

Aggregate While Searching: Online Partial-Trajectory Aggregation for Long-Horizon LLM Search Agents

Feixue Liu¹, Ruoyu Chen¹, Songyi Cui¹, Yimo Yan², Yong-Hong Kuo^{1,*}

¹Department of Data and Systems Engineering, The University of Hong Kong

{feixue_liu, ruoyuchen, echo07}@connect.hku.hk, yhkuo@hku.hk

²Department of Industrial Engineering, American University of Sharjah, United Arab Emirates
yyan@aus.edu

Abstract

Parallel test-time scaling treats each rollout as an independent candidate answer. For long-horizon search agents, however, these rollouts also produce rich traces of tool calls, retrieved evidence, and partial reasoning states. Intermediate failures such as reasoning drift are difficult to self-correct within a single rollout, yet useful corrective signals often already exist in other live rollouts before termination. This creates an opportunity to turn cross-trajectory information into online correction during execution, rather than harvesting it only after rollouts terminate. However, simple broadcast is unsafe because partial states can be redundant, irrelevant, or risky for the receiving trajectory. We propose OPTA, a live aggregation framework based on checkpoint-level synchronization: each rollout maintains a private report, while a controller distills a typed global shared state, and injects only relevant and useful feedback into each trajectory, preserving its exploration identity. On BrowseComp-Plus and DeepSearchQA across four LLM backbones, OPTA is consistently the strongest test-time scaling method, lifting Qwen3-30B on BrowseComp-Plus from 37.9% to 59.6%. Our results show that OPTA is a practical route toward timely and selective test-time scaling for long-horizon LLM search agents.

1 Introduction

Parallel test-time scaling is typically framed as generating multiple independent candidate answers through parallel rollouts. For long-horizon agents, this view is incomplete. A rollout is also a long trace of tool calls, evidence, and discarded hypotheses. Most of it never appears in the final answer. N parallel rollouts therefore produce not just N answers, but distributed experience across trajectories. However, raw trajectories are not directly

Actionability gap on BrowseComp-Plus

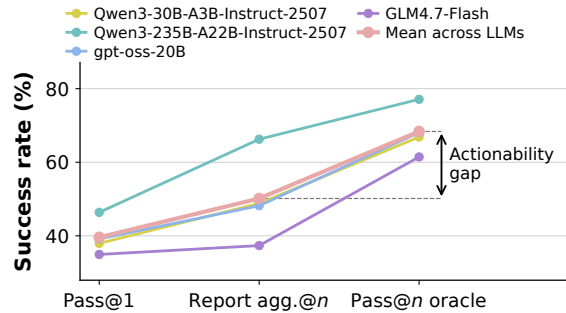


Figure 1: Actionability gap on BrowseComp-Plus: report-level terminal aggregation versus oracle Pass@ n across different LLM backbones.

consumable. Concatenated, they exceed the context window. Read individually, they mix supported findings with dead ends, stale assumptions, unresolved conflicts, and observations irrelevant to the trajectory at hand. Any cross-trajectory comparison, reuse, or feedback needs a representation that separates and exposes these signals. Therefore, test-time scaling for long-horizon agents is first a representation problem.

Existing multi-trajectory methods use such representations in two ways, both acting only after a trajectory has terminated. Selection scores or summarizes completed trajectories and picks or synthesizes a final answer; majority voting, self-consistency, best-of- n , judge-based selection, and report-level aggregation all fall into this regime (Wang et al., 2022; Chung et al., 2025; Qiao et al., 2025). Reuse goes further, distilling completed rollouts into experience banks or structured memories that condition later attempts (Wang et al., 2026d; Zhu et al., 2026). Neither acts in time: selection cannot reshape the reasoning behind its candidates, and reuse can spare future rollouts repeated work but cannot rescue an actively drifting trajectory.

* Corresponding author.
Our implementation is open-sourced at [OPTA](#).

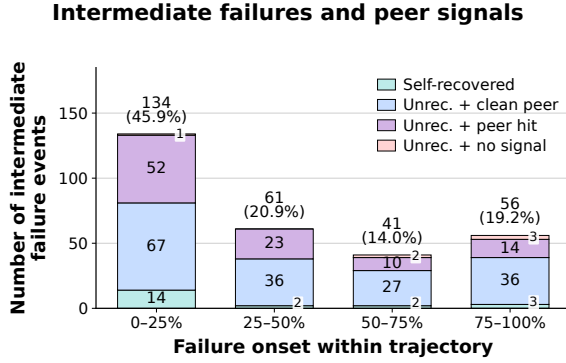


Figure 2: Intermediate failure events bucketed by trajectory stage. Each bar is split into self-recovered events and unrecovered events further decomposed by the strongest peer signal available at that stage (clean-peer support, peer-search hit, or none).

Moving feedback inside an active trajectory has been seen as risky, since mid-trajectory states are unreliable to evaluate and global updates can perturb ongoing reasoning and inflate variance.

These observations leave open a more precise form of online feedback. The question is not whether active trajectories should share everything with each other, but whether cross-trajectory information can be made available early enough to affect ongoing reasoning, and selective enough to avoid perturbing trajectories that do not need it. We therefore separate the motivation into three empirical claims: terminal aggregation leaves actionable information unused; relevant peer information often exists before trajectories terminate; and naive broadcast is too noisy to serve as the mechanism.

Figure 1 exposes the first claim. Oracle Pass@ n substantially exceeds report-level terminal aggregation, indicating that useful information, and sometimes even correct solutions, already exists in the sampled trajectories but is not recovered by final-stage synthesis. This is an actionability gap: the parallel computation holds it, but terminal aggregation fails to make it actionable.

Figure 2 asks whether this actionability gap can be localized before trajectories terminate. To answer this, we analyze 300 trajectories and identify 292 intermediate failure events, including reasoning drifts and retrieval misses, of which only 7.2% self-recover. Among the remaining unrecovered events, 97.8% already have some peer signal: another active trajectory either holds a clean evidence-supported state on the same semantic target (clean-peer), or has reached a relevant search result (peer-

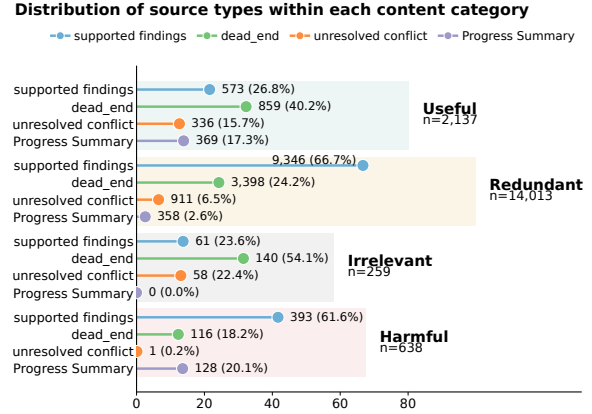


Figure 3: Receiver-specific usefulness of shared-state fragments. For each utility category, we show the distribution of atomic content types among the corresponding receiver-fragment pairs.

search hit). Thus, useful cross-trajectory information often exists while the failing trajectory is still active, but terminal aggregation and completed-rollout reuse expose it only after the opportunity to intervene has passed.

This does not imply that all intermediate states should be shared globally. Figure 3 reports such a simulation: only a subset of broadcast items are useful for the receiver. Many are redundant or irrelevant, and some are risky, introducing unsupported assumptions, irrelevant conflicts, or premature consensus. Checkpoint-level feedback must therefore be both timely and selective: cross-trajectory information should become actionable before trajectories terminate, but filtered against each trajectory’s private accumulated state.

These observations motivate a third operation enabled by trajectory representation: live synchronization. We present Online Partial-Trajectory Aggregation (OPTA), a test-time scaling framework that instantiates live synchronization for long-horizon LLM search agents. OPTA runs n search trajectories in parallel and, at fixed checkpoints, performs two-stage cross-trajectory aggregation: each trajectory maintains a private accumulated report of its progress; a controller then builds a typed shared state that separates supported findings, dead ends, unresolved conflicts, and progress; and it injects into each trajectory only fragments relevant to its state. Cross-trajectory information thus becomes an online correction signal during execution, rather than terminal evidence for final selection or offline experience for later attempts, with each trajectory’s exploration identity preserved.

Our contributions are threefold:

- We frame online partial-trajectory aggregation as a missing operation in test-time scaling for long-horizon agents: existing methods either select among completed trajectories or recycle finished rollouts as offline experience, leaving the active rollout window unaddressed.
- We propose OPTA, which realizes this operation through a typed shared state and trajectory-specific selective injection, so that cross-trajectory feedback is both timely and matched to each trajectory’s private state rather than broadcast globally or used to prune trajectories.
- On BrowseComp-Plus and DeepSearchQA across four LLM backbones, OPTA achieves the strongest result among test-time scaling methods and substantially reduces harmful information exposure compared to global broadcast, LeaP-routed reports, and RSE-style experience reuse.

2 Methodology

We propose OPTA (Online Partial-Trajectory Aggregation), a test-time scaling framework for long-horizon LLM search agents. OPTA runs multiple search trajectories in parallel and aggregates them at fixed checkpoints during search. At each checkpoint, every trajectory summarizes its recent progress into a compact report; an aggregation controller merges these reports into a shared state and selectively injects fragments into each trajectory. Trajectories thus benefit from each other’s discoveries while preserving independent exploration.

2.1 Problem Setup

Given an input problem x , a search agent generates a trajectory through a sequence of actions and observations:

$$\tau_i = (a_{i,1}, o_{i,1}, \dots, a_{i,L_i}, o_{i,L_i}, y_i), \quad (1)$$

where $a_{i,t}$ denotes the t -th reasoning step or tool action, $o_{i,t}$ denotes the corresponding observation or intermediate result, and y_i is the candidate answer produced by trajectory i .

A standard parallel test-time scaling approach samples n independent trajectories,

$$\tau_i \sim \pi_\theta(\cdot \mid x), \quad i = 1, \dots, n, \quad (2)$$

and performs aggregation only at the end:

$$y^* = \mathcal{A}_{\text{final}}(x, y_1, \dots, y_n). \quad (3)$$

This search-then-aggregate paradigm allows no cross-trajectory information flow during search. OPTA replaces $\mathcal{A}_{\text{final}}$ with periodic online aggregation, defined below.

2.2 Online Partial-Trajectory Aggregation

OPTA divides each trajectory into multiple search chunks. Let $\Delta\tau_i^k$ denote the k -th chunk of trajectory i . At checkpoint k , trajectory i updates its accumulated report by summarizing the new chunk and merging it with the previous enriched report:

$$r_i^k = f_{\text{update}}(x, \tilde{r}_i^{k-1}, \Delta\tau_i^k), \quad (4)$$

where $\tilde{r}_i^{k-1}$ is the enriched accumulated report from the previous checkpoint, and r_i^k is the updated local report before cross-trajectory injection. The report is a compact summary of the trajectory’s partial progress; its schema follows the four-category format introduced in Stage 1 below.

At each checkpoint, the controller collects the reports from all active trajectories:

$$\mathcal{R}^k = \{r_i^k \mid i \in \mathcal{I}^k\}, \quad (5)$$

where $\mathcal{I}^k$ denotes the set of active trajectories. OPTA then performs two-stage cross-trajectory aggregation.

Stage 1: Atomic Shared-State Construction.

The controller first aggregates all accumulated reports into a global shared state:

$$g^k = f_{\text{global}}(x, \mathcal{R}^k). \quad (6)$$

The shared state is organized into four types of information:

$$g^k = (\mathcal{F}^k, \mathcal{D}^k, \mathcal{C}^k, \mathcal{P}^k), \quad (7)$$

where $\mathcal{F}^k$ contains supported findings, $\mathcal{D}^k$ contains dead ends or unproductive directions, $\mathcal{C}^k$ contains unresolved conflicts across trajectories, and $\mathcal{P}^k$ summarizes overall progress.

Each component is written as a numbered list of atomic bullet points. We denote the union of all shared items as

$$\mathcal{B}^k = \{b_1^k, \dots, b_{M_k}^k\}. \quad (8)$$

Each item b_j^k is self-contained and traceable, with a specific type and supporting trajectory evidence. This atomic format makes the following selective injection step explicit and controllable.

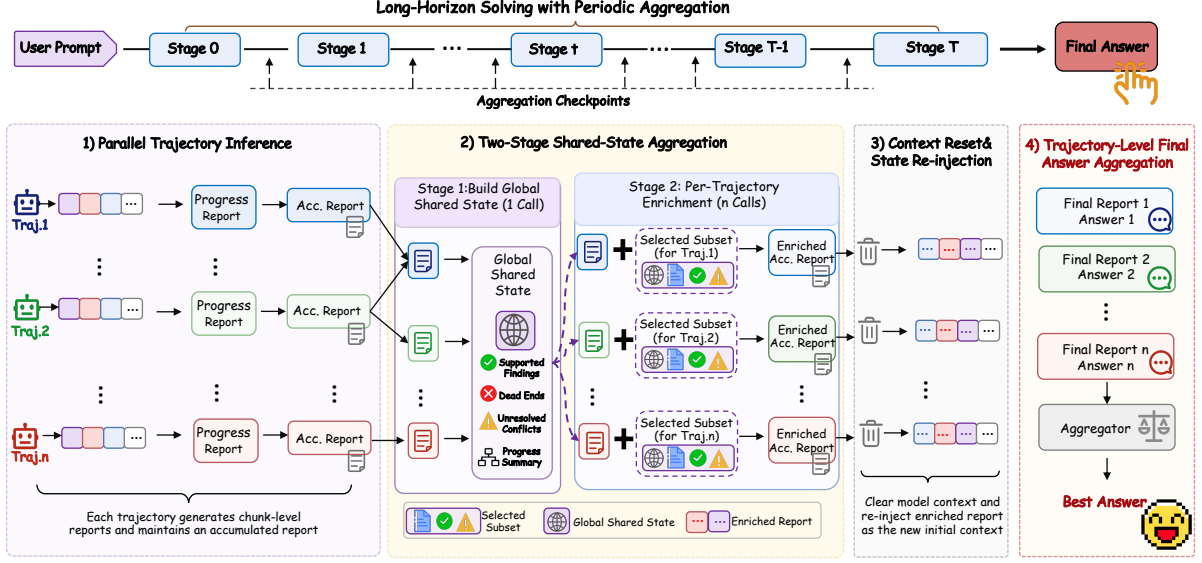


Figure 4: Overview of the long-horizon solving workflow with periodic synchronization.

Stage 2: Trajectory-Specific Selective Injection.

For each trajectory i , the controller reads its accumulated report r_i^k together with the atomic shared items $\mathcal{B}^k$, and selects only the items useful for that trajectory’s future search:

$$z_i^k = f_{\text{select}}(x, r_i^k, \mathcal{B}^k), \quad z_i^k \subseteq \mathcal{B}^k. \quad (9)$$

The selected items are then appended to the trajectory’s own accumulated report:

$$\tilde{r}_i^k = f_{\text{inject}}(x, r_i^k, z_i^k). \quad (10)$$

Equivalently, this can be written as

$$\tilde{r}_i^k = r_i^k \oplus z_i^k, \quad (11)$$

where $\oplus$ denotes report-preserving injection. The original report is preserved, and the selected cross-trajectory information is added as an auxiliary section. Each trajectory therefore receives shared information specific to its current progress.

After selective injection, OPTA resets the raw model context and starts the next search chunk from $\tilde{r}_i^k$, bounding context length while preserving each trajectory’s accumulated progress and the injected cross-trajectory information:

$$\Delta\tau_i^{k+1} \sim \pi_\theta(\cdot \mid x, \tilde{r}_i^k). \quad (12)$$

This context reset and re-injection mechanism prevents the raw context from growing indefinitely, while allowing each trajectory to continue from a concise summary of its own progress plus selected cross-trajectory information.

2.3 Final Answer Aggregation

OPTA does not prune trajectories during search. All trajectories run until completion and produce candidate answers. Let $\tilde{r}_i^{K_i}$ denote the final enriched accumulated report of trajectory i , and let y_i denote its final answer. The final aggregation stage receives

$$\mathcal{Y} = \left\{ \left(\tilde{r}_i^{K_i}, y_i \right) \mid i = 1, \dots, n \right\}. \quad (13)$$

The final answer is selected by an LLM-based aggregator:

$$i^* = \arg \max_{i \in \{1, \dots, n\}} s_\theta(x, \tilde{r}_i^{K_i}, y_i, \mathcal{Y}), \quad y^* = y_{i^*}. \quad (14)$$

The scoring function s_θ compares candidate answers according to evidence sufficiency, reasoning consistency, and unresolved conflicts. Therefore, OPTA uses online partial-trajectory aggregation to improve the search process itself, while the final aggregation stage selects the most strongly supported completed trajectory.

3 Experiment

3.1 Experiment Setup

We evaluate our method on two long-horizon search benchmarks: BrowseComp-Plus (BC+) and DeepSearchQA (DSQA). For BC+, we sample 20% of the benchmark, yielding 166 questions. For DSQA, we use 50 multi-select questions and 50 single-choice questions. Experiments are conducted on four model backbones: Qwen3-30B-

Model	Data	No scaling	Oracle	Terminal aggregation				Closed-loop aggregation				
		P@1	P@n	MV	WMV	BoN	SA	RSE	PP	OP	LeaP	OPTA
Qwen3-30B	BC+ DSQA	37.9%	66.8%	47.5%	48.3%	41.6%	48.8%	<u>51.2%</u>	48.2%	48.8%	40.3%	59.6%
		18.0%	49.0%	26.0%	42.0%	38.0%	30.0%	<u>43.0%</u>	28.0%	24.0%	36.0%	44.0%
OSS-20B	BC+ DSQA	39.1%	68.1%	46.3%	<u>64.1%</u>	59.5%	48.2%	45.1%	46.9%	46.9%	39.7%	66.3%
		23.0%	66.0%	37.0%	37.0%	<u>39.0%</u>	40.0%	<u>39.0%</u>	<u>39.0%</u>	29.0%	31.0%	40.0%
GLM-4.7-flash	BC+ DSQA	34.9%	61.4%	<u>50.6%</u>	46.3%	42.7%	37.3%	49.3%	46.9%	36.7%	41.5%	57.2%
		29.0%	67.0%	<u>43.0%</u>	<u>43.0%</u>	42.0%	31.0%	46.0%	41.0%	32.0%	41.0%	46.0%
Qwen3-235B	BC+ DSQA	46.3%	77.1%	48.8%	66.3%	<u>72.1%</u>	66.8%	69.8%	50.0%	63.8%	65.6%	74.7%
		43.0%	73.0%	48.0%	51.0%	48.0%	51.0%	<u>54.0%</u>	47.0%	52.0%	46.0%	55.0%

Table 1: Main results across two benchmarks and four model families. BC+ denotes BrowseComp-Plus; DSQA denotes DeepSearchQA. P@n denotes oracle Pass@n. MV denotes Majority Vote; WMV denotes Weighted Majority Vote; BoN denotes Best-of-n; SA denotes aggregation with summary; RSE and LeaP denote the experience-reuse and peer-sharing baselines of (Wang et al., 2026d) and (Luo et al., 2025), respectively; PP denotes Parallel-Probe (Zheng et al., 2026); OP denotes Online Prune. Qwen3-30B denotes Qwen3-30B-A3B-Instruct-2507; OSS-20B denotes gpt-oss-20B; Qwen3-235B denotes Qwen3-235B-A22B-Instruct-2507. The highest results are highlighted in **bold**, while the second-best results are underline.

A3B-Instruct-2507, gpt-oss-20B, GLM-4.7-flash, and Qwen3-235B-A22B-Instruct-2507.

All methods use a ReAct-style long-horizon agent with up to 300 turns. Each trajectory uses a 64K-token summary interval, which also defines the aggregation checkpoint interval in OPTA. At each checkpoint, the trajectory summarizes its newly generated history into its accumulated report, after which the controller performs shared-state construction and trajectory-specific selective injection.

For each query, we use a rollout budget of $n = 10$ with a sampling temperature of 0.2, and all baselines share the same rollout budget. We set the relatively low temperature to limit excessive sampling noise during long-horizon execution; nevertheless, the resulting rollouts exhibit substantial diversity at both the answer and retrieval-path levels, as quantitatively analyzed in Appendix §C. Answer correctness is evaluated by a GPT-4o-based critic.

We report P@1 for single-trajectory performance and Pass@n for oracle performance, with $n = 10$ throughout.

3.2 Main Results

We organize baselines by how the aggregate is used during test-time inference. In *terminal aggregation*, completed trajectories are aggregated only to produce a final decision; the aggregate does not influence any subsequent reasoning. In *closed-loop aggregation*, the aggregate is reused before the final answer is emitted, serving as a control or context signal for later inference. We fur-

ther divide closed-loop methods into two regimes. *Round/rollout-level feedback* aggregates completed reasoning units and uses the resulting experience to condition later units. In contrast, *checkpoint-level feedback* aggregates partial states from active trajectories and feeds the result back before those trajectories terminate. This distinction is important for long-horizon agents: feedback over completed units can help future attempts avoid repeated work, but only checkpoint-level feedback can intervene while a trajectory is still drifting.

Table 1 compares methods across two benchmarks and four model configurations under the same rollout budget. We organize the results by how aggregated information is used during test-time inference. Pass@1 reflects single-trajectory performance, while oracle Pass@n estimates whether the rollout budget contains at least one correct trajectory. The large gap between Pass@1 and Pass@n shows that additional rollouts often contain useful evidence, but the gap between existing methods and the oracle indicates that the challenge is recovering and using that evidence effectively.

Terminal aggregation methods improve over single-trajectory inference by selecting or synthesizing among completed trajectories. However, since their aggregates are used only after all trajectories terminate, they cannot influence the reasoning process that produced the candidates. This is limiting in long-horizon settings, where early errors, stale assumptions, or unverified intermediate conclusions can accumulate before the final aggregator is invoked.

Closed-loop aggregation addresses this limita-

tion by feeding aggregated information back into inference before the final answer is emitted. Within this category, rollout-level feedback, such as RSE, recycles information from completed reasoning units into later attempts. This can reuse experience across attempts, but it cannot correct an active trajectory while it is drifting. Checkpoint-level feedback instead aggregates partial trajectory states before termination, allowing cross-trajectory evidence to affect ongoing reasoning. PP, OP, LeaP, and OPTA all fall into this broad regime, but differ in how feedback is constructed and injected.

OPTA differs from prior checkpoint-level approaches by preserving each trajectory’s accumulated state while injecting only trajectory-relevant cross-trajectory information. Rather than pruning trajectories, routing from coarse reports, probing partial states, or exchanging global summaries, it lets evidence from one trajectory become actionable for another without overwriting local context or forcing trajectories into a shared summary.

Under the matched rollout budget, OPTA achieves the strongest or tied-strongest result in all evaluated settings. On BC+, it obtains the best result across all four model configurations: 59.6% on Qwen3-30B, 66.3% on OSS-20B, 57.2% on GLM-4.7, and 74.7% on Qwen3-235B. On DSQA, it is best on Qwen3-30B and Qwen3-235B, reaching 44.0% and 55.0%, and ties the best result on OSS-20B and GLM-4.7, both at 40.0% and 46.0%, respectively. Compared with the rollout-level closed-loop baseline RSE, OPTA is stronger or tied in every setting, suggesting that moving feedback from completed rollouts to intermediate checkpoints can make the same rollout budget more effective. However, since several checkpoint-level baselines underperform RSE in some settings, checkpoint timing alone is insufficient; the feedback must be specific to the receiving trajectory.

3.3 Cost Analysis

Table 2 shows that the gains of OPTA do not come from excessive controller-side computation. OPTA achieves the best score, 59.6%, with 27.1M total tokens, of which only 0.63M are controller-related tokens, corresponding to a 2.3% overhead. This indicates that checkpoint-level aggregation improves cross-trajectory information reuse with negligible coordination overhead.

Compared with GB, OPTA improves accuracy from 51.2% to 59.6% while reducing total tokens from 30.38M to 27.1M, a 10.8% reduction. It also

Method	Score $\uparrow$	Agent	Ctrl.	Total	Ctrl.%
RSE	51.2%	19.6M	0.71M	20.3M	3.5%
LeaP	40.3%	20.3M	0	20.4M	0.0%
GB	51.2%	29.57M	0.24M	30.38M	0.7%
OPTA	59.6%	25.69M	0.63M	27.1M	2.3%

Table 2: Cost-normalized comparison on BrowseComp-Plus (M = million tokens; GB = global broadcast). **Agent**: trajectory-execution tokens. **Ctrl.**: online coordination tokens (reports, feedback, probing/routing, shared-state construction and injection). **Total**: all measured tokens, also covering final aggregation and book-keeping, so $\text{Total} \geq \text{Agent} + \text{Ctrl.}$ $\text{Ctrl.\%} = \text{Ctrl.}/\text{Total}$.

Variant	Qwen		OSS		Avg.	
	Score (%)	Δ (pp)	Score (%)	Δ (pp)	Score (%)	Δ (pp)
Base agent	37.9	–	39.1	–	38.5	–
+ Structur. sum.	43.9	+6.0	46.3	+7.2	45.1	+6.6
+ Shared state	51.2	+13.3	51.8	+12.7	51.5	+13.0
OPTA	59.6	+21.7	66.3	+27.2	63.0	+24.5

Table 3: Component build-up on BrowseComp-Plus. All variants share the trajectory budget, checkpoint schedule, context-resetting scaffold, and final aggregation; Δ is the gain over **Base agent**. **Base agent**: each trajectory is rehydrated only from its own report. **+ Structur. sum.**: structured accumulated summaries. **+ Shared state**: non-selective injection of the full shared state into every subsequent trajectory. **OPTA**: adds trajectory-conditioned selective injection (full method).

uses fewer agent-execution tokens, suggesting that targeted checkpoint feedback is more cost-effective than uniform global broadcast. Although RSE uses fewer tokens, it requires costly offline construction of an experience library and runs about $2.5\times$ slower than OPTA in our experiments. LeaP has no controller-token overhead, but its limited summary exchange leads to much lower accuracy. Overall, OPTA provides a favorable high-accuracy operating point with lightweight online coordination.

3.4 Ablation Studies

Table 3 shows a progressive component build-up of OPTA on BrowseComp-Plus. Starting from the base fixed-window summarize-and-reset agent, which summarizes a trajectory when its context window is nearly full, adding structured summaries improves the average score from 38.5% to 45.1%, corresponding to a +6.6 percentage-point gain. This suggests that explicitly organizing accumulated evidence is already beneficial compared with relying on unstructured trajectory histories. Intro-

Policy	Useful $\uparrow$	Redundant	Irrelevant $\downarrow$	Harmful $\downarrow$
LeaP	12.76%	77.38%	2.33%	7.53%
RSE	18.64%	40.65%	28.31%	12.39%
GB	16.74%	76.15%	2.62%	4.48%
OPTA	20.40%	74.90%	2.00%	2.70%

Table 4: Item-level quality of injected feedback under different feedback policies. Percentages are computed over delivered atomic information items for each policy.

ducing shared state with non-selective injection further raises the average score to 51.5%, a +13.0 point improvement over the base agent, indicating that checkpoint-level information sharing across trajectories provides a substantial benefit. However, simply injecting the same shared state into every trajectory remains suboptimal. The full OPTA method, which performs trajectory-conditioned selective injection, achieves the best average score of 63.0%, improving over the base agent by +24.5 points and over non-selective injection by an additional +11.5 points. The same trend holds for both backbones: OPTA improves Qwen by +21.7 points and OSS by +27.2 points over the base agent. These results indicate that OPTA’s gains come not only from maintaining a shared state, but also from selectively routing the most relevant shared information to each trajectory, thereby reducing unnecessary context pollution while preserving useful cross-trajectory evidence.

3.5 Injection Policy Analysis

Table 4 analyzes the item-level quality of information delivered by different injection policies. While the preceding ablation shows that indiscriminately injecting all shared-state fragments weakens performance, it does not reveal whether the injected content is useful, redundant, irrelevant, or actively harmful to the receiving trajectory. We therefore evaluate injection quality at the receiver-item level.

For comparability, each delivered report, summary, shared-state fragment, or experience-pool entry is normalized into atomic information items. Native shared-state and experience-pool units are treated as atomic items, while summaries are decomposed using the same atomization prompt. A policy-blind judge labels each receiver-item pair as Useful, Redundant, Irrelevant, or Potentially harmful given the original problem, the receiver’s checkpoint report, and an anonymized item (rubric and human-audit agreement in Appendix § D).

OPTA delivers the highest Useful rate, 20.40%,

Stat.	Reasoning drift	Retrieval miss
Self-corr.	11.9%	1.5%
Events	183	235
Rel. sig.	137/183 (74.9%)	109/235 (46.4%)
Rec. rel. sig.	51/137 (37.2%) $\uparrow$	53/109 (48.6%) $\uparrow$
Rec. all	51/183 (27.9%) $\uparrow$	53/235 (22.6%) $\uparrow$

Table 5: Turning shared signals into online recovery. **Self-corr.:** recovery without cross-trajectory sharing. **Rel. sig.:** relevant signals (failure-type-specific; see text). **Rec. | rel. sig.** and **Rec. | all:** recovery rate among signal-conditioned events and over all observed failures. **Events:** diagnostic counts, not directly comparable across settings. $\uparrow$: rate exceeds the corresponding self-correction baseline.

compared with 18.64% for RSE, 16.74% for global broadcast, and 12.76% for LeaP. It also yields the lowest harmful exposure, with only 2.70% of items labeled Potentially harmful, versus 4.48% for global broadcast, 7.53% for LeaP, and 12.39% for RSE. OPTA’s Useful/Harmful ratio, 7.56, more than doubles every baseline. Trajectory-conditioned selection thus improves the precision of cross-trajectory information flow, routing actionable items to the trajectories that need them while reducing irrelevant or harmful exposure.

3.6 Mechanism Analysis

Online recovery from shared state signals. The motivation study (Figure 2) showed that intermediate failures rarely self-correct (11.9% for reasoning drift, 1.5% for retrieval miss), yet relevant peer signals exist while the failing trajectory is active. We ask whether OPTA converts these peer signals into in-trajectory recovery on the *same semantic target*, rather than only into better final answers.

Using the same failure types as in the motivation (reasoning drift, retrieval miss), we annotate each observed event for (i) whether the affected trajectory later receives a relevant shared-state or injection signal, and (ii) whether it subsequently recovers on that target. Retrieval miss is additionally conditioned on *recovery-eligible* events, where a peer cue, search hit, or evidence pointer becomes available; full protocol in Appendix § A.

Table 5 reports the result. Compared with the no-sharing self-recovery baseline, OPTA lifts reasoning-drift recovery from 11.9% to 27.9%, and retrieval-miss recovery from 1.5% to 22.6% (a 15 $\times$ gain). Conditioned on a relevant signal actually arriving, the rates rise further to 37.2% and 48.6%,

Model	Pass@10 $\uparrow$			Avg@10 $\uparrow$			MV@10 $\uparrow$		
	Standard	Ours	Δ	Standard	Ours	Δ	Standard	Ours	Δ
Qwen3-30B	66.87	67.46	+0.59	39.15	50.00	+10.85	47.59	57.83	+10.24
Qwen3-235B	77.11	76.50	-0.61	47.59	59.63	+12.04	48.80	71.10	+22.30
GPT-OSS-20B	68.07	69.27	+1.20	39.15	45.18	+6.03	46.38	64.45	+18.07
GLM-4.7-Flash	61.45	78.31	+16.86	37.34	49.39	+12.05	50.60	53.61	+3.01
Average	68.38	72.89	+4.51	40.81	51.05	+10.24	48.34	61.75	+13.41

Table 6: Trajectory-set quality across four base models: standard 10-run sampling vs. ours on Pass@10, Avg@10, and MV@10. Δ is the absolute improvement (percentage points) over the standard strategy.

indicating that recovery is driven by the routed signal rather than by unrelated trajectory dynamics. Thus shared-state aggregation is not bookkeeping for terminal selection: it returns peer evidence and cleaner semantic state to live trajectories as an on-line correction signal.

Trajectory-set coverage and quality. When injecting intermediate states into repeated sampling, a natural question is whether the additional guidance changes the behavior of the sampled trajectory set. In particular, if different samples become overly coupled, repeated sampling may no longer provide the same benefit as independent exploration. Moreover, noisy or misleading states from one trajectory may be propagated to other samples, biasing the entire trajectory set toward a suboptimal direction. We therefore compare the standard 10-run sampling strategy with OPTA using Pass@10, Avg@10, and MV@10.

As shown in Table 6, OPTA improves Avg@10 and MV@10 across all four base models, with average gains of 10.24 and 13.41 percentage points, respectively. It also improves Pass@10 on three out of four models with a positive average gain. This pattern suggests that the injected states do not merely help produce occasional successful trajectories. Instead, they improve the sampled trajectory set more broadly: individual trajectories become stronger on average, and the resulting set also supports more reliable majority voting. Overall, specificity-aware state injection improves the coverage and quality of repeated sampling without diminishing its benefit.

4 Related Work

Terminal Aggregation Test-time scaling aggregates completed trajectories via self-consistency, weighted voting, best-of- n , and verifier- or judge-based selection (Wang et al., 2022; Cobbe et al., 2021; Zheng et al., 2023; Chung et al., 2025), or

over trajectory-level Reports in long-horizon agents (Qiao et al., 2025; Li et al., 2025). Because aggregation acts only after completion, it cannot shape the search itself; we instead aggregate *during* search.

Rollout-Level Experience Reuse Completed trajectories can also be reused as shared experience for subsequent rollouts—distilled into an Experience Bank (Wang et al., 2026d) or structured state representations (Zhu et al., 2026). Feedback still arrives only between rollouts, and naive mid-trajectory synchronization is unreliable on noisy partial states (Wang et al., 2026d); Context Reset and Report Re-injection instead deliver shared information mid-rollout without destabilizing it.

Partial-Trajectory Feedback A third line uses intermediate states before termination—branch expansion in tree-structured reasoning (Yao et al., 2023; Hao et al., 2023), step scoring in process supervision (Lightman et al., 2024; Snell et al., 2024), consensus-based stopping (Zheng et al., 2026), and peer summary sharing (Luo et al., 2025)—but mainly for *control*, not for selectively enriching what each active trajectory itself reads. We instead use Two-Stage Cross-Trajectory Aggregation to selectively inject a Global Shared State into each trajectory’s own Accumulated Report. Due to space limitation, we include the complete discussion on related work in Appendix § B.

5 Conclusion

We study test-time scaling for long-horizon language agents from a single angle: turning information discovered across trajectories into on-line correction while those trajectories are still active. Our framework, checkpoint-level closed-loop aggregation, maintains a per-trajectory accumulated report, distills a global shared state, and selectively injects it back into each trajectory without overwriting its identity. On BrowseComp-Plus and DeepSearchQA, four base

models equipped with OPTA consistently outperform terminal-aggregation baselines under the same rollout budget. The shared state recovers a substantial fraction of reasoning-drift and retrieval-miss cases, and selective injection raises useful exposure while suppressing harmful exposure without collapsing diversity. The bottleneck in test-time scaling is not generating more trajectories, but using what they discover at the right time and with the right specificity.

Limitations

(1) **Task Domains.** We evaluate OPTA only on long-horizon information-seeking benchmarks (BrowseComp-Plus, DeepSearchQA). Extending checkpoint-level partial-trajectory aggregation to code agents, math-tool agents, GUI agents, and multimodal scientific research agents is left to future work; we expect the mechanism to transfer, but the schema of the shared state (supported findings, dead ends, unresolved conflicts, progress) will likely need to be retargeted to each domain’s characteristic failure modes.

(2) **Training the Controller.** The aggregation controller is a frozen prompted LLM, and OPTA is studied entirely in a training-free regime. We therefore do not explore training-based variants—fine-tuning the controller on aggregation traces, distilling it into a smaller specialized model, or jointly training the agent and controller via reinforcement learning so that the agent produces more aggregation-friendly partial trajectories. These directions should sharpen selective injection and lower coordination cost, and we leave them to future work.

Acknowledgments

This work was supported by the French National Research Agency ANR PRCI Project CyberRO (Ref: ANR-25-CE10-5230)/RGC Joint Research Scheme (Ref: A-HKU702/25) and the Mainland-Hong Kong Joint Funding Scheme, Innovation and Technology Fund (Ref: MHP/192/23). We also acknowledge the Information Technology Services (ITS) at The University of Hong Kong for providing computational resources and support.

Use of AI Assistant We used an AI assistant, ChatGPT, solely for language polishing, including improving clarity, flow, and readability. The AI assistant was not used for data analysis, experiments,

method design, or idea generation. All research design, methodology, results, and conclusions were developed by the authors. The authors reviewed and edited all AI-assisted text and take full responsibility for the accuracy, originality, and integrity of the manuscript.

References

- Zijian Chen, Xueguang Ma, Shengyao Zhuang, Ping Nie, Kai Zou, Andrew Liu, Joshua Green, Kshama Patel, Ruoxi Meng, Mingyi Su, Sahel Shari-fymoghaddam, Yanxi Li, Haoran Hong, Xinyu Shi, Xuye Liu, Nandan Thakur, Crystina Zhang, Luyu Gao, Wenhui Chen, and Jimmy Lin. 2025. [BrowseComp-Plus: A more fair and transparent evaluation benchmark of deep-research agent](#). *Preprint*, arXiv:2508.06600.
- Ho-Lam Chung, Teng-Yun Hsiao, Hsiao-Ying Huang, Chunerh Cho, Jian-Ren Lin, Zhang Ziwei, and Yun-Nung Chen. 2025. Revisiting test-time scaling: A survey and a diversity-aware method for efficient reasoning. *arXiv preprint arXiv:2506.04611*.
- Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, et al. 2021. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*.
- Nikita Gupta, Riju Chatterjee, Lukas Haas, Connie Tao, Andrew Wang, Chang Liu, Hidekazu Oiwa, Elena Gribovskaya, Jan Ackermann, John Blitzer, Sasha Goldshtein, and Dipanjan Das. 2026. [DeepSearchQA: Bridging the comprehensiveness gap for deep research agents](#). *Preprint*, arXiv:2601.20975.
- Shibo Hao, Yi Gu, Haodi Ma, Joshua Hong, Zhen Wang, Daisy Wang, and Zhiting Hu. 2023. Reasoning with language model is planning with world model. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 8154–8173.
- Baixuan Li, Dingchu Zhang, Jialong Wu, Wenbiao Yin, Zhengwei Tao, Yida Zhao, Liwen Zhang, Haiyang Shen, Runnan Fang, Pengjun Xie, Jingren Zhou, and Yong Jiang. 2025. Parallelmuse: Agentic parallel thinking for deep information seeking. *arXiv preprint arXiv:2510.24698*.
- Hunter Lightman, Vineet Kosaraju, Yuri Burda, Harrison Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. 2024. Let’s verify step by step. In *International Conference on Learning Representations*, volume 2024, pages 39578–39601.
- Tongxu Luo, Wenyu Du, Jiayi Bi, Stephen Chung, Zhengyang Tang, Hao Yang, Min Zhang, and Benyou Wang. 2025. Learning from peers in reasoning models. *arXiv preprint arXiv:2505.07787*.

- Ruotian Ma, Peisong Wang, Cheng Liu, Xingyan Liu, Jiaqi Chen, Bang Zhang, Xin Zhou, Nan Du, and Jia Li. 2025. S2R: Teaching LLMs to self-verify and self-correct via reinforcement learning. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 22632–22654.
- OpenAI. 2025. [gpt-oss-120b & gpt-oss-20b Model Card](#). Preprint, arXiv:2508.10925.
- Zile Qiao, Guoxin Chen, Xuanzhong Chen, Donglei Yu, Wenbiao Yin, Xinyu Wang, Zhen Zhang, Baixuan Li, Huifeng Yin, Kuan Li, et al. 2025. Webresearcher: Unleashing unbounded reasoning capability in long-horizon agents. *arXiv preprint arXiv:2509.13309*.
- Charlie Snell, Jaehoon Lee, Kelvin Xu, and Aviral Kumar. 2024. Scaling llm test-time compute optimally can be more effective than scaling model parameters. *arXiv preprint arXiv:2408.03314*.
- GLM Team, Aohan Zeng, Xin Lv, Qinkai Zheng, Zhenyu Hou, Bin Chen, Chengxing Xie, Cunxiang Wang, Da Yin, Hao Zeng, Jiajie Zhang, Kedong Wang, Lucen Zhong, Mingdao Liu, Rui Lu, Shulin Cao, Xiaohan Zhang, Xuancheng Huang, Yao Wei, and 152 others. 2025. [GLM-4.5: Agentic, reasoning, and coding \(arc\) foundation models](#). Preprint, arXiv:2508.06471.
- Peisong Wang, Bowen Liu, Zehua Li, Yuyao Wang, Zhiwei Ma, Yuhua Li, and Jia Li. 2026a. CP-Agent: A calibrated risk-controlled agent for feedback-driven competitive programming. *arXiv preprint arXiv:2605.24693*.
- Peisong Wang, Ruotian Ma, Bang Zhang, Xingyu Chen, Zhiwei He, Kang Luo, Qingsong Lv, Qingxuan Jiang, Zheng Xie, Shanyi Wang, et al. 2026b. RLVER: Reinforcement learning with verifiable emotion rewards for empathetic agents. In *International Conference on Learning Representations*, volume 2026, pages 28517–28545.
- Peisong Wang, Zhiwei Ma, Bowen Liu, Feixue Liu, Aochuan Chen, Chenyi Zi, Hongchuan Zeng, Yuhua Li, and Jia Li. 2026c. R^3 -Bench: LLMs struggle with resource-rational reasoning under shared budgets. *arXiv preprint arXiv:2608.16033*.
- Xinglin Wang, Jiayi Shi, Shaoxiong Feng, Peiwen Yuan, Yiwei Li, Yueqi Zhang, Chuyi Tan, Ji Zhang, Boyuan Pan, Yao Hu, et al. 2026d. Do not waste your rollouts: Recycling search experience for efficient test-time scaling. *arXiv preprint arXiv:2601.21684*.
- Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc Le, Ed Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. 2022. Self-consistency improves chain of thought reasoning in language models. *arXiv preprint arXiv:2203.11171*.
- An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, Chujie Zheng, Dayiheng Liu, Fan Zhou, Fei Huang, Feng Hu, Hao Ge, Haoran Wei, Huan Lin, Jialong Tang, and 41 others. 2025. [Qwen3 Technical Report](#). *arXiv preprint arXiv:2505.09388*.
- Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Tom Griffiths, Yuan Cao, and Karthik Narasimhan. 2023. Tree of thoughts: Deliberate problem solving with large language models. *Advances in neural information processing systems*, 36:11809–11822.
- Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric Xing, et al. 2023. Judging llm-as-a-judge with mt-bench and chatbot arena. *Advances in neural information processing systems*, 36:46595–46623.
- Tong Zheng, Chengsong Huang, Runpeng Dai, Yun He, Rui Liu, Xin Ni, Huiwen Bao, Kaishen Wang, Hongtu Zhu, Jiaxin Huang, et al. 2026. Parallel-probe: Towards efficient parallel thinking via 2d probing. *arXiv preprint arXiv:2602.03845*.
- Jialiang Zhu, Gongrui Zhang, Xiaolong Ma, Lin Xu, Miaosen Zhang, Ruiqi Yang, Song Wang, Kai Qiu, Zhirong Wu, Qi Dai, et al. 2026. Re-trac: Recursive trajectory compression for deep search agents. *arXiv preprint arXiv:2602.02486*.

A Online Recovery Annotation Protocol

We annotate online recovery at the level of semantic targets. Each intermediate failure event is associated with a trajectory, a checkpoint, a failure type, and the semantic target on which the failure occurs. We use the same two failure categories as in the motivation study: reasoning drift and retrieval miss.

A feedback item is counted as relevant only if it is introduced after the failure event and is related to the same semantic target. Relevant feedback may take the form of a shared-state aggregation item, a selective-injection item, a peer search hit, a missing-evidence pointer, a dead-end warning, or a conflict signal that could plausibly repair the failure.

For reasoning drift, an event is counted as recovered if the same trajectory later corrects or abandons the erroneous state on the same semantic target. This includes cases where the trajectory revises an incorrect constraint application, unbinds evidence from the wrong candidate, rejects an unsupported hypothesis, or replaces a faulty intermediate state with a better-supported one.

For retrieval miss, an event is counted as recovered if the same trajectory later retrieves, uses, or correctly incorporates the missing evidence after receiving a relevant shared-state cue, peer search hit, or evidence pointer. Because a retrieval miss can only be repaired when the missing evidence or a pointer to it becomes available, we report retrieval-miss recovery over recovery-eligible events. A retrieval miss is recovery-eligible if a relevant shared-state cue, peer search hit, or evidence pointer becomes available after the miss.

In the 300 trajectories generated by our method, we identify 183 reasoning drift events and 235 retrieval miss events. For reasoning drift, 137 out of 183 events receive a relevant shared-state aggregation or selective-injection item after the drift occurs. Among these feedback-conditioned drift events, 51 are later corrected, giving a conditional recovery rate of 37.2%. Measured over all observed reasoning drifts, 51 out of 183 events are recovered, giving an overall recovery rate of 27.9%. Both rates are higher than the 11.9% self-recovery rate observed in the no-sharing diagnostic.

For retrieval misses, 109 out of 235 observed events are recovery-eligible because a relevant shared-state cue, peer search hit, or evidence pointer becomes available. Among these eligible

events, 53 are recovered, giving a recovery rate of 48.6%. Measured over all observed retrieval misses, 53 out of 235 events are recovered, giving an overall recovery rate of 22.6%. This contrasts with the 1.5% self-recovery rate in the no-sharing diagnostic.

These raw event counts are used only for online recovery diagnostics. They are not interpreted as directly comparable failure rates across settings, because the analysis focuses on whether relevant feedback, once available to a live trajectory, leads to subsequent recovery.

B Complete Discussion on Related Work

Parallel trajectory search and terminal aggregation. Test-time scaling commonly improves reasoning by sampling multiple trajectories and aggregating their completed outputs. Self-consistency and majority voting aggregate final answers across sampled reasoning paths (Wang et al., 2022), while weighted voting, best-of- n , verifier-based selection, and judge-based selection score or select among completed candidates (Cobbe et al., 2021; Zheng et al., 2023; Chung et al., 2025). More broadly, recent work has also studied how reasoning agents allocate test-time computation under shared inference budgets (Wang et al., 2026c). In long-horizon agentic tasks, terminal aggregation can also operate over trajectory-level Reports rather than answer strings (Qiao et al., 2025; Li et al., 2025). For example, WebResearcher maintains report-like representations of long-horizon exploration, and a final aggregator can compare completed reports and candidate answers to select the best-supported output. These methods exploit diversity across parallel trajectories, but their aggregation point is terminal: the aggregate can improve final selection, but cannot affect the search process that produced the trajectories.

Rollout-level experience aggregation and reuse.

Another line of work reuses completed trajectories as experience for later attempts. RSE distills completed rollouts into a shared Experience Bank containing useful conclusions and failed directions, then conditions later rollout batches on this memory (Wang et al., 2026d). RE-TRAC similarly converts completed trajectories into structured state representations that summarize evidence, uncertainty, failures, and future plans, and uses them to guide subsequent trajectories (Zhu et al., 2026). Compared with terminal aggregation, rollout-level

reuse allows earlier trajectories to influence later ones. However, the feedback still occurs only after a rollout or batch has finished, so it cannot correct an active trajectory while it is drifting. RSE also notes that mid-trajectory synchronization is difficult because partial states are unreliable to evaluate and global updates may perturb the conditioning context or increase output variance (Wang et al., 2026d). This motivates a more selective form of aggregate-while-searching: shared experience should be available earlier, but should not be indiscriminately broadcast to every trajectory.

Online partial-trajectory aggregation and checkpoint feedback. Several methods use intermediate trajectory states or feedback before termination. Tree-structured reasoning and planning methods evaluate partial states to decide which branches to expand (Yao et al., 2023; Hao et al., 2023), while process-supervision and verifier-guided methods score intermediate steps for pruning, reranking, or budget allocation (Lightman et al., 2024; Snell et al., 2024). Related feedback-driven approaches also use self-verification, execution feedback, or verifiable interaction signals to refine agent behavior during reasoning or interaction (Ma et al., 2025; Wang et al., 2026a,b). Parallel-Probe periodically elicits intermediate answers from active branches for consensus-based early stopping and deviation-based pruning (Zheng et al., 2026), and online-pruning baselines similarly use checkpoint signals to decide which trajectories should continue. LeaP allows reasoning paths to summarize intermediate reasoning and share peer information during inference (Luo et al., 2025). These methods show that partial-trajectory information can be useful before termination, but they mainly use it for control, pruning, routing, feedback-driven refinement, or summary sharing. Our work instead studies Online Partial-Trajectory Aggregation for long-horizon search: during Parallel Trajectory Search, each trajectory maintains an Accumulated Report; Two-Stage Cross-Trajectory Aggregation first constructs a Global Shared State and then performs Trajectory-Specific Selective Injection to produce an Enriched Accumulated Report. Through Context Reset and Report Re-injection, selected cross-trajectory information becomes actionable during search while preserving trajectory-specific exploration. Final aggregation is still performed at the trajectory level after completion.

C Temperature Setting and Trajectory Diversity

We use a sampling temperature of $T = 0.2$ throughout our experiments. This relatively low temperature is chosen to balance trajectory diversity and reasoning reliability in long-horizon agent execution. Although a higher temperature can increase token-level sampling diversity, it may also introduce noisy generations whose effects accumulate over long reasoning and tool-interaction sequences. In long-horizon search, trajectory diversity does not rely solely on sampling temperature: small differences in intermediate reasoning can lead trajectories to formulate different search queries, retrieve different documents, and subsequently follow increasingly different reasoning paths. We therefore use $T = 0.2$ to limit excessive sampling noise while preserving the trajectory-level exploration induced by long-horizon search and tool interaction.

To verify that this setting still provides sufficient diversity for parallel test-time scaling, we analyze the resulting rollouts using Qwen3-30B on BrowseComp-Plus. Across ten rollouts, 79.7% of the questions produce at least two distinct normalized final answers, with an average of 3.35 distinct answers per question. We further examine diversity in the underlying retrieval paths. The average pairwise Jaccard distance is 0.67 for search-query sets and 0.495 for opened-document sets, showing substantial variation in both query formulation and evidence selection across trajectories.

These results indicate that the relatively low temperature does not collapse the rollouts into near-identical trajectories. Instead, $T = 0.2$ limits unnecessary sampling noise while maintaining meaningful diversity at both the final-answer and retrieval-path levels.

D LLM-as-Judge Annotation for injection policy analysis

We use a rubric-based LLM-as-judge protocol to label feedback-item relevance while minimizing bias toward our method. The judge is *policy-blind*: it is never told whether an atomic feedback item is selected by our method, produced by global broadcast, sampled by random injection, or generated by a baseline such as peer-summary routing. Instead, each annotation instance contains only the original problem, the receiving trajectory’s accumulated report at the checkpoint, and an anonymized atomic feedback item. The judge is also not given the final

answer, future trajectory states, or the eventual success of any method, so labels are checkpoint-local rather than hindsight-based.

We normalize all delivered feedback into atomic information items before relevance labeling. For our method and global broadcast, the delivered units are already typed shared-state fragments and are treated as atomic items. For the RSE baseline, we use its experience-bank entries as atomic feedback items. For LeaP peer routing baseline, which delivers longer peer summaries, we decompose each summary into atomic items using the same atomization prompt. All resulting receiver-item pairs are then labeled with the same fixed rubric: *Useful*, *Redundant*, *Irrelevant*, and *Potentially harmful*.

Policy-level statistics are computed only after annotation by mapping the blind labels back to the items delivered by each feedback policy. This ensures that the same label table is used to evaluate all policies and that the judge cannot favor our selective-injection policy during labeling.

To validate the annotation quality, we manually audit a random subset of 100 labeled receiver-item pairs sampled across item types, checkpoints, and feedback policies. For each audited example, a human annotator independently applies the same rubric without seeing the LLM judge’s label or the originating feedback policy. We first report four-way exact agreement. For each receiver-item pair, both the LLM judge and the human annotator assign one of four labels: *Useful*, *Redundant*, *Irrelevant*, or *Potentially harmful*. An audited example is counted as an exact agreement only when the two labels are identical. Formally, if $\hat{y}_i$ is the LLM label and y_i is the human label for item i , the four-way exact agreement is

$$\text{ExactAgreement} = \frac{1}{N} \sum_{i=1}^N \mathbf{1}[\hat{y}_i = y_i].$$

The resulting four-way exact agreement is 88%.

Because our downstream analysis is especially sensitive to which feedback items are labeled *Useful* or *Potentially harmful*, we also report label-specific overlaps. Let

$$\begin{aligned} \mathcal{U}_{\text{LLM}} &= \{i : \hat{y}_i = \text{Useful}\}, \\ \mathcal{U}_{\text{Human}} &= \{i : y_i = \text{Useful}\}. \end{aligned}$$

We define Useful-label overlap as the Jaccard overlap

$$\text{UsefulOverlap} = \frac{|\mathcal{U}_{\text{LLM}} \cap \mathcal{U}_{\text{Human}}|}{|\mathcal{U}_{\text{LLM}} \cup \mathcal{U}_{\text{Human}}|}.$$

Audit statistic	Value
Audited receiver-item pairs	100
Four-way exact agreement	88%
Useful-label Jaccard overlap	88.67%
Potentially-harmful-label Jaccard overlap	92.7%

Table 7: Human audit of feedback-item relevance labels.

Four-way exact agreement counts an item as correct only when the LLM judge and the human annotator assign the same label among *Useful*, *Redundant*, *Irrelevant*, and *Potentially harmful*. Useful-label and Potentially-harmful-label overlaps are Jaccard overlaps between the LLM-labeled and human-labeled item sets for the corresponding label.

Artifact	License / Terms
BrowseComp-Plus	MIT
DeepSearchQA	Apache-2.0
Qwen3-30B-A3B-Instruct-2507	Apache-2.0
gpt-oss-20B	Apache-2.0 + gpt-oss usage policy
GLM-4.7-Flash	MIT
Qwen3-235B-A22B-Instruct-2507	Apache-2.0

Table 8: Licenses and terms of use for the benchmarks and models used in this work.

The Useful-label overlap is 88.67%. We compute the Potentially-harmful-label overlap analogously over the sets of items labeled *Potentially harmful* by the LLM judge and by the human audit; this overlap is 92.7%.

E Artifact Licenses and Intended Use

We use two public benchmarks and four open-weight language models in our experiments. Table 8 summarizes the licenses and terms of use of these artifacts. All artifacts are used solely for research and evaluation purposes. We do not redistribute the benchmark data, model weights, or any raw web documents from the benchmark corpora.

Benchmarks. BrowseComp-Plus (Chen et al., 2025) is used as a controlled deep-research benchmark for evaluating retrieval-augmented agents. BrowseComp-Plus is released under the MIT License. Since its accompanying corpus consists of curated web documents, we use it only for evaluation and do not redistribute the corpus. DeepSearchQA (Gupta et al., 2026) is used as a multi-step information-seeking benchmark for evaluating deep-research agents. DeepSearchQA is released under the Apache License 2.0. Our use of both benchmarks is consistent with their intended evaluation purposes.

Models. We evaluate four open-weight language models: Qwen3-30B-A3B-Instruct-2507 (Yang et al., 2025), gpt-oss-20B (OpenAI, 2025), GLM-4.7-Flash (Team et al., 2025), and Qwen3-235B-A22B-Instruct-2507 (Yang et al., 2025). Qwen3-30B-A3B-Instruct-2507 and Qwen3-235B-A22B-Instruct-2507 are released under the Apache License 2.0. gpt-oss-20B is released under the Apache License 2.0 and is additionally subject to the gpt-oss usage policy. GLM-4.7-Flash is released under the MIT License. We use these models only for inference-based evaluation and do not redistribute their weights.

F Prompts for Two-Stage Cross-Trajectory Aggregation

We provide the complete prompts used by the aggregation controller in the two stages of OPTA. Stage 1 constructs the global shared state from the accumulated reports of all active trajectories, while Stage 2 performs trajectory-specific selective injection based on each trajectory’s private accumulated report.

F.1 Global Shared-State Construction

At each aggregation checkpoint, the controller receives the accumulated reports from all active trajectories and constructs the typed global shared state described in Section 2. The exact prompt used in our implementation is provided below.

```
You are the cross-trajectory aggregator. {
  sample_count} independent trajectories
have been working in parallel on the
same question. Each trajectory has just
produced its latest *accumulated
progress report* (covering everything it
has discovered up to chunk round {
  round_idx}).
```

```
**Original Question:**
{question}
```

```
**Your job:** Read every trajectory's
accumulated report and synthesise a
single SHARED STATE that captures, in a
structured way:
1. Findings or intermediate reasoning
results that are well-supported.
2. Dead ends (entities / paths /
hypotheses that have been definitively
ruled out).
3. Unresolved conflicts (claims that
genuinely disagree across trajectories
and still need verification).
```

4. A short **progress summary** (overall status + a 1 to 2 sentence note for each trajectory).

You must produce exactly the XML structure shown below. No prose outside the XML block.

Hard rules:

- Use ONLY information present in the provided accumulated reports. Do not invent findings, sources, URLs, tool outputs, calculations, or reasoning steps.
- An item belongs to `supported_findings` only if at least one trajectory provides concrete support for it. Support may be source evidence (a URL, search result, visited page content, PDF text, or tool output) OR reasoning evidence (a calculation, derivation, code/output trace, constraint check, or prompt-grounded inference). Mere unsupported assertion does NOT qualify.
- A target/path goes into `dead_ends` only if some trajectory has shown it definitively contradicts a hard constraint of the question, OR multiple trajectories independently established it as a wrong lead. Do NOT put a path here just because it is "weak". Cite which trajectories established it as dead.
- A disagreement goes into `unresolved_conflicts` only if (a) trajectories make competing claims about the same target/attribute AND (b) the conflict cannot be settled from the existing reports alone. If one side is clearly supported by strong evidence and the other is merely an unsupported assumption, do not create an unresolved conflict. Omit the unsupported claim unless it independently satisfies the dead-end criteria above.
- Each item must list the trajectory ids that contributed it (e.g. `<source_trajectories>[1, 3]</source_trajectories>`). Use 1-indexed ids exactly as labelled in the input.
- Keep every item short and grounded. Do not summarise the whole reports only extract reusable building blocks, including useful intermediate reasoning, that another trajectory could benefit from.
- If a bucket has no qualifying items, emit it as empty (e.g. `<dead_ends></dead_ends>`). Do not invent items just to fill it.

Below are the {sample_count} accumulated reports:

```
{trajectory_reports}
```

Return your answer in EXACTLY this XML format (no extra commentary):

```
<shared_state>
  <supported_findings>
    <finding>
      <id>SF1</id>
      <statement>...</statement>
      <evidence>Source URL/tool output/
calculation/derivation/constraint check
supporting the statement.</evidence>
      <source_trajectories>[...]</
source_trajectories>
    </finding>
    <!-- more <finding> blocks as needed;
omit entirely if none qualify -->
  </supported_findings>
  <dead_ends>
    <dead_end>
      <id>D1</id>
      <entity_or_path>...</entity_or_path>
      <reason>...</reason>
      <discovered_by>[...]</discovered_by>
    </dead_end>
    <!-- more <dead_end> blocks as needed -->
  </dead_ends>
  <unresolved_conflicts>
    <conflict>
      <id>C1</id>
      <issue>...</issue>
      <competing_views>...</competing_views>
      <verification_needed>...</
verification_needed>
      <involved_trajectories>[...]</
involved_trajectories>
    </conflict>
    <!-- more <conflict> blocks as needed -->
  </unresolved_conflicts>
  <progress_summary>
    <overall>One concise paragraph on where
the swarm collectively stands.</overall>
    <per_trajectory>
      <traj id="1">1 to 2 sentences on what
this trajectory is currently pursuing
and how far it has gotten.</traj>
      <!-- one <traj> block per alive
trajectory -->
    </per_trajectory>
  </progress_summary>
</shared_state>
```

E.2 Trajectory-Specific Selective Injection

Given the global shared state and the target trajectory's own accumulated report, the controller selects only the shared information that is useful for that trajectory's subsequent search. Importantly, the controller is allowed to inject no information when the shared state provides no actionable benefit. The target trajectory's original accumulated report is preserved verbatim, and selected cross-trajectory information is added only as an auxiliary section.

You are deciding which pieces of the SHARED STATE (built from all alive trajectories) should be injected into trajectory #{trajectory_id}'s next-session opening context, as additions to its own accumulated report.

****Original Question:****
{question}

****Trajectory #{trajectory_id}'s own accumulated report (this trajectory's private state):****
{own_report}

****Shared State (synthesised across all alive trajectories at chunk round {round_idx}):****
{shared_state_xml}

****Your task:**** Choose ONLY the shared-state items that are likely USEFUL for trajectory #{trajectory_id} to know going into its next session. Then emit an *enriched accumulated report* that the trajectory will see as its new starting context.

****Selection rules:****

1. Inject a ``<finding>`` ONLY if it gives this trajectory supported information or a useful reasoning result it does not already have, or directly resolves an open question it was investigating. Do NOT inject findings that this trajectory already knows.
2. Inject a ``<dead\_end>`` ONLY if this trajectory is currently pursuing (or might soon pursue) that path/entity. Use this to actively warn the trajectory away from a confirmed wrong lead. Do NOT inject dead ends about paths this trajectory has clearly abandoned or never considered.
3. Inject an ``<unresolved\_conflict>`` ONLY if this trajectory is well placed to help resolve it (e.g., it is investigating one side of the conflict, or it has tools/leads relevant to verification).
4. Inject the ``<overall>`` progress summary ONLY if it materially changes this trajectory's situational awareness (e.g., it shows that the swarm has converged on a different region of the search space). Otherwise omit it. NEVER inject other trajectories' per-trajectory summaries they are noise to this trajectory.
5. ****It is perfectly valid to inject NOTHING.** If the shared state offers no actionable benefit to this trajectory, return its original accumulated report

unchanged.

6. NEVER tell the trajectory which entity to commit to. The injected content is supplemental evidence or reasoning support, not orders.
7. NEVER paraphrase, weaken, or strengthen the original accumulated report. The trajectory's own findings must be preserved verbatim. You may only ADD a clearly demarcated "Cross-Trajectory Aggregator Notes" section.

****Output format:****

- First, list which shared-state ids you chose to inject (or ``[]`` if none) inside the ``<injection\_decision>`` block.
- Then emit the full ``<enriched\_accumulated\_report>`` block that the trajectory will see. It must contain:
 - a. The trajectory's own accumulated report, ****unchanged****, under the heading ``## Your Own Accumulated Progress`` (verbatim copy of the input own report).
 - b. A clearly separated section ``## Cross-Trajectory Aggregator Notes`` that contains ONLY the injected items (supported findings, dead-end warnings, unresolved conflicts that this trajectory could help resolve, and optionally the overall summary). If you injected nothing, emit this section with the literal text ``(No cross-trajectory notes to inject for this trajectory at this round.)``.

Return your answer in EXACTLY this XML format (no extra commentary outside it):

```

<injection_decision>
  <injected_findings>[...ids or empty list]</
  injected_findings>
  <injected_dead_ends>[...ids or empty list
  ]</injected_dead_ends>
  <injected_conflicts>[...ids or empty list
  ]</injected_conflicts>
  <injected_overall>true|false</
  injected_overall>
  <reasoning>One short paragraph: why these
  items (and only these) help this
  specific trajectory.</reasoning>
</injection_decision>
<enriched_accumulated_report>
## Your Own Accumulated Progress

<verbatim copy of the input own report>

## Cross-Trajectory Aggregator Notes

<only the injected items, neatly formatted as
  bullet points; or the literal "(No
  cross-trajectory notes to inject for
  this trajectory at this round.)" if none>

</enriched_accumulated_report>

```