measuring the additional accepted tokens supplied by trajectory retrieval after SmallDraft has already proposed its candidates.

Retrieval-based drafting. Retrieval methods obtain candidates from the prompt, external data, or generated text (Somasundaram et al., 2025; He et al., 2024; Fu et al., 2024; Hu et al., 2025; Oliaro et al., 2026). STAND exploits redundancy in reasoning trajectories (Song et al., 2025), and RASD combines retrieval and model-generated trees before verification (Quan et al., 2025). However, aggregate retrieval acceptance does not establish that useful tokens are specific to the current trajectory, and combining retrieval with a draft model does not show whether retrieval contributes candidates the drafter would otherwise miss. TLAR isolates trajectory-specific reuse through cross-trajectory, shuffled-history, and unigram controls, then measures its marginal contribution beyond SmallDraft and retrieval baselines under matched verification budgets.

Adaptive speculative computation. Prior work adapts candidate length, lookahead, or draft-tree structure using predicted acceptance or draft-model confidence (Huang et al., 2024; Mamou et al., 2024; Xiong et al., 2025; Wang et al., 2025a). These policies optimize model-generated speculation, but do not use the realized marginal value of trajectory retrieval to decide whether retrieval should run or how many retrieved candidates should be verified. TLAR uses only causally available outcomes from earlier target verification to control later retrieval activation and width. It connects source identification, marginal utility beyond an existing drafter, and feedback-based allocation of verification work in one evaluation.

7 Conclusion

Generated trajectories serve as runtime memory for speculative decoding. Source comparisons reveal trajectory-specific reuse and temporal locality, motivating TLAR's retrieval from generated history and adaptation through recent verification feedback. TLAR adds accepted tokens beyond a draft model and complements strong retrieval baselines under shared verification budgets. Its controller adjusts retrieval activation and candidate width to balance acceptance gains with verification cost. The complete adaptive union improves end-to-end throughput, with Qwen's relative gains increasing over the tested output lengths. These results connect trajectory reuse, candidate complementarity, and adaptive resource allocation to efficient inference.

Ethics Statement

TLAR accelerates language-model inference through reuse of generated text. Exact target-model verification preserves the target distribution, including its potential to produce biased, harmful, or inaccurate content. Deployments should retain appropriate safety safeguards as generation becomes faster. Generated trajectories may contain sensitive information; retrieval buffers should follow the access controls, retention policies, and deletion requirements applied to the original interaction. Our trajectory-local design keeps retrieval within the current generation. Resource use also warrants attention: throughput improvements can reduce execution time for a fixed workload, while energy consumption depends on hardware utilization and deployment scale. Responsible evaluation and release should document experimental conditions and protect sensitive data, credentials.

AI Use Statement

AI-assisted tools were used solely for grammar checking and presentation-quality review, including the identification of sentence fragments, dangling words, and formatting inconsistencies. The authors reviewed and approved every resulting edit and retain full responsibility for the manuscript.

Reproducibility Statement

The appendix documents model checkpoints, evaluation settings, retrieval configurations, verification budgets, controller parameters, and metric definitions. It also describes uncertainty calculations, causal-boundary and distribution-preservation audits, and the hardware, timing protocol, and repeated-run results for end-to-end evaluation. The accompanying anonymized code provides retrieval, candidate-tree construction, verification, adaptive control, and evaluation utilities.

References

Bradley Brown, Jordan Juravsky, Ryan Ehrlich, Ronald Clark, Quoc V Le, Christopher Ré, and Azalia Mirhoseini. Large language monkeys: Scaling inference compute with repeated sampling. *arXiv preprint arXiv:2407.21787*, 2024.

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. *arXiv preprint arXiv:2401.10774*, 2024.

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

Guanzheng Chen, Qilong Feng, Jinjie Ni, Xin Li, and Michael Qizhe Shieh. Rapid: Long-context inference with retrieval-augmented speculative decoding. *arXiv preprint arXiv:2502.20330*, 2025.

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

Mostafa Elhoushi, Akshat Shrivastava, Diana Liskovich, Basil Hosmer, Bram Wasti, Liangzhen Lai, Anas Mahmoud, Bilge Acun, Saurabh Agarwal, Ahmed Roman, et al. Layerskip: Enabling early exit inference and self-speculative decoding. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 12622–12642, 2024.

Angela Fan, Mike Lewis, and Yann Dauphin. Hierarchical neural story generation. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 889–898, 2018.

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

Aayush Gautam, Susav Shrestha, and Narasimha Reddy. Token-driven gammatune: Adaptive calibration for enhanced speculative decoding. *arXiv preprint arXiv:2504.00030*, 2025.

Zhenyu He, Zexuan Zhong, Tianle Cai, Jason 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, 2024.

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. Measuring mathematical problem solving with the math dataset. *arXiv preprint arXiv:2103.03874*, 2021.

Maximilian Holsman, Yukun Huang, and Bhuwan Dhingra. Fuzzy speculative decoding for a tunable accuracy-runtime tradeoff. In *Findings of the Association for Computational Linguistics: ACL 2025*, pp. 26257–26273, 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, 2025.

Kaixuan Huang, Xudong Guo, and Mengdi Wang. Specdec++: Boosting speculative decoding via adaptive candidate lengths. *arXiv preprint arXiv:2405.19715*, 2024.

Naman Jain, Alex Gu, Wen-Ding Li, Fanjia Yan, Tianjun Zhang, Sida Wang, Armando Solar-Lezama, Koushik Sen, and Ion Stoica. Livecodebench: Holistic and contamination free evaluation of large language models for code. In *International Conference on Learning Representations*, volume 2025, pp. 58791–58831, 2025.

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*, volume 2024, pp. 54107–54157, 2024.

Daniel Lee, Owen Queen, and James Zou. Reasonops: Operator segmentation for llm reasoning traces. *arXiv preprint arXiv:2605.29192*, 2026.

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

Minghan Li, Xilun Chen, Ari Holtzman, Beidi Chen, Jimmy Lin, Wen-tau Yih, and Xi V Lin. Nearest neighbor speculative decoding for llm generation and attribution. *Advances in Neural Information Processing Systems*, 37:80987–81015, 2024a.

Xingyi Li, Yadong Liu, Xiaojie Huang, Yiran Zhang, Shuai Wang, Shangguang Wang, Zhehao Lin, Yinben Xia, Chang Yu, Qihang Liu, Xuan Zhang, Hao Lu, Xiang Li, Zekun He, Yachen Wang, and Xianneng Zou. SwiftEP: Accelerating MoE inference with buffer fusion and TMA offloading. In *23rd USENIX Symposium on Networked Systems Design and Implementation (NSDI 26)*, pp. 1073–1089. USENIX Association, 2026a. URL `https://www.usenix.org/conference/nsdi26/presentation/li-xingyi`.

Yiwei Li, Jiayi Shi, Shaoxiong Feng, Peiwen Yuan, Xinglin Wang, Yueqi Zhang, Ji Zhang, Chuyi Tan, Boyuan Pan, Yao Hu, et al. Speculative decoding for multi-sample inference. In *EMNLP (Findings)*, pp. 12523–12533, 2025.

Yuhui Li, Fangyun Wei, Chao Zhang, and Hongyang Zhang. Eagle: Speculative sampling requires rethinking feature uncertainty. *arXiv preprint arXiv:2401.15077*, 2024b.

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, 2024c.

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

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

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, 2025.

Jonathan Mamou, Oren Pereg, Daniel Korat, Moshe Berchansky, Nadav Timor, Moshe Wasserblat, and Roy Schwartz. Dynamic speculation lookahead accelerates speculative decoding of large language models. *arXiv preprint arXiv:2405.04304*, 2024.

Xupeng Miao, Gabriele Oliaro, Zhihao Zhang, Xinhao Cheng, Zeyu Wang, Zhengxin Zhang, Rae Ying Yee Wong, Alan Zhu, Lijie Yang, Xiaoxiang Shi, et al. Specinfer: Accelerating generative large language model serving with tree-based speculative inference and verification. *arXiv preprint arXiv:2305.09781*, 2023.

Niklas Muennighoff, Zitong Yang, Weijia Shi, Xiang Lisa Li, Li Fei-Fei, Hannaneh Hajishirzi, Luke Zettlemoyer, Percy Liang, Emmanuel Candès, and Tatsunori B Hashimoto. s1: Simple test-time scaling. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 20286–20332, 2025.

Gabriele Oliaro, Zhihao Jia, Daniel Campos, and Aurick Qiao. Suffixdecoding: Extreme speculative decoding for emerging ai applications. *Advances in Neural Information Processing Systems*, 38: 126326–126354, 2026.

Rui Pan, Yinwei Dai, Zhihao Zhang, Gabriele Oliaro, Zhihao Jia, and Ravi Netravali. Specreason: Fast and accurate inference-time compute via speculative reasoning. *Advances in Neural Information Processing Systems*, 38:12730–12749, 2026.

Guofeng Quan, Wenfeng Feng, Chuzhan Hao, Guochao Jiang, Yuewei Zhang, and Hao Henry Wang. Rasd: Retrieval-augmented speculative decoding. In *Findings of the Association for Computational Linguistics: ACL 2025*, pp. 6167–6177, 2025.

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

Navid Rezazadeh and Arash Gholami Davoodi. The shape of overthinking: Backtracking bursts in long reasoning traces. *arXiv preprint arXiv:2605.27965*, 2026.

Junhan Shi, Yijia Zhu, Zhenning Shi, Dan Zhao, Qing Li, and Yong Jiang. SpecCoT: Accelerating chain-of-thought reasoning through speculative exploration. In *Findings of the Association for Computational Linguistics: EMNLP 2025*, pp. 24405–24415, November 2025.

Shwetha Somasundaram, Anirudh Phukan, and Apoorv Saxena. Pld+: Accelerating llm inference by leveraging language model artifacts. In *Findings of the Association for Computational Linguistics: NAACL 2025*, pp. 6090–6104, 2025.

Woomin Song, Saket Dingliwal, Sai Muralidhar Jayanthi, Bhavana Ganesh, Jinwoo Shin, Aram Galstyan, and Sravan Babu Bodapati. Accelerated test-time scaling with model-free speculative sampling. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 30611–30624, 2025.

Mitchell Stern, Noam Shazeer, and Jakob Uszkoreit. Blockwise parallel decoding for deep autoregressive models. *Advances in Neural Information Processing Systems*, 31, 2018.

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.

Jikai Wang, Juntao Li, Jianye Hou, Yan Bowen, Lijun Wu, and Min Zhang. Efficient reasoning for llms through speculative chain-of-thought. In *Findings of the Association for Computational Linguistics: ACL 2026*, pp. 1541–1553, 2026.

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc Le, Ed Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. Self-consistency improves chain of thought reasoning in language models. *arXiv preprint arXiv:2203.11171*, 2022.

Zhihai Wang, Jie Wang, Jilai Pan, Xilin Xia, Huiling Zhen, Mingxuan Yuan, Jianye Hao, and Feng Wu. Accelerating large language model reasoning via speculative search. *arXiv preprint arXiv:2505.02865*, 2025b.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35:24824–24837, 2022.

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. *Findings of the Association for Computational Linguistics: ACL 2024*, pp. 7655–7671, 2024.

Yunfan Xiong, Ruoyu Zhang, Yanzeng Li, and Lei Zou. Dyspec: Faster speculative decoding with dynamic token tree structure. *World Wide Web*, 28(3):36, 2025.

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.

Hongyi Yuan, Keming Lu, Fei Huang, Zheng Yuan, and Chang Zhou. Speculative contrastive decoding. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pp. 56–64, 2024.

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

Weilin Zhao, Yuxiang Huang, Xu Han, Wang Xu, Chaojun Xiao, Xinrong Zhang, Yewei Fang, Kaihuo Zhang, Zhiyuan Liu, and Maosong Sun. Ouroboros: Generating longer drafts phrase by phrase for faster speculative decoding. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pp. 13378–13393, 2024.

APPENDIX CONTENTS

A Related Work and Research Motivation . . . 15
A.1 Reasoning Length and Decoding Work . . . 15
A.2 Draft Construction and Exact Verification . . . 15
A.3 Text Sources as Draft Memory . . . 15
A.4 Feedback and the Allocation of Verification Work . . . 15
A.5 Trajectory Structure and the Role of TLAR . . . 16

B Evaluation Protocol, Metrics, and Implementation . . . 16
B.1 Models, Tasks, and Experimental Snapshots . . . 16
B.2 Retrieval Sources and Shared Implementations . . . 16
B.3 Acceptance, Recovery, and Verification Cost . . . 17
B.4 Uncertainty and Result Provenance . . . 17
B.5 Implementation Flow . . . 18

C Trajectory-Local Reuse and Source Controls . . . 18
C.1 Source Comparisons Across Models and Domains . . . 18
C.2 Repetition Audit and the Interpretation of Long Matches . . . 19
C.3 Same-Prompt, Different-Seed Pairing . . . 19
C.4 Temporal Dependence as a Control Signal . . . 20

D Candidate Complementarity and Verification Budgets . . . 20
D.1 Acceptance Beyond SmallDraft . . . 20
D.2 Domain Variation and Draft-Miss Opportunities . . . 21
D.3 Complementarity Under Shared Node Caps . . . 22
D.4 Supplementary Candidate-Coverage Analysis . . . 23

E Adaptive Retrieval and Verification Cost . . . 23
E.1 Candidate Settings and the Gain–Cost Frontier . . . 23
E.2 Activation, Width, and Periodic Probes . . . 24
E.3 Feedback Aligned with Incremental Acceptance . . . 24
E.4 Context Matching Under the Same Controller . . . 26

F Causal Construction and Exact Verification . . . 26
F.1 Historical Candidate Construction . . . 26
F.2 Exact Tree Verification and Distribution Preservation . . . 27
F.3 Event-Driven Replay and Controller Updates . . . 27

G End-to-End Execution and Length Scaling . . . 29
G.1 Execution Configuration and Target–Draft Pairs . . . 29
G.2 Throughput of the Complete Union Tree . . . 29
G.3 Repeated-Run Variation and Paired Summaries . . . 29
G.4 From Candidate Acceptance to Execution Efficiency . . . 30
G.5 Archived Model-Based Drafting Reference . . . 32

H Discussion: Trajectory Memory as an Inference Resource . . . 32
H.1 Generation Creates a Memory Whose Value Depends on the Drafter . . . 32
H.2 A Verification Budget Makes Candidate Composition Central . . . 32
H.3 Feedback Should Track the Value Added to the Existing System . . . 32
H.4 Exact Verification Supports Flexible Memory Search . . . 33
H.5 Longer Reasoning Changes Both the Memory and Its Economics . . . 33
H.6 An Evaluation Chain for Memory-Based Inference . . . 33

A Related Work and Research Motivation

Section 6 places TLAR among speculative decoding methods. We develop that connection through the sequence that motivates the method: longer reasoning creates a decoding workload, speculative verification accelerates token generation, retrieval supplies candidate continuations, and feedback allocates verification work.

A.1 Reasoning Length and Decoding Work

Chain-of-thought prompting elicits intermediate reasoning steps (Wei et al., 2022). Self-consistency and repeated sampling allocate computation across reasoning paths (Wang et al., 2022; Brown et al., 2024), while test-time scaling extends the amount of reasoning used to produce an answer (Muennighoff et al., 2025). These approaches make the execution cost of long trajectories a central inference problem. Acceleration can act at several levels. SpecReason delegates selected reasoning steps using semantic acceptance (Pan et al., 2026). Speculative search and speculative chain-of-thought methods organize parallel reasoning exploration (Wang et al., 2025b; 2026; Shi et al., 2025); multi-sample speculative decoding shares work across sampled outputs (Li et al., 2025). TLAR acts at the token level, retrieving continuations and applying exact target verification. This choice connects reasoning trajectories to lossless decoding.

A.2 Draft Construction and Exact Verification

Blockwise decoding proposes several future tokens and retains a verified prefix (Stern et al., 2018). Speculative sampling combines a cheaper drafter with parallel target verification and a sampling correction that preserves the target distribution (Leviathan et al., 2023; Chen et al., 2023). Surveys organize the resulting design space around drafting and verification (Xia et al., 2024).

Medusa introduces prediction heads (Cai et al., 2024); EAGLE and its successors use feature prediction and dynamic candidate trees (Li et al., 2024b;c; 2026b). SpecInfer verifies candidate trees (Miao et al., 2023), and Sequoia and OPT-Tree optimize tree structure for acceptance and hardware cost (Chen et al., 2024; Wang et al., 2025a). Ouroboros extends drafts phrase by phrase (Zhao et al., 2024); self-speculation, early exit, and layer skipping reuse the target model's computation (Zhang et al., 2024; Elhoushi et al., 2024). Lookahead Decoding builds candidates through parallel Jacobi-style iteration (Fu et al., 2024). TLAR supplies an additional candidate source to this draft–verify architecture: continuations already present in the generated trajectory.

A.3 Text Sources as Draft Memory

LLMA copies and verifies reference spans (Yang et al., 2023). REST retrieves from an external datastore (He et al., 2024), PLD+ exploits repeated prompt and generation text (Somasundaram et al., 2025), RAPID combines retrieval with long-context speculation (Chen et al., 2025), and NEST uses nearest-neighbor text for generation and attribution (Li et al., 2024a). Generated and cached sequences provide another memory source: SuffixDecoding uses suffix-tree statistics (Oliaro et al., 2026), SAM-Decoding uses suffix automata (Hu et al., 2025), Token Recycling reuses candidate transitions (Luo et al., 2025), and STAND uses stochastic n-gram statistics from reasoning trajectories (Song et al., 2025). These systems motivate a source-level question: which continuations become useful because they belong to the trajectory currently being generated? Section 3 addresses this question with cross-trajectory, shuffled-history, unigram, and same-prompt comparisons. Sections 5.2 and 5.3 then measure the extra acceptance supplied by that source alongside existing drafters.

A.4 Feedback and the Allocation of Verification Work

The benefit of a candidate depends on its acceptance and its verification cost. SpecDec++ predicts acceptance (Huang et al., 2024); DISCO adapts lookahead using draft-model signals (Mamou et al., 2024); EAGLE-2 and DySpec grow trees using draft confidence or distributional information (Li et al., 2024c; Xiong et al., 2025). OPT-Tree and Sequoia optimize expected acceptance and tree cost (Wang et al., 2025a; Chen et al., 2024), while token-driven calibration adapts speculation during execution (Gautam et al., 2025).

TLAR uses completed target-verification events to control retrieval activation and width. Periodic probes refresh the signal after retrieval becomes inactive. The controller ablations in Appendix E connect this feedback to incremental acceptance, and Appendix F specifies how the feedback is generated along the executed decoding path.

A.5 Trajectory Structure and the Role of TLAR

ReasonOps identifies recurring reasoning operations (Lee et al., 2026), and analyses of overthinking describe clustered backtracking (Rezazadeh & Davoodi, 2026). These observations motivate studying structure within a trajectory. TLAR measures that structure through context matches and exactly accepted continuations. Its verifier preserves the target distribution; speculative contrastive decoding and fuzzy speculation explore altered decoding objectives and accuracy–runtime trade-offs (Yuan et al., 2024; Holsman et al., 2025).

RASD provides a closely related hybrid architecture: it prunes retrieval candidates using draft probabilities and merges retrieval and model-generated trees (Quan et al., 2025). Building on retrieval and tree verification, TLAR connects source identification, added acceptance, and feedback-driven resource allocation. Table 3 summarizes how candidate sources and control signals fit together. The remaining appendices follow that experimental chain.

Table 3: Candidate sources and control mechanisms in closely related systems. Feedback control uses earlier target-verification outcomes to change later retrieval work. Dashes indicate absent components.

Method	Candidate source	Draft-tree fusion	Exact verification	Feedback control	Selection principle
STAND	reasoning trajectories	—	yes	—	stochastic n-gram sampling
SuffixDecoding	cached/generated suffixes	—	yes	tree structure	suffix statistics
RASD	retrieval database	yes	yes	—	probability pruning and tree fusion
TLAR	current trajectory	yes	yes	yes	source isolation and causal budget allocation

B Evaluation Protocol, Metrics, and Implementation

This section supplies the common setup for Section 5.1. We first define the tasks and candidate sources, then the acceptance and cost counters, and finally the estimation and implementation procedures. These definitions connect each result table to its reference drafter and evaluated positions.

B.1 Models, Tasks, and Experimental Snapshots

The main suite contains 100 code-debugging, 24 mathematical-reasoning, and 24 open-ended-writing problems per target. Target generation uses temperature 0.6, top-p $= 0.95$, and a maximum of 4,096 new tokens. Within each target, retrieval comparisons share fixed trajectories and SmallDraft predictions. Table 4 gives the target–draft pairs.

Table 4: Target–draft pairs for the offline comparisons and the B200 execution analysis.

Target	SmallDraft checkpoint
Qwen3-8B	`Qwen/Qwen3-0.6B`
Mistral-24B	`mistralai/Ministral-8B-Instruct-2410`
Llama-3.1-8B	`meta-llama/Llama-3.2-1B-Instruct`

Code prompts contain SWE-bench Verified issue descriptions and repository metadata (Jimenez et al., 2024); gold patches and hidden tests remain in the evaluation data. The MATH test problems span algebra, counting/probability, geometry, and number theory (Hendrycks et al., 2021). WritingPrompts supplies the open-ended prompts, with reference stories retained as evaluation data (Fan et al., 2018). Examples are sorted by canonical identifier and sampled with seed 20260217 before retrieval evaluation. The same-prompt comparison uses two generation seeds for each of 24 code-debugging problems.

Table 5 records the two experimental snapshots. The historical source audit includes DeepSeek and five writing prompts; the main suite includes Mistral and 24 writing prompts. Their roles are source analysis and the expanded utility evaluation, respectively.

B.2 Retrieval Sources and Shared Implementations

PLD searches the current prompt. The REST proxy indexes the other prompts in the trace pool, separated by sentinel tokens; a 100-trace pool supplies 99 external prompt entries for each query

Table 5: Experimental snapshots and their roles. Code, Math, and Open give problem counts per target.

Snapshot	Models	Code	Math	Open	Role
Legacy source audit	Qwen, DeepSeek, Llama	100	24	5	nine-cell source diagnostic in Appendix C
Expanded main suite	Qwen, Mistral, Llama	100	24	24	RQ1–RQ3 and B200 evaluation

trajectory. The SAM proxy combines exact history retrieval with this prompt datastore. STAND-style and SuffixDecoding candidates use the common ranking, tree, and verification interfaces summarized in Table 6. This shared implementation supports comparisons of candidate sources under the same evaluation rules.

Table 6: Retrieval sources and components implemented in the common evaluation harness.

Displayed label	Search source	Implemented components	Evaluation role
PLD	current prompt	exact n-gram lookup, $k = 4$, $d = 4$	prompt-source candidate comparison
REST proxy	other prompts	external prompt index, exact context, sentinel boundaries	shared-harness prompt-datastore proxy
SAM proxy	history and other prompts	exact suffix lookup and union in the shared tree	shared-harness history/datastore proxy
STAND-style	generated history	stochastic n-gram candidates, common gate and budget	shared-harness STAND-style candidates
SuffixDecoding	generated/cache suffixes	exact suffix candidates and prefix tree	common verifier and budgets; offline accepted-token comparison

B.3 Acceptance, Recovery, and Verification Cost

Following Section 2, let $M_t = \mathbf{1}[a_{\text{small}}(t) = 0]$, let g_t indicate an active history gate, and let $a_{\text{retr}}(t)$ be the longest exactly accepted retrieval continuation. Let N_t count retrieval work under the counter specified for an experiment. The first-token-miss analysis uses

$$A_{\text{miss}} = \frac{\sum_t M_t g_t}{\sum_t M_t}, \qquad R = \frac{\sum_t M_t g_t \mathbf{1}[a_{\text{retr}}(t) \geq 1]}{\sum_t M_t g_t},$$

$$G_{\text{miss}} = \frac{1}{T} \sum_t M_t g_t a_{\text{retr}}(t), \qquad C_{\text{miss}} = \frac{1}{T} \sum_t M_t g_t N_t.$$

Here A_{miss} is the active fraction of draft failures, recorded as `active_rate_over_miss`. Recovery R uses active failures as its denominator; G_{miss} and C_{miss} average over all evaluated positions. Their ratio measures recovered tokens per verification node.

The hybrid comparison also credits retrieval for extending an accepted SmallDraft prefix. Its acceptance and cost are

$$\Delta = \frac{1}{T} \sum_t \left[\max\{a_{\text{small}}(t), g_t a_{\text{retr}}(t)\} - a_{\text{small}}(t)\right], \qquad C_{\text{hyb}} = \frac{1}{T} \sum_t g_t N_t.$$

This definition connects candidate usefulness directly to the existing drafter. The fixed-candidate hybrid and static grid count summed branch lengths, including shared prefixes. The adaptive replay, baseline-addition analysis, and budgeted union count distinct prefix nodes. Each gain–cost comparison uses its experiment's counter. The budgeted comparison gives every method the same maximum of B distinct retrieval nodes per position.

B.4 Uncertainty and Result Provenance

Trajectory-cluster bootstrap resamples trajectory identifiers with replacement, retains all positions from each sampled trajectory, and recomputes the pooled metric. We use 2,000 resamples and report the 2.5th and 97.5th percentiles. This procedure preserves dependence among positions within a trajectory. Same-prompt source comparisons pair the two rollouts by problem identifier.

The tables report experimental measurements. Dagger-marked entries in Table 14 identify updated experimental summaries. The reported acceptance intervals and node-cost summaries accompany the corresponding configurations. Table 18 reports point estimates. Appendix G gives the B200 measurements and repeated-run summaries, including the calculation of paired throughput intervals.

B.5 Implementation Flow

The scripts in Table 7 follow the evaluation pipeline: construct prompts, generate trajectories, record draft predictions, compare candidate sources, and aggregate acceptance and cost. Context matching selects historical locations; the `accepted_len` routine checks continuation tokens exactly until the first mismatch or depth cap. Appendix F connects this replay computation to incremental history insertion and executed decoding events.

Table 7: Evaluation scripts and their roles in the data-to-metric pipeline.

Script	Role
`build_code_debug_pool.py`	Constructs the code-debugging prompt pool.
`generate_traces.py`	Generates target-model trajectories under the decoding configuration above.
`analyze_draft_hits.py`	Teacher-forces the small draft model over each trajectory; records $\text{hit}_{\text{draft}}(t)$.
`analyze_selfcopy.py`	Computes within/cross/shuffled/unigram E_{exact} (Section 3).
`analyze_tree_hybrid.py`	Computes TLAR and hybrid (TLAR + SmallDraft) accepted lengths per position; evaluates individual retrieval trees.
`compare_retrieval_baselines.py`	Shared retrieval-baseline harness; source of the static grid and union hybrid.
`bootstrap_metrics.py`	Trajectory-cluster bootstrap over the outputs of the scripts above.
`summarize_analysis.py`	Computes development-time aggregate summaries.

C Trajectory-Local Reuse and Source Controls

Section 3 asks where reusable continuations originate. We develop the evidence from source comparisons to repetition auditing, same-prompt pairing, and temporal dependence. Each step explains a design choice in TLAR: the memory source, the interpretation of accepted spans, and the feedback signal.

C.1 Source Comparisons Across Models and Domains

Table 8 supplies the Qwen, Mistral, and Llama values plotted in Figure 1. Within searches the generated prefix; Cross uses another trajectory with the same available history length; Shuffled reorders history blocks; Unigram uses token frequencies. Within, Cross, and Shuffled share a matching rule, making the source the experimental variable.

Table 8: Source means underlying Figure 1. Entries are exactly accepted tokens; Unigram is the strongest control in each row.

Task	Model	Within	Cross	Shuffled	Unigram
Code debugging	Qwen3-8B	1.2184	0.0139	0.0161	0.0658
Code debugging	Mistral-24B	2.7266	0.0069	0.0326	0.0585
Code debugging	Llama-3.1-8B	1.0612	0.0046	0.0236	0.0539
Mathematics	Qwen3-8B	1.0620	0.0037	0.0155	0.0627
Mathematics	Mistral-24B	1.1888	0.0039	0.0268	0.0713
Mathematics	Llama-3.1-8B	2.2378	0.0141	0.0539	0.0761
Open-ended	Qwen3-8B	0.0274	0.0016	0.0003	0.0535
Open-ended	Mistral-24B	0.1849	0.0003	0.0011	0.0435
Open-ended	Llama-3.1-8B	0.3283	0.0020	0.0121	0.0561

Current history yields higher accepted lengths than the controls in code and mathematics. Open-ended writing has smaller accepted lengths and a different ordering for Qwen: Within is 0.0274 and Unigram is 0.0535. The historical Qwen/DeepSeek/Llama snapshot in Table 9 adds the archived intervals and within-to-control ratios. Unigram is the strongest control in all nine cells, and the ratios range from $0.5\times$ to $29.4\times$.

Table 9: Historical source snapshot: mean exactly accepted length and archived 95% bootstrap intervals. Lift is Within divided by Unigram. The measurement cap is 32 tokens. † marks the cleaned Llama mathematics cell after removal of seven flagged trajectories.

Model	Within	Cross	Shuffled	Unigram	Lift
Code-debugging					
Qwen3-8B	1.2184 [1.1543, 1.2844]	0.0139 [0.0113, 0.0165]	0.0161 [0.0139, 0.0187]	0.0658 [0.0631, 0.0686]	18.5×
DeepSeek-R1-Qwen-7B	0.6599 [0.6038, 0.7146]	0.0171 [0.0147, 0.0197]	0.0109 [0.0092, 0.0128]	0.0756 [0.0733, 0.0780]	8.7×
Llama-3.1-8B	1.0612 [0.8361, 1.3284]	0.0046 [0.0022, 0.0080]	0.0236 [0.0135, 0.0351]	0.0539 [0.0511, 0.0568]	19.7×
Mathematical reasoning					
Qwen3-8B	1.0620 [0.9294, 1.2131]	0.0037 [0.0028, 0.0047]	0.0155 [0.0119, 0.0196]	0.0627 [0.0574, 0.0687]	16.9×
DeepSeek-R1-Qwen-7B	0.8183 [0.6486, 1.0127]	0.0025 [0.0018, 0.0032]	0.0142 [0.0106, 0.0187]	0.0742 [0.0670, 0.0837]	11.0×
Llama-3.1-8B†	2.2378 [1.5816, 2.7283]	0.0141 [0.0062, 0.0222]	0.0539 [0.0187, 0.1165]	0.0761 [0.0616, 0.0898]	29.4×
Open-ended writing					
Qwen3-8B	0.0274 [0.0189, 0.0405]	0.0016 [0.0005, 0.0027]	0.0003 [0.0001, 0.0006]	0.0535 [0.0471, 0.0592]	0.5×
DeepSeek-R1-Qwen-7B	0.1170 [0.0519, 0.1713]	0.0084 [0.0021, 0.0169]	0.0007 [0.0000, 0.0015]	0.0521 [0.0464, 0.0596]	2.2×
Llama-3.1-8B	0.3283 [0.0675, 0.7089]	0.0020 [0.0000, 0.0042]	0.0121 [0.0025, 0.0204]	0.0561 [0.0480, 0.0682]	5.9×

C.2 Repetition Audit and the Interpretation of Long Matches

Long accepted spans can arise from repeated output loops. To identify this source of acceptance, the historical audit ranks trajectories by E_{exact} and inspects the ten highest-ranked trajectories per cell, or all five writing trajectories. A trajectory is flagged when its mean is at least five tokens and its maximum match reaches the 32-token measurement cap. A cell is classified as repetition-dominated when over half of the inspected trajectories meet this rule.

Table 10: Historical repetition audit. Counts describe the inspected high-repetition trajectories in each model–domain cell.

Domain	Model	Inspected	Flagged
code-debug100	DeepSeek-R1-Qwen-7B	10	0
code-debug100	Llama-3.1-8B	10	0
code-debug100	Qwen3-8B	10	0
math	DeepSeek-R1-Qwen-7B	10	0
math	Llama-3.1-8B	10	7
math	Qwen3-8B	10	0
open-ended	DeepSeek-R1-Qwen-7B	5	0
open-ended	Llama-3.1-8B	5	0
open-ended	Qwen3-8B	5	0

The audit flags seven of the ten inspected Llama mathematics trajectories and zero in each other inspected cell. Table 11 gives the task-level values. For example, task 22 repeats the same four-sentence geometric argument more than a dozen times, ending with "$\angle ABC = 180° - y - 90° = 90° - y$". This repeated passage produces long exact matches and illustrates why repetition structure is relevant to interpreting retrieval acceptance.

Removing the seven flagged trajectories changes the Llama mathematics source mean from 7.03 to 2.2378. Figure 1 and Tables 8 and 9 use the cleaned mean. The 32-token source-measurement cap quantifies reuse; TLAR's depth-four candidate tree evaluates how that reuse supplies a practical draft.

C.3 Same-Prompt, Different-Seed Pairing

The next comparison holds the problem fixed and changes the sampled trajectory. Each target has 24 paired problems. Qwen, Mistral and Llama use exact context matching and an alternative-history

Table 11: The ten highest-repetition Llama mathematics trajectories. Flagged trajectories are removed when computing the cleaned source mean.

Rank	Task ID	Within E_{exact}	Flagged
1	3	16.28	yes
2	22	16.24	yes
3	7	11.67	yes
4	16	8.87	yes
5	18	6.77	yes
6	4	5.89	yes
7	6	5.61	yes
8	8	3.91	no
9	10	3.35	no
10	2	2.61	no

prefix of the same available length. Each row in Table 12 compares own and other-seed history under that target's protocol, as plotted in Figure 2.

Table 12: Same-prompt source comparisons on 24 paired code-debugging problems per model. Brackets give archived 95% intervals; the alternative-source column specifies the matching protocol.

Model	Alternative source	Within	Other seed
Qwen3-8B	Prefix; exact context	2.0814 [1.4340, 2.9241]	1.2653 [0.9282, 1.6487]
Mistral-24B	Prefix; exact context	1.4354 [1.1416, 1.7132]	0.8614 [0.6644, 1.1190]
Llama-3.1-8B	Prefix; exact context	0.8243 [0.3755, 1.4092]	1.1134 [0.6498, 1.6673]

Qwen and Mistral have higher own-history means, while Llama has a higher other-seed mean. Mistral's paired difference is 0.5740 with interval $[0.1060, 1.0065]$; the table gives the marginal intervals for each source. These within-model contrasts identify how trajectory identity affects candidate availability and motivate the downstream complementarity tests in Appendix D.

C.4 Temporal Dependence as a Control Signal

On code debugging, next-position hit probabilities after a hit are 0.76, 0.86, and 0.81 for Qwen, Mistral, and Llama. The overall probabilities on the same position pairs are 0.46, 0.50, and 0.38. A recent successful retrieval provides information about the next position. Equation 2 compresses this information into a running score, and the probes in Section 4.3 refresh that score as generation moves between regions. Appendix E measures the resulting gain–cost trade-off.

D Candidate Complementarity and Verification Budgets

The source tests establish candidate availability. The next question is how those candidates extend an existing drafter. We follow RQ1 with the SmallDraft hybrid comparison, examine domain and draft-miss effects, and then follow RQ2 with a shared-budget comparison against a strong retrieval source.

D.1 Acceptance Beyond SmallDraft

Table 13 expands Table 1 with verification cost. The fixed configuration uses a 512-token history gate and $k = d = 4$. The union has the largest increment for each target, and its summed branch cost also increases. Table 14 gives the corresponding intervals and identifies updated experimental summaries using the convention in Appendix B.4.

The fixed-candidate TLAR increments are 0.1709, 0.0253, and 0.0180 for Qwen, Mistral, and Llama. Their meaning is incremental: they credit tokens that extend the same SmallDraft's accepted prefix. The union's additional acceptance motivates testing how both retrieval sources compete for a common node budget below.

Table 13: Fixed-candidate hybrid acceptance and cost on 100 code-debugging tasks per target, with a 512-token gate and $k = d = 4$. e is SmallDraft acceptance and Δ its hybrid increment. Nodes sum branch lengths, including shared prefixes; the union sums both branch costs. Table 14 gives intervals and revision markers.

	Qwen3-8B		Mistral-24B		Llama-3.1-8B	
Added candidates	Δ	nodes	Δ	nodes	Δ	nodes
SmallDraft alone	$e = 2.383$		$e = 3.148$		$e = 2.823$	
SuffixDecoding	0.128	3.38	0.026	3.27	0.013	1.34
STAND	0.164	4.76	0.017	2.11	0.011	0.86
TLAR (fixed k)	0.171	7.21	0.025	4.10	0.018	1.42
STAND∪TLAR	0.228	11.98	0.032	6.22	0.026	2.28

Table 14: Full fixed-candidate hybrid measurements. Brackets give 95% bootstrap intervals. Dagger-marked cells identify updated experimental summaries, as described in Appendix B.4.

Model	Added retriever	Δ	Nodes / position
Qwen3-8B	SuffixDecoding	0.1278 [0.1182, 0.1381]	3.3840 [3.1768, 3.5937]
Qwen3-8B	STAND	$0.1640^{\dagger}$ [0.1529, 0.1755]	$4.7635^{\dagger}$ [4.6361, 4.8879]
Qwen3-8B	TLAR	0.1709 [0.1606, 0.1809]	7.2146 [6.9291, 7.4923]
Qwen3-8B	STAND∪TLAR	0.2280 [0.2141, 0.2422]	11.9781 [11.5626, 12.3594]
Mistral-24B	SuffixDecoding	0.0262 [0.0214, 0.0312]	3.2738 [2.5212, 4.1417]
Mistral-24B	STAND	$0.0170^{\dagger}$ [0.0109, 0.0233]	$2.1141^{\dagger}$ [1.7659, 2.4891]
Mistral-24B	TLAR	0.0253 [0.0189, 0.0324]	4.1033 [3.1826, 5.0426]
Mistral-24B	STAND∪TLAR	0.0325 [0.0249, 0.0403]	6.2174 [4.9751, 7.4900]
Llama-3.1-8B	SuffixDecoding	$0.0130^{\dagger}$ [0.0090, 0.0173]	$1.3422^{\dagger}$ [1.1674, 1.5177]
Llama-3.1-8B	STAND	$0.0110^{\dagger}$ [0.0078, 0.0143]	$0.8609^{\dagger}$ [0.7485, 0.9717]
Llama-3.1-8B	TLAR	$0.0180^{\dagger}$ [0.0151, 0.0211]	$1.4209^{\dagger}$ [1.2036, 1.6435]
Llama-3.1-8B	STAND∪TLAR	$0.0260^{\dagger}$ [0.0225, 0.0298]	$2.2817^{\dagger}$ [1.9761, 2.6201]

D.2 Domain Variation and Draft-Miss Opportunities

Table 15 extends RQ1 to mathematics and writing. Recovery measures success at active first-token misses; G_{miss} averages that recovery over all positions; Δ also credits extensions of accepted draft prefixes. For Mistral, source acceptance is higher in code (2.7266) than mathematics (1.1888), while the hybrid increment is higher in mathematics (0.0610) than code (0.0253). This reversal connects source reuse to the coverage already supplied by SmallDraft, as illustrated in Figure 4.

Draft failures reveal one route through which retrieval adds value. Table 16 reports their frequency and recovery before and after history gating. Mistral has 11,401 first-token misses over 110,159 positions; 0.4514 of those misses pass the gate. Its recovery rises from 0.0946 over all misses to 0.1069 over gated misses. The gate changes the evaluated set by selecting positions with enough accumulated history.

Expanded cross-domain evaluation. We extend the mathematics and open-ended writing evaluation to 100 problems per domain and target. This expanded evaluation supports the cross-domain findings in RQ1; the full recovery and acceptance breakdown appears here to connect those findings to draft-miss opportunities. Table 17 preserves the domain ordering observed in the original evaluation: all three targets obtain larger acceptance increments in mathematics than in open-ended writing. Within mathematics, Llama achieves the highest recovery rate, while Qwen obtains the largest increment over SmallDraft. Recovery measures retrieval success at active first-token misses.

Table 15: Mathematics and writing results on 24 problems per domain and target. Brackets give archived 95% bootstrap intervals. Metric definitions appear in Appendix B.3.

Domain	Model	Recovery	G_{miss}	Δ
math	Qwen3-8B	0.1645 [0.1536, 0.1771]	0.0521 [0.0454, 0.0600]	0.0942 [0.0817, 0.1095]
math	Mistral-24B	0.1213 [0.1107, 0.1335]	0.0346 [0.0292, 0.0392]	0.0610 [0.0516, 0.0691]
math	Llama-3.1-8B	0.2718 [0.2336, 0.3056]	0.0271 [0.0220, 0.0317]	0.0557 [0.0445, 0.0657]
open-ended	Qwen3-8B	0.0141 [0.0094, 0.0209]	0.0060 [0.0047, 0.0074]	0.0082 [0.0057, 0.0112]
open-ended	Mistral-24B	0.0380 [0.0282, 0.0490]	0.0060 [0.0040, 0.0082]	0.0104 [0.0066, 0.0145]
open-ended	Llama-3.1-8B	0.0235 [0.0105, 0.0439]	0.0032 [0.0013, 0.0048]	0.0057 [0.0021, 0.0088]

Table 16: SmallDraft first-token misses and retrieval recovery on code debugging. Brackets give 95% trajectory-cluster bootstrap intervals.

Model	Draft miss fraction	Ungated recovery	Gated recovery
Qwen3-8B	0.207 [0.203, 0.212]	0.193 [0.184, 0.202]	0.222 [0.212, 0.232]
Mistral-24B	0.1035 [0.0991, 0.1082]	0.0946 [0.0789, 0.1121]	0.1069 [0.0898, 0.1313]
Llama-3.1-8B	0.145 [0.139, 0.152]	0.051 [0.045, 0.058]	0.089 [0.073, 0.106]

Its contribution across the trajectory also depends on the frequency of these opportunities and the accepted continuation length, while Δ additionally credits extensions of accepted draft prefixes. The expanded results reinforce the role of the existing drafter's coverage in determining the value of trajectory-local retrieval.

Table 17: Expanded cross-domain evaluation on 100 problems per domain and target. Cells report point estimates with intervals in brackets. Recovery measures success at active draft misses; G_{miss} measures recovered tokens per position; Δ measures additional acceptance over SmallDraft. Metric definitions appear in Appendix B.3.

Domain	Model	Recovery	G_{miss}	Δ
Math	Qwen3-8B	0.1682 [0.1628, 0.1741]	0.0534 [0.0500, 0.0572]	0.0971 [0.0907, 0.1038]
	Mistral-24B	0.1187 [0.1134, 0.1248]	0.0338 [0.0313, 0.0364]	0.0594 [0.0549, 0.0636]
	Llama-3.1-8B	0.2651 [0.2476, 0.2823]	0.0264 [0.0239, 0.0288]	0.0542 [0.0489, 0.0595]
Open-ended	Qwen3-8B	0.0153 [0.0127, 0.0184]	0.0063 [0.0056, 0.0070]	0.0087 [0.0073, 0.0102]
	Mistral-24B	0.0356 [0.0309, 0.0408]	0.0057 [0.0048, 0.0068]	0.0098 [0.0079, 0.0118]
	Llama-3.1-8B	0.0251 [0.0184, 0.0329]	0.0034 [0.0025, 0.0043]	0.0061 [0.0043, 0.0079]

D.3 Complementarity Under Shared Node Caps

Table 18 supplies the point estimates for Figure 5. The strongest single source is STAND for Qwen and Mistral and SuffixDecoding for Llama. Every method receives a cap of $B \in \{4, 8\}$ distinct retrieval nodes per position. The union merges the two candidate sources, deduplicates prefixes, and prunes to that cap. Ranking uses source-local recency and completed earlier verification events.

The union has the largest increment in all six comparisons. Relative to the best single source, its gains for Qwen, Mistral, and Llama are 13.7%, 11.5%, and 5.0% at $B = 4$, and 16.8%, 14.8%, and 4.8% at $B = 8$. These results connect the source evidence to better candidate composition under a common verification allowance.

Table 18: Point estimates under shared verification-node caps, plotted in Figure 5. Best single is STAND for Qwen and Mistral and SuffixDecoding for Llama. The union combines that source with TLAR.

	$B = 4$ nodes			$B = 8$ nodes		
Model	TLAR	Best single	Budgeted union	TLAR	Best single	Budgeted union
Qwen3-8B	0.128	0.168	**0.191**	0.161	0.184	**0.215**
Mistral-24B	0.022	0.026	**0.029**	0.025	0.027	**0.031**
Llama-3.1-8B	0.011	0.020	**0.021**	0.012	0.021	**0.022**

D.4 Supplementary Candidate-Coverage Analysis

Table 19 identifies the additional continuations supplied by TLAR to individual retrieval baselines. It retains each baseline's prefixes, then appends previously absent TLAR prefixes up to the larger of the configured width and the baseline prefix count. Acceptance and distinct-node increments are reported together. This construction maps source overlap, while the preceding shared-budget test evaluates competition for the available verification nodes.

Table 19: Supplementary coverage analysis on 100 code-debugging problems per target. Added tokens use the named retrieval baseline as reference; added nodes count the extra distinct prefixes. Baseline candidates are retained before new TLAR prefixes are appended.

Model	Baseline	Added tokens	95% interval	Added nodes
Qwen3-8B	PLD	0.6341	[0.6082, 0.6614]	2.3890
Qwen3-8B	REST	0.6365	[0.6097, 0.6623]	2.3906
Qwen3-8B	SAM-proxy	0.1850	[0.1798, 0.1900]	0.9884
Qwen3-8B	STAND	0.0238	[0.0225, 0.0250]	0.2567
Qwen3-8B	Suffix	0.1879	[0.1829, 0.1928]	1.0006
Mistral-24B	PLD	0.5667	[0.4295, 0.7136]	1.3207
Mistral-24B	REST	0.5743	[0.4406, 0.7265]	1.3314
Mistral-24B	SAM-proxy	0.1002	[0.0862, 0.1156]	0.3672
Mistral-24B	STAND	0.0200	[0.0161, 0.0242]	0.1351
Mistral-24B	Suffix	0.1029	[0.0879, 0.1195]	0.3828
Llama-3.1-8B	PLD	0.2023	[0.1696, 0.2368]	0.5564
Llama-3.1-8B	REST	0.2022	[0.1674, 0.2373]	0.5552
Llama-3.1-8B	SAM-proxy	0.0571	[0.0478, 0.0665]	0.2213
Llama-3.1-8B	STAND	0.0131	[0.0100, 0.0165]	0.0894
Llama-3.1-8B	Suffix	0.0586	[0.0492, 0.0681]	0.2299

PLD and REST have larger increments than the history-based baselines. This pattern relates the value of the added source to the text already searched by the baseline. The next section uses feedback to allocate retrieval work across decoding positions.

E Adaptive Retrieval and Verification Cost

RQ3 connects candidate quality to the amount of work spent obtaining accepted tokens. We first vary the candidate set, then the activation policy, and finally the feedback reward. This sequence links the retrieval mechanism in Section 4.1 to the controller in Section 4.3.

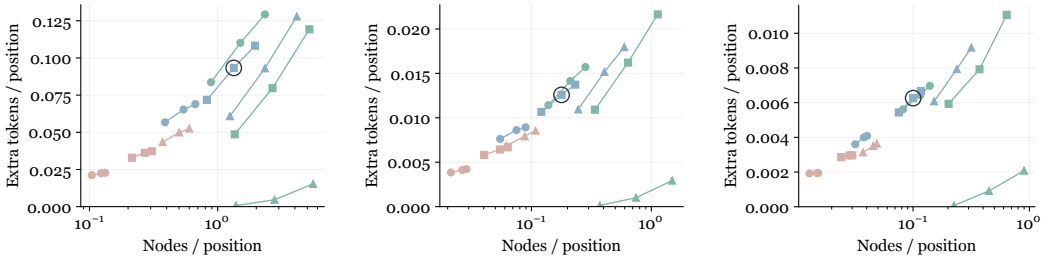
E.1 Candidate Settings and the Gain–Cost Frontier

The static grid contains all 27 combinations of $c \in \{2, 4, 8\}$, $\epsilon \in \{0, 1, 2\}$, and $k \in \{2, 4, 8\}$ per target on 100 code-debugging problems. Depth four and the 512-token gate remain fixed. The 81 archived settings provide the full grid and the within-grid selections in Tables 21 and 20. Figure 8 shows how context length, tolerance, and width move the gain–cost operating point.

The default uses fewer nodes and recovers fewer tokens than the largest-gain setting. Qwen's highest-gain configuration yields $G = 0.1293$ at $C = 2.3356$; its highest-efficiency configuration yields $G = 0.0212$ at $C = 0.1043$. Candidate settings jointly control total recovery and recovery per node.

Table 20: Selections from the archived static grid. $G = G_{\text{miss}}$ measures recovered tokens, $C = C_{\text{miss}}$ measures verification cost, and G/C measures efficiency.

Model	Selection	(c, ϵ, k)	G	C	G/C
Qwen3-8B	Default	4,1,4	0.0933	1.3344	0.0699
Qwen3-8B	Largest observed gain	2,0,8	0.1293	2.3356	0.0554
Qwen3-8B	Highest gain per node	8,0,2	0.0212	0.1043	0.2034
Mistral-24B	Default	4,1,4	0.0126	0.1783	0.0706
Mistral-24B	Largest observed gain	2,1,8	0.0216	1.1308	0.0191
Mistral-24B	Highest gain per node	8,0,2	0.0038	0.0213	0.1807
Llama-3.1-8B	Default	4,1,4	0.0063	0.1003	0.0624
Llama-3.1-8B	Largest observed gain	2,1,8	0.0111	0.6384	0.0173
Llama-3.1-8B	Highest gain per node	8,0,2	0.0019	0.0129	0.1496



(a) Qwen3-8B (b) Mistral-24B (c) Llama-3.1-8B

Figure 8: The 27 candidate settings per model. Green, blue, and pink indicate $c = 2, 4, 8$; circles, squares, and triangles indicate $\epsilon = 0, 1, 2$. Lines connect $k = 2, 4, 8$; the outlined point is $(4, 1, 4)$. Horizontal axes are logarithmic, and panels use separate scales.

E.2 Activation, Width, and Periodic Probes

The adaptive comparison holds the distinct-prefix counter fixed and measures each policy against the same fixed gate and width. Table 22 reports acceptance and cost, followed by half-life and probe-interval sweeps. Full control reduces both quantities. Gain per node rises for Qwen and Llama; Mistral's ratio changes from $0.0252/2.691$ to $0.0168/1.817$.

Removing periodic probes reduces the increment from 0.1270 to 0.0079 for Qwen, from 0.0168 to 0.0055 for Mistral, and from 0.0069 to 0.0016 for Llama. Probes provide observations that reactivate retrieval after inactive periods. Varying H changes the persistence of feedback, while varying P changes how often the controller checks for renewed reuse.

The hit-dependence measurements in Appendix C.4 motivate recent-outcome feedback. The exponential score in Equation 2 is a constant-memory summary with half-life H. The archived empirical dependence has a long tail; the sweep over $H \in \{8, 16, 32, 64, 128\}$ evaluates several timescales for summarizing it.

E.3 Feedback Aligned with Incremental Acceptance

The default reward $z_t = \mathbf{1}[a_{\text{retr}}(t) \geq 1]$ records a retrieval hit. A marginal-gain reward directly credits extension beyond SmallDraft:

$$r_t = \delta_t = \max\{a_{\text{small}}(t), a_{\text{retr}}(t)\} - a_{\text{small}}(t).$$

Both rewards become available after verification and inform later events. Gate thresholds are selected using node counts to match the default controller's cost. Table 23 compares random, periodic, last-hit, hit-EMA, and marginal-gain-EMA activation at those matched costs.

Marginal-gain EMA has the highest point estimate for every target. The progression from random activation to recent-hit feedback and marginal-gain feedback connects the controller's information to its acceptance objective.

Table 21: Complete static grid. R is recovery at active draft misses; G and C average recovered tokens and verification nodes over all positions. The archived CSV summaries supply the per-setting bootstrap intervals.

	Qwen3-8B			Mistral-24B			Llama-3.1-8B		
(c, ϵ, k)	R	G	C	R	G	C	R	G	C
2,0,2	0.2127	0.0837	0.8871	0.1065	0.0114	0.1385	0.0887	0.0056	0.0820
2,0,4	0.2710	0.1101	1.5031	0.1286	0.0142	0.2108	0.1009	0.0064	0.1156
2,0,8	0.3093	0.1293	2.3356	0.1391	0.0157	0.2823	0.1070	0.0070	0.1394
2,1,2	0.1538	0.0486	1.3583	0.1232	0.0109	0.3388	0.1075	0.0059	0.2021
2,1,4	0.2424	0.0797	2.6755	0.1801	0.0162	0.6393	0.1460	0.0079	0.3727
2,1,8	0.3453	0.1193	5.1864	0.2334	0.0216	1.1308	0.1947	0.0111	0.6384
2,2,2	0.0034	0.0007	1.3873	0.0029	0.0001	0.3729	0.0025	0.0001	0.2244
2,2,4	0.0231	0.0046	2.7746	0.0163	0.0010	0.7459	0.0264	0.0009	0.4493
2,2,8	0.0735	0.0153	5.5492	0.0486	0.0029	1.4920	0.0583	0.0021	0.8990
4,0,2	0.1261	0.0567	0.3882	0.0597	0.0076	0.0548	0.0507	0.0036	0.0319
4,0,4	0.1426	0.0653	0.5415	0.0651	0.0086	0.0751	0.0538	0.0040	0.0378
4,0,8	0.1488	0.0689	0.6692	0.0667	0.0089	0.0894	0.0543	0.0041	0.0402
4,1,2	0.1757	0.0718	0.8231	0.0933	0.0107	0.1210	0.0801	0.0054	0.0755
4,1,4	0.2218	0.0933	1.3344	0.1069	0.0126	0.1783	0.0892	0.0063	0.1003
4,1,8	0.2516	0.1082	1.9539	0.1141	0.0137	0.2321	0.0923	0.0067	0.1171
4,2,2	0.1697	0.0608	1.2454	0.1088	0.0109	0.2457	0.1024	0.0061	0.1518
4,2,4	0.2507	0.0930	2.3340	0.1492	0.0152	0.4059	0.1334	0.0079	0.2378
4,2,8	0.3309	0.1279	4.1397	0.1698	0.0180	0.5953	0.1491	0.0092	0.3168
8,0,2	0.0428	0.0212	0.1043	0.0276	0.0038	0.0213	0.0254	0.0019	0.0129
8,0,4	0.0449	0.0225	0.1232	0.0291	0.0041	0.0265	0.0254	0.0019	0.0149
8,0,8	0.0454	0.0228	0.1329	0.0295	0.0042	0.0288	0.0254	0.0019	0.0154
8,1,2	0.0692	0.0329	0.2145	0.0424	0.0058	0.0403	0.0370	0.0029	0.0241
8,1,4	0.0751	0.0363	0.2700	0.0459	0.0064	0.0547	0.0375	0.0030	0.0285
8,1,8	0.0769	0.0373	0.3061	0.0470	0.0067	0.0634	0.0375	0.0030	0.0301
8,2,2	0.0950	0.0434	0.3717	0.0523	0.0069	0.0617	0.0441	0.0031	0.0372
8,2,4	0.1075	0.0499	0.5012	0.0581	0.0079	0.0880	0.0482	0.0035	0.0455
8,2,8	0.1121	0.0525	0.6007	0.0614	0.0086	0.1081	0.0487	0.0036	0.0489

Table 22: Adaptive replay using the distinct-prefix counter. Half-life sweeps keep $P = 32$; probe-interval sweeps keep $H = 32$. Gain is additional acceptance over the same SmallDraft.

	Qwen3-8B		Mistral-24B		Llama-3.1-8B	
Setting	Δ	nodes	Δ	nodes	Δ	nodes
Fixed gate and width	0.1709	5.688	0.0252	2.691	0.0124	1.077
Adaptive gate, fixed width	0.1599	5.272	0.0195	2.149	0.0090	0.688
Periodic probes removed	0.0079	0.258	0.0055	0.439	0.0016	0.082
Full controller	0.1270	3.810	0.0168	1.817	0.0069	0.482
$H = 8$	0.1176	3.544	0.0163	1.849	0.0080	0.581
$H = 16$	0.1255	3.762	0.0168	1.846	0.0081	0.555
$H = 32$	0.1270	3.810	0.0168	1.817	0.0069	0.482
$H = 64$	0.1189	3.567	0.0157	1.614	0.0054	0.349
$H = 128$	0.0917	2.715	0.0129	1.327	0.0030	0.207
$P = 8$	0.1292	3.865	0.0194	2.058	0.0087	0.673
$P = 16$	0.1302	3.904	0.0183	1.995	0.0075	0.576
$P = 32$	0.1270	3.810	0.0168	1.817	0.0069	0.482
$P = 64$	0.1195	3.592	0.0151	1.586	0.0048	0.263
$P = 128$	0.0961	2.901	0.0128	1.341	0.0032	0.163

Table 23: Controller rewards at matched node cost on code debugging. Each cell gives Δ with its 95% trajectory interval and distinct retrieval nodes per position.

Gate	Qwen3-8B	Mistral-24B	Llama-3.1-8B
Random activation	0.096 [0.087, 0.105] 3.806 nodes	0.0137 [0.0082, 0.0192] 1.812 nodes	0.0053 [0.0025, 0.0081] 0.480 nodes
Fixed periodic	0.105 [0.096, 0.114] 3.803 nodes	0.0144 [0.0088, 0.0200] 1.819 nodes	0.0057 [0.0029, 0.0085] 0.481 nodes
Last-hit gate	0.118 [0.109, 0.127] 3.808 nodes	0.0157 [0.0099, 0.0215] 1.815 nodes	0.0064 [0.0035, 0.0093] 0.483 nodes
Hit EMA (z_t; default)	0.127 [0.117, 0.137] 3.810 nodes	0.0168 [0.0108, 0.0228] 1.817 nodes	0.0069 [0.0039, 0.0099] 0.482 nodes
Marginal-gain EMA (δ_t)	**0.136 [0.126, 0.146]** 3.811 nodes	**0.0176 [0.0115, 0.0237]** 1.816 nodes	**0.0074 [0.0043, 0.0105]** 0.482 nodes

E.4 Context Matching Under the Same Controller

Table 24 holds the marginal-gain controller and node budget fixed while changing exact to approximate context matching. The increments rise from 0.121 to 0.136 for Qwen, 0.0162 to 0.0176 for Mistral, and 0.0070 to 0.0074 for Llama. Candidate construction and feedback contribute at successive stages: matching supplies continuations, and the reward guides their use. The causal execution of these stages is specified next.

Table 24: Exact and approximate context matching under the marginal-gain controller and the same verification-node budget.

Model	Exact Δ	Exact nodes	Approx. Δ	Approx. nodes
Qwen3-8B	0.121 [0.111, 0.131]	3.807	**0.136 [0.126, 0.146]**	3.811
Mistral-24B	0.0162 [0.0103, 0.0221]	1.814	**0.0176 [0.0115, 0.0237]**	1.816
Llama-3.1-8B	0.0070 [0.0040, 0.0100]	0.481	**0.0074 [0.0043, 0.0105]**	0.482

F Causal Construction and Exact Verification

This section supports Section 4 by following a decoding event from candidate construction to exact verification and controller feedback. Prefix audits check information availability, the verifier supplies a distributional argument, and event-driven replay evaluates the resulting execution path.

F.1 Historical Candidate Construction

For each retrieved continuation, we record query position t, source position j, context length c, and depth d. The historical boundary is

$$j + d - 1 \leq t - 1.$$

Every continuation token is present in $h_t = y_{<t}$ before ranking, pruning, prefix deduplication, and activation. Three checks validate this construction: reconstruction from the explicit prefix, perturbation of all tokens at and after t, and comparison with an index that inserts tokens after target acceptance.

Table 25: Historical-boundary audit. Candidate agreement compares the archived replay with explicit-prefix reconstruction; index agreement compares complete-prefix construction with incremental insertion.

Model	Boundary violations	Candidate agreement	Ranking agreement	Index agreement
Qwen3-8B	0	100.0%	100.0%	100.0%
Mistral-24B	0	100.0%	100.0%	100.0%
Llama-3.1-8B	0	100.0%	100.0%	100.0%

Table 25 reports zero boundary violations and 100% candidate, ranking, and index agreement. Candidate sets and controller decisions remain identical under future-token perturbation. The reconstruction

rerun in Table 26 is paired with the earlier fixed-width snapshot, whose Llama increment is 0.0124. The updated Llama measurement of 0.0180 is recorded separately in Appendix B.4.

Table 26: Prefix-reconstruction rerun paired with the earlier fixed-width snapshot. Δ measures added acceptance over the same SmallDraft.

Model	Original Δ	Boundary-safe Δ	Relative change
Qwen3-8B	0.1709	0.1680	-1.7%
Mistral-24B	0.0253	0.0244	-3.6%
Llama-3.1-8B	0.0124	0.0117	-5.6%

F.2 Exact Tree Verification and Distribution Preservation

At a committed prefix h, SmallDraft and retrieval supply child-token labels for a prefix-sharing tree. The target distribution is $p(\cdot \mid h)$. The verifier follows this target-coupled procedure:

1. Evaluate target logits at the candidate tree nodes in one masked forward pass.
2. Draw $x = F^{-1}_{p(\cdot|h)}(u)$ using the next shared uniform u.
3. If x labels a child, commit x, move to that child, and continue.
4. Otherwise, commit x as the correction token and end the event.

At depth d, the event ends after the d accepted tree tokens. The reported implementation emits zero bonus tokens. Candidate labels determine which precomputed logits can be reused along the target-sampled path.

Distribution preservation. For a tree constructed from the committed prefix and a sequence of uniforms, the emitted tokens equal those of target ancestral sampling driven by the same uniforms. At the root, both procedures apply the same inverse CDF to the same distribution and uniform, producing the same token. A token outside the tree is committed as a correction and ends the event. A token inside the tree extends both prefixes identically, allowing the argument to repeat at the child. Induction over committed tokens gives equality of the conditional and joint output distributions. Event termination determines the timing of the next target forward pass. □

Table 27: Target-coupled implementation audit using 100 held prefixes and 100 shared-uniform streams per target. Exact-sequence agreement includes correction tokens.

Model	Coupled trials	Exact sequences	Token mismatches	Max logit error
Qwen3-8B	10,000	100.0%	0	3.1×10^{-6}
Mistral-24B	10,000	100.0%	0	3.6×10^{-6}
Llama-3.1-8B	10,000	100.0%	0	2.9×10^{-6}

The implementation audit uses 100 held prefixes and 100 shared-uniform streams per model. Table 27 reports identical complete emitted sequences across 30,000 coupled trials, including correction tokens. This audit connects the verifier's distributional argument to the implemented tree execution.

F.3 Event-Driven Replay and Controller Updates

Position-wise replay evaluates candidate availability at each recorded position. Speculative execution advances by accepted blocks. Let s_n be an executed position and a_n the accepted tree edges. The next position is

$$s_{n+1} = \begin{cases} s_n + a_n + 1, & \text{if verification leaves the tree,} \\ s_n + d, & \text{if all } d \text{ tree edges are accepted.} \end{cases}$$

Event-driven replay applies this transition, performs retrieval and controller updates at visited positions, and inserts historical tokens after acceptance. The controller at event n uses outcomes from completed earlier events. This schedule aligns candidate construction, feedback, and index state with the positions executed by the decoder.

The event-driven increments are 0.1231, 0.0162, and 0.0066 for Qwen, Mistral, and Llama, respectively, with lower node counts than position-wise replay. Each gain changes by less than 6%. The

Table 28: Position-wise and event-driven replay. Both use the same SmallDraft reference; event-driven cost counts retrieval nodes at executed verification events.

	Position-wise replay		Event-driven replay	
Model	Δ	Nodes/position	Δ	Nodes/position
Qwen3-8B	0.1270	3.810	0.1231	3.681
Mistral-24B	0.0168	1.817	0.0162	1.751
Llama-3.1-8B	0.0069	0.482	0.0066	0.461

comparison connects the offline candidate analysis to an executable adaptive policy. Appendix G then accounts for the serving costs of that complete policy.

G End-to-End Execution and Length Scaling

Section 5.4 links acceptance and node cost to runtime. The B200 experiments measure the complete adaptive union tree end to end, followed by repeated-run summaries and a length sweep. We present the execution configuration first, then throughput, variation, and amortization as generated history grows. We then connect these outcomes to event-level progress, resource costs, and a controlled systems-evaluation framework (Appendix G.4).

G.1 Execution Configuration and Target–Draft Pairs

The target–draft pairs are Mistral-24B/Ministral-8B-Instruct-2410 and Qwen3-8B/Qwen3-0.6B on one NVIDIA B200, matching Table 4. Each configuration uses the same 100 code-debugging prompts, 1,024 generated tokens per prompt, precision, KV-cache configuration, and serving engine. The accounting covers lookup, tree construction, deduplication, target verification, controller updates, cache updates, and sampling correction. The execution follows Appendix F.

G.2 Throughput of the Complete Union Tree

Tables 29 and 30 compare SmallDraft, single-source hybrids, and the budgeted union at batch sizes 1 and 32. The measured union gains are 4.1% and 3.5% for Mistral and 8.3% and 7.6% for Qwen. Qwen's larger acceptance increment accompanies a larger throughput gain. Mistral's standalone TLAR gain is 0.2% at batch one, while its union gain is 4.1%, highlighting the role of candidate composition in the complete system.

Table 29: Mistral/B200 end-to-end measurements. Each row represents 102,400 output tokens. Relative throughput uses SmallDraft at the same batch size.

Batch	Method	Seconds	Tokens/s	Relative
1	SmallDraft	1305.76	78.42	1.000
1	STAND + SmallDraft	1282.70	79.83	1.018
1	TLAR + SmallDraft	1303.18	78.58	1.002
1	Budgeted STAND∪TLAR + SmallDraft	1254.36	**81.64**	**1.041**
32	SmallDraft	81.23	1260.69	1.000
32	STAND + SmallDraft	79.95	1280.86	1.016
32	TLAR + SmallDraft	80.26	1275.82	1.012
32	Budgeted STAND∪TLAR + SmallDraft	78.48	**1304.81**	**1.035**

Table 30: Qwen/B200 end-to-end measurements with the protocol of Table 29. Relative throughput uses the matching SmallDraft control.

Batch	Method	Seconds	Tokens/s	Relative
1	SmallDraft	715.08	143.20	1.000
1	STAND + SmallDraft	686.93	149.07	1.041
1	TLAR + SmallDraft	675.86	151.51	1.058
1	Budgeted STAND∪TLAR + SmallDraft	660.26	**155.09**	**1.083**
32	SmallDraft	42.58	2405.00	1.000
32	STAND + SmallDraft	41.22	2484.40	1.033
32	TLAR + SmallDraft	40.47	2530.10	1.052
32	Budgeted STAND∪TLAR + SmallDraft	39.57	**2587.80**	**1.076**

G.3 Repeated-Run Variation and Paired Summaries

We measure throughput over five warmed runs per configuration. Tables 31 and 32 preserve those values and their medians. CV is the sample coefficient of variation. Table 33 computes Mistral's paired percentage gains against SmallDraft and two-sided 95% t intervals over the five measured pairs; the Qwen table includes the corresponding paired summaries. These calculations summarize the observed variation across repeated runs.

Table 31: Measured Mistral throughput distribution underlying Table 29. Values are tokens/s. CV is the sample coefficient of variation across the five measured runs.

Batch	Method	Run 1	Run 2	Run 3	Run 4	Run 5	Median	CV
1	SmallDraft	78.42	78.11	78.68	78.29	78.57	78.42	0.29%
1	STAND + SmallDraft	79.96	79.52	80.17	79.83	79.68	79.83	0.31%
1	TLAR + SmallDraft	78.58	78.24	78.90	78.51	78.65	78.58	0.30%
1	Budgeted union	81.80	81.29	82.02	81.64	81.47	81.64	0.35%
32	SmallDraft	1260.69	1251.80	1270.40	1258.20	1265.70	1260.69	0.56%
32	STAND + SmallDraft	1284.90	1270.60	1291.70	1280.86	1277.50	1280.86	0.62%
32	TLAR + SmallDraft	1279.80	1265.40	1287.10	1275.82	1271.60	1275.82	0.64%
32	Budgeted union	1310.20	1294.60	1317.50	1304.81	1300.30	1304.81	0.68%

Table 32: Measured Qwen throughput distribution. Values are tokens/s. The final column gives paired mean gains and 95% t intervals; SmallDraft rows give CV.

Batch	Method	Run 1	Run 2	Run 3	Run 4	Run 5	Median	Paired summary
1	SmallDraft	143.20	142.55	144.03	142.86	143.67	143.20	CV 0.42%
1	STAND + SmallDraft	149.44	148.22	150.01	149.07	148.73	149.07	4.071 [3.643, 4.499]%
1	TLAR + SmallDraft	151.93	150.55	152.36	151.51	150.97	151.51	5.726 [5.215, 6.237]%
1	Budgeted union	155.62	154.08	156.41	155.09	154.71	155.09	**8.320 [7.795, 8.846]%**
32	SmallDraft	2405.0	2389.0	2422.0	2398.0	2413.0	2405.0	CV 0.53%
32	STAND + SmallDraft	2491.0	2468.0	2505.0	2484.4	2477.0	2484.4	3.313 [2.831, 3.795]%
32	TLAR + SmallDraft	2538.0	2509.0	2551.0	2530.1	2520.0	2530.1	5.164 [4.598, 5.731]%
32	Budgeted union	2598.0	2561.0	2614.0	2587.8	2576.0	2587.8	**7.564 [6.869, 8.260]%**

Table 33: Paired percentage gains calculated from the measured Mistral runs in Table 31. Intervals are two-sided 95% t intervals over the five pairs.

Batch	Comparison with SmallDraft	Mean gain	95% interval
1	STAND + SmallDraft	1.808%	[1.522, 2.095]%
1	TLAR + SmallDraft	0.207%	[0.111, 0.302]%
1	Budgeted union	**4.119%**	**[3.801, 4.438]%**
32	STAND + SmallDraft	1.566%	[1.086, 2.047]%
32	TLAR + SmallDraft	1.157%	[0.639, 1.674]%
32	Budgeted union	**3.498%**	**[2.922, 4.075]%**

G.4 From Candidate Acceptance to Execution Efficiency

From reusable content to elapsed time. The source, hybrid, and matched-budget comparisons connect reusable content to incremental candidate coverage. Execution adds the time required to produce a fixed output. SwiftEP connects data movement and GPU resource competition to serving performance through buffer fusion and transfer offloading (Li et al., 2026a). For TLAR, the corresponding questions concern execution dependencies, resource costs, and their effect on complete-request time.

Let N be the fixed number of output tokens, and let T_0 and T_1 be the elapsed times for SmallDraft and the budgeted union. The throughput ratio and fractional time saving are

$$S = \frac{N/T_1}{N/T_0} = \frac{T_0}{T_1}, \qquad R_T = \frac{T_0 - T_1}{T_0} = 1 - \frac{1}{S}. \tag{3}$$

Table 34 calculates these savings directly from the reported durations. Mistral's 4.1%/3.5% throughput gains correspond to 3.94%/3.39% time savings at batches 1/32; Qwen's 8.3%/7.6% gains correspond to 7.67%/7.07% time savings. These are net outcomes after drafting, retrieval, verification, and cache management. They quantify the benefit available to the complete system.

Separating progress per event from event cost. For configuration i, let M_i be the number of executed decoding events, $\bar{C}_i = N/M_i$ the mean committed tokens across requests per event, and

Table 34: Time savings calculated from the measured durations in Tables 29 and 30. Each row covers 102,400 output tokens. Percentages use the corresponding SmallDraft duration.

Target	Batch	T_0 (s)	T_1 (s)	Saved (s)	R_T
Mistral	1	1305.76	1254.36	51.40	3.94%
Mistral	32	81.23	78.48	2.75	3.39%
Qwen	1	715.08	660.26	54.82	7.67%
Qwen	32	42.58	39.57	3.01	7.07%

$\bar{t}_i = T_i/M_i$ the mean elapsed time per event, with prefill amortized across these events. Then

$$Q_i = \frac{\bar{C}_i}{\bar{t}_i}, \qquad S = \frac{\bar{C}_1/\bar{C}_0}{\bar{t}_1/\bar{t}_0}. \tag{4}$$

Committed progress includes accepted candidate tokens and target-sampled correction tokens. The offline acceptance experiments describe candidate coverage at recorded positions; the event-driven execution determines which positions are visited and how many tokens each event commits. More useful candidates can increase $\bar{C}$, while wider verification and memory operations can increase $\bar{t}$. Acceleration occurs when the relative increase in committed progress exceeds the relative increase in event cost. This identity specifies the two quantities needed to explain the different Mistral and Qwen runtime gains. Per-event traces supply their separate contributions.

Locating resource costs and overlap. The released implementation constructs history matches and candidate metadata on the CPU and verifies the tree on the GPU. Token IDs, positions, and masks cross the host–device boundary. Batched KV packing and accepted-path extraction move device tensors; unequal trees introduce padding. CPU time, transfer and cache-copy bytes, useful and padded nodes, target-kernel duration, and peak allocated memory provide corresponding resource measurements.

Verification consumes the completed candidate tree, and subsequent retrieval uses committed tokens and observed outcomes. Drafting and history lookup share the same committed prefix, providing an opportunity for concurrent scheduling. A timeline can measure overlap and exposed waiting time. Concurrent GPU operations may compete for bandwidth and compute resources. Critical-path time and resource demand jointly determine the value of overlap; summing component durations counts overlapping intervals multiple times.

A controlled component study. Hold prompts, checkpoints, candidates, precision, and target-sampling streams fixed while varying buffer reuse, metadata packing, KV compaction, and concurrent scheduling separately. Evaluate their combination to measure interactions. Record committed tokens, event count, critical-path duration, and complete-request time alongside resource counters. This design tests the execution cost of a fixed candidate policy. A subsequent width-and-budget sweep jointly measures changes in $\bar{C}$ and $\bar{t}$.

The length sweep adds a temporal axis: Qwen's gains change from $-0.6\%/-0.7\%$ at 512 tokens to $12.4\%/10.8\%$ at 4,096 tokens (Table 2). Longer execution creates more history and retrieval-eligible positions. Tracking index size, lookup time, cache traffic, and committed progress would reveal how memory maintenance and reuse develop together.

A framework for runtime-aware retrieval. The resulting framework connects source quality, candidate complementarity, committed progress, critical-path cost, and serving performance. A subsequent controller can combine incremental acceptance with measured event time, memory pressure, and queue load to select retrieval width and verification budget. Its objective is committed tokens per unit time under latency and memory constraints. Evaluation should cross batch size, concurrent-request count, and output length, reporting throughput, first-token and inter-token latency, completion latency, GPU activity, memory bandwidth, CPU time, and peak device memory. Paired repetitions and sampling audits connect efficiency to generation behavior, making each mechanism testable through its effect on the complete inference workload.

G.5 Archived Model-Based Drafting Reference

The EAGLE-3 archive (job 40233870) provides a separate model-based reference using 100 code-debugging prompts, 256 output tokens, and five speculative tokens. Qwen vanilla and EAGLE-3 throughput are 13,063.70 and 20,436.75 tokens/s; Llama values are 19,436.63 and 16,721.52 tokens/s. The checkpoints are `RedHatAI/Qwen3-8B-Thinking-speculator.eagle3` and `yuhuili/EAGLE3-LLaMA3.1-Instruct-8B`. These archived point measurements describe that execution configuration. The shared-condition SmallDraft controls in Tables 29 and 30 define the reference for the TLAR runtime analysis.

H Discussion: Trajectory Memory as an Inference Resource

The results connect three properties of generated text: it stores reusable continuations, those continuations can extend an existing drafter, and their value changes during decoding. This connection gives trajectory memory an operational role in inference. The research problem becomes how to identify, select, and verify useful parts of an evolving memory while accounting for the cost of maintaining and accessing it. The source tests, hybrid comparisons, controller ablations, causal audits, and runtime analysis each resolve one part of that problem.

H.1 Generation Creates a Memory Whose Value Depends on the Drafter

The source comparisons show that a generated trajectory contains continuations that can be recovered from recurring contexts. The same-prompt comparison adds a more specific observation: changing the rollout of a fixed problem changes the available drafts. Qwen and Mistral favor their own history, while Llama favors the other rollout (Table 12). A mechanistic hypothesis is that choices made during generation—such as notation, local phrasing, and the order of intermediate steps—create a continuation vocabulary that a matching history makes accessible. Paired analyses of these choices would connect the token-level source effect to the structure of reasoning.

The value of this memory is also conditional on what the drafter already supplies. Mistral has stronger source acceptance in code than mathematics, yet its hybrid increment is larger in mathematics (Figure 4). SmallDraft coverage mediates the route from available reuse to additional accepted tokens. This reversal makes incremental coverage a useful design objective for memory-based inference: a memory should be evaluated through the predictions it adds to the system receiving it. A direct next experiment would vary draft-model capacity on fixed target trajectories and measure how source coverage, overlap, and marginal gain change together.

H.2 A Verification Budget Makes Candidate Composition Central

The supplementary addition test identifies new accepted continuations from TLAR across retrieval sources (Table 19). The matched-budget test then requires both sources to share a fixed node allowance. Its union gains of 4.8%–16.8% over the strongest single source (Table 18) show the value of composing candidates from different retrieval mechanisms. At $B = 4$, Qwen's increment rises from 0.168 to 0.191; at $B = 8$, it rises from 0.184 to 0.215.

This result motivates selection based on the contribution of a candidate to the tree already chosen. For a fixed target continuation, adding a candidate changes accepted length through the longest accepted path. Two candidates that recover the same prefix overlap in value; a candidate that extends another path changes coverage. Prefix sharing also changes cost, because overlapping paths share verification nodes. Candidate selection couples these two forms of overlap.

An ensuing research direction is to estimate the expected increase in accepted length per added tree node, conditional on the existing draft and retrieval branches. Such a selector could jointly choose the source, continuation depth, and branching width. The two-budget experiment provides an initial evaluation template: hold the node cap fixed, vary the selection rule, and record coverage, prefix sharing, and target latency. This would turn multi-source retrieval into an explicit resource-allocation problem.

H.3 Feedback Should Track the Value Added to the Existing System

Temporal dependence supplies a reason to adapt: next-position hit probabilities after a hit exceed the overall probabilities for all three targets (Appendix C.4). The controller ablations then reveal that the choice of feedback matters. At almost identical node costs, marginal-gain EMA has higher point

estimates than hit EMA, which in turn exceeds random activation (Table 23). Qwen progresses from 0.096 under random activation to 0.127 under hit EMA and 0.136 under marginal-gain EMA. The feedback becomes more closely aligned with the hybrid's acceptance objective along this sequence.

Periodic probes expose a second property of adaptive memory access: the policy affects which outcomes it observes. An inactive retriever obtains its next reuse observation when it probes. Removing probes reduces Qwen's gain from 0.1270 to 0.0079, with reductions for Mistral and Llama as well (Table 22). Probing is part of information acquisition for the controller. Future policies could allocate probes according to uncertainty about local reuse and compare the recovered gain against the cost of gathering that information. Event-driven replay supplies the execution schedule for this experiment, ensuring that feedback corresponds to completed decoding events.

H.4 Exact Verification Supports Flexible Memory Search

Approximate matching and exact acceptance operate at successive stages. The matched-controller comparison shows higher increments with approximate contexts for each target (Table 24); the verifier still commits tokens through the target distribution. The distributional argument and 30,000 coupled trials in Appendix F.2 connect that rule to the implemented tree. The prefix reconstruction and event-driven audits complete the connection from available history to executed feedback.

This separation creates a useful interface for future retrieval research. Context embeddings, compressed history indexes, and structured references to earlier reasoning steps could be studied as candidate generators feeding the same exact verifier. The research questions become how each representation affects candidate recall, incremental acceptance, lookup latency, and memory use. The existing source controls and shared-budget tests provide a common way to compare those representations while preserving the generation objective.

H.5 Longer Reasoning Changes Both the Memory and Its Economics

The static grid shows that the highest total gain and the highest gain per node occur at different settings (Table 20). Runtime adds further costs: lookup, index maintenance, tree construction, and cache updates. The B200 measurements connect those costs to output length. At 512 tokens, controller and index overhead produce changes of -0.6% and -0.7%; at 4,096 tokens, measured gains reach 12.4% and 10.8% (Table 2). The length sweep describes an amortization process as more positions become eligible for retrieval.

Long generation has two linked effects: it creates more history to search and more future decoding work over which to recover memory-access cost. This suggests jointly adapting retention, lookup effort, and verification width as a trajectory grows. A useful next study would measure the age and reuse frequency of retained spans alongside device latency, then evaluate eviction policies against full-history retrieval. Repeating the analysis across target–draft pairs, batch sizes, and hardware would connect memory policy to the actual serving objective of accepted tokens per unit time.

H.6 An Evaluation Chain for Memory-Based Inference

The repetition audit gives the broader evaluation framework its first step: inspect what creates a strong reuse statistic. Removing seven looping Llama mathematics trajectories changes source acceptance from 7.03 to 2.2378 (Appendix C.2). Source identity, task progress, and accepted span length carry complementary information. The hybrid reversal for Mistral adds a second step: measure what the memory contributes beyond the current drafter. Shared-budget unions add resource control, controller ablations identify the feedback objective, and causal replay connects the policy to an executable decoder.

Together, these steps provide a reusable methodology for studying inference-time memory: identify the source signal, measure incremental coverage, allocate a common resource budget, validate the execution path, and account for runtime. Future benchmarks can pair these measurements with task outcomes and actual device traces, with documented measurement procedures and reproducible execution settings. Appendix G.4 develops this evaluation into a component study that connects candidate coverage, committed progress, critical-path time, and serving performance. The central contribution of this perspective is a connection between reasoning content and computational scheduling. A generated trajectory becomes a resource whose useful continuations can guide the allocation of subsequent inference work.