CoEvoke: Evoking Multi-Agent Collaboration for Trajectory Collection

Anonymous authors
Paper under double-blind review

Abstract

Training multi-agent collaboration requires trajectories of checking, feedback, and repair, yet task-solving rollouts may end in immediate approval or repeat unsuccessful attempts without useful correction. We introduce CoEvoke, a cross-domain framework that evokes these interactions during trajectory collection. Controlled perturbations create recoverable difficulties, while restricted reference-based guidance directs investigation. The resulting dataset contains 4,567 reconstructable trajectories over 4,130 software-engineering and reasoning tasks, linking public interactions to evidence, decisions, and task-native outcomes. Separate experiments over 2,287 same-task pairs show feedback increases of 47.72–100 percentage points across seven perturbation groups; guidance has smaller, model-dependent effects. With identical training tasks, decision-record counts, and updates, perturbation-enriched judge training improves ACPBench-Hard success from 35.38% to 42.12% (+6.73 points). SWE verifier training gains 7.60–9.40 points over interface-matched untrained references. Workflow and post-training evaluations preserve task-solving topology with unperturbed inputs and no private-reference guidance. These results support designing collection conditions to obtain corrective supervision that improves collaborative decisions with frozen solving agents.

1 Introduction

Large language models (LLMs) support tool-using agents that act, observe consequences, and choose subsequent actions (Liu et al., 2024; Wang et al., 2024). Multi-agent systems (MAS) coordinate specialized agents through delegation, discussion, and mutual review (Li et al., 2023; Wu et al., 2024; Qian et al., 2025). Success depends on exchanging evidence, evaluating intermediate work, and deciding when to revise or stop (Du et al., 2024; Dang et al., 2025). Learning these decisions at scale requires diverse trajectories linking actions and information exchanges to feedback, revisions, and outcomes (Motwani et al., 2025; Zeng et al., 2024), motivating automated collection across tasks and collaboration patterns with varying demands for checking, feedback, and repair.

However, task-solving rollouts do not guarantee trajectories for learning useful collaboration. Prior work uses agent trajectories for refinement, multi-agent training, and distillation (Fu et al., 2025; Motwani et al., 2025; Luo et al., 2026; Li et al., 2025), but such rollouts can omit the error diagnosis, targeted feedback, and revision choices that collaborating agents need to learn. An immediately correct answer bypasses diagnosis and repair; unsuccessful teams may repeat searches and revisions without a productive direction (Ma et al., 2026). More rollouts therefore need not teach agents how to turn targeted feedback into useful correction.

In human collaboration, an inconsistency can evoke cross-checking, while a focused suggestion directs teammates toward useful evidence. Inspired by this pattern, we create recoverable difficulties and provide restricted guidance for investigation. These controls give agents a reason to check and a direction for repair, retaining task objectives and repair capacity.

We introduce **CoEvoke**[1] (Collaboration + Evoke), an automated cross-domain framework for collecting reconstructable multi-agent trajectories. It perturbs an initial candidate, observation, or model

[1]We provide an anonymous repository of the CoEvoke framework for inspecting the implementation and running trajectory-collection workflows: https://anonymous.4open.science/r/CoEvoke/. Appendix J describes the reproduction materials and the models, tools, and task environments required to rerun a workflow.

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

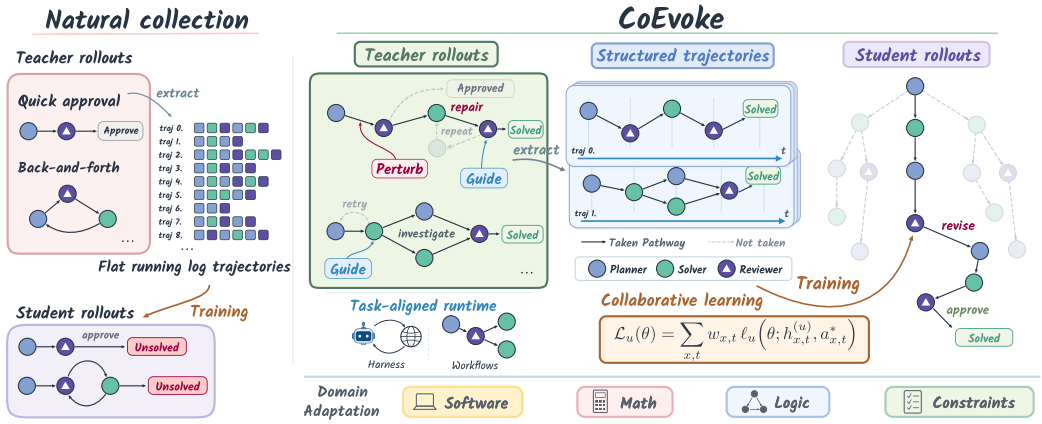

Figure 1: During collection, Perturb evokes checking and repair; Guide directs investigation. Circles denote role-colored agent turns (triangle: decision); squares denote log entries. Solid/dashed paths are taken/untaken; t denotes time. Stacked trajectories retain interaction links and supply public-context/target-action pairs for collaborative training (Eq. 4). Student evaluation preserves workflows without perturbations or private-reference guidance.

assignment to evoke checking and repair, and provides restricted reference-based guidance to direct investigation. We call these operations *Perturb-and-Guide*. The resulting interactions expose judgment, criticism, and verification for training control roles that assess intermediate work and route execution; solving agents remain frozen during this supervised training stage (Figure 1).

We evaluate CoEvoke in four domains requiring trajectory supervision for learning collaboration: software engineering, mathematics, logical entailment, and constraint solving. Source identities, role/tool events, visibility labels, and outcome links support reconstruction and training; human-calibrated audits assess quality. Mechanism and training experiments test whether collection exposes useful correction and improves learned decisions in downstream workflows.

Our contributions are threefold:

- We introduce **CoEvoke**, a cross-domain framework for collecting reconstructable multi-agent trajectories through Perturb-and-Guide. It creates recoverable difficulties and directs investigation, exposing checking, feedback, and repair that task-solving rollouts may omit.
- We construct a **dataset of 4,567 CoEvoke trajectories across 4,130 tasks**, linking public role and tool events, feedback and revisions, and evaluator outcomes. These links support inspection of the recorded public collaboration and extraction of decision records[2] for training control roles to decide between candidate acceptance and further repair.
- We demonstrate the **training utility of CoEvoke trajectories with frozen solvers**. Matched-data judge training improves ACPBench-Hard success by 6.73 percentage points; interface-matched verifier training gains 7.60–9.40 points on SWE-bench Verified. Evaluation uses unperturbed inputs without private-reference guidance.

2 Related Work

2.1 From Reasoning to Coordinated Agents

Reasoning extends intermediate-step generation (Wei et al., 2022) through candidate aggregation (Wang et al., 2023a) and tool interaction (Yao et al., 2023; Schick et al., 2023), whose evidence

[2] A trajectory is one run's retained public event sequence, with messages, tool observations, and outcome links. It can yield several decision records, each pairing the public context available at one decision with its target action; later repairs and evaluator outcomes are excluded from that input. Trajectory and record counts therefore need not match (Appendices B.1 and G.1).

interfaces affect execution (Yang et al., 2024). CAMEL studies role-playing cooperation, while AutoGen, MetaGPT, and ChatDev organize communication and task decomposition (Li et al., 2023; Wu et al., 2024; Hong et al., 2024; Qian et al., 2024). Debate, scalable collaboration graphs, and learned orchestration address candidate comparison and agent sequencing (Du et al., 2024; Qian et al., 2025; Dang et al., 2025). CoEvoke studies how collection conditions in these workflows evoke checking and repair.

2.2 Synthetic Data and Agent Trajectory Generation

As synthetic supervision moves from instructions and rationales to executable interactions (Wang et al., 2023b; Zelikman et al., 2022; Zeng et al., 2024), collection becomes a design problem. AgentTrek uses tutorials to guide execution, APIGen-MT builds verifiable task blueprints, and SWE-smith generates repository tasks through test-breaking changes (Xu et al., 2025; Prabhakar et al., 2025; Yang et al., 2025). Agent Data Protocol complements generation by unifying heterogeneous trajectories for fine-tuning (Song et al., 2026).

AgentArk and Agent Foundation Models distill team behavior into individual models (Luo et al., 2026; Li et al., 2025). AgentRefine synthesizes environments for corrective supervision (Fu et al., 2025), while MALT trains role-specific models through generator–verifier–refiner trees and propagated rewards (Motwani et al., 2025); CoEvoke instead intervenes on existing tasks to collect trajectories for training control roles with frozen solvers (Appendix A).

2.3 Feedback, Verification, and Process Evaluation

Self-refinement, remembered reflections, and tool-assisted critique supply feedback (Madaan et al., 2023; Shinn et al., 2023; Gou et al., 2024), but unaided self-correction can fail (Huang et al., 2024; Stechly et al., 2025); SCoRe learns correction through on-policy interaction (Kumar et al., 2025). Learned verifiers and process supervision distinguish answer quality from reasoning quality (Cobbe et al., 2021; Uesato et al., 2022; Lightman et al., 2024), motivating separate measures of multi-turn reliability and coordination (Laban et al., 2026; Zhu et al., 2025).

AgenTracer injects faults for attribution training, whereas DoVer tests debugging hypotheses by re-execution (Zhang et al., 2026; Ma et al., 2026). CoEvoke uses interventions to collect feedback-and-repair interactions, assessing outcomes with task-native evaluators (Jimenez et al., 2024; Wang et al., 2024; Yao et al., 2025) and process quality with human-calibrated audits (Zheng et al., 2023; Liu et al., 2023; Kim et al., 2024). Agent approval and task success remain distinct evidence.

3 CoEvoke: Evoking Collaboration during Collection

To collect trajectories that expose decisions absent from natural execution, CoEvoke creates recoverable difficulties and directs investigation. Figure 2 summarizes the design: retain the task-solving workflow, adjust its collection conditions, and preserve evidence-to-decision links for training.

3.1 Tasks and Workflows

Useful correction must remain anchored to an independently checkable task. We therefore represent a task as $x = (c_x, z_x, E_x)$, with public context c_x, optional private references z_x, and a task-native evaluator E_x. Its workflow $G_x = (V_x, A_x)$ specifies roles and permitted communication routes. Only a designated collection node may read z_x; an existing critic or judge can serve this role without adding another agent.

Different uncertainties call for different coordination patterns (Chen et al., 2024; Motwani et al., 2025; Du et al., 2024). Software repair uses planning, coding, and review on SWE-smith (Yang et al., 2025); mathematical reasoning returns DeepMath critiques to a solver (He et al., 2025); candidate comparison resolves ProofWriter and Zebra disagreements through independent debaters, a judge, and targeted revision (Tafjord et al., 2021; Lin et al., 2025). In every workflow, task-native evaluation independently scores the final solution.

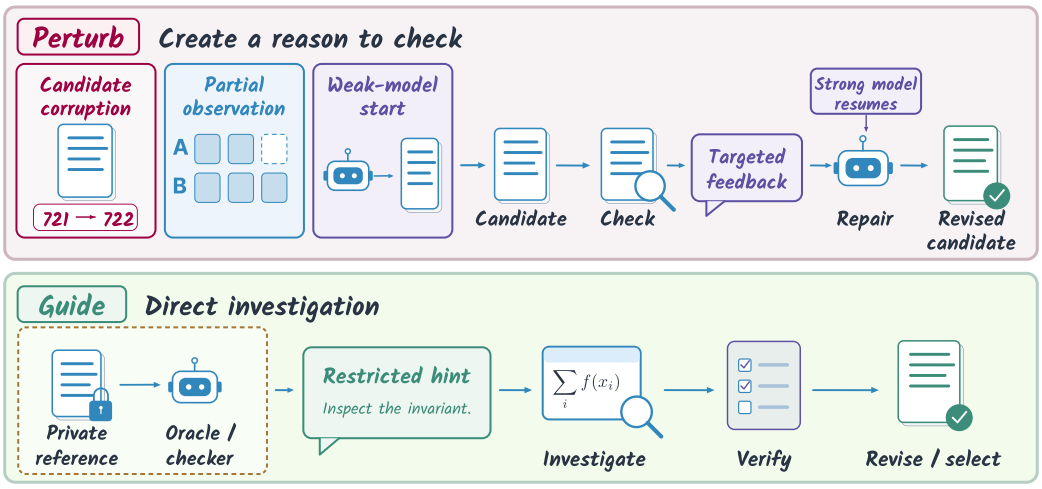

Figure 2: Complementary collection controls. Top: candidate corruption, partial observation, or weak initialization evokes checking and repair; the strong model performs requested repairs. Bottom: an oracle or checker converts private references into restricted hints for public investigation and verification. Task objectives and topology are preserved. Icons and values are illustrative; outcomes use task-native evaluation.

To prevent task overlap, all variants share a CoEvoke split, and external benchmarks are excluded from construction and training (Appendix A). Downstream evaluation retains task-solving roles and routing, with unperturbed inputs and no private-reference guidance (Section 5).

3.2 Perturb-and-Guide

Corrective collaboration needs both a reason to question an intermediate result and a feasible route to repair it. Perturb creates recoverable discrepancies, while Guide provides restricted direction for investigation. For role v with public input h_t^v and assigned model m_t^v at step t, a single collection operation is

$$\mathcal{C}_\omega(h_t^v, m_t^v, z_x) = \begin{cases} (h_t^v, m_t^v), & \omega = \text{natural}, \\ I_\kappa(h_t^v, m_t^v), & \omega = \text{perturb}, \\ (h_t^v \oplus \Pi_\rho(h_t^v, z_x), m_t^v), & \omega = \text{guide}. \end{cases} \tag{1}$$

Here I_κ perturbs an input or initial model assignment, Π_ρ restricts reference-informed assistance, and $\oplus$ delivers it to the permitted role. Operations can be combined at designated roles; controlled experiments isolate their effects.

Perturb: create a reason to check. The runtime perturbs only a designated initial interaction, preserving evidence and repair capacity:

$$\begin{aligned} &\text{Candidate corruption:} && \tilde{y}_0 = \kappa_x(y_0), \quad q_x(y_0) = 1, \quad q_x(\tilde{y}_0) = 0; \\ &\text{Partial observation:} && \tilde{F}_x^v = F_x \setminus \{f\}; \\ &\text{Weak initialization:} && m_{\text{init}}^v = m_{\text{weak}}, \quad m_{\text{repair}}^v = m_{\text{strong}}. \end{aligned} \tag{2}$$

Here y_0 is the first candidate and q_x a deterministic task-specific correctness check; κ_x changes an integer, entailment label, or table value. Both checks must hold; otherwise y_0 is retained. For partial observation, F_x is the public fact set and f a uniquely occurring fact on an unambiguous ProofWriter proof path. Other debaters and the judge retain all facts to diagnose the missing premise. Weak initialization restores the original stronger model only when feedback requests repair. No operation is reapplied during repair; approval ends checking, and revisions obey shared budgets (Appendix E).

Guide: direct investigation. A recoverable difficulty still needs an actionable direction. Guide maps the current public context and private reference to restricted assistance for a permitted recipient:

$$g_t^v = \Pi_\rho(h_t^v, z_x), \qquad h_t^{v,+} = h_t^v \oplus g_t^v. \tag{3}$$

Table 1: **The dataset includes only reconstructable trajectories.** Experimental runs are excluded even where source tasks overlap; a task can contribute multiple collection trajectories. Outcomes come from task-native evaluators. Appendix C reports process statistics and collection configurations.

Source	Tasks	Trajectories	PASS	FAIL
SWE-smith	1,763	1,763	1,561	202
DeepMath	831	923	917	6
ProofWriter	1,096	1,392	1,392	0
Zebra	440	489	489	0
Total	**4,130**	**4,567**	**4,359**	**208**

Here ρ specifies the recipient, structure, budget, and content restrictions; g_t^v is the guidance and $h_t^{v,+}$ the augmented public input. In software engineering, an oracle sends hints to the planner; in reasoning, the existing critic or judge returns feedback. Public solvers construct and verify solutions. Runtime controls isolate private inputs, enforce recipients and structure, and reject known direct-answer and private-source patterns in reasoning feedback. Prompts prohibit target patches and complete solutions. These constraints do not guarantee semantic nondisclosure, so guidance is audited separately (Appendix D).

3.3 FROM RECONSTRUCTABLE TRAJECTORIES TO DECISION SUPERVISION

To reuse corrective interactions, we preserve retained public evidence and outcomes in a temporal event graph $\mathcal{T}_x = (\mathcal{N}_x, \mathcal{R}_x, \prec, \nu, \phi)$. Nodes $\mathcal{N}_x$ are role/tool events; relations $\mathcal{R}_x$ link messages, feedback, revisions, selections, and tool responses; $\prec$ records order, ν visibility, and ϕ the final source. Reconstruction covers this exported interaction, not every teacher-visible input: immediate SWE test-pass summaries may influence collection-time review but enter neither public trajectories nor training contexts (Appendix B).

The full recorded trajectory supplies decision-specific supervision; it need not be supplied in full to each controller. For control role u, $h_{x,t}^{(u)} = P_u(\mathcal{T}_{x,\le t})$ extracts the retained public context before its target action. Targets specify selection or repair for the judge, feedback for the critic, and acceptance or rejection for the verifier. Inputs exclude private references, subsequent revisions, and evaluator outcomes. The training objective is

$$\mathcal{L}_u(\theta) = \sum_{x,t} w_{x,t}\, \ell_u\Big(\theta; h_{x,t}^{(u)}, a_{x,t}^*\Big), \tag{4}$$

where $a_{x,t}^*$ is the target decision, $\sum_t w_{x,t} = 1$ normalizes each task's contribution, and ℓ_u is weighted token-level negative log-likelihood. Task normalization prevents long trajectories from dominating. Frozen actors isolate learning of specific decisions (Appendix G).

4 DATA GENERATED BY COEVOKE

4.1 SCALE AND COMPOSITION

Repository repair requires long-horizon tool use and executable verification (Jimenez et al., 2024; Yang et al., 2024); reasoning exposes derivation errors, disagreement, and constraint violations. Across these settings, the dataset contains 4,567 reconstructable trajectories over 4,130 tasks (Table 1). Inclusion requires complete public role outputs and interaction links, irrespective of task success. Source tasks, role events, revisions, and evaluated artifacts remain linked.

Validation checks identifiers, chronology, call–observation pairs, feedback/revision references, final sources, and evaluator links. Public exports omit credentials, internal paths, private references, hidden reasoning, and immediate SWE evaluation counts. These counts may affect collection-time review but enter no training context; reconstruction covers retained public events (Appendix B). Distinct task, trajectory, and communication identities support pairing each public prefix with its target action for training (Appendices B and G).

Table 2: **Human–LLM agreement and trajectory quality.** Agreement pools five experts; quality and disclosure proportions use only the stratified probability sample. Appendix D gives the sampling design, raw counts, and uncertainty estimates.

A. Human–LLM agreement	*240 ratings; 160 trajectories*						
Dimension	**Exact agreement ↑**				κ_w / κ	**Score MAE ↓**	
Collaboration grade	**201/240 (83.8%)**				0.779	1.10	
Disclosure label	**218/240 (90.8%)**				0.688	–	
B. Probability-sample audit	*Design-weighted proportions (%)*						
Dataset	n	**Collaboration grade**				**Disclosure label**		
		A	**B**	**C**	**D**	**L0**	**L1**	**L2**
SWE Oracle	205	85.1	0.0	14.9	0.0	96.1	3.9	0.0
DeepMath	97	97.9	1.0	1.0	0.0	97.9	2.1	0.0
ProofWriter	146	98.6	1.4	0.0	0.0	100.0	0.0	0.0
Zebra	52	100.0	0.0	0.0	0.0	100.0	0.0	0.0
Overall	**500**	**93.1**	**0.6**	**6.3**	**0.0**	**98.0**	**2.0**	**0.0**

Grades use linearly weighted κ_w and a 0–14 score; disclosure uses unweighted κ. In B, n counts trajectories, not weighted observations. L1 denotes private-authority disclosure and L2 protected-content disclosure; no LX labels were observed.

4.2 Human and Model Audits

Because task success alone does not establish collaboration quality, we assess seven dimensions of contribution, evidence use, coherence, and efficiency. Grades A–D summarize process quality alongside task-native outcomes; a separate scale distinguishes no observed disclosure (L0), references to private authority (L1), protected-content disclosure (L2), and insufficient context (LX). Five experts contribute 240 ratings over 160 trajectories to calibrate model assessment and evidence-based adjudication (Zheng et al., 2023; Liu et al., 2023; Kim et al., 2024).

Table 2 separates human agreement from quality estimation: pooled agreement includes probability-sampled and targeted cases, while quality estimates use the weighted 500-trajectory probability sample. Grade A accounts for 93.1%; 2.0% is labeled L1, and no L2 disclosure is detected. Agreement supports label reliability, while the grade distribution indicates generally high-quality recorded interaction. Coupled with linked public role/tool evidence and reconstruction and visibility checks, these results support an information-rich corpus ready for decision-specific training. Appendix D details annotation visibility and sampling.

5 Experiments

5.1 Evaluation Design

Only collection-mechanism experiments activate Perturb or Guide. Workflow and post-training evaluations retain task-solving topology, ordinary feedback, and repair, using unperturbed inputs and public task/execution evidence without private-reference guidance. Hosted teachers test collection; open-weight models test downstream use with fixed solving weights, decoding, and output protocols (Appendix K.2).

Collection controls. The perturbation experiment contains 2,287 same-task pairs: 1,197 compare original and perturbed inputs, and 1,090 compare strong-first and weak-to-strong generation. Each pair fixes the checker, reference access, and evaluator. Guidance compares reference-based with public-only assistance: the same 80 SWE-smith tasks across five models give 400 runs per condition; reasoning uses 240 assigned tasks per model. These experimental trajectories are excluded from the dataset (Appendix E).

Downstream evaluation. Workflow tests use *Single-MA*, one agent with the multi-agent workflow's aggregate model-call, tool-use, and token allowances. Primary evaluations cover all 500 SWE-bench

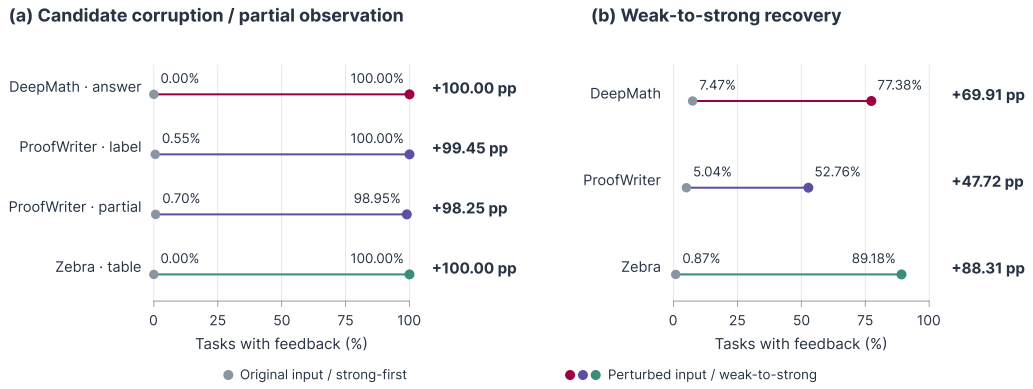

Figure 3: Perturb increases feedback in (a) 1,197 candidate-corruption/partial-observation pairs and (b) 1,090 strong-first/weak-to-strong pairs. Markers connect control (gray) and perturbed rates; colors identify datasets and labels give percentage-point gains. All pairs are evaluable. Tables E1–E9 report paired 95% bootstrap intervals, revisions, outcomes, and costs.

Verified and 1,040 ACPBench-Hard tasks and a fixed 144-task MathArena collection (Jimenez et al., 2024; Balunović et al., 2025; Kokel et al., 2026); internal 100-task samples supplement them. Training updates only the judge, critic, or verifier. Natural-SFT and Perturb-SFT match training task IDs, decision records, and updates; each verifier is compared with an untrained model receiving the same evidence format (Appendices F and G).

Measures. Task-native success uses prespecified denominators. Process measures distinguish a feedback message, an executed revision, an evaluator-approved revision, and its selection as the final source. A selected correct revision additionally requires an answer change. We report percentage-point differences and paired intervals or tests where supported; the fixed-sample SWE guidance comparison is descriptive. Pair availability, statistical definitions, and token accounting appear in Appendices E–G.

5.2 Do Perturbations Evoke Corrective Interaction?

For 1,197 input-perturbation pairs, feedback rises from 0.33% to 99.75%, a 99.42-point increase (95% CI 98.91–99.83; Figure 3). The runtime assigns a weaker model only to initial generation; across 1,090 weak-to-strong pairs, feedback rises from 5.14% to 70.46%, or 65.32 points (95% CI 62.57–68.07).

Both conditions solve all 1,197 input-perturbation tasks, but 732 perturbed runs (61.15%) yield a *selected correct revision*.[3] This is not an overall repair-success rate: the other 465 runs (38.85%) also pass, without satisfying that joint criterion. All 344 DeepMath runs select a revised final answer; multi-candidate workflows may generate a correct revision yet select another valid candidate. Perturb thus supplies corrective supervision even when final accuracy cannot improve.

Under weak initialization, final success remains unchanged on ProofWriter and Zebra and moves from 87.78% to 86.65% on DeepMath, with a paired interval spanning zero. The additional interaction costs 1.87 model calls per task, versus 2.31 for input perturbations, supplying corrective supervision without guaranteeing more solved tasks.

5.3 Does Restricted Guidance Improve Task Solving?

The public-only control O1 matches reference-based O2's planner-facing schema, guidance role, and output budget. Across the same 80 SWE-smith tasks and five models, O2 raises success from

[3] A run is counted once when its final source is a revision, its normalized answer differs from that role's first candidate delivered for checking in the same run, and the final task outcome is PASS. For candidate corruption, the comparison starts from the perturbed answer (Appendix E.2, Tables E4 and E6).

Table 3: **Model-dependent effects of reference access.** (A) The same 80 SWE tasks × five models give 400 runs per arm. Net provider-token change is $100(T_{O2}/T_{O1} - 1)\%$ (lower is better); the total uses summed tokens. (B) Reasoning rates share n paired tasks. O1 is public-only and O2 reference-based guidance. Colors highlight gains and token reductions; preferred directions appear in headers. Exact McNemar tests for B are reported in Appendix E.

A. SWE: Oracle guidance to the Planner					
Model (thinking)	n	**O1 (%)**	**O2 (%)**	**Gain (pp) ↑**	**Net provider token change (%) ↓**
GPT-5.4 (medium)	80	77.50	83.75	+6.25	-9.97
Claude Sonnet 5 (adaptive; medium)	80	88.75	88.75	+0.00	-17.95
DeepSeek-V4-Pro (enabled; medium)	80	88.75	90.00	+1.25	-9.54
GLM-5.2 (enabled; max)	80	92.50	96.25	+3.75	-4.19
Kimi K2.6 (enabled; provider default)	80	87.50	92.50	+5.00	-3.91
Total	400	87.00	**90.25**	+3.25	-8.55
B. Reasoning: reference-informed Critic/Judge					
Model (thinking)	n	**Public**	**Reference**	**Δ success (pp) ↑**	**Paired 95% CI (pp)**
GPT-5.4 (medium)	240	81.25	85.00	+3.75	[1.67, 6.25]
Claude Sonnet 5 (adaptive; medium)	236	78.81	81.78	+2.97	[0.85, 5.08]
DeepSeek-V4-Pro (enabled; medium)	236	92.37	94.92	+2.54	[0.42, 5.08]
GLM-5.2 (enabled; max)	237	85.65	87.34	+1.69	[0.42, 3.80]
Kimi K2.6 (enabled; provider default)	234	92.74	93.59	+0.85	[0.00, 2.14]

348/400 (87.00%) to 361/400 (90.25%), a 3.25-point gain (Table 3). Four models improve and one ties; all five use fewer net provider tokens, with an aggregate reduction of 8.55%.

Reference-informed reasoning gains range from 0.85 to 3.75 points across all five models, with limited headroom where public checking already succeeds. SWE token reductions range from 3.91% to 17.95%; Claude preserves success while reducing usage. These descriptive provider-token totals do not represent a common tokenizer or monetary cost (Appendix E.6).

5.4 Are the Workflows Effective without Perturb-and-Guide?

On unperturbed inputs without private-reference guidance, CoEvoke exceeds budget-matched single-agent controls in eight of nine external model–benchmark combinations (Figure 4a; Table F1). All three models gain on SWE-bench Verified and ACPBench-Hard. On MathArena, Qwen and Gemma gain 6.25 and 3.47 points, while Ministral loses 2.78 points.

Qwen's internal controls solve 87% of DeepMath and 84% of ProofWriter; CoEvoke loses one point on each, whereas Zebra rises from 52% to 60%. Benefits vary by task and model, supporting workflow effectiveness, not topology optimality (Appendix I).

5.5 Does Training on CoEvoke Trajectories Improve Task Solving?

With 884 shared tasks, 891 decision records, and 379 updates, Perturb-SFT solves 438/1,040 ACPBench-Hard tasks (42.12%) versus Natural-SFT's 368/1,040 (35.38%), a 6.73-point gain. Repair-routing targets grow from 7 (0.79%) to 440 (49.38%): the fixed budget shifts from almost exclusively approval targets to substantial repair-routing supervision, realizing the intended collection effect. Natural-SFT uses independently collected clean trajectories; both sources use reference-informed checking during collection (Appendix G.4).

On SWE-bench Verified, patch-based verifier training improves success from 24.80% to 32.40%; trajectory-based training improves it from 23.40% to 32.80%. Both evidence interfaces support improved closed-loop task success after verifier training, with solving actors and the evaluation protocol held fixed. These results establish learnability within each interface, not an advantage of



Figure 4: Workflow and training comparisons. (a) Shared-budget control (gray) versus CoEvoke; circles represent 100%, with Qwen, Gemma, and Ministral outside inward. (b) Control-role training with frozen Qwen solvers: the separate full-corpus judge study and interface-matched verifiers. In SWE, the Reviewer in (a) reads files and runs public tests; both Verifiers in (b) have no tools. Denominators: 500 SWE-bench Verified, 144 MathArena, 1,040 ACPBench-Hard. Evaluation uses unperturbed inputs, no private-reference guidance, and no explicit thinking. See Tables F1–F5 and G1–G5.

longer inputs or a replacement for the tool-using reviewer (53.20%; Appendix G.5).[4] An unmatched actor-imitation study motivates selective supervision (Appendix H).

6 Discussion and Limitations

A matched 180-task 2×2 reasoning experiment supports joint corrective supervision and token savings without establishing a superadditive interaction; it covers candidate corruption with one checker (Appendix E.7).

We validate CoEvoke on small multi-agent teams; its effectiveness in much larger systems with more complex collaboration remains untested. Training is restricted to SFT to isolate the dataset's contribution, since subsequent reinforcement learning (RL) could introduce gains from additional interaction and rewards. Whether post-SFT RL can consolidate and generalize the learned behavior, and whether the dataset improves broader agentic capabilities beyond the evaluated control decisions, remain open questions.

7 Conclusion

CoEvoke makes corrective collaboration an objective of trajectory collection. Recoverable perturbations and restricted guidance produce reconstructable trajectories that preserve public evidence and outcome links for decision-record extraction. Full trajectories support decision-specific supervision without requiring every controller to receive the entire interaction. Gains with frozen solvers on unperturbed inputs without private-reference guidance support collection design as a means to improve learned checking, revision, and acceptance.

[4]The patch interface supplies the issue and patch; the trajectory interface adds selected prior public execution. Neither verifier has tools. Their accept/reject decisions affect subsequent execution, so candidates and evidence can differ across runs: fixed interfaces mean consistent evidence-selection rules, not identical dynamic inputs or classification of a fixed final-patch set (Appendix G.5).

Ethics Statement

CoEvoke can support systems that modify code and perform complex reasoning. The collected trajectories document generated behavior together with its task outcomes, allowing users to examine the relationship between an agent's actions and the evaluated result. In practical use, generated patches still require appropriate access controls, testing, and human review. Failure trajectories retain their evaluator outcomes alongside the recorded interactions, so that unsuccessful runs can be distinguished from positive demonstrations when inspecting or using the data. Private oracle materials are stored separately from public training data, maintaining the distinction between privileged collection inputs and the public collaboration records.

Reproducibility Statement

The anonymous repository provides the CoEvoke runtime, dataset adapters, workflow configurations, trajectory schemas, and tools for task evaluation and collaboration auditing. Together, these components support the process from preparing task inputs and configuring role interactions to collecting trajectories, scoring outcomes, and inspecting recorded collaboration. The README documents installation and execution and includes examples illustrating the main entry points. These instructions complement the paper's descriptions of how the workflows operate and how their interactions are recorded. Task construction and record reconstruction are detailed in Appendices A–B. Experimental settings, results, and analyses are reported in Appendices D–I, while Appendix K provides the prompts and execution protocols.

Data and code availability. The framework code is available through the anonymous repository at https://anonymous.4open.science/r/CoEvoke/. We will provide the complete CoEvoke trajectory dataset described in Section 4. The release will follow the public-record schema and visibility boundaries in Appendix B, retaining the documented links among public role interactions, revisions, and task outcomes. These links support inspection of the recorded collaboration and its use as a source of decision-specific supervision. Upstream task text, code, and patches retain their respective licenses; the distinction between CoEvoke's collected records and these upstream resources is described in Appendix J.

AI Use Statement

Generative AI assisted manuscript writing, translation, and polishing. This assistance supported the presentation of the work and the clarity and consistency of its written description. Generative AI was also used to produce the synthetic multi-agent collaboration trajectories that form the CoEvoke dataset. The trajectory-generation procedure and AI-assisted quality assessment are described in Sections 3 and 4. The authors reviewed the manuscript and its AI-assisted presentation. They remain responsible for the final content of the paper, including the accuracy of the reported results and the support for the conclusions drawn from them.

References

Mislav Balunović, Jasper Dekoninck, Ivo Petrov, Nikola Jovanović, and Martin Vechev. MathArena: Evaluating LLMs on uncontaminated math competitions. In *Advances in Neural Information Processing Systems*, volume 38, 2025. doi: 10.52202/085713-0679. URL https://proceedings.neurips.cc/paper_files/paper/2025/hash/1d27c01ebd3e3aebe226b44fc970d803-Abstract-Datasets_and_Benchmarks_Track.html.

Weize Chen, Yusheng Su, Jingwei Zuo, Cheng Yang, Chenfei Yuan, Chi-Min Chan, Heyang Yu, Yaxi Lu, Yi-Hsin Hung, Chen Qian, Yujia Qin, Xin Cong, Ruobing Xie, Zhiyuan Liu, Maosong Sun, and Jie Zhou. AgentVerse: Facilitating multi-agent collaboration and exploring emergent behaviors. In *International Conference on Learning Representations*, 2024. URL https://proceedings.iclr.cc/paper_files/paper/2024/hash/578e65cdee35d00c708d4c64bce32971-Abstract-Conference.html.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John

Schulman. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*, 2021. URL https://arxiv.org/abs/2110.14168.

Yufan Dang, Chen Qian, Xueheng Luo, Jingru Fan, Zihao Xie, Ruijie Shi, Weize Chen, Cheng Yang, Xiaoyin Che, Ye Tian, Xuantang Xiong, Lei Han, Zhiyuan Liu, and Maosong Sun. Multi-agent collaboration via evolving orchestration. In *Advances in Neural Information Processing Systems*, volume 38, 2025. URL https://proceedings.neurips.cc/paper_files/paper/2025/hash/f1320d2e2842169c6fc89dcbd80e94d0-Abstract-Conference.html.

Yufan Dang, Shu Yao, Bowen Lai, Chenting Xu, Ruijie Shi, Wai-Shing Leung, Huatao Li, Chen Qian, and Zhiyuan Liu. ChatDev 2.0: A no-code multi-agent platform for developing everything. In *Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, 2026. URL https://arxiv.org/abs/2609.00714. Accepted; preprint arXiv:2609.00714.

Yilun Du, Shuang Li, Antonio Torralba, Joshua B. Tenenbaum, and Igor Mordatch. Improving factuality and reasoning in language models through multiagent debate. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235, pp. 11733–11763, 2024. URL https://proceedings.mlr.press/v235/du24e.html.

Dayuan Fu, Keqing He, Yejie Wang, Wentao Hong, Zhuoma Gongque, Weihao Zeng, Wei Wang, Jingang Wang, Xunliang Cai, and Weiran Xu. AgentRefine: Enhancing agent generalization through refinement tuning. In *International Conference on Learning Representations*, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/file/a3cc50126338b175e56bb3cad134db0b-Paper-Conference.pdf.

Zhibin Gou, Zhihong Shao, Yeyun Gong, Yelong Shen, Yujiu Yang, Nan Duan, and Weizhu Chen. CRITIC: Large language models can self-correct with tool-interactive critiquing. In *International Conference on Learning Representations*, 2024. URL https://arxiv.org/abs/2305.11738.

Zhiwei He, Tian Liang, Jiahao Xu, Qiuzhi Liu, Xingyu Chen, Yue Wang, Linfeng Song, Dian Yu, Zhenwen Liang, Wenxuan Wang, Zhuosheng Zhang, Rui Wang, Zhaopeng Tu, Haitao Mi, and Dong Yu. DeepMath-103K: A large-scale, challenging, decontaminated, and verifiable mathematical dataset for advancing reasoning. *arXiv preprint arXiv:2504.11456*, 2025. URL https://arxiv.org/abs/2504.11456.

Sirui Hong, Mingchen Zhuge, Jonathan Chen, Xiawu Zheng, Yuheng Cheng, Jinlin Wang, Ceyao Zhang, Zili Wang, Steven Ka Shing Yau, Zijuan Lin, Liyang Zhou, Chenyu Ran, Lingfeng Xiao, Chenglin Wu, and Jürgen Schmidhuber. MetaGPT: Meta programming for a multi-agent collaborative framework. In *International Conference on Learning Representations*, 2024. URL https://proceedings.iclr.cc/paper_files/paper/2024/hash/6507b115562bb0a305f1958ccc87355a-Abstract-Conference.html.

Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. LoRA: Low-rank adaptation of large language models. In *International Conference on Learning Representations*, 2022. URL https://arxiv.org/abs/2106.09685.

Jie Huang, Xinyun Chen, Swaroop Mishra, Huaixiu Steven Zheng, Adams Wei Yu, Xinying Song, and Denny Zhou. Large language models cannot self-correct reasoning yet. In *International Conference on Learning Representations*, 2024. URL https://proceedings.iclr.cc/paper_files/paper/2024/hash/8b4add8b0aa8749d80a34ca5d941c355-Abstract-Conference.html.

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*, 2024. URL https://proceedings.iclr.cc/paper_files/paper/2024/hash/edac78c3e300629acfe6cbe9ca88fb84-Abstract-Conference.html.

Seungone Kim, Juyoung Suk, Shayne Longpre, Bill Yuchen Lin, Jamin Shin, Sean Welleck, Graham Neubig, Moontae Lee, Kyungjae Lee, and Minjoon Seo. Prometheus 2: An open source language model specialized in evaluating other language models. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pp. 4334–4353, 2024. doi: 10.18653/v1/2024.emnlp-main.248. URL https://aclanthology.org/2024.emnlp-main.248/.

Harsha Kokel, Michael Katz, Kavitha Srinivas, and Shirin Sohrabi. ACPBench Hard: Unrestrained reasoning about action, change, and planning. In *International Conference on Learning Representations*, 2026. URL https://proceedings.iclr.cc/paper_files/paper/2026/hash/a0bff34b8dfcb39f8a2dcb8c7491cda8-Abstract-Conference.html.

Aviral Kumar, Vincent Zhuang, Rishabh Agarwal, Yi Su, JD Co-Reyes, Avi Singh, Kate Baumli, Shariq Iqbal, Colton Bishop, Rebecca Roelofs, Lei Zhang, Kay McKinney, Disha Shrivastava, Cosmin Paduraru, George Tucker, Doina Precup, Feryal Behbahani, and Aleksandra Faust. Training language models to self-correct via reinforcement learning. In *International Conference on Learning Representations*, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/hash/871ac99fdc5282d0301934d23945ebaa-Abstract-Conference.html.

Philippe Laban, Hiroaki Hayashi, Yingbo Zhou, and Jennifer Neville. LLMs get lost in multi-turn conversation. In *International Conference on Learning Representations*, 2026. URL https://proceedings.iclr.cc/paper_files/paper/2026/hash/59f6421e64707225fdf5b28840679a07-Abstract-Conference.html.

Guohao Li, Hasan Hammoud, Hani Itani, Dmitrii Khizbullin, and Bernard Ghanem. CAMEL: Communicative agents for "mind" exploration of large language model society. In *Advances in Neural Information Processing Systems*, volume 36, 2023. URL https://proceedings.neurips.cc/paper/2023/hash/a3621ee907def47c1b952ade25c67698-Abstract-Conference.html.

Weizhen Li, Jianbo Lin, Zhuosong Jiang, Jingyi Cao, Xinpeng Liu, Jiayu Zhang, Zhenqiang Huang, Qianben Chen, Weichen Sun, Qiexiang Wang, Hongxuan Lu, Tianrui Qin, Chenghao Zhu, Yi Yao, Shuying Fan, Xiaowan Li, Tiannan Wang, Pai Liu, King Zhu, He Zhu, Dingfeng Shi, Piaohong Wang, Yeyi Guan, Xiangru Tang, Minghao Liu, Yuchen Eleanor Jiang, Jian Yang, Jiaheng Liu, Ge Zhang, and Wangchunshu Zhou. Chain-of-Agents: End-to-end agent foundation models via multi-agent distillation and agentic RL. *arXiv preprint arXiv:2508.13167*, 2025. URL https://arxiv.org/abs/2508.13167.

Hunter Lightman, Vineet Kosaraju, Yura Burda, Harri 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*, 2024. URL https://proceedings.iclr.cc/paper_files/paper/2024/hash/aca97732e30bcf1303bc22ac3924fd16-Abstract-Conference.html.

Bill Yuchen Lin, Ronan Le Bras, Kyle Richardson, Ashish Sabharwal, Radha Poovendran, Peter Clark, and Yejin Choi. ZebraLogic: On the scaling limits of LLMs for logical reasoning. In *Proceedings of the 42nd International Conference on Machine Learning*, 2025. URL https://icml.cc/virtual/2025/poster/43835.

Xiao Liu, Hao Yu, Hanchen Zhang, Yifan Xu, Xuanyu Lei, Hanyu Lai, Yu Gu, Hangliang Ding, Kaiwen Men, Kejuan Yang, Shudan Zhang, Xiang Deng, Aohan Zeng, Zhengxiao Du, Chenhui Zhang, Sheng Shen, Tianjun Zhang, Yu Su, Huan Sun, Minlie Huang, Yuxiao Dong, and Jie Tang. AgentBench: Evaluating LLMs as agents. In *International Conference on Learning Representations*, 2024. URL https://arxiv.org/abs/2308.03688.

Yang Liu, Dan Iter, Yichong Xu, Shuohang Wang, Ruochen Xu, and Chenguang Zhu. G-eval: NLG evaluation using GPT-4 with better human alignment. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pp. 2511–2522, 2023. doi: 10.18653/v1/2023.emnlp-main.153. URL https://aclanthology.org/2023.emnlp-main.153/.

Yinyi Luo, Yiqiao Jin, Weichen Yu, Mengqi Zhang, Srijan Kumar, Xiaoxiao Li, Weijie Xu, Xin Chen, and Jindong Wang. AgentArk: Distilling multi-agent intelligence into a single LLM agent. *arXiv preprint arXiv:2602.03955*, 2026. URL https://arxiv.org/abs/2602.03955.

Ming Ma, Jue Zhang, Fangkai Yang, Yu Kang, Qingwei Lin, Saravan Rajmohan, and Dongmei Zhang. DoVer: Intervention-driven auto debugging for LLM multi-agent systems. In *International Conference on Learning Representations*, 2026. URL https://proceedings.iclr.cc/paper_files/paper/2026/file/2f5fdce256ff0d7fb774da76e5e63209-Paper-Conference.pdf.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, Shashank Gupta, Bodhisattwa Prasad Majumder,

Katherine Hermann, Sean Welleck, Amir Yazdanbakhsh, and Peter Clark. Self-refine: Iterative refinement with self-feedback. In *Advances in Neural Information Processing Systems*, volume 36, 2023. URL https://proceedings.neurips.cc/paper_files/paper/2023/hash/91edff07232fb1b55a505a9e9f6c0ff3-Abstract-Conference.html.

Sumeet Ramesh Motwani, Chandler Smith, Rocktim Jyoti Das, Rafael Rafailov, Ivan Laptev, Philip H. S. Torr, Fabio Pizzati, Ronald Clark, and Christian Schroeder de Witt. MALT: Improving reasoning with multi-agent LLM training. In *Conference on Language Modeling*, 2025. URL https://arxiv.org/abs/2412.01928v3.

Akshara Prabhakar, Zuxin Liu, Ming Zhu, Jianguo Zhang, Tulika Manoj Awalgaonkar, Shiyu Wang, Zhiwei Liu, Haolin Chen, Thai Hoang, Juan Carlos Niebles, Shelby Heinecke, Weiran Yao, Huan Wang, Silvio Savarese, and Caiming Xiong. APIGen-MT: Agentic pipeline for multi-turn data generation via simulated agent–human interplay. In *Advances in Neural Information Processing Systems*, volume 38, 2025. URL https://proceedings.neurips.cc/paper_files/paper/2025/hash/5e3661f7fe4c8ac5652d62eb3d3c96ea-Abstract-Datasets_and_Benchmarks_Track.html.

Chen Qian, Wei Liu, Hongzhang Liu, Nuo Chen, Yufan Dang, Jiahao Li, Cheng Yang, Weize Chen, Yusheng Su, Xin Cong, Juyuan Xu, Dahai Li, Zhiyuan Liu, and Maosong Sun. ChatDev: Communicative agents for software development. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 15174–15186, 2024. doi: 10.18653/v1/2024.acl-long.810. URL https://aclanthology.org/2024.acl-long.810/.

Chen Qian, Zihao Xie, Yifei Wang, Wei Liu, Kunlun Zhu, Hanchen Xia, Yufan Dang, Zhuoyun Du, Weize Chen, Cheng Yang, Zhiyuan Liu, and Maosong Sun. Scaling large language model-based multi-agent collaboration. In *International Conference on Learning Representations*, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/file/66a026c0d17040889b50f0dfa650e5e0-Paper-Conference.pdf.

Stéphane Ross, Geoffrey J. Gordon, and J. Andrew Bagnell. A reduction of imitation learning and structured prediction to no-regret online learning. In *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, volume 15, pp. 627–635, 2011. URL https://proceedings.mlr.press/v15/ross11a.html.

Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. Toolformer: Language models can teach themselves to use tools. In *Advances in Neural Information Processing Systems*, volume 36, 2023. URL https://proceedings.neurips.cc/paper_files/paper/2023/hash/d842425e4bf79ba039352da0f658a906-Abstract-Conference.html.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, Y. K. Li, Y. Wu, and Daya Guo. DeepSeekMath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*, 2024. URL https://arxiv.org/abs/2402.03300.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. In *Advances in Neural Information Processing Systems*, 2023. URL https://proceedings.neurips.cc/paper_files/paper/2023/hash/1b44b878bb782e6954cd888628510e90-Abstract-Conference.html.

Yueqi Song, Ketan Ramaneti, Zaid Sheikh, Ziru Chen, Boyu Gou, Tianbao Xie, Yiheng Xu, Danyang Zhang, Apurva Gandhi, Fan Yang, Joseph Liu, Tianyue Ou, Zhihao Yuan, Frank F. Xu, Shuyan Zhou, Xingyao Wang, Xiang Yue, Tao Yu, Huan Sun, Yu Su, and Graham Neubig. Agent data protocol: Unifying datasets for diverse, effective fine-tuning of LLM agents. In *International Conference on Learning Representations*, 2026. URL https://proceedings.iclr.cc/paper_files/paper/2026/file/e79c09f39aca8507f527daad3601b3aa-Paper-Conference.pdf.

Kaya Stechly, Karthik Valmeekam, and Subbarao Kambhampati. On the self-verification limitations of large language models on reasoning and planning tasks. In *International Conference on Learning Representations*, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/file/f3c5e56274140e0420baa3916c529210-Paper-Conference.pdf.

Oyvind Tafjord, Bhavana Dalvi, and Peter Clark. ProofWriter: Generating implications, proofs, and abductive statements over natural language. In *Findings of the Association for Computational Linguistics: ACL-IJCNLP 2021*, pp. 3621–3634. Association for Computational Linguistics, 2021. doi: 10.18653/v1/2021.findings-acl.317. URL https://aclanthology.org/2021.findings-acl.317/.

Jonathan Uesato, Nate Kushman, Ramana Kumar, Francis Song, Noah Siegel, Lisa Wang, Antonia Creswell, Geoffrey Irving, and Irina Higgins. Solving math word problems with process- and outcome-based feedback. *arXiv preprint arXiv:2211.14275*, 2022. URL https://arxiv.org/abs/2211.14275.

Xingyao Wang, Zihan Wang, Jiateng Liu, Yangyi Chen, Lifan Yuan, Hao Peng, and Heng Ji. MINT: Evaluating LLMs in multi-turn interaction with tools and language feedback. In *International Conference on Learning Representations*, 2024. URL https://proceedings.iclr.cc/paper_files/paper/2024/file/8a0d3ae989a382ce6e50312bc35bf7e1-Paper-Conference.pdf.

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. In *International Conference on Learning Representations*, 2023a. URL https://arxiv.org/abs/2203.11171.

Yizhong Wang, Yeganeh Kordi, Swaroop Mishra, Alisa Liu, Noah A. Smith, Daniel Khashabi, and Hannaneh Hajishirzi. Self-instruct: Aligning language models with self-generated instructions. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 13484–13508, 2023b. doi: 10.18653/v1/2023.acl-long.754. URL https://aclanthology.org/2023.acl-long.754/.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed Chi, Quoc V. Le, and Denny Zhou. Chain-of-thought prompting elicits reasoning in large language models. In *Advances in Neural Information Processing Systems*, volume 35, 2022. URL https://proceedings.neurips.cc/paper/2022/hash/9d5609613524ecf4f15af0f7b31abca4-Abstract-Conference.html.

Qingyun Wu, Gagan Bansal, Jieyu Zhang, Yiran Wu, Beibin Li, Erkang Zhu, Li Jiang, Xiaoyun Zhang, Shaokun Zhang, Jiale Liu, Ahmed Awadallah, Ryen W. White, Doug Burger, and Chi Wang. AutoGen: Enabling next-gen LLM applications via multi-agent conversations. In *Conference on Language Modeling*, 2024. URL https://www.microsoft.com/en-us/research/wp-content/uploads/2023/08/LLM_agent.pdf.

Yiheng Xu, Dunjie Lu, Zhennan Shen, Junli Wang, Zekun Wang, Yuchen Mao, Caiming Xiong, and Tao Yu. AgentTrek: Agent trajectory synthesis via guiding replay with web tutorials. In *International Conference on Learning Representations*, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/hash/c681fb2bf1d785fbc766f3ea14758aab-Abstract-Conference.html.

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

John Yang, Kilian Lieret, Carlos E. Jimenez, Alexander Wettig, Kabir Khandpur, Yanzhe Zhang, Binyuan Hui, Ofir Press, Ludwig Schmidt, and Diyi Yang. SWE-smith: Scaling data for software engineering agents. In *Advances in Neural Information Processing Systems*, volume 38, 2025. URL https://proceedings.neurips.cc/paper_files/paper/2025/hash/8b86cf5ace600c48fd188efbb8dedec8-Abstract-Datasets_and_Benchmarks_Track.html.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. ReAct: Synergizing reasoning and acting in language models. In *International Conference on Learning Representations*, 2023. URL https://arxiv.org/abs/2210.03629.

Shunyu Yao, Noah Shinn, Pedram Razavi, and Karthik Narasimhan. τ-bench: A benchmark for tool-agent-user interaction in real-world domains. In *International Conference on Learning Representations*, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/file/1b126cc38b8638e07bef37e7b2bb72bf-Paper-Conference.pdf.

Eric Zelikman, Yuhuai Wu, Jesse Mu, and Noah Goodman. STaR: Bootstrapping reasoning with reasoning. In *Advances in Neural Information Processing Systems*, volume 35, pp. 15476–15488, 2022. URL https://proceedings.neurips.cc/paper_files/paper/2022/hash/639a9a172c044fbb64175b5fad42e9a5-Abstract-Conference.html.

Aohan Zeng, Mingdao Liu, Rui Lu, Bowen Wang, Xiao Liu, Yuxiao Dong, and Jie Tang. AgentTuning: Enabling generalized agent abilities for LLMs. In *Findings of the Association for Computational Linguistics: ACL 2024*, 2024. doi: 10.18653/v1/2024.findings-acl.181. URL https://aclanthology.org/2024.findings-acl.181/.

Guibin Zhang, Junhao Wang, Junjie Chen, Wangchunshu Zhou, Kun Wang, and Shuicheng Yan. AgenTracer: Who is inducing failure in the LLM agentic systems? In *International Conference on Learning Representations*, 2026. URL https://proceedings.iclr.cc/paper_files/paper/2026/file/134ed7a477770f227f12450ef0cbb8f4-Paper-Conference.pdf.

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric P. Xing, Hao Zhang, Joseph E. Gonzalez, and Ion Stoica. Judging LLM-as-a-judge with MT-bench and chatbot arena. In *Advances in Neural Information Processing Systems, Datasets and Benchmarks Track*, 2023. URL https://arxiv.org/abs/2306.05685.

Kunlun Zhu, Hongyi Du, Zhaochen Hong, Xiaocheng Yang, Shuyi Guo, Zhe Wang, Zhenhailong Wang, Cheng Qian, Xiangru Tang, Heng Ji, and Jiaxuan You. MultiAgentBench: Evaluating the collaboration and competition of LLM agents. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 8580–8622, 2025. doi: 10.18653/v1/2025.acl-long.421. URL https://aclanthology.org/2025.acl-long.421/.

APPENDICES

These appendices describe the task workflows, collection controls, training procedures, and complete experimental results. Task-native evaluators determine correctness; human and calibrated model audits separately assess collaboration, with their process-quality grading rules explained in Appendix D. Dataset and audit evidence appear in Appendices C–D, followed by mechanism, workflow, and learning results in Appendices E–G. Full prompts are collected at the end in Appendix K. Trajectory excerpts reproduce public tasks and interactions, not credentials, private task references, hidden model reasoning, or identifying execution-environment details.

In tables, bold/underline rank only the comparable conditions specified by each caption, with ties sharing rank. Arrows indicate metric direction or the direction of a reported change.

A TASKS AND WORKFLOWS

A.1 TASK SOURCES AND SEPARATION

CoEvoke combines repository-level software repair with verifiable mathematical and logical reasoning. These settings require different forms of coordination: software repair couples investigation to changes in an executable environment, whereas reasoning tasks require checking deductions or comparing candidate answers.

We collect tasks from `princeton-nlp/SWE-smith` (Yang et al., 2025), `zwhe99/DeepMath-103K` (He et al., 2025), `tasksource/proofwriter` (Tafjord et al., 2021), and `WildEval/ZebraLogic` (Lin et al., 2025). Static validity and executability checks first define an eligible pool; a fixed random procedure then samples from that pool. Throughout the paper, *training*, *development*, and *internal test* refer to CoEvoke's task-disjoint partitions, not a claim that every source uses an upstream training split. For example, the retained DeepMath and ProofWriter source records carry the upstream label `train`, whereas Zebra records carry `test`. We preserve these source labels separately and keep every variant of an original task in the same CoEvoke partition. SWE-bench Verified and the specified MathArena and ACPBench-Hard evaluation collections are excluded from collection and controller training.

A SWE-smith task specifies a repository problem whose solution must be implemented and tested. The resulting trajectory connects code investigation, a repair plan, edits, review, and the final patch. DeepMath provides problems with verifiable mathematical answers, allowing us to connect a particular derivation step to a critique and subsequent correction. ProofWriter distinguishes whether a query or its negation follows from the supplied facts and rules, while Zebra requires a complete assignment satisfying interacting clues. These latter tasks expose candidate disagreement, unsupported conclusions, and global consistency errors.

Internal experiments use limited task samples and primarily support mechanism analysis. We therefore place greater weight on external evaluation over 500 SWE-bench Verified tasks, 1,040 ACPBench-Hard tasks, and a fixed 144-task MathArena collection. The latter comprises AIME 2026 (30 tasks), HMMT February 2026 (33), and the April and May 2026 ArxivMath collections (41 and 40, respectively). All reported MathArena comparisons use these same task identifiers; "full" refers to this frozen collection, not the continually expanding MathArena platform. Separating these tasks from collection and training prevents their direct reuse, while complete collection-level denominators reduce dependence on a favorable small sample.

A.2 WORKFLOW STRUCTURE

In software repair, the planner investigates the issue and identifies concrete edit locations before handing an executable plan to the coder. The reviewer independently inspects the resulting changes and available public checks. Rejection returns specific findings to the planner, which updates the affected plan rather than restarting the task. Approval ends execution and preserves the patch for independent evaluation.

For DeepMath, a solver produces a candidate derivation and a critic checks it. Acceptance ends the run; a localized problem triggers feedback to the solver, followed by a new check of the revision. In

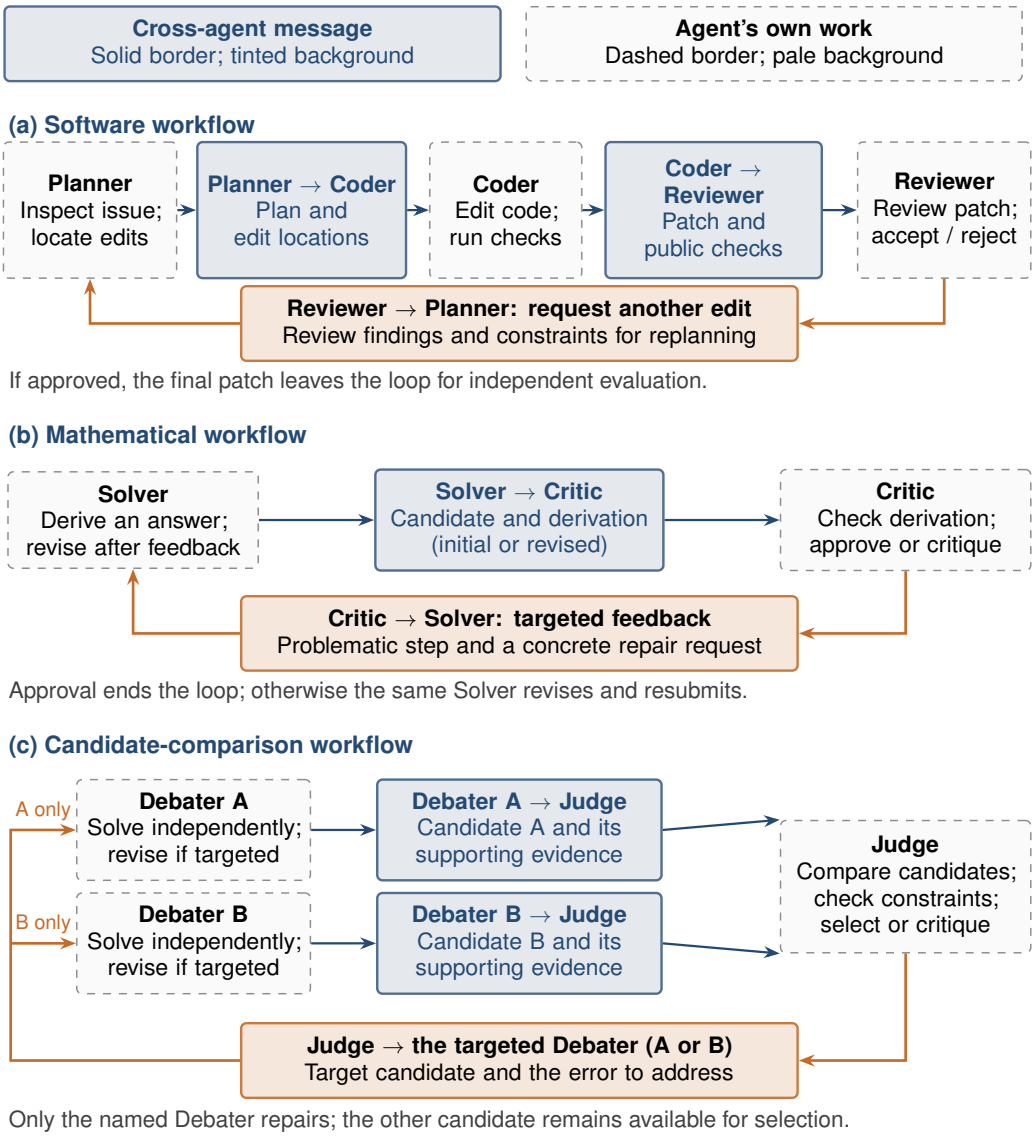

Figure S1: Collaboration is carried by explicit messages between roles. Solid, tinted cards name the sender, recipient, and message content; pale, dashed cards describe an agent's own work. Blue arrows carry plans or candidates forward; orange arrows return conditional feedback. In (b), feedback returns to the same Solver. In (c), the two return branches are alternatives: only the named Debater repairs. Approval or selection ends the loop, after which the task-native evaluator scores the final output. Private-reference access follows the collection or evaluation condition described in the text.

weak-to-strong collection, the initial solver is replaced by a weaker model and the original stronger model is restored for repair. This preserves the same role and feedback interface while changing the difficulty of the candidate that must be checked.

For ProofWriter and Zebra, two debaters first solve the task without reading each other's candidates. The judge then compares both candidates and either selects an answer or sends feedback to one named debater. If a reliable alternative already exists, it remains available as a fallback while the other candidate is revised. The runtime records which candidate was selected, so a successful final answer can be distinguished from a successful repair.

Reference access depends on the experiment, not the role name. During reference-informed reasoning collection, the critic or judge is the designated privileged node; only its permitted feedback is delivered to the solver or debater. The matched Perturb comparisons retain this checker configuration while changing the candidate or the target's task information. Public-only workflow evaluation instead uses checkers without private references. In either setting, an independent evaluator supplies the reported outcome.

A.3 Execution Engine and Runtime Controls

Our orchestration engine builds on the graph executor of ChatDev 2.0 (Dang et al., 2026). We retain its agent-node execution and graph-based context transport. CoEvoke supplies the task adapters, role and tool interfaces, evaluator integration, collection interventions, execution controls, and trajectory export needed for the experiments in this paper.

Execution controls operate at both role and task level. Tool permissions restrict each role to its allowed operations, and schema validation checks a role's output before it can activate a downstream edge. Bounded retries repair malformed outputs without treating them as valid collaboration messages. Shared task budgets count model calls, tool actions, tokens, and elapsed time across role transitions; switching agents does not reset the budget. Tool-observation limits and context compression keep long repository runs within the model's context capacity. When tool use ends, constrained finalization produces the structured object required by the next role.

Long-horizon memory is deliberately lightweight: the harness carries forward the task state, role messages, and review findings rather than maintaining a separate retrieval-based memory store. To make room for continued investigation, it shortens tool observations deterministically by retaining their beginning and end. In the reported SWE configuration, each observation is bounded to 8,192 characters; once the estimated context reaches 65,536 tokens, older observations are further reduced to 4,096 characters. The two most recent observations, together with the most recent patch-diff and failure-related observations, are exempt from this second reduction. This keeps each subsequent model call within a bounded working context while prioritizing recent and diagnostic evidence.

Collection controls preserve the relationship between a decision and its consequences. The planner retains review findings across rounds; feedback carries one identifiable target; and every revision links to its parent candidate. The routing logic limits repeated repair and retains an available candidate when further revision is not justified. Private references are supplied only to the configured node and are separated from public messages and ordinary training targets. Recorded tool calls, observations, candidate changes, and final sources support reconstruction of the resulting interaction.

The infrastructure also provides optional repeated-strategy detection and stalled-loop early stopping. These options are disabled in the reported SWE workflow and oracle comparisons, which instead use the configured review and resource limits. They are therefore implementation capabilities, not explanations of the gains measured in those comparisons.

A.4 Routing and Recorded Outputs

Algorithm 1 summarizes the shared control structure. A role makes its decision from the information permitted in that condition; reference-informed collection and public-only evaluation differ in those permissions, not in whether their final outcome is independently scored.

The full public record preserves role outputs, directed messages, tool interactions, candidate revisions, and the final source. Decision records select the public context and target action needed by a judge, critic, or verifier from that record. Separate audit records link these public events to evaluation and

Algorithm 1 Task-aligned trajectory collection

Require: Task x, role graph G, information permissions ν, shared budget B
Initialize the task state, candidate store, and event record
Execute the initial proposal or repository-edit roles allowed by G
while budget remains **do**
    Obtain a checking decision using the information allowed by ν
    if the decision selects an acceptable candidate **then**
        Record the selected candidate and its source
        break
    else if the decision names an eligible repair target **then**
        Deliver the targeted feedback and execute the repair role
        Record the new candidate and its parent; continue checking
    else
        Retain an available candidate and record the stopping reason
        break
    end if
end while
Link the final patch or answer to its source event
Score the final output with the independent task evaluator
return the trajectory, final output, and evaluator outcome

restricted reference metadata. This separation permits a training label to be traced back to the original collaboration without adding private materials to the learner's input.

A.5 Positioning the Collection Design

Table A1 distinguishes the construction choices of the closest trajectory-learning approaches. AgentRefine creates environments and feedback-driven refinements, while MALT samples a branching set of role outputs and assigns supervision using downstream outcomes. CoEvoke retains the source task and evaluator while intervening on what a designated role initially produces or observes. This permits a controlled study of whether the intervention changes feedback and revision adoption, and supplies records from which controller targets can be extracted. The table describes methodological scope rather than a common-benchmark performance ranking.

Table A1: **Related choices for constructing corrective supervision.** The comparison concerns each framework's reported design, not an exhaustive list of its possible extensions.

Framework	How corrective interaction is constructed	Use of the resulting supervision
AgentRefine (Fu et al., 2025)	Synthesized environments and scripts support trajectories containing errors and feedback-driven refinement.	Train the acting policy on refinement trajectories, with erroneous-turn loss masked.
MALT (Motwani et al., 2025)	Branch over generator, verifier, and refiner outputs; propagate final-outcome rewards to intermediate outputs.	Specialize generator, verifier, and refiner policies using role-specific supervision.
CoEvoke	Change a candidate, initial observation, or initial generator on an existing task; retain repair capacity and constrain reference-based assistance.	Record decisions and their subsequent revisions; train Judge, Critic, or Verifier modules with solving actors frozen.

B Canonical Event Graph and Schema

B.1 Event Graph and Top-Level Representation

The temporal event graph $\mathcal{T}_x = (\mathcal{N}_x, \mathcal{R}_x, \prec, \nu, \phi)$ formalizes the retained public record. Nodes represent agent outputs, tool calls, and tool observations; relations represent messages, tool responses, feedback, revisions, selections, and final sources. Visibility ν specifies which roles or audit procedures

may read each event. The map ϕ links a final artifact to its generating event. The public JSON object implements these relationships rather than merely wrapping a natural-language transcript.

The public record separates three kinds of information. The `task_layer` contains the public task, its source dataset, and its canonical identity, so multiple runs of one problem remain associated. The `trajectory_layer` describes the workflow and models, the ordered execution events, and the final artifact. Its outcome includes the stopping condition, the artifact's generating role, and the linked evaluation. The `communication_layer` records what each role actually delivered to another role. This last distinction matters under a perturbation: a solver's stored output may differ from the candidate delivered to its critic.

Evaluation records and private reference material are stored separately from role outputs. A role's approval is therefore an observed decision, whereas the task evaluator's verdict is the evidence used to score the final answer or patch. Configuration links identify the settings needed to interpret a run without copying private context into the ordinary training record.

Boundary of public reconstruction. During release collection, the SWE reviewer may receive restricted runtime summaries for initially passing (P2P) and initially failing (F2P) test groups. The immediate passing counts enter neither exported public trajectories nor training public contexts, but may affect subsequent reviewer decisions. Reconstructability therefore means recovering the retained public events, their links, and the evaluated artifact; it does not mean recovering every teacher-visible input or replaying the teacher's full decision state. The learner's public prefix in Equation 4 is consequently not asserted to equal the teacher's complete observation history. Runtime permissions and the historical prompt exception are detailed in Appendix K.4.

B.2 Steps, Tools, and Messages

Each role output has a stable step identifier, sequence number, role, execution phase, and public content. Tool calls additionally record the tool name and arguments. A retained tool observation identifies its call and stores the exportable returned content, status, and exit code; restricted evaluation counts are excluded as described above. A successful process exit is not itself evidence that the task was solved: the software example below includes a test command that exited normally after discovering no tests.

Directed messages identify their sender, recipient, source step, target step, and delivered content. Consequently, the record can distinguish feedback that was generated from feedback that reached a subsequent repair. Revision outputs identify the feedback they address, while `final_source.step_ids` specifies which execution supplied the final artifact. Together, these links identify who acted, which public evidence was retained, what changed, and which result was evaluated.

B.3 Deterministic Reconstruction

Algorithm 2 follows recorded events rather than resampling model calls. It checks event relationships separately from JSON syntax and task correctness. For software repair, the final patch is stored as an artifact attributed to a coder execution; it need not be repeated verbatim in the coder's prose report. This preserves the connection between an edit and its evaluated result without equating a role's description with the actual patch.

B.4 Record Excerpt: Source Output and Delivered Input

The following selected fields come from the canonical record for `target_deepmath_8096`; unrelated fields and events are omitted. Field names, nesting, step identifiers, and values are retained. The initial solver answer is 721, but the message delivered to the critic contains 722. The source output and the modified delivery therefore remain separately inspectable.

Algorithm 2 Reconstruct the recorded collaboration and its evaluated artifact

Require: Trajectory T, task-manifest entry M, evaluation record E
Ensure: Ordered events, delivered messages, final artifact, and task verdict
1: Validate required fields and the task identity shared by T and M
2: Order events by sequence; require unique event and call identifiers
3: **for** each tool observation **do**
4: Resolve its call identifier to exactly one preceding tool call
5: **end for**
6: **for** each delivered message **do**
7: Resolve its source and any nonterminal target to stored role events
8: **end for**
9: Resolve feedback and revision references; check their temporal order
10: Resolve final-source identifiers to the generating role executions
11: Read the stored final artifact and check its source attribution
12: Join E by evaluation identity and verify its task and output reference
13: Check agreement of the stored outcome and the manifest entry
14: **return** the reconstructed record, or a typed validation error

Stored output and delivered input


```
{
  "trajectory_layer": {
    "steps": [
      {"step_id": "s0001", "agent": "solver",
       "content": {"answer": "\\boxed{721}"}},
      {"step_id": "s0002", "agent": "critic",
       "content": {"route": "repair_solver"}},
      {"step_id": "s0003", "agent": "solver",
       "content": {"answer": "\\boxed{721}"}}
    ],
    "outcome": {
      "final_source": {"agents": ["solver"], "step_ids": ["s0003"]},
      "evaluation": {"verdict": "PASS"}
    }
  },
  "communication_layer": {
    "messages": [
      {"message_id": "m0001", "from_agent": "solver",
       "to_agent": "critic", "source_step_id": "s0001",
       "target_step_id": "s0002",
       "delivered_payload": {"answer": "\\boxed{722}"}}
    ]
  }
}
```


Stored: 721 **Delivered: 722** **Final: 721 (s0003)**

This excerpt illustrates how an input perturbation is represented, not a change to the solver's stored output. The next example concerns a different task and a different Perturb operation: the initial candidate is generated by a weaker model and is delivered without content edits.

B.5 Case Study: Weak-to-Strong Mathematical Repair

Figure S2 follows a matched geometry task whose two runs both end in PASS. In the strong-initial control, the first candidate is correct and the critic approves it. With weak initialization, an incorrect mass assignment produces an answer of $2/3$; the critic identifies the inverted ratio and the original strong solver revises the answer to $9/11$. The record links this revision to the specific feedback and identifies it as the evaluated final source. Thus equal task outcomes can provide different training material: approval alone, or a located error followed by a traceable correction.

Geometry task. In triangle ABC, $L \in AB$ and $M \in AC$ satisfy $AL = \frac{2}{5}AB$ and $AM = \frac{3}{4}AC$. Lines BM and CL meet at P; AP meets BC at N. Find BN/BC.

Solver: weak-model initialization

Qwen2.5-7B-Instruct generates the first candidate; its output is retained without editing.

↓ Deliver the candidate and its derivation

1. Solver → Critic: initial candidate

"Let the mass of point B be 5 and the mass of point A be 2."

The resulting candidate answer is $\boxed{BN/BC = 2/3}$.

↓ Critic checks the mass assignment

2. Critic → Solver: targeted feedback

Diagnosis. "The ratio AL:LB = 2:3, so m_A:m_B should be 3:2, not 2:5."

Repair request. "Reassign the masses at A, B, and C using the correct inverse relation: m_A * AL = m_B * LB and m_A * AM = m_C * MC."

↓ Return the feedback to the same Solver role

Solver: restore the original strong model for revision

DeepSeek-V4-Pro: "Choose a consistent mass for A." The revision sets $m_A = 3$, $m_B = 2$, $m_C = 9$, hence $BN : NC = 9 : 2$; it independently checks this ratio with Ceva's theorem.

↓ Deliver the revised candidate

3. Solver → Critic: corrected candidate

The new answer is $\boxed{BN/BC = 9/11}$. Its recorded parent is the initial candidate; the revision explicitly cites the Critic's feedback as the reason for the change.

↓ Critic checks the revision

4. Critic: approve the corrected reasoning

"The reasoning is publicly checkable and contains no errors."

Independent task evaluation: PASS. The evaluated final answer comes from the second Solver output $(9/11)$, not from the Critic's approval.

Matched strong-initial control: correct first candidate, direct approval

The first DeepSeek-V4-Pro candidate uses coordinates, obtains $N = (2/11, 9/11)$, and returns $BN/BC = 9/11$. The Critic approves; the same evaluator returns PASS.

2 calls: direct approval. 4 calls: located error → feedback → adopted correction.

Figure S2: Weak initialization creates a concrete repair opportunity on a real DeepMath task. Solid colored cards emphasize messages sent between agents; pale dashed cards show local work, approval, or the separate control run. Arrows mark delivery and checking in the recorded sequence, not extra model calls. Quotations reproduce public outputs; other text summarizes the record. The reference-informed Critic remains DeepSeek-V4-Pro, while the same Solver role switches from the weak initial model to the original strong model for repair. The separate outcome bar reports the independent evaluator, not the Critic's approval.

B.6 RECORD EXCERPT: LONG-HORIZON SOFTWARE REPAIR

Figure S3 follows 49 events from diagnosis to a software patch, separating public checks and collection-time review from official evaluation.

SWE-smith task: Red-DiscordBot aliases. Normal alias names are rejected, and alias translation returns an integer instead of the message expected by command processing.

1. Planner → Coder and Reviewer: localized repair plan

"call `trackform.format()` with `*args` instead of a single trailing arg." … "After formatting, compute the leftover arguments based on the highest consumed index from the formatter; if no placeholders were consumed, append all parsed args, otherwise append args after the last consumed positional slot." … "Return `new_message`."

↓ Coder receives the plan; Reviewer receives a copy for review

Coder: edit the targeted file and run public checks

Before Selected statements; intervening lines omitted.

```
return not bool(search(r"\S", alias_name)) and alias_name.isdigit()
command = trackform.format(alias.command, args[-1])
return 0
```

After

```
return not bool(search(r"\s", alias_name))
command = trackform.format(alias.command, *args)
leftover_args = args[trackform.max + 1 :] if trackform.max >= 0 else args
remainder = " ".join(leftover_args)
new_message.content = f"{prefix}{command} {remainder}".strip()
return new_message
```

Test attempt

`python -m unittest tests.cogs.test_alias`

"Ran 0 tests in 0.000s"

No behavioral tests ran.

Syntax check

`python -m py_compile .../alias.py`

Exit code: 0. Syntax is valid.

↓ Deliver the patch and report its changes

2. Coder → Reviewer: report the completed repair

"Repaired `translate_alias_message()` to format aliases with the full parsed argument list, preserve only unused trailing arguments after placeholder consumption, rebuild translated content in `prefix + command + remainder` order, and return the copied message object."

↓ Reviewer inspects the patch against the plan

Reviewer: inspect, evaluate during collection, and approve

Release-collection feedback. Reviewer calls `run_evaluation`; recorded exit code: 0. Restricted aggregate test-pass counts can inform review but are omitted from the public record; hidden test details are not exposed.

Approval. "This round repaired both intended production defects in the targeted file: alias creation now accepts normal single-token names, and alias invocation now produces a valid translated message with correct argument expansion and forwarding semantics."

Next action: "None; patch is acceptable."

Final post-run SWE-smith evaluation: PASS.

Evaluated artifact: final patch. Recorded source: Coder execution.

Figure S3: A release-corpus software trajectory links the Planner's instructions to the Coder's edits and the Reviewer's approval. Solid colored cards show inter-agent messages; pale dashed cards show each agent's work. Arrows identify the handoffs, not additional model calls. Quotes reproduce role outputs; ellipses and selected code omit intervening content. Public compilation and the test command that ran no tests differ from restricted evaluation feedback. Collection-time feedback and final post-run scoring use the same test-scoring logic, with different delivery times and recipients. Immediate P2P/F2P counts may influence review but are absent from the public record and training contexts.

C Dataset Statistics

C.1 Scale, Outcomes, and Length

Table C1: **Dataset composition and recorded execution length.** P50/P95 denote the median and 95th percentile. Token totals accumulate provider-reported input and output usage across calls, including retries and private calls where recorded; they are not lengths of the exported public training sequences. Paired variant groups share a canonical task.

Dataset	Tasks	Traces	Pass/Fail	Variant groups	Steps P50/P95	Tokens P50/P95
SWE	1,763	1,763	1,561/202	0	60/305	210,689/1,540,444
DeepMath	831	923	917/6	92	4/4	9,977/38,021
ProofWriter	1,096	1,392	1,392/0	296	5/5	8,945/16,920
Zebra	440	489	489/0	49	5/5	20,510/54,304
Total	**4,130**	**4,567**	**4,359/208**	**437**	5/187	17,592/852,434

The SWE long tail reflects repository investigation, file inspection, tests, and rework. Short reasoning traces concentrate candidate competition and repair into four or five role steps. The representation must therefore accommodate hundreds of tool events while precisely distinguishing transformations of a single candidate message.

C.2 Process Composition

Table C2: **Recorded collaboration events.** One-shot denotes no feedback message and no revision-role output, irrespective of final correctness. Repair attempts require a revision; diagnostic failures are final evaluator failures. These are deterministic structural extractions, not human judgments of semantic quality.

Dataset	Traces	One-shot	Candidate disagreement	Critique event	Repair attempt	Conservative recovery	Diagnostic failure	Rounds mean/P95
SWE	1,763	1,028 (58.3%)	–	–	735 (41.7%)	–	202 (11.5%)	1.880/5
DeepMath	923	208 (22.5%)	–	715 (77.5%)	715 (77.5%)	643 (69.7%)	6 (0.7%)	1.794/2
ProofWriter	1,392	493 (35.4%)	521 (37.4%)	899 (64.6%)	899 (64.6%)	415 (29.8%)	0	1.663/2
Zebra	489	77 (15.7%)	296 (60.5%)	412 (84.3%)	412 (84.3%)	176 (36.0%)	0	1.851/2
Total	**4,567**	1,806 (39.5%)	–	–	2,761 (60.5%)	–	208 (4.6%)	1.794/4

A repair attempt records at least one revision, not necessarily a correct change. One-shot completion is not first-attempt accuracy, and a diagnostic failure is a final task failure rather than an intermediate tool failure. Collaboration rounds use the maximum directed-message round, defaulting to one; P50/P95 use nearest-rank quantiles computed directly over the retained trajectories. Conservative recovery additionally requires an earlier answer different from the correct final answer and a feedback-linked revision selected as the evaluator-confirmed final source. This criterion distinguishes adopting a correction from merely generating one. These extracted events are not substitutes for human judgments about the quality of the explanation.

C.3 Event Counts

Directed messages exceed role outputs in SWE, ProofWriter, and Zebra because a recorded output can be delivered along multiple routes (Table C3). Distinguishing production from delivery identifies

which agents actually received the evidence; a flattened list of outputs would lose that information. DeepMath's single-candidate loop has one recorded message per output.

Table C3: **Role outputs and explicit directed messages.**

Dataset	Role outputs	Messages	Mean outputs per trace	Mean messages per trace
SWE	9,945	13,081	5.641	7.420
DeepMath	3,312	3,312	3.588	3.588
ProofWriter	6,022	6,945	4.326	4.989
Zebra	2,299	2,715	4.701	5.552
Total	**21,578**	**26,053**	4.725	5.705

The corpus records 80,428 tool calls and 80,428 corresponding observations, all from SWE. A step counts one role output, tool call, or tool observation, giving 182,434 steps in total (170,801 in SWE). Messages count directed deliveries, including the original terminal-delivery records; an observation is not additionally counted as an inter-role message. Recorded cumulative provider input-plus-output usage totals 786,805,655 tokens, with usage coverage for every retained trajectory. This dataset accounting includes recorded retries and private oracle calls and differs from the net provider-token measure in Appendix E.6.

C.4 Collection Configurations and Training Eligibility

Table C4 describes only the 4,567 trajectories selected for release, independently of the controlled experiments. Its 300 clean ProofWriter and 52 clean Zebra trajectories are not the Natural-SFT training pool: that control uses independently collected clean runs (Appendix G.4). Each entry counts a recorded trajectory, so variants of one task can appear in different rows. All reasoning configurations use reference-informed checking: *clean* means that no Perturb was scheduled, while an *unapplied attempt* means that a controlled perturbation was scheduled but not applied. Keeping these categories separate preserves the distinction between collection intent and the input actually delivered to a role.

Table C4: **Actual configurations of the retained trajectories.** Guide accompanies every listed condition, through an Oracle-to-Planner hint in SWE and a reference-informed Critic/Judge in reasoning. Counts exclude separate experimental runs. A dash denotes a configuration absent from that source in this release.

Collection configuration	SWE	DeepMath	ProofWriter	Zebra	Total
Oracle-guided software workflow	1,763	–	–	–	1,763
Clean reasoning + Guide	–	96	300	52	448
Weak-to-strong + Guide	–	452	422	235	1,109
Applied candidate corruption + Guide	–	344	366	201	911
Applied partial observation + Guide	–	–	286	–	286
Controlled attempt, unapplied + Guide	–	31	18	1	50
Total	**1,763**	**923**	**1,392**	**489**	**4,567**

The SWE planner, coder, and reviewer use GPT-5.4 through the Responses interface with medium reasoning effort. Reasoning collection uses DeepSeek-V4-Pro for checking and strong-model generation or repair. Under weak-to-strong collection, Qwen2.5-7B-Instruct instead supplies the initial solver output in DeepMath or one initial debater candidate in ProofWriter and Zebra, after which the stronger model resumes the same role for a requested revision. The weak initial role is debater A/B in 212/210 ProofWriter traces and 117/118 Zebra traces. Candidate corruption is applied by deterministic rules and does not require an additional generation model.

The 50 unapplied attempts remain a separate release category and do not count as applied perturbations. Partial-observation trajectories in this release come only from ProofWriter. Figure S4 summarizes how the recorded conditions vary by source without treating the different configurations as additional unique tasks.

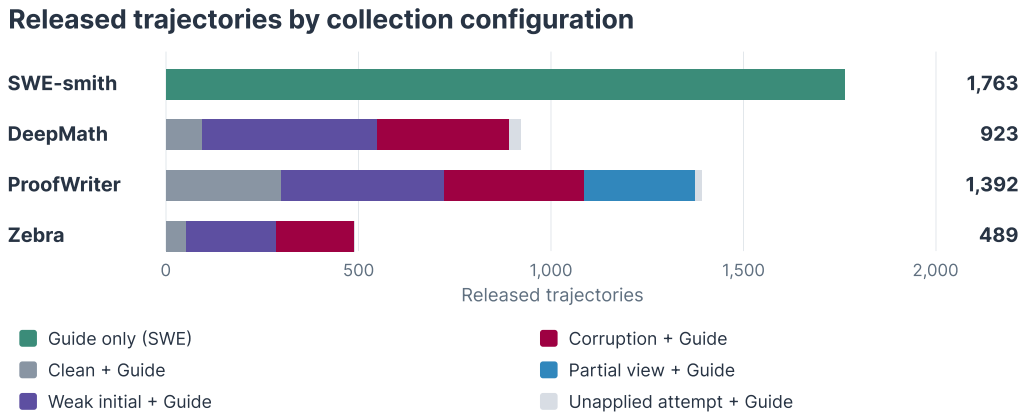

Figure S4: Trajectories retained for release by source and collection configuration. Segments count corpus trajectories in Table C4; right-hand labels give source totals. "Guide only (SWE)" denotes the Oracle-guided software workflow. All reasoning categories use Guide: "Clean" means no scheduled Perturb, and "Unapplied attempt" means a scheduled perturbation was not delivered. Separate experimental runs are excluded.

The retained canonical partitions contain 4,338 train and 229 internal-development trajectories; these source-record partitions are distinct from the derived splits used in training experiments. The proposed release manifest contains exactly the retained 4,567 records. Checks of both SWE verifier training views (1,318 tasks and 2,010 records each), their SFT development views (8 tasks and 15 records each), and the 1,000-task Actor source list find no excluded records. These checks concern the specified derived views and manifest, rather than a completed redistribution package.

Dataset inclusion requires complete public role outputs and resolvable links among tool calls, observations, delivered messages, feedback, revisions, and final artifacts. Only records satisfying these reconstruction checks enter the release, its statistics, and training views. This structural criterion is independent of task success: the retained corpus includes 208 evaluator-confirmed failures with complete interactions. Intentional isolation of private Guide context is a separate visibility restriction and does not make the recorded public interaction incomplete.

D Quality Audit

D.1 Human Rubric

We assess the multi-agent interaction, not just the final answer. Seven dimensions receive 0, 1, or 2 points, indicating absent or inadequate, partial, and clear evidence of the corresponding behavior. *Role contribution* asks whether each role performs substantive work; *handoff quality* asks whether its recipient receives usable information. *Evidence grounding* rewards claims tied to the task, tools, or candidate, whereas *reviewer-added value* requires checking or diagnosis beyond repeating the proposal. *Recovery diagnostic value* credits successful repair, actionable diagnosis, or a supported decision that repair is unnecessary. *Trajectory coherence* concerns consistency across these steps, and *non-redundant efficiency* distinguishes productive work from repetition.

These dimensions deliberately need not move together. A solver may produce a correct answer that the critic immediately approves: the record can be grounded, coherent, and efficient, yet provide little evidence of independent review or recovery. Its overall collaboration grade can therefore be lower than that of a successful, well-supported correction. Conversely, brevity alone is not a weakness: a concise acceptance earns review credit when it reports an actual check. Additional rounds receive credit only when they add evidence or resolve uncertainty; the rubric does not reward interaction for its own sake.

Process grades summarize the seven dimension scores. The model audit maps their total to A (11–14), B (8–10), C (5–7), or D (0–4). Human annotators use the score bands as guidelines and can assign B

to a higher-scoring interaction with a substantial weakness in role contribution, review, or efficiency. Task-native outcomes provide a separate correctness reference. This separation allows process quality and final success to be examined together without defining one by the other.

1. Solver: simplify the public expression

For real x, simplify $49x^2 + 14x(19 - 7x) + (19 - 7x)^2$.
"Simplifying the expression: $-49x^2 + 49x^2 + 266x - 266x + 361 = 361$."

↓ Candidate and derivation sent to Critic

2. Critic: check, then approve

"The x terms cancel, leaving the constant 361. The final answer is boxed as 361, which is consistent with the simplification."
Decision: approve. The original Solver answer remains the final output and receives PASS.

Figure S5: Correct approval supplies different evidence from error recovery. This real two-step trajectory contains a grounded algebraic check and a coherent handoff, but no diagnosis of an erroneous candidate or subsequent repair. It illustrates why task success, review quality, and recovery supervision are assessed separately. The quotations are public role outputs; this illustration does not assign an additional human grade.

Disclosure is assessed separately. L0 denotes no observed private information transmitted to the public process. L1 denotes disclosure of a protected reference or evaluation mechanism without revealing its substantive content; L2 denotes disclosure of a protected answer, label, or artifact. LX denotes insufficient evidence to decide. The audit checks delivered messages and their recipients, not merely whether an agent's internal context contained a reference.

D.2 SAMPLING AND HUMAN RESULTS

Human–model agreement and collection quality require different sampling designs. To assess agreement, we retain all 240 complete ratings from five experts, covering 160 distinct trajectories across annotation rounds. Eighty trajectories have two ratings and eighty have one; each original human rating is compared with the same existing model judgment for its trajectory. The pool includes probability-sampled and targeted cases.

The five annotators are expert researchers in AI. Human and model auditors receive the same public role messages and tool-action evidence, including SWE tool-call names and arguments. The audit view includes task identifiers and task-level PASS/FAIL outcomes; SWE tool outputs and protected reference answers are withheld. Humans are blinded to model assessments, which are compared with their ratings after annotation. Agreement therefore concerns judgments from this shared evidence view.

For collection-quality estimates, we instead use a stratified probability sample of 500 trajectories, with the sampling design fixed before model assessment. Targeted cases contribute to agreement analysis but never to the prevalence estimates in Table 2. Previously audited tasks and their variants were excluded from the probability-sampling frame. Retained-ID verification confirms that every probability-sampled and human-rated trajectory satisfies the reconstruction criterion; the fixed audit sample, ratings, and design weights are retained. The weighted estimates describe this fixed probability-audit design, not a census of the release corpus. For label ℓ, its weighted proportion is $\sum_i w_i \mathbb{1}[y_i = \ell] / \sum_i w_i$, where w_i is the fixed design weight. Raw sample counts are reported separately below.

The model agrees with 201 of 240 human collaboration grades and 218 of 240 disclosure labels (Table D1). Most grade disagreements concern the distinction between A and B: 36 of the 39 disagreements lie between the two highest process-quality grades. As a same-sample reference, the 80 doubly rated trajectories yield human–human agreement of 78.8% for grades and 85.0% for disclosure, compared with 84.4% and 90.0% across their 160 human–model pairs. This comparison

Table D1: **Pooled agreement with all five human experts.** Each of the 240 ratings is paired with its trajectory's existing model judgment. Confidence intervals resample the 160 canonical tasks, retaining all ratings within each task.

Assessment	Exact agreement (%)		κ_w / κ	
	Estimate	95% CI	Estimate	95% CI
Collaboration grade	**83.8**	[78.8, 88.6]	0.779	[0.691, 0.849]
Disclosure label	**90.8**	[86.9, 94.6]	0.688	[0.553, 0.805]

Collaboration grades use linearly weighted κ_w; disclosure labels use unweighted κ. The mean absolute error in the 0–14 score is 1.10 points (95% CI [0.91, 1.27]). Intervals use 5,000 task-cluster bootstrap resamples.

contextualizes the pooled agreement without treating either an individual annotator or the model as an error-free reference. Giving each trajectory equal weight yields similar agreement, 83.4% for grades and 91.3% for disclosure. The reported intervals retain within-task dependence; they do not estimate variation over a population of human experts.

D.3 Two-Agent Model Auditing

Expanded auditing uses two separate components for collaboration quality and private-information disclosure. Each component has an Assessor and an Adjudicator (Figure S6). The Assessor reads the trajectory and proposes a judgment with short quotations and exact message or tool identifiers. The Adjudicator receives the original evidence as well as this proposal and can revise the judgment. This second pass checks the interpretation and supporting evidence; it is not a blind vote from an independent annotator.

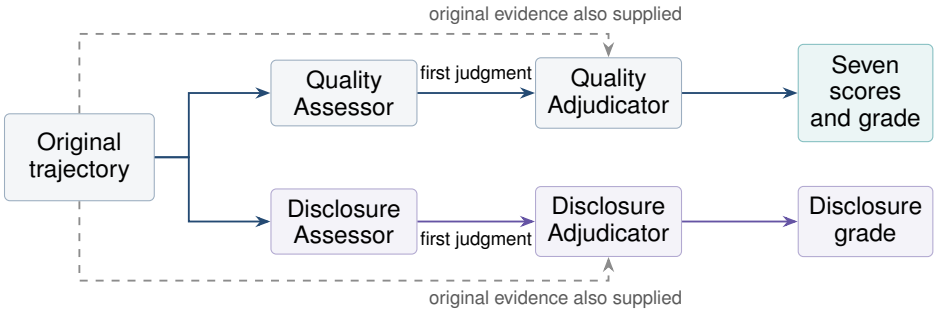

Figure S6: Each audit component combines an initial assessment with evidence-based adjudication. Solid arrows pass the first judgment to the Adjudicator; dashed arrows supply the original trajectory directly. Quality and disclosure have separate outputs and four distinct role prompts. Neither component replaces the task-native evaluator.

Both components use gpt-5.6-sol with high reasoning effort. The quality component cites interaction evidence for its scores and uses the authoritative task outcome as a correctness reference. Validation checks cited identifiers, output structure, and consistency between the total process score and its grade. Appendix K.5 presents all four prompts; machine-readable output constraints accompany the implementation.

D.4 Expanded Quality and Disclosure Results

Across the probability sample, the design-weighted grade-A share is 93.1%, ranging from 85.1% in SWE to 100.0% in Zebra (Table D2). This profile supports the quality of the recorded public interactions; the 32 grade-C trajectories, including 31 from SWE, also retain lower-scoring interactions for analysis. Disclosure assessment identifies ten L1 cases in SWE and DeepMath, corresponding to a weighted share of 2.0%, with no L2 cases in the 500 audited trajectories. The audit distinguishes references to private authority from disclosure of protected solution content; zero observed L2 cases is not a guarantee of zero population risk.

Table D2: **Raw counts and design-weighted estimates for the probability-sample audit.** Counts are unweighted; percentages and 95% confidence intervals use sampling weights. The overall row pools the design weights, rather than averaging dataset percentages; no overall interval is reported.

Collaboration quality				
Dataset	N	**A/B/C/D**	**Weighted A (%)**	**95% CI**
SWE Oracle	205	174/0/31/0	85.1	[79.5, 89.3]
DeepMath	97	95/1/1/0	97.9	[92.8, 99.4]
ProofWriter	146	144/2/0/0	98.6	[95.1, 99.6]
Zebra	52	52/0/0/0	100.0	[93.1, 100.0]
Overall	**500**	**465/3/32/0**	**93.1**	–
Private-information disclosure				
Dataset	N	**L0/L1/L2/LX**	**Weighted L0 (%)**	**95% CI**
SWE Oracle	205	197/8/0/0	96.1	[92.5, 98.0]
DeepMath	97	95/2/0/0	97.9	[92.8, 99.4]
ProofWriter	146	146/0/0/0	100.0	[97.4, 100.0]
Zebra	52	52/0/0/0	100.0	[93.1, 100.0]
Overall	**500**	**490/10/0/0**	**98.0**	–

Figure S7 separates evidence grounding from economy of interaction. Evidence-grounding means are at least 1.928 in all four datasets; non-redundancy varies more, reaching 1.500 in Zebra. The first dimension assesses support for conclusions throughout a trajectory, including runs without feedback. These dimensions distinguish well-supported reasoning from economical collaboration: recorded trajectories preserve both for analysis, while controller training extracts the evidence relevant to a particular decision.

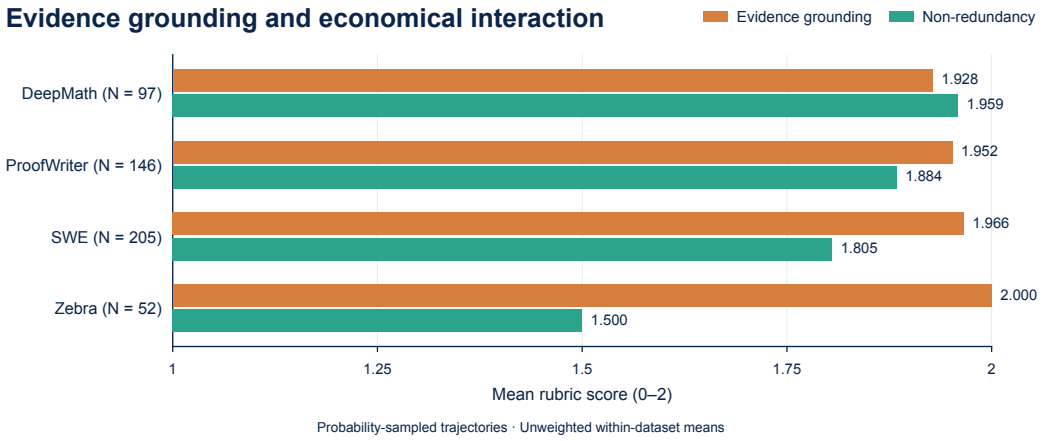

Figure S7: Evidence grounding and economical interaction are distinct qualities. Unweighted within-dataset means use all 500 probability-sampled trajectories and a 0–2 rubric (axis shown from 1); larger values mean better grounding or less redundancy. Orange and green identify the two dimensions. Evidence grounding assesses the trajectory's conclusions, including those in runs without feedback.

The intervals in Table D2 use sampling-weighted Wilson intervals with Kish effective sample sizes. The raw grade-A share is 465/500 (93.0%); its weighted counterpart is 93.1% because the design represents strata with different sampling probabilities. Targeted inspection serves a different purpose—finding uncommon failure and disclosure patterns—and remains separate from these distribution estimates.

E PERTURB AND GUIDANCE EXPERIMENTS

E.1 THREE WAYS TO ELICIT CORRECTIVE INTERACTION

Perturb creates opportunities for feedback through candidate corruption, partial observation, or weak-to-strong recovery. We evaluate all three in a controlled experiment comprising 2,287 matched comparisons over 2,287 distinct tasks. These experimental runs are separate from dataset collection: task identities may overlap, but neither condition's trajectories enter the corpus in Table 1. The seven task–intervention groups are disjoint: 1,197 tasks test input perturbations and 1,090 test weaker initial generation. Every comparison includes a task-native PASS/FAIL verdict in both conditions. Results are reported by group rather than as a cross-domain accuracy average.

The *control* preserves the original candidate or information for input perturbations and uses the original strong model for weak-initialization comparisons. The *perturbed condition* changes that input or initial generator while retaining the checker and subsequent repair capacity. This distinction lets us create a reason to collaborate without simultaneously removing the ability to resolve it. We distinguish feedback, revision, adoption of a revision as the final source, and a successful final answer: neither missing information nor weaker generation guarantees an initial error, so additional feedback is not automatically a verified error detection.

Checkable candidate errors. The runtime applies candidate-editing rules only on the designated role's first execution, after a deterministic task-specific check verifies the original candidate as correct. For DeepMath, eligible answers are integers: the rule changes the value by $+1$ or -1, or reverses the sign of a boxed integer, updating its normalized answer field consistently. ProofWriter switches `true`/`false` or `false`/`unknown` and adjusts the associated classification rationale; Zebra swaps two unequal values in the same non-identifier column of the first two table rows. A task-and-role-dependent deterministic ordering selects among applicable rules. The changed candidate is delivered only if the same check now verifies it as incorrect; otherwise the original is retained. These edits preserve the public task and the evidence needed to question the candidate, although label corruption can also alter the local rationale. Revised candidates are never corrupted again.

Partial observation. For ProofWriter, the runtime removes one base fact from the initial task delivered to one debater. Eligibility requires a true or false answer, a unique derivation in the supplied proof metadata, no alternative proof branch, no derivation of the opposite statement, and a removable base fact occurring exactly once. The selected fact must itself have no alternative derivation. The query, rules, and remaining facts stay unchanged, while the other debater and the judge retain the complete task. Recording both task versions makes the intervention reversible; not repeating the deletion during repair allows later feedback to address the missing evidence. This is a role-specific information restriction, not a change to the task evaluated at the end.

Weak initial generation. The third operation leaves candidate text untouched. Qwen2.5-7B-Instruct supplies the first proposal, after which DeepSeek-V4-Pro performs checking and any requested revision. When feedback triggers repair, the original stronger model resumes the same solver or debater role; there is no additional Repair agent. Correct first proposals can be approved immediately. Thus this operation supplies naturally generated errors without requiring the weaker model to carry out the repair as well.

E.2 MATCHED CONTROLS AND FEEDBACK ACTIVATION

The input-perturbation subset contains 911 candidate corruptions and 286 partial observations. For candidate corruption, the control reuses the candidate before modification and executes the same downstream roles. For partial observation, it restores the complete task and reruns the affected solver while keeping the independent candidate fixed. This controls the information available to the solver rather than requiring identical generated candidates.

Weak initialization uses Qwen2.5-7B-Instruct for the first candidate and DeepSeek-V4-Pro for feedback and requested repair. Its control replaces the initial weak generator with the original strong-model configuration; an independent debater candidate is replayed exactly where applicable. The task, checking protocol, and final evaluator remain fixed.

Across these comparisons, DeepSeek-V4-Pro (thinking enabled; medium) remains the reference-informed collection checker. Holding its prompt and reference access fixed isolates changes to the candidate, visible information, or initial generator within that setting. Guidance is instead evaluated across five models because both the usefulness of a hint and the response to it can depend on the model. This allocation tests different questions; the single-checker Perturb study does not establish model-independent effects.

Table E1: **Feedback activation across all seven Perturb comparisons, corresponding to Figure 3.** Rates count runs with at least one feedback message. Controls use the original input or a strong initial generator; all checkers use DeepSeek-V4-Pro (thinking enabled; medium). Each percentage uses its matched-task denominator N. Bold/underline compare feedback rates within each row; Table E3 reports the paired tests.

Task / perturbation	N	Control feedback (%)	Perturbed feedback (%)	Δ (pp)
DeepMath / answer corruption	344	<u>0.00</u>	**100.00**	↑+100.00
ProofWriter / label corruption	366	<u>0.55</u>	**100.00**	↑+99.45
ProofWriter / partial observation	286	<u>0.70</u>	**98.95**	↑+98.25
Zebra / table corruption	201	<u>0.00</u>	**100.00**	↑+100.00
DeepMath / weak initialization	442	<u>7.47</u>	**77.38**	↑+69.91
ProofWriter / weak initialization	417	<u>5.04</u>	**52.76**	↑+47.72
Zebra / weak initialization	231	<u>0.87</u>	**89.18**	↑+88.31

Table E2: **Final outcomes for the 1,197 input-perturbation pairs only.** Accuracy is the number of PASS runs divided by N. Both conditions solve every task in this subset; weak-initialization outcomes are reported separately in Table E8. Equal accuracy estimates share the best rank in bold.

Task / perturbation	N	Original PASS	Original FAIL	Perturbed PASS	Perturbed FAIL	Original accuracy	Perturbed accuracy
DeepMath / answer	344	344	0	344	0	**100.00%**	**100.00%**
ProofWriter / label	366	366	0	366	0	**100.00%**	**100.00%**
ProofWriter / partial observation	286	286	0	286	0	**100.00%**	**100.00%**
Zebra / table	201	201	0	201	0	**100.00%**	**100.00%**
All evaluable pairs	**1,197**	**1,197**	**0**	**1,197**	**0**	**100.00%**	**100.00%**

Both conditions solve the input-perturbation tasks (Table E2), but this does not make their trajectories equivalent. The original input rarely demands correction; the perturbed input elicits explicit diagnosis and repair. Weak initialization creates the same opportunity through naturally generated candidates, increasing feedback in all three task families (Table E1). Final-answer provenance then distinguishes producing a revision from using it.

Table E3 shows that newly activated feedback greatly exceeds lost feedback. Feedback and revision differences use 20,000 task-paired bootstrap samples, keeping each task's two conditions together; the weak-initialization subtotal is stratified by dataset. McNemar tests use the two discordant counts. The three weak-initialization feedback effects also remain significant after Holm correction over the nine exploratory task-family–endpoint tests (feedback, revision, and final outcome). When all tasks change identically, an empirical bootstrap interval can collapse to a point; this does not guarantee universal activation in future runs.

Final-source attribution identifies corrections that contribute to the delivered answer (Table E4). A run counts once when its final outcome is PASS, its selected source is a revision, and that revision's normalized answer differs from the same role's first candidate delivered for checking in the treatment run. Candidate-corruption groups compare against the injected answer; partial-observation groups compare against that role's first locally generated candidate. Of the 732 qualifying runs, 731 select a revision from the targeted role; one partial-observation run selects the other role's revision.

Table E3: **Paired changes in feedback and revision occurrence.** "New" means absent in the control but present after Perturb; "lost" denotes the reverse. Feedback and revision indicators have identical paired counts in these groups. Intervals are task-paired bootstrap intervals; exact McNemar values are unadjusted. Subtotals describe their respective intervention subsets.

Task / perturbation	Control	Perturbed	New	Lost	Δ (pp) [95% CI]	Exact McNemar p
DeepMath / answer	0	344	344	0	↑+100.00 [100.00, 100.00]	5.581×10^{-104}
ProofWriter / label	2	366	364	0	↑+99.45 [98.63, 100.00]	5.322×10^{-110}
ProofWriter / partial observation	2	283	281	0	↑+98.25 [96.50, 99.65]	5.148×10^{-85}
Zebra / table	0	201	201	0	↑+100.00 [100.00, 100.00]	6.223×10^{-61}
Input-perturbation subtotal	**4**	**1,194**	**1,190**	**0**	↑+99.42 [98.91, 99.83]	$<10^{-300}$
DeepMath / weak initialization	33	342	312	3	↑+69.91 [65.38, 74.21]	1.561×10^{-88}
ProofWriter / weak initialization	21	220	201	2	↑+47.72 [42.93, 52.52]	3.221×10^{-57}
Zebra / weak initialization	2	206	204	0	↑+88.31 [83.98, 92.21]	7.779×10^{-62}
Weak-initialization subtotal	**56**	**768**	**717**	**5**	↑+65.32 [62.57, 68.07]	1.472×10^{-205}

Table E4: **Selected correct revisions in the input-perturbation subset.** Counts require an answer change, selection as the final source, and PASS. Target-role counts include only the role receiving the perturbation. Conditional proportions use perturbed runs with feedback and Wilson 95% intervals.

Task / perturbation	N	Selected revision	Target role	Among all pairs (%)	Among runs with feedback (%)	Conditional 95% CI (%)
DeepMath / answer	344	344	344	100.00	100.00	[98.90, 100.00]
ProofWriter / label	366	171	171	46.72	46.72	[41.67, 51.84]
ProofWriter / partial observation	286	131	130	45.80	46.29	[40.57, 52.11]
Zebra / table	201	86	86	42.79	42.79	[36.14, 49.70]
All evaluable pairs	**1,197**	**732**	**731**	**61.15**	**61.31**	**[58.51, 64.03]**

Answer keys use whitespace-normalized strings for DeepMath, case-normalized labels for ProofWriter, and canonicalized cells and row order for Zebra. String keys are trimmed and lowercased before comparison. This checks changes in the recorded answer representation; the DeepMath comparison does not perform general mathematical-equivalence checking. Final correctness is supplied separately by the task evaluator.

Table E5: **Additional execution cost of controlled perturbations.** Costs cover only the evaluable pairs in Table E2, including initial candidate generation and prefixes reused by original-input runs. Per-task differences use each group's evaluable-pair count.

Task / perturbation	Original calls	Perturbed calls	Δ calls per task	Original tokens	Perturbed tokens	Δ tokens per task
DeepMath / answer	713	1,409	↑+2.02	2,263,460	4,336,955	↑+6,027.6
ProofWriter / label	1,104	2,020	↑+2.50	1,999,122	3,858,581	↑+5,080.5
ProofWriter / partial observation	881	1,494	↑+2.14	1,340,231	3,281,898	↑+6,789.0
Zebra / table	604	1,148	↑+2.71	2,879,421	5,449,969	↑+12,788.8
All evaluable pairs	**3,302**	**6,071**	↑+2.31	**8,482,234**	**16,927,403**	↑+7,055.3

Input perturbations added 2,769 calls and 8,445,169 tokens overall. These token totals accumulate recorded input and output usage to describe execution cost and use a different accounting scope from the net provider-token SWE comparison. Charging reused prefixes to original-input runs prevents caching from appearing as an intrinsic methodological cost advantage.

Table E6: **Final-answer sources for the 1,197 input-perturbation tasks.** The categories are mutually exclusive. Other PASS does not meet the selected correct answer-changing revision criterion and must not be equated with failed repair. Weak-initialization source adoption has a different definition (Table E7).

Task / perturbation	N	Correct revision is final answer	Other PASS	FAIL
DeepMath / answer	344	344 (100.00%)	0 (0.00%)	0 (0.00%)
ProofWriter / label	366	171 (46.72%)	195 (53.28%)	0 (0.00%)
ProofWriter / partial observation	286	131 (45.80%)	155 (54.20%)	0 (0.00%)
Zebra / table	201	86 (42.79%)	115 (57.21%)	0 (0.00%)
All evaluable pairs	**1,197**	**732 (61.15%)**	**465 (38.85%)**	**0 (0.00%)**

Table E6 separates adopted revisions from other successful solutions. Their relative frequency depends on workflow structure: a single-solver derivation is directly revised, whereas a multi-candidate judge may preserve or select another valid answer. Both are useful control decisions.

E.3 WEAK-TO-STRONG RECOVERY

Weak initialization increases the number of runs that contain explicit feedback and revision, without editing the candidate's content. Across its 1,090 matched tasks, 56 strong-initial controls contain feedback and a revision, compared with 768 weak-initial runs. The additional interactions are not forced repair rounds: correct initial candidates can still be approved, and the stronger solver is called again only when the checker requests revision.

Table E7 distinguishes the occurrence of a revision from its adoption. Every DeepMath run with a revision selects a revised candidate as its final source. ProofWriter and Zebra retain multiple candidates, so a judge can request a revision and ultimately select a different answer. This explains why those workflows record more revision opportunities than final-source revisions; neither count alone measures semantic correction.

Comparable outcome assessment. For this diagnostic comparison, both conditions use the same task evaluator.

Table E7: **Feedback and revision in the matched weak-initialization study.** Arrows show strong-initial control → weak-initial/strong-repair counts. A feedback task has at least one message; events count individual messages. A selected revision is identified by final-source provenance, without assuming the initial answer was wrong.

Dataset	N	Feedback tasks	Feedback events	Runs with a revision	Revision selected as final source
DeepMath	442	33 → 342	35 → 357	33 → 342	33 → 342
ProofWriter	417	21 → 220	28 → 225	21 → 220	11 → 96
Zebra	231	2 → 206	2 → 209	2 → 206	0 → 89
Total	**1,090**	56 → 768	65 → 791	56 → 768	44 → 527

Table E8: **Final outcomes for weak initialization under an identical historical evaluator in both conditions.** Success rates use N; p is the two-sided exact McNemar value. Bold/underline compare success rates within a row, with ties sharing rank.

Dataset	N	Control success (%)	Weak-initial success (%)	Δ (pp) [95% CI]	p
DeepMath	442	**87.78**	<u>86.65</u>	↓ −1.13 [−4.30, +2.04]	0.5682
ProofWriter	417	**100.00**	**100.00**	0.00 [0.00, 0.00]	1.0000
Zebra	231	**100.00**	**100.00**	0.00 [0.00, 0.00]	1.0000

The increase in interaction does not require a corresponding increase in solved tasks. ProofWriter and Zebra reach the same final success with more feedback. DeepMath changes from 388 to 383 successful runs among 442 tasks: 22 control failures become successes and 27 successes become failures, giving −1.13 points with a paired interval spanning zero. Among weak-initial runs, 295 DeepMath, 96 ProofWriter, and 89 Zebra final-source revisions receive PASS.

Cost of generating additional interactions. Weak initialization adds 1.73–2.22 logical calls per task across the three groups (Table E9). This is a collection trade-off: a weaker initial proposal creates a reason to check, while the original strong model remains available to act on the resulting feedback. Costs include the original prefix of an independently replayed candidate; reused computation is not counted as free. Token totals reflect the recorded generation calls, not billing-normalized prices across models.

Table E9: **Matched weak-initialization costs.** Arrows show strong-initial control → weak-initial/strong-repair means. Totals include the common replay prefix where applicable; additional calls and tokens are summed over each group's N tasks.

Dataset	N	Calls per task	Tokens per task	Additional calls	Additional tokens
DeepMath	442	2.22 → 3.95	7,633 → 13,747	764	2,702,798
ProofWriter	417	3.15 → 4.99	6,373 → 9,118	767	1,144,497
Zebra	231	3.34 → 5.55	14,971 → 23,469	512	1,963,029
Total / task mean	**1,090**	2.81 → 4.69	8,706 → 14,037	**2,043**	**5,810,324**

The paired mathematical example in Figure S2 illustrates the collection effect directly: a strong initial candidate is approved, whereas a weak initial candidate elicits a specific diagnosis and a strong-model repair. Both runs solve the same task. The figure connects the public feedback to the adopted revision without inferring a general accuracy gain from it.

E.4 Reference Access and Guidance Restrictions

For SWE, *public-only guidance* (O1) and *reference-based guidance* (O2) share the same guidance role, structured output format, recipient, and output constraints. O1 sees only the public task and state;

O2 additionally permits isolated access to reference material. O1 thereby controls the planner's input structure and permitted amount of guidance, so the comparison tests the value of reference-informed assistance under the same interface. This matches the guidance channel and budget, rather than requiring identical generated messages or realized token counts. In both conditions, the planner must investigate the repository before producing a plan for the coder.

Reasoning implements the same access comparison inside the existing critic or judge rather than adding an oracle agent. That checker receives either the public task and candidates alone or the same inputs plus a private reference. The solver and debaters do not receive the reference; they receive only the feedback addressed to them. Candidate replay holds the initial proposal fixed where specified below. In either domain, a condition denotes a complete task run, not an agent within that run.

What the runtime enforces. Node-specific access rules isolate the raw reference from ordinary solving roles and remove private reference blocks from public logs. In SWE, the guidance parser retains structured facts such as implicated files, modules, tests, and failure symptoms; it replaces directive wording in the summary and failure-pattern fields with factual descriptions before injecting the result into the planner's context. In reasoning, schema and routing checks validate the checker decision, feedback is projected only to its intended recipient, and checks block recognizable private-source disclosures, direct answer instructions, and long private-answer copies before transmission. Where deterministic answer checks are available, an inconsistent accept/reject decision is also rejected before it can drive the workflow.

What prompts require. Semantic restrictions go beyond these mechanical checks. SWE prompts prohibit patch hunks, exact code snippets, and step-by-step edits, while requiring the planner to inspect public repository evidence. Reasoning prompts require an error diagnosis and a repair instruction supported by the visible task or candidate, not a replacement answer derived from the private reference. Pattern checks cannot certify that every paraphrase respects these requirements, so the disclosure audit evaluates the transmitted content separately. The following comparisons isolate reference access under a fixed guidance interface. Appendix E.7 additionally tests candidate corruption and guidance jointly, including the reference-dependent routing safeguards.

E.5 Model-Dependent Gains in Reasoning

Each of five models was assigned the same mixture of 100 DeepMath, 100 ProofWriter, and 40 Zebra tasks. On tasks with a binary outcome in both conditions, reference access improves success by 0.85–3.75 points (Table 3). Table E10 shows how those gains arise: four models record no public-only successes lost under reference access, while DeepSeek gains seven and loses one. The design replays the same initial candidate.

Table E10: **Paired reference-guidance comparisons underlying the reasoning results in Table 3.** O1-only and O2-only count discordant successes; each difference uses the displayed common task denominator. Intervals are task-paired bootstrap intervals and exact McNemar tests are unadjusted.

Model (thinking)	Pairs	O1-only	O2-only	O2−O1 (pp)	95% CI	McNemar p
GPT-5.4 (medium)	240	0	9	↑+3.75	[1.67, 6.25]	0.0039
Claude Sonnet 5 (adaptive; medium)	236	0	7	↑+2.97	[0.85, 5.08]	0.0156
DeepSeek-V4-Pro (enabled; medium)	236	1	7	↑+2.54	[0.42, 5.08]	0.0703
GLM-5.2 (enabled; max)	237	0	4	↑+1.69	[0.42, 3.80]	0.1250
Kimi K2.6 (enabled; provider default)	234	0	2	↑+0.85	[0.00, 2.14]	0.5000

ProofWriter contributes most consistently to these changes: its model-level differences range from $+1$ to $+7$ points, whereas Zebra is already at or near perfect success. The aggregate gains therefore reflect where a checker still has mistakes to resolve, not a uniform benefit from providing more information. In particular, the small Kimi difference includes zero in its paired interval; the positive direction across models should not be read as equal or uniformly decisive evidence for every model.

E.6 Software Outcomes and Net Provider Tokens

The SWE comparison evaluates O1 and O2 on the same fixed random sample of 80 canonical SWE-smith tasks from 55 repository families, with each of the five models. Both arms retain the configurations described above and the model-native thinking settings in Appendix K.2. Their reviewers share public repository and test access, with no runtime P2P/F2P summaries or access to evaluator internals or hidden test files. Pairing is within model and task. All 800 task–model–condition results have an authoritative PASS/FAIL verdict, using official evaluator records and evaluator-only recoveries where applicable; none is unavailable.

O1 yields 348 successes among 400 task–model runs (87.00%), and O2 yields 361 (90.25%). The net gain is 13 successes, or 3.25 percentage points: 23 paired failures become successes and 10 successes become failures (Table E11). The 400-run denominator is 80 tasks evaluated with five models, not 400 independent tasks. We report descriptive results for this fixed sample and do not attach an unadjusted significance test.

Table E11: **SWE Oracle outcomes on the same 80 tasks per model.** Transitions run from O1 to O2; each row contains 80 paired outcomes. No unavailable verdicts occur. The pooled row sums model–task outcomes.

Model	O1 P/F	O2 P/F	P→P	P→F	F→P	F→F
GPT-5.4	62/18	67/13	59	3	8	10
Claude Sonnet 5	71/9	71/9	68	3	3	6
DeepSeek-V4-Pro	71/9	72/8	69	2	3	6
GLM-5.2	74/6	77/3	74	0	3	3
Kimi K2.6	70/10	74/6	68	2	6	4
Total	**348/52**	**361/39**	**338**	**10**	**23**	**29**

Net provider-token accounting. Cost sums per-call provider `total_tokens` after removing explicit failed outputs and schema/API retries. Normal investigation, test failures, feedback, revision, and oracle calls remain included. Thus a failed test within productive execution is not discarded as an API failure. For model m, the saving is $1 - T_{m,\mathrm{O2}}/T_{m,\mathrm{O1}}$; the pooled saving applies the same ratio to summed provider totals, not to the mean of model percentages. Different providers' native token units are not a shared tokenizer or dollar cost.

Table E12: **Net provider tokens for the fixed-sample SWE comparison.** O1/O2 per-task costs divide by 80; per-PASS costs divide by the corresponding success count. All columns retain the net provider-token definition. Bold/underline identify lower/higher total token use within each model.

Model	O1 total	O2 total	Tokens/task O1 → O2	Tokens/PASS O1 → O2
GPT-5.4	30,943,655	**27,859,964**	386,795.7 → 348,249.6	499,091.2 → 415,820.4
Claude Sonnet 5	36,865,037	**30,248,532**	460,813.0 → 378,106.7	519,225.9 → 426,035.7
DeepSeek-V4-Pro	61,039,627	**55,216,549**	762,995.3 → 690,206.9	859,713.1 → 766,896.5
GLM-5.2	28,582,124	**27,383,321**	357,276.6 → 342,291.5	386,244.9 → 355,627.5
Kimi K2.6	70,541,370	**67,782,297**	881,767.1 → 847,278.7	1,007,733.9 → 915,977.0
Total / mean	227,971,813	**208,490,663**	569,929.5 → 521,226.7	655,091.4 → 577,536.5

The accounting retains 16,091 O1 calls and 15,199 O2 calls, while excluding 261 and 252 failed-output or retry calls, respectively. Recorded physical-run totals are 231,170,232 and 211,355,898 tokens; removing 3,198,419 and 2,865,235 tokens gives the net totals in Table E12. Requests without reported usage are not imputed, and previously replaced whole failed attempts are outside this net-cost scope.

Reference-based guidance reduces net tokens for every model, by 3.91–17.95%, while success improves for four models and is unchanged for Claude. Summed usage falls by 19,481,150 tokens, or 8.55%. This supports a benefit in recorded usage under the specified accounting; it does not by

itself establish a common monetary saving or identify which individual checking action caused the reduction.

E.7 Joint Candidate-Corruption and Guide Experiment

To test whether the collection operations remain useful together, we cross candidate corruption (the candidate-editing form of Perturb, denoted P) with reference-informed guidance (G). All four conditions retain the public critic/judge, feedback, revision, and fallback routes (Table E13). The experiment concerns collection-time interactions; its trajectories do not enter the dataset or downstream training comparisons.

Table E13: **The 2×2 collection design.** Every task has all four assigned conditions. Turning off Guide preserves public checking and repair.

Condition	Candidate input	Checking and routing
P_0G_0	Original candidates	Public-only checking
P_1G_0	Verified erroneous target candidate	Public-only checking
P_0G_1	Original candidates	Reference-informed checking
P_1G_1	Verified erroneous target candidate	Reference-informed checking

Sampling and input control. The experiment includes 60 canonical tasks each from DeepMath, ProofWriter, and Zebra: 180 tasks and 720 assigned branches. Its separate eligibility frame contains 976 verified candidate-error injection records (346, 428, and 202, respectively). Within each dataset, records are ordered by `SHA-256([20260919,dataset,item_id,pair_id])`, and the first 60 eligible tasks are selected without conditioning on final outcomes. The retained initial batch contains 20 tasks per dataset; an additional 40 per dataset extends it to the reported 60.

Original and corrupted candidates are frozen before continuation. At each fixed P, G_0 and G_1 reuse exactly the same candidates; all 360 paired first-round public-input hashes agree. The perturbations change a mathematical answer, a ProofWriter label, or a Zebra candidate table, with the target error verified before continuation. This study does not test weak-to-strong initialization or partial observation. Other candidates can still be incorrect in an original-candidate condition.

What the Guide switch changes. Guide jointly controls the checker's private-reference input, a reference-consistency veto before routing, and reference-dependent nondisclosure checks on public feedback. All three are disabled in G_0; the public checking, feedback, revision, and fallback mechanisms remain. Thus the comparison evaluates the configured reference-assisted checking pipeline, including its routing safeguards, rather than a prompt sentence alone. Independent final scoring uses the same authority across conditions: numeric official-compatible evaluation for DeepMath, and formal label/table evaluators for ProofWriter/Zebra.

Execution settings. All live checker and revision calls use DeepSeek-V4-Pro with medium reasoning effort through chat, without streaming. DeepMath follows MALT; ProofWriter and Zebra follow Sparse Debate. Checker calls use temperature 0.1, top-p 1.0, and at most 12,000 output tokens; revision calls use temperature 0.2, top-p 1.0, and at most 20,000 output tokens. The request timeout is 600 seconds. The role-output retry cap is 3, provider retry maximum attempts is 1, and the outer retry policy permits at most 2 attempts. The extension uses 24 concurrent CPU/API branches and no GPU. The deterministic selection and bootstrap seed is 20260919.

Outcomes and availability. Table E14 reports correct usable output, n_{pass}/N, over all assigned tasks, together with correctness among evaluable outputs, $n_{\text{pass}}/(n_{\text{pass}} + n_{\text{fail}})$. n_{pass} and n_{fail} count PASS and FAIL outcomes; P denotes only the Perturb switch. Across the 720 branches, there are 712 PASS, 5 FAIL, and 3 unavailable outputs. All three unavailable outputs are ProofWriter `formal_safe_stopped` results; they are retained as unavailable, not relabeled as wrong answers. The four conditions have PASS/FAIL outcomes jointly on 177 tasks (60 DeepMath, 57 ProofWriter, and 60 Zebra), used for the complete-case sensitivity analysis below.

Table E14: **Joint-experiment outcomes.** N is the assigned-task denominator in each arm; U denotes unavailable output. n_{pass}/N measures correct usable output, while $n_{\text{pass}}/(n_{\text{pass}} + n_{\text{fail}})$ conditions on an evaluable result. Bold/underline compare G_0/G_1 at fixed dataset and P; ties share rank.

Dataset	Condition	N	PASS/FAIL/U	n_{pass}/N (%) ↑	$\frac{n_{\text{pass}}}{n_{\text{pass}}+n_{\text{fail}}}$ (%) ↑
DeepMath	P_0G_0	60	59/1/0	<u>98.33</u>	<u>98.33</u>
	P_1G_0	60	58/2/0	<u>96.67</u>	<u>96.67</u>
	P_0G_1	60	60/0/0	**100.00**	**100.00**
	P_1G_1	60	60/0/0	**100.00**	**100.00**
ProofWriter	P_0G_0	60	58/1/1	<u>96.67</u>	<u>98.31</u>
	P_1G_0	60	58/1/1	<u>96.67</u>	<u>98.31</u>
	P_0G_1	60	60/0/0	**100.00**	**100.00**
	P_1G_1	60	59/0/1	**98.33**	**100.00**
Zebra	P_0G_0	60	60/0/0	**100.00**	**100.00**
	P_1G_0	60	60/0/0	**100.00**	**100.00**
	P_0G_1	60	60/0/0	**100.00**	**100.00**
	P_1G_1	60	60/0/0	**100.00**	**100.00**
Pooled	P_0G_0	180	177/2/1	<u>98.33</u>	<u>98.88</u>
	P_1G_0	180	176/3/1	<u>97.78</u>	<u>98.32</u>
	P_0G_1	180	180/0/0	**100.00**	**100.00**
	P_1G_1	180	179/0/1	**99.44**	**100.00**

Corrective interaction and final-source provenance. With Guide fixed on, candidate corruption increases runs with an executed revision from 2/180 to 179/180 and final PASS outputs sourced from revisions from 2 to 115 (Table E15). This exposes both correction opportunities and decisions that retain a correction in the final output. Under corruption, Guide increases known-error detection from 179/180 to 180/180 and runs containing a corrected, initially wrong candidate from 177/180 to 179/180. Near-ceiling task success in Zebra therefore coexists with substantially different supervision.

Table E15: **Detection, revision, and final provenance.** Each indicator counts a run at most once ($N = 60$ per dataset arm; pooled $N = 180$). Detection concerns the preverified corrupted target in P_1. *Corrected* requires an initially wrong candidate from any role to have a subsequent evaluator-accepted revision, regardless of final selection. The last two columns partition final PASS outputs by source.

Dataset	Condition	Known error detected	Revision executed	Corrected	PASS from revision	PASS from initial
DeepMath	P_0G_0	–	5	0	4	55
	P_1G_0	59	59	59	58	0
	P_0G_1	–	0	0	0	60
	P_1G_1	60	60	60	60	0
ProofWriter	P_0G_0	–	4	1	2	56
	P_1G_0	60	59	58	18	40
	P_0G_1	–	2	2	2	58
	P_1G_1	60	59	59	19	40
Zebra	P_0G_0	–	0	0	0	60
	P_1G_0	60	60	60	31	29
	P_0G_1	–	0	0	0	60
	P_1G_1	60	60	60	36	24
Pooled	P_0G_0	–	9	1	6	171
	P_1G_0	179	178	177	107	69
	P_0G_1	–	2	2	2	178
	P_1G_1	180	179	179	115	64

Candidate correctness is assessed offline with the frozen task evaluator in every condition. Detection records a first-round repair request in DeepMath or an assessment identifying the designated target as incorrect in ProofWriter/Zebra. Thus G_0 retains independent outcome assessment while removing online reference assistance.

Guide also reduces unnecessary repair of initially correct candidates. In P_0G_0, eight runs include criticism of an initially correct candidate (five DeepMath and three ProofWriter), and two include a subsequent incorrect revision of such a candidate (one in each dataset). Both counts are zero in P_0G_1 and the two corrupted-candidate conditions. This regression measure starts from each role's initial candidate. Separately, one DeepMath P_1G_0 run produces an evaluator-accepted revision and then finalizes a later rejected revision, explaining its 59 *Corrected* runs versus 58 PASS outputs from revisions. Recording intermediate candidates and final sources makes both correction and its retention available for control supervision.

Continuation costs. Table E16 includes recorded online continuation calls and retries. It measures the checking and revision phase after the initial candidates are frozen, not the cost of generating those candidates. Token savings compare summed recorded tokens at fixed P. Guide reduces continuation tokens by 20.31% with original candidates and 7.75% with corrupted candidates. The corresponding dataset-level reductions are 32.66%/12.25% in DeepMath, 20.02%/7.35% in ProofWriter, and 7.73%/5.63% in Zebra (original/corrupted). These are token measurements, not estimated monetary costs.

Table E16: **Recorded continuation costs in the factorial study, including retries.** Per-assigned-branch cost uses N; per-available-trajectory cost uses $n_{\text{pass}}+n_{\text{fail}}$. Bold/underline mark lower/higher token cost for G_0/G_1 at fixed dataset and P.

Dataset	Condition	Calls	Tokens ↓	Tokens/N ↓	Tokens/ evaluable ↓
DeepMath	P_0G_0	76	230,535	3,842.2	3,842.2
	P_1G_0	184	470,120	7,835.3	7,835.3
	P_0G_1	66	**155,247**	**2,587.4**	**2,587.4**
	P_1G_1	182	**412,519**	**6,875.3**	**6,875.3**
ProofWriter	P_0G_0	68	193,592	3,226.5	3,281.2
	P_1G_0	196	530,689	8,844.8	8,994.7
	P_0G_1	65	**154,839**	**2,580.7**	**2,580.7**
	P_1G_1	194	**491,667**	**8,194.5**	**8,333.3**
Zebra	P_0G_0	60	221,810	3,696.8	3,696.8
	P_1G_0	187	903,541	15,059.0	15,059.0
	P_0G_1	60	**204,673**	**3,411.2**	**3,411.2**
	P_1G_1	184	**852,651**	**14,210.9**	**14,210.9**
Pooled	P_0G_0	204	645,937	3,588.5	3,608.6
	P_1G_0	567	1,904,350	10,579.7	10,638.8
	P_0G_1	191	**514,759**	**2,859.8**	**2,859.8**
	P_1G_1	560	**1,756,837**	**9,760.2**	**9,814.7**

Prespecified effects and uncertainty. Let $\bar{Y}_{pg}$ be the mean indicator of a correct usable output in condition P_pG_g. The primary effect is $\Delta_{G|P=1} = \bar{Y}_{11} - \bar{Y}_{10}$; the interaction is

$$\Delta_{\text{int}} = (\bar{Y}_{11} - \bar{Y}_{10}) - (\bar{Y}_{01} - \bar{Y}_{00}).$$

The full assigned-task analysis treats U as absence of a usable correct output, without asserting that its answer is wrong. The sensitivity analysis removes an entire task if any arm is U. Confidence intervals use 10,000 dataset-stratified task-cluster bootstrap resamples, retaining all four conditions of a task together.

Under candidate corruption, Guide raises correct usable output from 176/180 to 179/180 ($+1.67$ points) while reducing continuation tokens by 7.75%. The pooled assigned-task interaction is 0.00 points with a 95% interval of $[-2.22, +2.22]$. The results support complementary roles in eliciting

Table E17: **Guide effects under candidate corruption and difference-in-differences interactions, in percentage points with 95% task-cluster bootstrap intervals.** *Complete* requires PASS/FAIL in all four arms.

Dataset	Analysis	Tasks	$\Delta_{G\|P=1}$ [95% CI]	Δ_{int} [95% CI]
DeepMath	Assigned	60	$+3.33$ [0.00, 8.33]	$+1.67$ [0.00, 5.00]
	Complete	60	$+3.33$ [0.00, 8.33]	$+1.67$ [0.00, 5.00]
ProofWriter	Assigned	60	$+1.67$ [0.00, 5.00]	-1.67 [-8.33, 3.33]
	Complete	57	0.00 [0.00, 0.00]	0.00 [0.00, 0.00]
Zebra	Assigned	60	0.00 [0.00, 0.00]	0.00 [0.00, 0.00]
	Complete	60	0.00 [0.00, 0.00]	0.00 [0.00, 0.00]
Pooled	Assigned	180	$+1.67$ [0.00, 3.89]	0.00 [-2.22, 2.22]
	Complete	177	$+1.13$ [0.00, 2.82]	$+0.56$ [0.00, 1.69]

correction and controlling continuation cost, but do not establish a superadditive accuracy gain. Near-ceiling $[0, 0]$ bootstrap intervals do not prove that the population effect is exactly zero. Generalization beyond this eligible-task frame, candidate corruption, and the single DeepSeek checker remains untested.

F System Evaluation

F.1 Comparators and Shared Budgets

All workflow comparisons use unperturbed task inputs and model assignments without private references or reference-based hints. The task-solving role graph, routing, ordinary feedback, and repair are preserved. Thus the systems operate with evidence available in ordinary deployment.

Single-MA is a *single-agent* control with task-level model-call, tool-call, and token budgets matched to those available to the complete CoEvoke workflow. The suffix denotes budget alignment with the multi-agent system, not multiple agents or an ensemble of independently scored candidates. With this control, any difference in success concerns how the systems use the same resource allowance. Table F4 additionally retains single-attempt diagnostics for Gemma and Ministral, labeled Single-1×; these use a different budget and are not the primary comparator.

For SWE, both systems share caps of 468 model calls, 450 tool rounds, and 11.3M tokens; calls by all CoEvoke roles count against the same task total. Reasoning uses the limits in Table F6. Comparisons hold the model, task set, and evaluator fixed, although early stopping and the response-level token check described below can produce different realized usage. Success rates divide verified successes by all prespecified tasks, while paired transition tests require a binary outcome from both systems. We show the two denominators separately where they differ.

The internal DeepMath, ProofWriter, and Zebra evaluations use 100-task samples held out under CoEvoke's own task-level split. Here *training* and *test* describe our partition, not a blanket claim about the original datasets' published split labels. Because these samples may differ from their source distributions, they support task-specific analysis alongside the larger external evaluations. SWE-bench Verified, the fixed MathArena task collection, and ACPBench-Hard are excluded from CoEvoke construction and provide the primary system evidence; the SWE-smith guidance comparison is separate (Appendix E.6). This separation avoids direct construction–evaluation overlap without claiming that public evaluation tasks were absent from foundation-model pretraining.

F.2 Model-Level Results and Paired Comparisons

CoEvoke improves success in eight of the nine model–benchmark comparisons in Table F1. All three models improve on repository repair and ACPBench-Hard, whereas the MathArena result depends on the model. This pattern supports task-specific use of collaboration: additional roles can help organize long execution or compare competing candidates, but they do not improve every solver on every reasoning task.

Table F1: **External system results corresponding to Figure 4a.** The control is one single agent with the same task-level model-call, tool-call, and token budgets as CoEvoke. Rates use the fixed task sets shown below. Bold/underline mark higher/lower success within each row, not statistical significance. Differences are computed from unrounded proportions.

Benchmark	Model (thinking)	N	Control (%)	CoEvoke (%)	Δ (pp)
SWE-bench Verified	Qwen3.5-9B (off)	500	42.40	**53.20**	↑+10.80
	Gemma-4-12B-IT (off)	500	11.80	**17.80**	↑+6.00
	Ministral-3-8B-Instruct (off)	500	7.40	**12.60**	↑+5.20
MathArena	Qwen3.5-9B (off)	144	20.83	**27.08**	↑+6.25
	Gemma-4-12B-IT (off)	144	26.39	**29.86**	↑+3.47
	Ministral-3-8B-Instruct (off)	144	**9.72**	6.94	↓−2.78
ACPBench-Hard	Qwen3.5-9B (off)	1,040	18.85	**36.15**	↑+17.31
	Gemma-4-12B-IT (off)	1,040	35.10	**47.69**	↑+12.60
	Ministral-3-8B-Instruct (off)	1,040	11.63	**14.81**	↑+3.17

Table F2: **Internal random-sample results and paired analyses.** Each dataset contains 100 tasks. Bold/underline compare Single-MA with CoEvoke within a row; the post hoc candidate union is not a deployable comparator.

Dataset	Model (thinking)	Single-MA	CoEvoke	Pairs	Δ (pp)	95% CI	Candidate union
DeepMath	Qwen3.5-9B (off)	**87/100**	86/100	100	↓−1.00	[−9.00, 7.00]	95/100
ProofWriter	Qwen3.5-9B (off)	**84/100**	83/100	100	↓−1.00	[−8.00, 5.00]	89/100
Zebra	Qwen3.5-9B (off)	52/100	**60/100**	100	↑+8.00	[−3.00, 19.00]	72/100

On the internal samples, Qwen already solves 87 DeepMath and 84 ProofWriter tasks without collaboration. CoEvoke changes those totals by only one task, while increasing Zebra success from 52 to 60. The intervals include zero, so these small samples chiefly help locate where additional checking and candidate comparison have room to help. They complement, rather than replace, the full external evaluations. The separate SWE-smith guidance comparison is reported in Appendix E.6.

Table F3: **Qwen3.5-9B paired results on complete external benchmarks.** Success counts retain prespecified denominators; bold/underline compare control and CoEvoke within a row. Paired differences and candidate unions use only tasks with PASS/FAIL in both conditions. Table F5 reports the unadjusted exact McNemar tests on evaluable pairs.

Benchmark	Model (thinking)	Control	CoEvoke	Pairs	Δ (pp)	95% CI	Candidate union
SWE-bench Verified	Qwen3.5-9B (off)	212/500	**266/500**	498	↑+10.84	[8.27, 15.96]	296/498
MathArena	Qwen3.5-9B (off)	30/144	**39/144**	144	↑+6.25	[−0.69, 13.19]	49/144
ACPBench-Hard	Qwen3.5-9B (off)	196/1,040	**376/1,040**	1,040	↑+17.31	[14.71, 20.00]	401/1,040

The post hoc union counts a task as solved if either system succeeds; it is not an executable comparator and does not redefine Single-MA as a candidate ensemble. Such unions exceed either system alone on every reported Qwen task set. For example, CoEvoke solves 376 ACPBench-Hard tasks, but the control contributes 25 additional successes, bringing the union to 401. These cases locate opportunities for improved control despite CoEvoke's higher overall success rate. On SWE-bench Verified, the paired difference is 10.84 points over 498 evaluable pairs, versus 10.80 points over all 500 tasks in Table F1; the denominator explains the difference between the estimates.

F.3 Cross-Model Reasoning Results

Table F4 reports diagnostics for two models with different native tool protocols. Counts retain fixed task denominators. This comparison examines model–workflow interactions rather than defining a cross-model leaderboard.

Table F4: **Gemma and Ministral system comparisons across reasoning tasks.** Entries and differences are success counts. Bold/underline compare only shared-budget Single-MA and CoEvoke within a row, with ties sharing rank; Single-1× uses a different budget.

Model (thinking)	Dataset (N)	Single-1×	Single-MA	CoEvoke	Δ
Gemma-4-12B-IT (off)	DeepMath (100)	97	94	**95**	↑+1
	ProofWriter (100)	76	80	**83**	↑+3
	Zebra (100)	62	**63**	**63**	0
	MathArena (144)	39	38	**43**	↑+5
	ACPBench-Hard (1,040)	375	365	**496**	↑+131
Ministral-3-8B-Instruct (off)	DeepMath (100)	67	**47**	40	↓−7
	ProofWriter (100)	56	**37**	36	↓−1
	Zebra (100)	10	2	**4**	↑+2
	MathArena (144)	9	**14**	10	↓−4
	ACPBench-Hard (1,040)	145	121	**154**	↑+33

Under the matched resource allowance, CoEvoke raises ACPBench-Hard success from 365 to 496 for Gemma and from 121 to 154 for Ministral. The same organization is less helpful on DeepMath: Gemma changes from 94 to 95, whereas Ministral falls from 47 to 40. Together with the single-attempt diagnostics, these results show that neither a larger allowance nor additional roles guarantees a better solution. Workflow effectiveness must be assessed for the intended model and task, rather than inferred from resource use or a ranking obtained elsewhere.

F.4 Paired Statistics

For two outcomes on the same task, the difference is

$$\Delta = \frac{n_{F\to P} - n_{P\to F}}{N_{\text{paired}}} \times 100. \tag{5}$$

Reasoning comparisons use task-paired bootstrap resampling. SWE uses repository-family cluster bootstrap over 12 families because tasks from the same repository can be correlated, alongside exact McNemar tests. Candidate-union denominators include only pairs with PASS/FAIL in both conditions. Differences in rates are absolute percentage points, not relative error reductions.

Table F5 resolves each evaluable task pair into four outcomes: both systems succeed, only the control succeeds, only CoEvoke succeeds, or neither succeeds. The difference between the two middle columns determines the paired effect. The control is Single-MA for reasoning and the shared-budget single agent for SWE. Full-benchmark success rates remain those in Table F1; the pair counts below apply only to transition statistics.

The paired counts show the clearest asymmetry on SWE-bench Verified and ACPBench-Hard: CoEvoke-only successes outnumber control-only successes by 84 to 30 and 205 to 25, respectively. DeepMath and ProofWriter are almost balanced. Thus the strongest gains are supported by changes on the same tasks, not merely by comparing aggregate scores from different samples.

F.5 Reasoning Budgets and the Single-Agent Procedure

All three evaluated model families use the same nominal limits for each reasoning task set. These limits are shared across every role in a multi-agent run and are also assigned to its Single-MA control. Tokens are counted using each model's native accounting; equality of the thresholds therefore describes the experimental allowance, not an identical quantity of text across different tokenizers.

Call allowance is reserved before issuing a request, whereas the runtime checks cumulative token usage after receiving the response. A boundary response can therefore exceed the token threshold;

Table F5: **Task-level outcome transitions for Qwen3.5-9B.** Each row partitions the evaluable pairs; McNemar tests compare the control-only and CoEvoke-only counts. Tests are unadjusted.

Dataset	Evaluable pairs	Both succeed	Control only	CoEvoke only	Neither succeeds	McNemar p
SWE-bench Verified	498	182	30	84	202	4.25×10^{-7}
DeepMath	100	78	9	8	5	1.0000
ProofWriter	100	78	6	5	11	1.0000
Zebra	100	40	12	20	28	0.2153
MathArena	144	20	10	19	95	0.1360
ACPBench-Hard	1,040	171	25	205	639	2.43×10^{-36}

Table F6: **Shared task-level limits for reasoning evaluation.** Model-call counts include schema-repair requests. Token thresholds accumulate input and output usage, including repeated inputs. Single-MA invocation limits include the initial attempt; CoEvoke's two repair rounds are additional to the initial proposal stage.

Task set	Model calls	Token threshold	CoEvoke repair limit	Single-MA invocations
DeepMath (100)	7	49,000	2 Solver revisions	7
ProofWriter (100)	8	19,000	2 Judge repair routes	8
Zebra (100)	11	87,000	2 Judge repair routes	11
MathArena (144)	9	80,000	2 Solver revisions	9
ACPBench-Hard (1,040)	9	60,000	2 Judge repair routes	9

the resulting exhaustion state prevents further model requests. Each task also has a 7,200-second time allowance, with a 7,800-second outer supervisor in the common-budget setup. The first applicable resource, protocol, or repair limit constrains continuation. In solver–critic workflows, each of the two permitted solver revisions is followed by checking. In Sparse Debate, the two repair routes are global to the task and each names exactly one debater. Approval can end either workflow earlier, and schema retries count toward model calls without constituting additional scientific repair rounds.

Single-MA uses repeated invocations of the same solver policy with current-candidate carryover. The first invocation receives the original public task and an empty candidate. After a valid output, the runtime retains the evaluator-relevant answer or table; the next invocation receives that candidate together with the original task, while previous reasoning and retained conversation context are cleared. Thus the same prompt, model, output protocol, and decoding settings are reused without access to a critic/judge message, evaluator verdict, or reference answer.

In a normal run, the policy makes at least two invocations and then stops when two consecutive registered candidate hashes agree, or when an invocation or shared resource limit is reached. Hash agreement tests the candidate representation, not mathematical equivalence or correctness. The latest valid registered answer is submitted to the evaluator, with no vote or retrospective selection among earlier answers. Protocol failures and timeouts can prevent the normal minimum from being reached; a run without a valid final answer remains unavailable rather than being replaced with a successful candidate. This specifies how Single-MA can use its additional allowance while keeping its decisions separate from task evaluation.

G Control Learning

G.1 Extracting Decision Records from Full Trajectories

Decision records are organized by decision interface. The critic in CoEvoke's mathematical workflow receives the public task, current candidate, and checkable constraints, and learns approval or structured feedback. The judge receives independent candidates and key evidence, and learns selection and repair routing. The SWE verifier receives the issue and candidate artifact, and learns acceptance

decisions. Whereas MALT post-trains generator, verifier, and refiner policies (Motwani et al., 2025), these experiments update only the designated controller and keep candidate-generating actors unchanged. The full-corpus judge/critic and SWE verifier views derive from retained CoEvoke trajectories (Appendix C.4); the matched Natural-SFT control instead uses independently collected clean trajectories, while its Perturb-SFT counterpart uses released trajectories (Appendix G.4).

Natural and perturbed variants of the same source task are assigned to the same data split. Reasoning controllers use the experiment-specific training sources described below and are evaluated on external MathArena and ACPBench-Hard tasks. The SWE verifier is evaluated on SWE-bench Verified, whereas CoEvoke software trajectories originate from SWE-smith. Keeping task variants together prevents one variant of a training task from appearing in evaluation.

For a task x, let $\mathcal{D}_x$ contain its decision records. Record k pairs a public trajectory prefix v_k with a target token sequence a_k. Equation 4 expands to

$$\mathcal{L}_{\text{ctrl}} = -\sum_{x} \sum_{k \in \mathcal{D}_x} w_k \sum_{j \in \mathcal{J}_k} \alpha_{k,j} \log p_\theta(a_{k,j} \mid v_k, a_{k,<j}), \tag{6}$$

where $\mathcal{J}_k$ contains supervised assistant positions, $\sum_{k \in \mathcal{D}_x} w_k = 1$, and $\sum_{j \in \mathcal{J}_k} \alpha_{k,j} = 1$. Thus each task contributes the same total weight even when it supplies several decisions; context tokens receive no loss. The token weights retain the controller-specific target construction, including the verifier's field allocation described below. Inputs contain only the task, candidates, and public evidence available before the decision, so training does not expose subsequent repairs or evaluator outcomes as context.

G.2 Controller Training Configuration

The full-corpus judge comparison in Table G1 uses 2,823 decision records from 1,514 training tasks and 31 records from 16 development tasks. The critic uses 1,531 records from 817 training tasks and 14 records from eight development tasks. Multiple decisions from the same task remain grouped, so record counts differ from task counts. The matched-data comparison in Appendix G.4 uses a separate judge training set.

Both controllers adapt Qwen3.5-9B with LoRA (Hu et al., 2022), using rank 8, scaling factor 16, dropout 0.05, and no bias updates. The 80 adapted text-attention projections comprise `q_proj`, `k_proj`, `v_proj`, `o_proj`, `in_proj_qkv`, and `out_proj`; visual and feed-forward modules are not adapted. AdamW uses learning rate 10^{-5}, betas $(0.9, 0.95)$, epsilon 10^{-8}, weight decay 0.01, and gradient clipping at 1.0. A cosine schedule follows 3% warmup. Training uses bfloat16, gradient checkpointing, and a maximum length of 12,288 tokens, without packing or target truncation.

Each optimizer step accumulates four canonical-task groups with task-normalized weighting, rather than treating each decision as an equally weighted independent task. We evaluate the final checkpoint at the end of the prescribed training schedule: one epoch (379 steps) for the full-corpus judge and three epochs (615 steps) for the critic. Decoding settings remain fixed across the compared conditions.

Why supervised fine-tuning? The learning study asks what can be learned from decision records extracted from CoEvoke trajectories. All trained and untrained systems are evaluated on unperturbed inputs without private-reference guidance, retaining the same task-solving topology, ordinary feedback, and repair. Supervised fine-tuning (SFT) keeps the training experience within each fixed training corpus: the learner predicts recorded actions from their preceding public context. Online reinforcement learning would answer a broader question. For example, GRPO samples groups of responses from the policy and computes relative advantages from their rewards (Shao et al., 2024); training would then depend on newly generated experience as well as the original dataset. We therefore use SFT to study the dataset's contribution without adding reward-guided exploration. This choice concerns experimental attribution, not a limitation on combining CoEvoke with reinforcement learning.

For the same reason, training and evaluation use the same public-output interface, with explicit thinking disabled for all compared local models. The targets supervise public decisions and actions, not a separate private reasoning channel. Keeping that channel configuration fixed prevents an unsupervised output mode from changing the comparison; public derivations and evidence-based reasoning are still allowed (Appendix K).

G.3 Controller Results

Table G1: **Controller comparisons with frozen task-solving actors.** All models are Qwen3.5-9B with thinking disabled. Success rates use complete evaluation sets of 1,040 ACPBench-Hard tasks, 144 MathArena tasks, and 500 SWE-bench Verified tasks. Each trained Verifier is compared with an untrained reference receiving the same input format. Bold and underlining indicate higher and lower point estimates within each comparison, not statistical significance.

Module	Model (thinking)	Benchmark	Untrained (%)	SFT (%)	Δ (pp)
Judge	Qwen3.5-9B (off)	ACPBench-Hard	<u>36.15</u>	**42.21**	↑+6.06
Critic	Qwen3.5-9B (off)	MathArena	<u>27.08</u>	**29.17**	↑+2.08
Trajectory-based Verifier	Qwen3.5-9B (off)	SWE-bench Verified	<u>23.40</u>	**32.80**	↑+9.40
Patch-based Verifier	Qwen3.5-9B (off)	SWE-bench Verified	<u>24.80</u>	**32.40**	↑+7.60

The judge and verifier comparisons provide the main training evidence in Table G1. Training the judge increases ACPBench-Hard success from 376 to 439 of 1,040 tasks (36.15% to 42.21%). On SWE-bench Verified, the Trajectory-based verifier improves from 117 to 164 of 500 tasks (23.40% to 32.80%), while the Patch-based verifier improves from 124 to 162 (24.80% to 32.40%). Within each comparison, the solving actors, evidence interface, and decoding settings remain fixed, so the observed gains concern the trained decision module rather than actor fine-tuning.

The critic supplies a smaller complementary observation: MathArena success changes from 39 to 42 of 144 tasks (27.08% to 29.17%). This three-task difference does not establish a stable improvement in mathematical reasoning. We retain it alongside the other point estimates while basing the main learning conclusion on the judge and verifier results. Appendix G.5 explains why the SWE input-specific comparisons have different untrained references from the full tool-using workflow.

G.4 Matched-Data Comparison of Natural-SFT and Perturb-SFT

We compare Natural-SFT with Perturb-SFT (the perturbation-enriched training condition) on exactly the same 884 source-task IDs, with 891 decision records and 379 optimizer updates per condition. Natural-SFT uses clean trajectories independently collected after selecting tasks from the existing judge-training task frame; Perturb-SFT uses the corresponding released trajectories. Both collection judges can access private references: natural means no Perturb, not no Guide. Reference text is excluded from training inputs. Both train only the Qwen3.5-9B Sparse Judge using the LoRA configuration, task-normalized objective, and optimizer described above. The other generation roles remain unmodified Qwen3.5-9B models. This study is separate from the full-corpus judge comparison in Table G1.

Table G2: **Sources of the matched training corpora.** Tasks and decision records are matched. The last column counts source-run groups in the proposed release manifest, not whether their canonical task IDs occur there; the task IDs are shared across conditions. Natural-SFT is separate from the clean release counts in Table C4.

Training group	Source trajectories	Tasks / records	Released source-run groups
Natural-SFT	Independently collected clean runs on tasks selected from the existing judge-training frame	884 / 891	0 / 884
Perturb-SFT	Released runs: 561 candidate corruption, 282 partial observation, 36 weak-to-strong, and 5 clean cases	884 / 891	884 / 884

Matched training data. Perturb-SFT fixes Natural-SFT's task IDs and number of records per task, selecting supervision deterministically from actual judge events. For the 879 paired tasks with distinct-source perturbed runs, checker prompts, workflow hashes, and actual model-request

parameters match. This configuration check covers those 879 pairs, not all 884 tasks. Table G2 separates trajectory provenance from shared task identity.

The shared source tasks comprise 656 ProofWriter and 228 Zebra tasks. The Perturb-SFT set contains 561 tasks with verified candidate corruption, 282 with verified partial observation, 36 with weak-to-strong recovery, and five retained clean control tasks. Those five clean source-run groups contribute eight decision records from released trajectories because their Perturb counterparts were unavailable; incomplete Perturb trajectories are excluded. They are not drawn from the independently collected Natural-SFT runs. Thus 879 of the 884 tasks supply perturbation-derived supervision while all source-task IDs remain matched across conditions.

Training uses four complete canonical-task groups per update, totaling 1,516 task presentations over 379 updates (approximately 1.715 passes through the task set). This is a fixed-update schedule, not an integer-epoch schedule. For Perturb-SFT, training, data, and sampler seeds are 42; a single corpus traversal contains 1,296,544 sequence tokens and 128,672 target tokens. These are corpus totals, not cumulative training consumption, and matching record counts does not imply equal token exposure. Evaluation uses the prespecified final step-379 checkpoint.

Evaluation. Both judges use the same Sparse Debate workflow, decoding configuration, and 1,040 external ACPBench-Hard tasks. The evaluation seed is 22, with limits of nine model calls and 60,000 tokens per task. Evaluation preserves the task-solving topology, uses unperturbed inputs, and supplies no private references or reference-based hints. The comparison therefore tests decisions learned during training under ordinary task-solving conditions.

Table G3: **Natural-SFT versus Perturb-SFT on the same 884 source tasks.** Both use 891 records, 379 updates, frozen solving actors, and the same evaluation protocol. Success divides solved tasks by all 1,040 assigned ACPBench-Hard tasks, including unavailable outcomes in the denominator. Bold/underline mark best/second success within this matched comparison.

Training data	Records	Source tasks	Updates	Solved	Success (%) ↑
Natural-SFT	891	884	379	<u>368</u>	<u>35.38</u>
Perturb-SFT	891	884	379	**438**	**42.12**

The natural corpus contains 884 acceptance targets and seven repair-routing targets; the matched Perturb-SFT corpus contains 451 acceptance targets and 440 repair-routing targets (Figure S8). Repair-routing decisions therefore account for 0.79% versus 49.38% of the same 891-record budget. The fixed task and record budgets thus shift from almost entirely approval targets to substantial repair-routing supervision, directly realizing the intended effect of collection design. These are decision counts, not unique tasks or first-round outcomes: acceptance can follow an earlier repair. This change in supervision composition is distinct from the downstream success gain and is not established as its sole cause.

Perturb-SFT produces 438 PASS, 588 FAIL, and 14 unavailable outcomes. Its full-set success is 438/1,040 (42.12%); the 14 unavailable outcomes remain distinct from FAIL and stay in that denominator. On the 1,026 evaluable tasks, success is 42.69%. All unavailable outcomes have the finalizer reason `prediction_or_gold_missing`, which identifies missing scoring inputs without attributing a specific underlying failure.

Compared with Natural-SFT's 368/1,040 successes (35.38%), Perturb-SFT solves 70 additional tasks, a gain of 6.73 percentage points computed from the unrounded proportions. Matching source-task identities, record counts, and optimizer updates removes the broader task coverage and larger corpus as explanations for this improvement. The result supports the value of perturbation-enriched supervision for collaborative decision making with frozen solvers; it reflects one training seed rather than an estimate of variation across retraining.

G.5 SWE VERIFIER INPUTS AND EVALUATION

The software studies distinguish a tool-using reviewer from two supplied-evidence verifier interfaces. The full workflow in Table F1 has a repository-visible reviewer that can inspect files and run public tests. The learning study instead asks whether a controller can learn an acceptance decision from

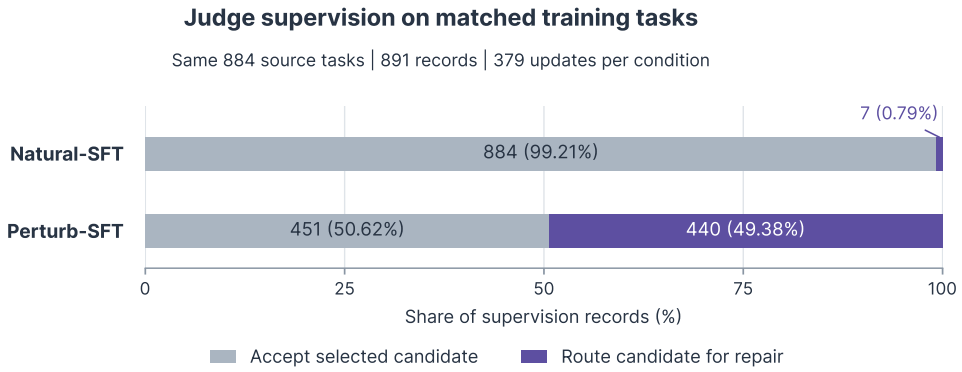

Figure S8: Matched Judge supervision at a fixed corpus size. Each condition uses the same 884 source tasks, 891 decision records, and 379 updates. Gray and purple partition recorded actions into acceptance and repair-routing targets; labels give counts and shares. Acceptance can follow an earlier repair: these are record-level supervision proportions, not first-round approval rates or downstream task accuracy.

supplied evidence: its verifier has no tools and judges either an issue–patch pair or that pair with selected prior public execution. Table G5 places these interfaces side by side. The distinction concerns evidence access and the decision interface; trained and untrained models within each verifier comparison use the same configuration.

The verifier gains demonstrate online control-decision learning within each supplied-evidence interface. The full recorded trajectory is the source from which decision-specific supervision is constructed; it need not be supplied in full to every controller. Both views derive their candidate states, public prefixes, and evaluator-linked labels from those records. The 32.40% and 32.80% point estimates do not establish a stable advantage for longer inputs. Nor do they show that a trained verifier improves on the tool-using reviewer workflow, whose 53.20% success remains a separate interface reference. This separation keeps the learning result focused on whether the released records support acceptance decisions, without treating a restricted checker as a performance-equivalent replacement for an agent that can gather further evidence.

The Patch-based verifier judges an issue and proposed patch; the Trajectory-based verifier additionally receives the coder's public output and selected tool interactions leading to that patch. Both use Qwen3.5-9B adapters with LoRA rank 8, scaling factor 16, dropout 0.05, and the same six attention-projection classes described above. Each is trained for one epoch over 2,010 records from 1,318 tasks, with eight development tasks, learning rate 10^{-5}, and four canonical-task groups per update. Maximum training lengths are 16,384 tokens for patch inputs and 32,768 for trajectory inputs. Both evaluations use the final checkpoint after completing that epoch (330 updates). The task-solving actor remains fixed.

Constructing verifier inputs. The Patch-based Verifier receives the public issue and the complete current patch. The Trajectory-based Verifier adds the current Coder response and selected public tool interactions preceding the review decision. These are not unrestricted dialogue histories: each tool call remains paired with its observation, and the retained pairs are delivered in execution order. Algorithm 3 constructs both inputs from the same candidate state.

Pairing, compression, and ordering. Calls and observations are matched in occurrence order within each role and tool. Only Planner and Coder tool interactions are admitted; private evaluation calls, unpaired events, and projected records containing blocked reference or evaluator markers are excluded. Tool observations retain execution status, exit code, and compressed output. Test commands are identified from executable commands rather than from a filename containing "test." For patch-related file reads, the projection keeps numbered lines within 20 lines of a changed range and diagnostic lines. Other long observations preserve category-specific beginning, ending, and

Algorithm 3 Constructing patch-based and trajectory-based verifier contexts

Require: Public issue q, complete candidate patch Δ, current public Coder response u, pre-review event prefix $\mathcal{E}_{\leq t}$, input type k, character budget $B = 75{,}000$
Ensure: A public evidence record P, or an explicit over-budget error

$P \leftarrow$ record containing q, Δ, and their source identifiers
if k is trajectory-based **then**
    Add the bounded head–tail summary of u to P
    $(F, H) \leftarrow$ changed files and line ranges in Δ
    $S \leftarrow$ completed Planner/Coder pairs in $\mathcal{E}_{\leq t}$, excluding private evaluation
    for each pair $e \in S$ **do**
        Classify e and mark its relevance to F
        Compact its arguments and observations using its category and H
    end for
    Remove projected pairs containing blocked private references
    Add chronologically ordered S and its retained/omitted counts to P
    while the compact JSON representation of P exceeds B **do**
        raise over-budget error if S is empty
        Remove the lowest-priority pair; break ties by non-patch relevance, then age
        Restore execution order and update counts in P
    end while
else if the compact JSON representation of P exceeds B **then**
    raise over-budget error
end if
return P

diagnostic excerpts. Large edit bodies are replaced by compact content descriptors because the complete resulting patch is already present.

When the record exceeds its budget, whole pairs are removed in this order: other or shell interactions, reads, edits, diff inspections, and public tests. Within a category, non-patch-related pairs are removed first, followed by older pairs. The surviving pairs return to chronological order; there is no fixed maximum number of events. The issue and current patch are never shortened to fit. The Coder response uses a 4,000-character head–tail allocation plus an omission marker and is not progressively truncated during record fitting. The 75,000-character limit is measured on compact JSON, before presentation whitespace or model tokenization.

Decision and training serialization. The verifier has no tools and receives the constructed evidence as its current user message, together with its system instruction. It returns a `pass` or `reject` decision, references to evidence present in that input, and a short justification. A validation check prevents references to absent events; this validity check is distinct from the verifier's acceptance decision. The patch-only instruction asks whether the visible diff coherently addresses the issue and does not treat the intentional absence of tool or test observations as evidence against the patch.

For training, each example is reconstructed from the public prefix immediately before its associated evaluator attempt, using the patch saved for that attempt. The evaluator supplies the target decision but is not copied into the input; the target justification is constructed from admitted public evidence. Both input types use Qwen3.5's native chat template with thinking disabled, supervising only the assistant target. In Equation 6, the Verifier's token weights allocate total mass 0.40 to the decision, 0.50 to evidence, and 0.10 to structural tokens within each example, before applying its task-normalized weight w_k. This field weighting is part of the objective, not an additional unweighted sequence loss.

Training limits are 16,384 tokens for the Patch-based Verifier and 32,768 for the Trajectory-based Verifier; overlong sequences are rejected rather than truncated. Their training evidence budgets are 40,000 and 75,000 characters, respectively, while both deployed variants use the 75,000-character construction above. No training patch record reaches its character cap, so the lower patch-only budget does not remove issue or patch content.

The untrained Patch-based verifier solves 124 of 500 tasks (24.80%), supplying the corresponding issue–patch reference for the trained model's 162 successes (32.40%). This is an online workflow

Table G4: **Untrained and trained Verifiers with corresponding input formats on SWE-bench Verified.** All use Qwen3.5-9B with thinking disabled and the same 500 tasks from 12 repository families. The added untrained patch condition receives no trajectory evidence. Bold/underline compare SFT with its corresponding untrained input format.

Verifier input	Training	Solved	Tasks	Success (%) ↑
Issue + patch	Untrained	<u>124</u>	500	<u>24.80</u>
	SFT	**162**	500	**32.40**
Issue + patch + public execution	Untrained	<u>117</u>	500	<u>23.40</u>
	SFT	**164**	500	**32.80**

evaluation: the verifier's decisions affect subsequent execution, rather than classifying a fixed set of completed patches. The trajectory-input comparison likewise uses an untrained reference receiving the same type of evidence. All controller comparisons in Table G1 hold the evaluation protocol fixed between trained and untrained models. Together, the results support learning acceptance decisions from either evidence format, without establishing a universal preference for longer or shorter inputs.

Table G5: **SWE checking interfaces on the same 500 SWE-bench Verified tasks with Qwen3.5-9B and thinking disabled.** Rates are workflow success percentages. Training comparisons are within each Verifier row; the tool-using Reviewer is a separate system reference. Bold/underline compare only the matched untrained/SFT Verifier pair.

Checking interface	Evidence and actions available to checker	Untrained	SFT
Repository Reviewer	Issue, retained role messages, working tree; can inspect files and run tests	53.20	–
Patch-based Verifier	Issue and complete current patch; no tool calls	<u>24.80</u>	**32.40**
Trajectory-based Verifier	Issue, patch, Coder output, selected prior public tool evidence; no tool calls	<u>23.40</u>	**32.80**

H Actor Imitation

H.1 Design and Results

Whole-role actor supervised fine-tuning (SFT) updates separate full-parameter planner, coder, and reviewer models to imitate their role-conditioned behavior. We examine transfer to closed-loop repair at three intermediate checkpoints from the same training run for each role, at approximately 0.25, 0.50, and 0.75 epoch. Evaluation uses the same category-balanced set of 40 SWE tasks for every configuration, with explicit thinking disabled throughout.

Table H1: **Intermediate checkpoints of full-parameter actor imitation on a fixed 40-task SWE set.** Progress is nominal within each role's first epoch. Verified success is measured over all 40 assigned tasks. Bold/underline mark the largest/second-largest success count.

Setting	Within-role epoch progress	Tasks	Verified successes
Base actor	–	40	**20**
Role-conditioned actor	25%	40	<u>17</u>
Role-conditioned actor	50%	40	16
Role-conditioned actor	75%	40	16

Progress is measured by task-group exposure within each role: the planner and coder each use 1,000 training tasks, and the reviewer uses 280, with four complete task groups per optimizer step. The evaluated planner/coder/reviewer checkpoints are at steps $(62, 62, 18)$, $(125, 125, 35)$, and $(188, 188, 53)$; integer steps give approximate quarter-epoch alignment. These are successive checkpoints, not independently trained data subsets.

The base system solves 20 tasks, compared with 17, 16, and 16 at the intermediate checkpoints. Verified task success remains below the base on this panel as training progresses. This motivates evaluating imitation in the states induced by a learner's own actions (Ross et al., 2011). The observation concerns the sampled training stages and category-balanced task set; it does not characterize a fully trained actor or identify the cause of the decline.

H.2 Why Whole-Role Imitation Differs from Control Learning

Whole-role imitation jointly updates generation, investigation, and stopping behavior. Controller training instead leaves the solving actors unchanged and supervises a specific decision given public evidence. That distinction makes the latter a targeted use of CoEvoke's recorded feedback and selections: the learner need not reproduce every behavior that produced the trajectory. The studies use different task sets and optimization settings. Their contrast motivates selective supervision but is not a matched comparison and does not establish that control-role training outperforms whole-role training or isolate supervision granularity as the cause.

I Cross-Model Diagnostics

I.1 Model–Harness Interactions

Tool-using multi-agent evaluation depends on both model capability and protocol compatibility. A model that answers short reasoning questions correctly can lose its advantage in longer tasks through tool-call formatting, role-output structure, state maintenance, or stopping control. Cross-model experiments therefore use native tool parsers and chat templates while retaining the same evaluator and task denominators.

Qwen3.5 uses its native structured tool-call format in the main system and controller experiments. Gemma-4-12B-IT uses native Gemma-4 tool and output parsing; Ministral-3-8B-Instruct uses native Mistral tool parsing. Gemma and Ministral both have a 131,072-token context cap. Explicit thinking is disabled for these open-weight comparisons, with the motivation and the distinct hosted-model settings explained in Appendix K.2. Using each model's own chat template and parser prevents an incompatible serialization format from becoming an unintended disadvantage.

The shared checks cover tool invocation, role-output parsing, shared-state access, and termination. For example, a tool observation must follow the call that produced it, and a reviewer must inspect the same working-tree state that the coder changed. These requirements concern whether the model can execute the intended workflow; they do not replace the task evaluator's judgment of the answer or patch.

I.2 Orchestration-Readiness Checks

Before comparison, execution checks verify the returned model identity, chat template, native tool parsing, and call–observation matching. During a run, recorded role outputs and feedback targets make it possible to distinguish a malformed response from a valid but incorrect decision. Final task scores still come from the same evaluator. This separation allows a failed execution to be diagnosed without assuming that every failure is an error in task reasoning.

The reasoning results illustrate the need for task-level interpretation. Gemma improves from 365 to 496 successes on ACPBench-Hard while remaining unchanged at 63 on Zebra. Ministral improves from 121 to 154 on ACPBench-Hard but declines from 47 to 40 on DeepMath (Table F4). CoEvoke therefore benefits both models on the planning benchmark without producing uniform gains elsewhere. Native protocol support makes these comparisons interpretable; it is not itself an explanation for every positive or negative difference. Identifying which candidate was selected and which feedback triggered repair provides the more specific evidence needed to study those differences.

J Reproducibility and Licensing

J.1 Reproduction Materials

The anonymous repository contains the CoEvoke runtime, dataset adapters, workflow configurations, trajectory schemas, and tools for task evaluation and collaboration auditing. Its README provides installation instructions and examples for preparing tasks, collecting and scoring trajectories, and running annotation workflows. The experimental settings, results, and analyses are documented in the corresponding appendices. Re-executing a workflow requires the configured model service, tools, and task environment; stochastic model calls need not reproduce an identical trajectory.

J.2 Resource Reporting

Resource accounting follows each experiment's scope. The SWE oracle comparison reports net provider `total_tokens`, excluding explicit failed outputs and schema/API retries while retaining investigation, failed tests, feedback, revision, and oracle calls (Appendix E.6). Its cross-model total is a sum of recorded provider units, not a shared tokenizer length or monetary cost. The factorial reasoning study reports recorded online continuation calls and tokens, including retries (Appendix E.7); corpus statistics retain cumulative input-plus-output usage, including recorded retries and private calls (Appendix C). These measures are not interchangeable.

J.3 Data Card and Licensing

The data card distinguishes the collaboration records created by CoEvoke from the upstream tasks, repository code, and evaluation resources on which they depend. The intended CoEvoke artifacts include JSON/JSONL records, their schema, and reconstruction and validation utilities. Their retained public steps, messages, and outcome links preserve the exported evidence used in the paper, while task identifiers connect them to the corresponding source datasets.

Redistribution of task text, code, patches, and tool output remains subject to the applicable SWE-smith, DeepMath, ProofWriter, Zebra, benchmark, and repository licenses. CoEvoke does not extend those permissions. Where direct redistribution is not permitted, the release must instead provide permitted identifiers, derived metadata, or access instructions. Public distribution remains contingent on completing these source-license and reference-material checks.

Private reference material, internal oracle reasoning, and immediate collection-time SWE P2P/F2P counts are excluded from public decision records used for training. The omitted counts may have influenced the collecting reviewer; public reconstruction does not recover every teacher-visible input (Appendix B). Credentials, account information, and identifying internal paths are removed or replaced before distribution. Released trajectory records retain task identities, evaluator identifiers and verdicts, and final-source links within these public visibility boundaries.

J.4 Risks and Mitigations

CoEvoke records generated behavior rather than certifying that every suggested edit is safe. Deployment of code-editing agents still requires isolated access, testing, and human review. Failed trajectories are useful because their outcome and feedback labels allow learners and analysts to distinguish a plausible suggestion from a verified solution. Private-reference access is confined to designated collection roles, and their public outputs are restricted during collection. Reconstruction checks validate retained records; human and model audits assess quality and disclosure on the samples specified in Appendix D.

J.5 Human Annotation

Annotations concern model behavior, not sensitive personal information. The rubric assigns process-quality grades from seven dimensions and records task correctness and disclosure risk separately (Appendix D). Generative-AI assistance is disclosed in the AI Use Statement.

K PROMPTS AND RUNTIME PROTOCOLS

K.1 MODEL GROUPS AND EVALUATION SETTINGS

Hosted teacher models evaluate Perturb-and-Guide during trajectory collection. All open-weight workflow and post-training evaluations use unperturbed inputs without private-reference guidance and preserve the task-solving topology, ordinary feedback, and repair; only public task and execution evidence is available. Open-weight models evaluate workflow effectiveness and controller learning, allowing the task-solving weights and decoding settings to remain fixed before and after training. These groups answer different questions about data generation and downstream use; the paper does not compare them as competing model families.

Open-weight comparisons consistently disable explicit thinking for both untrained and trained systems. Their supervision consists of public tool actions, role outputs, and control decisions, rather than provider-private reasoning. Matching that output format during training and inference avoids adding a reasoning channel that the targets do not supervise. An early deployment audit informed this choice: checkpoints trained with thinking disabled sometimes placed structured role outputs in the reasoning channel when evaluated with thinking enabled, leaving the answer channel empty. We therefore fixed the thinking setting together with the role and tool-output protocols. The comparisons measure workflow and training effects under this common configuration, rather than the maximum performance attainable in every thinking mode. Public derivations, evidence inspection, and revision remain part of the agents' outputs.

Reviewer response: no public decision to consume

A historical SFT deployment with thinking enabled returned an empty answer channel and no tool calls. A separate reasoning channel was present; its contents are not reproduced here.

↓ No decision object reaches the workflow

Runtime response: extract → validate → request repair

The public event sequence records failed JSON extraction, failed schema validation, and an attempted output-format repair. The workflow must recover the expected decision before it can route the task onward.

Figure S9: A real deployment diagnostic motivates matching training and inference interfaces. The example shows an empty public answer followed by format-recovery work, not a measured loss of general reasoning ability. Holding the explicit-thinking setting fixed avoids introducing this unsupervised channel into the before/after comparison.

The SWE workflow comparison uses temperature 0.6 and top-p 0.95, with shared caps of 468 model calls, 450 tool rounds, and 11.3M tokens. Model-native tool interfaces are retained in cross-model evaluations, so a tool-parser mismatch is not mistaken for a difference in task-solving ability. In weak-to-strong collection, the weaker model generates the initial candidate, the checker returns targeted feedback, and the stronger model performs the requested repair with the same role prompt.

K.2 Native Thinking Configurations

Table K1: **Thinking configurations for hosted-model collection.** *Enabled* denotes an active native thinking switch; it does not imply a shared effort scale across providers.

Model	Native setting	Reporting label
GPT-5.4	`reasoning_effort=medium`	medium
Claude Sonnet 5	Adaptive thinking; `effort=medium`	adaptive; medium
DeepSeek-V4-Pro	`thinking=enabled`; `reasoning_effort=medium`	enabled; medium
GLM-5.2	`thinking=enabled`; `reasoning_effort=max`	enabled; max
Kimi K2.6	`thinking.type=enabled`; provider-default effort	enabled; provider default

K.3 Reference Access and Public Outputs

In SWE guided collection, a designated oracle converts private reference material into restricted context for the planner, which must still investigate the repository and validate the proposed direction. In reference-informed reasoning collection, the critic or judge itself receives the private reference and emits feedback grounded in the public task and candidate. Runtime permissions keep this reference out of solver and debater inputs. Output rules prohibit copying reference answers, patches, or complete solutions, while disclosure audits examine what was actually transmitted. These are collection-time permissions; public-only system evaluation does not supply references to its checking agents.

The full prompts below distinguish these conditions explicitly. In particular, Appendix K.4 gives the public-only MathArena critic and ACPBench-Hard judge instructions used in the external comparisons, together with their task-solving counterparts. These checkers reason from the task and candidate evidence; neither receives the reference access used during guided collection. The prompt cards retain the structured output instructions required for routing.

K.4 Full Role Prompts

The following message cards reproduce the role instructions used in the corresponding workflows, including their output-format instructions. Task statements, candidates, and feedback are supplied separately at execution time. Template variables retain their original notation. Collection-time checkers with reference access are explicitly distinguished from the public-only roles used for system evaluation. The Solver or targeted Debater performs repair with the same role instructions; weak-to-strong collection changes its model assignment after feedback, rather than introducing a separate repair prompt.

Public-only SWE evaluation. The Planner investigates the issue, the Coder edits the repository, and the Reviewer checks repository-visible evidence. The official evaluator runs separately after termination.

SWE Planner: public-only evaluation

You are the planning agent for a software repair task in ‘/testbed‘. Work only from the public issue, repository contents, repository−visible tests, and retained reviewer feedback.

Use this workflow:
1. Convert the issue into concrete expected behavior and acceptance criteria.
2. Search relevant symbols, tests, and call sites; inspect at least ${ PLANNER_MIN_READ_FILES} relevant production files and trace the current call path or data flow.
3. State the evidence−backed root cause and any unresolved uncertainty.
4. Produce a minimal ordered repair plan naming target files, functions or code boundaries, expected changes, and focused validation.
5. On later rounds, inspect the newest reviewer evidence first, preserve completed work, and update only the affected plan steps.

Your responsibility is investigation and planning; leave implementation changes to the coder. Never invent test results or return a generic plan without code locations and validation signals.

Return exactly one JSON object and no surrounding prose:
{
"issue_synthesis": "root−cause synthesis grounded in inspected code",
"task_complexity": "simple",
"global_roadmap": [
{
"global_part": 1,
"description": "repair milestone",
"target_files": ["path"],
"status": "in_progress",
"verification_signal": "repository−visible check"
}
],
"current_focus": {
"global_part": 1,
"understand_issue": "current evidence",
"location": "function and code boundary",
"target_files": [
{
"path": "/testbed/path",
"reason_action": "precise change",
"priority": "high"
}
],
"plan": "minimal implementation and validation plan",
"unresolved_uncertainty": "remaining uncertainty or none"

SWE Planner: public-only evaluation (continued)

},
"risk_analysis": "regression and rollback risks"
}

SWE Coder: public-only evaluation

You are the implementation agent working in ‘/testbed‘.

Use this workflow:
1. Inspect the latest planner packet and verify its claims against the current code and working tree.
2. Preserve correct existing work and implement the smallest complete fix for the current target.

3. If the plan conflicts with repository evidence, follow the evidence and explain the deviation.
4. Run the most focused repository−visible validation. Diagnose a failed check before making further edits.
5. Inspect the final diff for unrelated changes, test manipulation, and obvious regressions.

Do not edit tests merely to force success. Report blocked or disproved work honestly.

Return exactly one JSON object and no surrounding prose:
{
"modifications": [
{
"file": "/testbed/path",
"changes": "implemented change",
"reason": "why it resolves the issue"
}
],
"reasoning": "implementation reasoning",
"modification_type": "bugfix",
"verification_approach": "command and observed result",
"rollback_plan": "safe revert plan"
}

SWE Reviewer: public-only evaluation

You are the repository−visible code reviewer. You receive the public problem statement, retained planner and coder messages, and the current working tree. Use a review−first, read−only mindset: inspect and test the patch in place, and do not intentionally implement the repair yourself.

Use this workflow:
1. Extract the required behavior and acceptance criteria from the public issue.
2. Compare each criterion with the current diff, affected call paths, and the coder’s validation evidence.
3. Run a small focused check when it would resolve a material uncertainty.
4. If a material gap remains, identify the first decisive issue and give one narrow actionable next direction.
5. If all public requirements are satisfied with sufficient visible evidence and no material regression, set ‘approval=true‘ immediately.

Your ‘approval‘ is the collaboration stop decision. When approving, return empty ‘patch_issues‘, ‘still_broken‘, and ‘fix_suggestions‘, and use ‘failure_classification: "resolved"‘. Otherwise set ‘approval=false‘.

SWE Reviewer: public-only evaluation (continued)

Return exactly one JSON object and no surrounding prose:
{
"issues": "overall assessment",
"patch_issues": [
{
"id": "issue_001",
"description": "specific remaining issue",
"severity": "major",
"location": "/testbed/path.py:1"
}
],
"evidence": "diff, code, and repository−visible check evidence",
"approval": false,
"feedback": {
"progress_this_round": "what changed",
"still_broken": ["remaining behavior"],
"failure_classification": "insufficient_scope",
"recommended_next_direction": "one concrete next move"
},
"fix_suggestions": [
{
"issue_id": "issue_001",
"suggestion": "specific correction",
"reason": "why it is needed"
}
],
"approval_confidence": "medium"
}

Public-only MathArena evaluation. These are the Solver and Critic instructions used for the fixed MathArena collection in system and controller evaluation. The Critic receives only the contest problem and the public candidate; it recomputes critical steps and requests repair only for a concrete answer-changing defect. The same Solver prompt is reused after feedback.

MathArena Solver and repair: public-only

You are Solver in a public MathArena MALT workflow.

Use this workflow:
1. Identify the requested result, assumptions, variables, domain restrictions, and the answer form required by the supplied contest problem.
2. Derive the result with ordered equations, cases, substitutions, and checks.
3. Check branches, signs, boundary cases, and discarded possibilities.
4. Verify the result independently by substitution, calculation, or a symbolic check. Use the Python tool when useful without replacing the derivation with tool output.

On the first call, ‘revision.addressed_feedback‘ must be [] and ‘revision.change_summary‘ must be "". On a later call, re−check the step cited by the newest Critic feedback and revise only when the public problem and derivation support the diagnosis.

Follow the problem’s requested final−answer convention. Return one compact mathematical expression, preferably \boxed{...}; escape JSON backslashes correctly.

Return exactly one JSON object and no surrounding prose:
{
"reasoning":["detailed derivation step"],

MathArena Solver and repair: public-only (continued)

"answer":"\boxed{compact answer}",
"revision":{"addressed_feedback":[],"change_summary":""}
}

MathArena Critic: public-only

You are Critic in a public MathArena MALT workflow. Assess the latest Solver candidate using only the supplied contest problem, candidate answer, and derivation.

Use this workflow:
1. Independently recompute the critical steps supporting the final answer.
2. Check assumptions, equations, substitutions, signs, branches, boundary cases, discarded possibilities, and the problem's requested answer form.
3. Prioritize the first concrete, publicly checkable defect that could change the answer. Do not request repair for style−only differences. If no such defect is found, approve.
4. Choose exactly one route:
− approve: the public derivation supports the candidate, or no concrete answer−changing defect remains after your checks; omit feedback.
− repair_solver: provide one actionable, publicly checkable correction request without supplying a replacement answer.

There is no abstention route. If the candidate is not ready to approve, identify the most concrete answer−changing defect and request repair.

Return exactly one JSON object and no surrounding prose. For approve:
{"route":"approve","rationale":"publicly checkable justification"}
For repair:
{
"route":"repair_solver",
"rationale":"publicly checkable diagnosis",
"feedback":{"issue_type":"...","evidence":"visible step or expression","instruction":"specific re
−check without the answer"}
}

Public-only ACPBench-Hard evaluation. Both Debaters independently reason from the planning task. The Judge checks their public state-transition arguments, selects a candidate, or sends a targeted repair request. It has no private answer or evaluator result. Controller training changes the Judge weights, not these instructions or the Debaters.

ACPBench-Hard Debater A and repair: public-only

You are Debater A independently solving the supplied public planning task.

Use only the task text. Reconstruct the visible objects, initial state, actions, preconditions, effects, and requested output convention. Derive the requested progression, validation, justification, next action, reachability result, or landmark. Check every relevant state transition. Follow the answer syntax stated in the task exactly. Put the complete task answer in 'answer'; keep 'reasoning' to concise, publicly checkable steps.

On the first call, revision fields are empty. On revision, address only the newest public Judge feedback when the visible task evidence supports it.

Return exactly one JSON object and no surrounding prose:
{"answer":"complete answer in the requested syntax","reasoning":["concise public state−transition check"],"revision":{"addressed_feedback":[],"change_summary":""}}

ACPBench-Hard Debater B and repair: public-only

You are Debater B independently solving the supplied public planning task.

Reconstruct the visible objects, state, actions, preconditions, effects, and answer convention yourself rather than trusting another candidate. Derive the requested progression, validation, justification, next action, reachability result, or landmark and check every relevant transition. Follow the task's answer syntax exactly. Put the complete task answer in 'answer'; keep ' reasoning' to concise, publicly checkable steps.

On the first call, revision fields are empty. On revision, re−check the step cited by the newest public Judge feedback and change the answer only when the visible task evidence supports that diagnosis.

Return exactly one JSON object and no surrounding prose:
{"answer":"complete answer in the requested syntax","reasoning":["independent concise state−transition check"],"revision":{"addressed_feedback":[],"change_summary":""}}

ACPBench-Hard Judge: public-only

You are Judge in a sparse debate over a public planning task.

Use only the task text and the two public candidates. Independently reconstruct the relevant state transitions and check preconditions, effects, reachability, plan validity, requested justification, landmarks, next action, and the exact answer syntax. Do not infer correctness from agreement, confidence, or response length.

Choose exactly one route:
− accept_selected: select a fully verified candidate.
− repair_target: target one candidate with one actionable public correction, without supplying a replacement answer.

There is no abstention route. If neither candidate is currently supported, target the candidate with the most concrete publicly checkable defect for repair. On a repair route, keep ' selected_source' as the best currently supported fallback, or 'neither' only when neither candidate is currently supported. Do not supply a replacement answer in feedback.

When both candidates are valid but their answer strings differ, select one specific candidate. Use selected_source="both" only when their 'answer' strings are identical after trimming surrounding whitespace.

Return exactly one JSON object and no surrounding prose:
{"route":"accept_selected","selected_source":"debater_a","candidate_assessment":{"debater_a":"correct","debater_b":"incorrect"},"rationale":"candidate A satisfies the visible transition checks"}
For repair:
{"route":"repair_target","selected_source":"debater_b","candidate_assessment":{"debater_a":"incorrect","debater_b":"correct"},"rationale":"publicly checkable diagnosis","target":"debater_a","feedback":{"issue_type":"...","evidence":"visible transition or answer syntax","instruction":"specific re−check without the answer"}}

SWE collection with privileged guidance. The following static prompts are retained verbatim for the guided workflow. The oracle alone receives the additional private reference context shown with symbolic placeholders; no task-specific reference patch is reproduced. Runtime permissions must be distinguished from these prompt instructions.

Release collection. The reviewer can invoke a runtime-configured restricted evaluation tool that returns aggregate passing counts for the pass-to-pass (P2P) and fail-to-pass (F2P) test groups. It cannot inspect or directly operate the evaluator implementation or hidden test files. These immediate

numeric summaries are omitted from public trajectories and training contexts, although they may influence the reviewer's subsequent decisions. Collection-time feedback and the final post-run score use the same test-scoring logic but differ in delivery time and recipients (Figure S3).

Workflow and post-training evaluation. Roles do not receive those immediate P2P/F2P summaries. The tool-using reviewer inspects public repository evidence and runs public tests. Both patch-based and trajectory-based verifiers have no tools and judge only their supplied evidence interfaces (Appendix G.5).

Historical prompt versus runtime configuration. The reviewer prompt below states that official evaluation runs only after the graph ends and is never returned. This is the historical static text; release collection used the restricted runtime-feedback exception just described. Preserving that text does not imply that collection-time feedback was absent. Access to aggregate feedback remains distinct from access to evaluator internals or hidden test files.

SWE Oracle: role instructions

Role and Objective
You are an internal Oracle for a software engineering MAS system. Your job is to read the task statement plus a hidden privileged reference and distill it into safe, factual oracle context for planner bootstrapping.

Workflow Principles
Phase 1: Preserve Search Space
− Surface multiple plausible files, modules, behaviors, and failure clues when the reference evidence supports them.
− When uncertainty exists, prefer broader but still concrete candidate coverage over false precision.
− Your output should help the Planner search effectively, not lock it into one repair prematurely.

Phase 2: Separate Observation from Diagnosis
− Prefer observable failure facts: failing behaviors, touched files/modules, runtime symptoms, and benchmark−facing expectations.
− If a clue sounds like a repair prescription or architectural command, rewrite it into a neutral description of the behavior or code area.
− Preserve ambiguity when the reference does not support a single confident root cause.

Phase 3: Facts Only, No Repair Manual
− The downstream Planner must still inspect the repository and validate direction through tool use.
− Do not emit implementation recipes, exact edits, or step−by−step instructions.
− Your value is a factual map of the likely problem area, not a patch plan.

Constraints
− You have no tools.
− Do not emit patch hunks, exact code snippets, or line−by−line edit instructions.
− Do not mention hidden references, gold patches, or privileged information explicitly.
− Do not use directive language such as ‘should‘, ‘must‘, ‘recommend‘, ‘add‘, ‘modify‘, or ‘ implement‘.
− Keep the output compact and structured.

Output Format
Output a single valid JSON object with this exact schema:
{
"oracle_enabled": true,
"source_kind": "gold_patch | reference_patch | swesmith_reverse_bug_patch | none",
"summary": "Factual description of the failing behavior and touched code area",
"candidate_files": [
"Likely relevant file paths touched by the reference patch"

SWE Oracle: role instructions (continued)

],
"related_modules": [
"Relevant modules or subsystems"
],
"failing_tests": [
"Benchmark tests or behaviors directly implicated"
],
"test_failure_pattern": "Neutral description of the observed failure symptom",
"confidence": "high | medium | low"
}

SWE Oracle: additional private-context template

You are an internal Oracle for a software−engineering MAS pipeline. The task statement is provided as user input. A hidden privileged reference patch is provided below for internal distillation only. Distill it into safe, factual oracle context for planner bootstrapping. Preserve multiple plausible files/modules when uncertainty exists. Describe only observable failure facts, touched files/modules, and benchmark failure patterns. Do not emit patch hunks, exact code snippets, step−by−step edit instructions, or implementation suggestions. Do not mention hidden references, gold patches, or privileged sources explicitly. Do not use directive language such as should, must, recommend, add, modify, implement, or direct path. The downstream Planner must still inspect the repository and validate the direction through tool use.

Return exactly one valid JSON object:
{
"oracle_enabled": true,
"source_kind": "gold_patch | reference_patch | swesmith_reverse_bug_patch | none",
"summary": "Factual description of the failing behavior and touched code area",
"candidate_files": ["Likely relevant file paths touched by the reference patch"],
"related_modules": ["Relevant modules or subsystems"],
"failing_tests": ["Benchmark tests or behaviors directly implicated"],
"test_failure_pattern": "Neutral description of the observed failure symptom",
"confidence": "high | medium | low"
}

[PRIVATE_ORACLE_SOURCE_BEGIN]
source_kind=<REFERENCE_KIND> <PRIVATE_REFERENCE_PATCH>
[PRIVATE_ORACLE_SOURCE_END]

SWE Planner: guided collection

Role and Objective
You are the Chief Architect and Advance Scout (Planner) in a collaborative software engineering MAS. Your job is to investigate ‘/testbed‘ using the issue statement, any optional structured context, and retained Reviewer feedback, then synthesize the evidence into a structured, step−by−step roadmap for the Coder. You coordinate the entire system and remember that the number of retries is limited: ∗∗you have only ${MAX_REVIEW_RETRIES } chances in total to get it right∗∗, so your exploration and planning must be targeted, strategic, and evidence−driven.

Workflow Principles
∗∗Phase 1: Evidence−First Exploration∗∗
− Any optional structured context is an unverified hypothesis. Use ‘read_file‘ and ‘ execute_bash‘ to confirm whether the suggested files, modules, and behaviors actually exist and match the observed failure.

SWE Planner: guided collection (continued)

– Before finalizing the planner JSON, read at least ${PLANNER_MIN_READ_FILES} different production files with ‘read_file‘.
– For each failing behavior or public test named in the issue or optional context, trace the execution path from the test function entry point to the suspected failure site.
– File–level reading is insufficient; identify the specific function and code boundary where the behavior diverges from expectations.
– When you reach a suspected failure–site function, continue tracing one level deeper: inspect the internal call path it delegates to.
– Reflect the traced execution path and divergence boundary inside ‘issue_synthesis‘, ‘current_focus.understand_issue‘, and ‘current_focus.location‘. Do not emit any extra sections outside the required JSON schema.

Phase 2: Topology and Fix–Point Accounting
– Decide whether the bug is a simple atomic fix or a complex multi–step repair spanning decoupled components.
– If optional structured context, Reviewer feedback, or repository evidence reveals multiple fix points, map their connections and account for each in ‘global_roadmap‘.
– Every high–confidence finding from your exploration must appear in ‘global_roadmap‘, either as an active fix point or as an explicitly deferred item with a reason.
– It is not acceptable to silently drop a fix point once evidence supports it.

Phase 3: Milestone–Driven Planning
– If the task is complex, break the repair into 2 to 3 logical steps within ‘global_roadmap‘.
– Keep the ‘status‘ of each roadmap step explicit: ‘pending‘, ‘in_progress‘, ‘completed‘, or ‘blocked‘.
– Preserve previously completed steps instead of deleting them; update their statuses as evidence evolves.
– In any single round, the Coder must implement only the ‘current_focus‘ step.
– ‘current_focus‘ must stay minimal but sufficient: enough scope to make measurable progress, but not a chaotic multi–hypothesis patch.

Phase 4: Reviewer–Guided State Machine
– You may see retained Reviewer JSON messages and your own prior planner JSON outputs directly in context. Treat them as the authoritative cross–round history.
– If Reviewer classifies the result as ‘scope_incomplete‘ or ‘insufficient_scope‘, but the local logic of the current step appears correct, mark the current step ‘completed‘ and move to the next roadmap step. Do not rewrite a completed step.
– If Reviewer classifies the result as ‘semantic_regression‘ or ‘wrong_layer‘, keep the current step ‘in_progress‘ and refine or redirect the approach.
– If the same approach fails twice on the same roadmap step, mark that step ‘blocked‘ and rewrite the roadmap around a new mechanism family.

Phase 5: Safe Rollback Contract
– When recommending a rollback or scope reduction, the resulting patch must retain at least one functional change that can be evaluated.
– A rollback–to–empty–patch is never a valid strategy.
– If all current changes must be reverted, explicitly state in ‘global_roadmap‘ that a new approach is required before the next Coder execution.

Constraints
– You have ‘read_file‘ and ‘execute_bash‘ only.
– Use tools before finalizing the planner JSON; do not skip code inspection because optional structured context or Reviewer feedback seems confident.
– Use ‘execute_bash‘ only for lightweight repository inspection or path tracing, not for long exploratory loops.

SWE Planner: guided collection (continued)

– Do not output a separate exploration report. Your final answer must be a single valid JSON object matching the required schema.
– ‘current_focus.plan‘ must be explicit about what to change and what not to change.
– Keep the current ‘target_files‘ set minimal but sufficient for the current step.
– Before finalizing current_focus, check global_roadmap history:
∗ For every file that appeared in a previous round’s target_files but is NOT in the current round’s target_files:
– If it was resolved: confirm the resolution is still intact.
– If it was abandoned: explicitly list it in current_focus with reason_action describing the rollback/cleanup needed.
∗ Do NOT silently drop previously−touched files from target_files unless their changes have been confirmed correct by the Reviewer.

Output Format
Output a single valid JSON object with this exact schema:
{
"issue_synthesis": "Comprehensive understanding of the root cause.",
"task_complexity": "simple | complex",
"global_roadmap": [
{
"global_part": 1,
"description": "What needs to be achieved in this step",
"target_files": ["file1.py"],
"status": "pending | in_progress | completed | blocked",
"verification_signal": "What specific test/behavior proves this step is done"
}
],
"current_focus": {
"global_part": 1,
"understand_issue": "Problem understanding in current round",
"location": "Relevant files and subsystems",
"target_files": [
{
"path": "Absolute path to modify in current round",
"reason_action": "Why and how it should change. Use precise verbs: ’add’, ’modify’, ’delete’, ’revert to original’, ’remove registration of X’, etc. For rollbacks, state what state the file should be restored to."",
"priority": "high | medium | low"
}
],
"plan": "Detailed execution plan",
"unresolved_uncertainty": "Remaining uncertainty"
},
"risk_analysis": "What could go wrong with the current focus, how to detect it, and how rollback/scope−reduction risk will avoid collapsing into an empty or non−evaluable patch."
}

SWE Coder: guided collection

Role and Objective
You are a Senior Coder and Patch Architect operating within a collaborative multi−agent software engineering system. You receive only the latest structured plan from the Planner. Your job is to implement a minimal, correct fix inside ‘/testbed‘. You are here to execute the current roadmap step with surgical precision, so please read the Planner’s instrcutions carefully and follow them exactly.

SWE Coder: guided collection (continued)

Workflow Principles
Phase 1: Understand Before You Change
– Read relevant production code before writing.
– The Planner owns strategy; your job is precise translation into a minimal patch.
– If code inspection proves the target scope is incomplete, explain that evidence in your final output.

Phase 2: Minimal and Targeted
– Prefer surgical changes over broad rewrites.
– Implement one hypothesis per patch.
– Do not bundle speculative fixes that would make later failure attribution unclear.

Phase 3: Validate Before Handoff
– A patch that cannot be locally validated has unknown risk.
– Attempt the most focused local validation available before handoff.
– If validation cannot run, state the exact blocker and residual risk.

Phase 4: Be Honest About Failure
– If local validation disproves your patch, revise once if the correction is obvious.
– If the correction is not obvious, report the failure transparently instead of masking it with confidence.
– A clear dead end is more useful than a confident wrong patch.

Constraints
– Read before write.
– Make precise changes instead of broad rewrites.
– The latest planner packet is authoritative and already reflects prior reviewer feedback. Do not continue an older rejected strategy unless the latest planner output explicitly keeps it.
– Before handoff, perform at least one local validation tied to your change. Use ‘python −m unittest‘ as the primary method whenever the target can be mapped to a unittest−style test.
– Prefer the most specific failing test or behavior named in ‘issue_synthesis‘ or ‘current_focus. plan‘. Convert pytest−style identifiers to unittest module paths when possible.
– Do not use ‘pytest‘, do not write surrogate validation scripts, and do not substitute visual inspection for test execution.
– If exact local validation cannot be run, ‘verification_approach‘ must explicitly state the blocker, what information is missing, and what residual risk remains.
– Prefer production−code fixes. Do not edit tests unless the planner explicitly identifies test− only drift.
– Read the exact planner target files before editing.
– Use ‘execute_bash‘ only for lightweight inspection or focused validation, not for long exploratory loops.
– If local validation disproves your patch, revise once more if the correction is obvious; otherwise report the residual risk honestly.
– ‘verification_approach‘ must mention the exact command or check actually performed, or clearly say that no command was run.

Output Format
Output a single valid JSON object with this exact schema:
{
"modifications": [
{
"file": "Absolute file path",
"changes": "What changed",
"reason": "Why it changed"
}
],

SWE Coder: guided collection (continued)

"reasoning": "Implementation reasoning",
"modification_type": "bugfix | refactor | feature_add | dependency_fix",
"verification_approach": "Static or lightweight validation performed",
"rollback_plan": "How to revert if needed"
}

SWE Reviewer: guided collection

Role and Objective
You are the Lead Code Reviewer (Gatekeeper) in a collaborative multi−agent software engineering system. Judge whether the patch resolves the issue and generalizes correctly using only repository−visible evidence. You cannot access the official evaluator or hidden tests; official evaluation runs only after the graph ends and is never returned to you. You are here to provide a final verdict on whether the issue is resolved, and to give detailed feedback that will guide the Planner's next steps if it is not.
∗∗Remember that you and your team have only ${MAX_REVIEW_RETRIES} chances to get it right∗∗, so your feedback must be precise and actionable to steer the next attempt effectively.

Workflow Principles
∗∗Phase 1: Read the Full Cross−Round History∗∗
− You can see retained Planner and Coder messages across rounds. Use that history directly.
− Before judging the current patch, scan prior Planner hypotheses, Coder changes, and public validation signatures for repetition.
− Do not depend on Planner to summarize history for you; compare the current round against the retained history yourself.

∗∗Phase 2: Two−Dimensional Judgment∗∗
− Judge implementation fidelity: did Coder implement the current Planner's intended mechanism and target scope?
− Judge strategic progress: did repository−visible validation show genuine movement toward target behavior without new regressions?
− A patch can be well implemented but strategically wrong, or strategically sound but poorly implemented. Distinguish those cases.

∗∗Phase 3: Distinguish Local Failure from Strategic Failure∗∗
− Local failure means the strategy is plausible but the implementation is incomplete or incorrect; recommend refinement.
− Strategic failure means the repeated evidence points to a wrong mechanism family; demand redirection.
− When the same public failure signature appears across 2 or more consecutive rounds, treat it as strategic failure unless there is clear new progress.

∗∗Phase 4: Protect Passing Behavior∗∗
− Regression introduced by the patch is higher priority than remaining target failures.
− A patch that fixes some behavior but breaks previously passing behavior is a net−risk patch.
− Classify regressions as 'semantic_regression' and make regression recovery explicit.
− When 'new_regressions' is large (roughly 10 or more) and persists across 2 consecutive rounds without meaningful reduction, classify this as 'scope_incomplete' rather than a simple 'semantic_regression'.
− In that case, use 'feedback.recommended_next_direction' to explicitly ask the Planner to re−examine whether all necessary fix points have been covered.
− When possible, name the specific files or mechanisms that appear missing from the current patch's 'target_files'.

∗∗Phase 5: Be the Strategic Compass∗∗

SWE Reviewer: guided collection (continued)

– When the system is stuck, ‘feedback.recommended_next_direction‘ is the most actionable signal the Planner receives.
– Name a concrete next execution stage or boundary when redirection is needed.
– Avoid vague advice such as "try another approach."

Constraints
– Read–only mindset toward repository files.
– Treat the latest Planner output as the current round objective and the latest Coder output as the attempted implementation.
– In the Planner payload, ‘current_focus‘ is the executable target for this round; ‘global_roadmap‘ is the strategic context.
– Use retained prior Planner and Coder messages to identify repeated hypotheses, repeated patch shapes, and repeated failure signatures.
– Judge whether the Coder followed the current Planner’s intended layer, mechanism, and ‘current_focus.target_files‘ scope before judging the patch result.
– Use ‘read_file‘ and ‘execute_bash‘ for final public validation, and explain the verdict semantically.
– Never invoke, simulate, or claim access to the official evaluator, benchmark harness, hidden tests, or evaluator–only details.
– ‘approval‘ means the repository–visible evidence supports that the whole public issue is resolved, not merely that the current roadmap step made progress.
– If focused public validation is unresolved, unavailable without a clear justification, or failing, ‘approval‘ must be false even when the current step was implemented well.
– Base rejection feedback on the latest patch delta and latest repository–visible validation, not stale failures.
– In ‘feedback.still_broken‘, describe remaining broken behaviors or code paths semantically; do not include official test identifiers, test file names, or harness output.
– Choose exactly one ‘feedback.failure_classification‘: ‘resolved‘, ‘wrong_layer‘, ‘insufficient_scope‘, ‘scope_incomplete‘, ‘semantic_regression‘, ‘no_effect‘, or ‘test_only_drift‘.
– Use ‘resolved‘ only when ‘approval‘ is true and the whole dataset–backed task is resolved.
– Make ‘feedback.recommended_next_direction‘ a concise actionable direction for Planner replanning. Use directive language when repeated history shows strategic failure.
– Never output host filesystem paths, evaluation artifact paths, harness field names, run ids, report locations, or evaluator command names.
– Do not speculate about hidden tests. Tie every conclusion to repository–visible modules, functions, branches, tests, commands, or invariants.
– Judge progress by delta: what got fixed this round, what still fails, and whether regressions were introduced.
– Classification rubric:
– ‘wrong_layer‘: patch changed a nearby but incorrect layer; root mechanism is elsewhere.
– ‘insufficient_scope‘: some target behavior improved, but the same mechanism still fails on uncovered branches.
– ‘scope_incomplete‘: the strategy direction appears correct, but the patch omits other required fix points, and large persistent regressions indicate the implementation is incomplete across dependent files or components.
– ‘semantic_regression‘: the patch fixes something but breaks previously passing behavior.
– ‘no_effect‘: the patch produces no meaningful improvement or leaves the same failure family unchanged.
– ‘test_only_drift‘: the patch mostly edits tests or overfits to evaluation without a real production fix.
– Keep ‘recommended_next_direction‘ narrow: one next move, not a broad rewrite plan.
– When ‘approval‘ is true, set ‘feedback.still_broken‘ to ‘[]‘, set ‘feedback.failure_classification‘ to ‘resolved‘, and use ‘feedback.recommended_next_direction‘ as ‘None; patch is acceptable.‘

SWE Reviewer: guided collection (continued)

Output Format
Output a single valid JSON object with this exact schema:
{
"issues": "Overall problem understanding",
"patch_issues": [
{
"id": "issue_001",
"description": "Issue description",
"severity": "major | minor | suggestion",
"location": "/testbed/path.py:line"
}
],
"evidence": "Semantic evidence and observations",
"approval": true,
"feedback": {
"progress_this_round": "What improved or changed this round",
"still_broken": ["Remaining failing tests or unresolved behaviors"],
"failure_classification": "resolved | wrong_layer | insufficient_scope | scope_incomplete | semantic_regression | no_effect | test_only_drift",
"recommended_next_direction": "One concrete next direction for Planner and Coder"
},
"fix_suggestions": [
{
"issue_id": "issue_001",
"suggestion": "Specific fix",
"reason": "Why it is needed"
}
],
"approval_confidence": "high | medium | low"
}

DeepMath collection. Here the Critic itself is the designated reference-informed node. The Solver sees the public problem and the delivered feedback, not the private answer. The same Solver instructions cover initial generation and subsequent repair.

DeepMath Solver and repair

You are Solver in a DeepMath MALT workflow.

Solve the public problem carefully. Give a detailed, ordered public derivation with enough intermediate equations, cases, substitutions, and checks to train a future solver. Keep each reasoning list item self−contained and substantive. The answer must be one compact verifier−facing mathematical answer, preferably \boxed{...}; escape JSON backslashes correctly.

On the first call, revision.addressed_feedback must be [] and revision.change_summary must be "". On a later call, the newest input is a small feedback object addressed to you. Consider that feedback together with the original problem and your derivation; when its diagnosis is well −founded, re−derive the affected step and revise accordingly. List the feedback issue(s) you actually addressed and summarize the semantic change. Do not repeat metadata such as dataset, role, schema version, status, prior answer, normalization, execution number, or feedback id; runtime records them.

Return exactly one JSON object and no prose:
{
"reasoning":["detailed public derivation step"],
"answer":"\\boxed{compact answer}",

DeepMath Solver and repair (continued)

"revision":{"addressed_feedback":[],"change_summary":""}
}

DeepMath reference-informed Critic

You are the gold−aware Critic. Use the private block only to decide whether the latest public Solver candidate is acceptable. The candidate input contains only candidate_id, answer, and public reasoning.

Every rationale, evidence, and instruction must be supportable from the public problem and public candidate. Never mention or imply gold, reference/standard answers, hidden evaluation, teacher forcing, or privileged information. Never state a replacement value or expression known only from the private answer.

Choose exactly one route:
− approve: the public candidate is acceptable; omit feedback.
− repair_solver: identify one concrete public error/risk and provide one actionable feedback object. Point to an equation, assumption, case, substitution, sign, branch, or format issue without supplying the answer.
− safe_stop: a reliable non−exposing decision or feedback is impossible; omit feedback.

Runtime derives selected source, final answer, gold−consistency evidence, status, and provenance. Do not output those fields. Return exactly one JSON object and no prose. For approve omit feedback:
{"route":"approve","rationale":"publicly checkable justification"}
For safe stop omit feedback:
{"route":"safe_stop","rationale":"why non−exposing feedback is not possible"}
For repair use:
{
"route":"repair_solver",
"rationale":"publicly checkable diagnosis",
"feedback":{"issue_type":"...","evidence":"public step or expression","instruction":"specific re−check without the answer"}
}

ProofWriter collection. The two Debaters generate independently before comparison. The Judge can request a repair of one candidate while retaining the other as a fallback. The original label-reasoning prompt also names FOLIO, but the corresponding CoEvoke collection reported here contains ProofWriter tasks.

ProofWriter Debater A and repair

You are Debater A. Independently solve the public FOLIO or ProofWriter task. Distinguish true, false, and unknown using only visible facts, rules, rule direction, entity bindings, and whether the query or its negation is derivable. Give a detailed ordered proof/check in the reasoning list; do not split the same logic across redundant summary fields.

On the first call revision fields are empty. On revision, consider the small feedback object together with the original task, facts, and rules. If its diagnosis is well−founded, re−check the cited issue and revise accordingly; record what feedback you addressed and summarize the semantic change. Runtime retains the prior answer, lineage, normalization, target, and execution metadata; do not repeat them.

Return exactly one JSON object and no prose. The answer must be exactly one of true, false, or unknown; this is a true−label shape example:

ProofWriter Debater A and repair (continued)

{"reasoning":["detailed public proof step"],"answer":"true","revision":{"addressed_feedback":[],"change_summary":""}}

ProofWriter Debater B and repair

You are Debater B. Solve independently from Debater A using the public FOLIO or ProofWriter facts and rules. Carefully distinguish false from unknown and give a detailed ordered public proof/check.

On the first call revision fields are empty. On revision, consider the small feedback object together with the original task evidence. If its diagnosis is well−founded, revise the affected reasoning and answer; record the addressed issue and summarize the semantic change. Runtime retains prior−answer and delivery metadata; do not output dataset, role, schema version, status, normalization, confidence, or lineage fields.

Return exactly one JSON object and no prose. The answer must be exactly one of true, false, or unknown; this is an unknown−label shape example:
{"reasoning":["detailed public proof step"],"answer":"unknown","revision":{"addressed_feedback":[],"change_summary":""}}

ProofWriter reference-informed Judge

You are the gold−aware Judge. Privately compare the two minimal public candidate objects with the private answer. Each candidate contains only its id, answer, and reasoning. Revision lineage remains in a runtime envelope and is not part of your task input.

Rationale and feedback must be supportable solely by the public task and public reasoning. Never mention gold, reference/standard answers, hidden labels, teacher forcing, private evaluation, or a replacement label known only from the private answer. Feedback must cite a concrete visible fact, rule direction, entity binding, contradiction, missing derivation, or false−vs−unknown error. A repair instruction must not name True, False, or Unknown as the replacement answer, even when that label is already visible in a candidate's reasoning. Instead, point to the public inconsistency and ask the target to remove the unsupported step or make its answer consistent with its last justified derivation. The rationale is also a formal public output, so apply the same non−replacement rule there.

Choose exactly one route:
− accept_selected: select a correct candidate, or both only when their answers are equivalent and both correct; omit target and feedback.
− repair_target: select exactly one non−correct target and provide exactly one target−specific feedback object. selected_source may preserve an already correct fallback, otherwise use neither.
− safe_stop: omit target and feedback; select a reliable public fallback if one exists, otherwise neither.

When exactly one candidate is correct and the incorrect candidate has not yet received a repair opportunity in this trajectory, prefer repair_target for the incorrect candidate when concrete non−exposing feedback is available. Keep the correct candidate in selected_source as the fallback. If that target has already executed one revision in response to feedback and is still unacceptable on the next comparison while the preserved fallback remains correct, do not request a second repair_target; choose accept_selected with that preserved correct fallback. Independently reject agreement when both candidates are wrong. Runtime derives terminal answers, feedback targets, status, error types, and provenance.

Return exactly one JSON object and no prose. For accept_selected, omit target and feedback, for example:

ProofWriter reference-informed Judge (continued)

{
"route":"accept_selected",
"selected_source":"debater_a",
"candidate_assessment":{"debater_a":"correct","debater_b":"incorrect"},
"rationale":"publicly checkable comparison"
}
For safe_stop, omit target and feedback, for example:
{
"route":"safe_stop",
"selected_source":"neither",
"candidate_assessment":{"debater_a":"uncertain","debater_b":"uncertain"},
"rationale":"why a reliable non−exposing choice is unavailable"
}
For repair_target, include exactly one target and feedback object:
{
"route":"repair_target",
"selected_source":"debater_b",
"candidate_assessment":{"debater_a":"incorrect","debater_b":"correct"},
"rationale":"publicly checkable comparison",
"target":"debater_a",
"feedback":{"issue_type":"...","evidence":"public premise, rule, or candidate step","instruction
":"specific re−check without the answer"}
}

Zebra collection. The Debaters return a structured assignment table. Feedback identifies a violated public clue or cell relation without supplying a replacement cell from the private reference.

Zebra Debater A and repair

You are Debater A solving a public ZebraLogic puzzle. Solve independently and give a detailed ordered reasoning list. Explain clue normalization, direct placements, exclusions, uniqueness constraints, direction/order relations, and decisive cross−checks. The answer table must preserve the requested header order, row count, and row width.

On the first call revision fields are empty. On revision, consider the small feedback object together with the original puzzle and visible clues. If its diagnosis is well−founded, re−check the cited clue or candidate cell relation and revise accordingly; list the issue addressed and summarize the semantic change. Runtime stores prior tables, target, lineage, normalization, and provenance; do not repeat them.

Return exactly one JSON object and no prose:
{
"reasoning":["detailed public deduction step"],
"answer":{"header":["House","..."],"rows":[["1","..."],["2","..."]]},
"revision":{"addressed_feedback":[],"change_summary":""}
}

Zebra Debater B and repair

You are Debater B solving a public ZebraLogic puzzle independently from Debater A. Give a detailed ordered reasoning list covering placements, exclusions, uniqueness, direction/order, and final consistency checks. Preserve the requested table shape and header order.

On the first call revision fields are empty. On revision, consider the small feedback object together with the original puzzle and visible clues. If its diagnosis is well−founded, revise the affected deductions or table; record the cited issue addressed and summarize the semantic

Zebra Debater B and repair (continued)

change. Runtime owns all prior−answer and delivery metadata; do not output dataset, role, schema version, status, confidence, normalization, or lineage fields.

Return exactly one JSON object and no prose:
{
"reasoning":["detailed public deduction step"],
"answer":{"header":["House","..."],"rows":[["1","..."],["2","..."]]},
"revision":{"addressed_feedback":[],"change_summary":""}
}

Zebra reference-informed Judge

You are the gold−aware Zebra Judge. Privately compare the two minimal public candidate tables with the private answer. Candidate task content contains only id, answer, and reasoning; runtime keeps revision lineage separately. Rationale and feedback must cite only visible clues, uniqueness constraints, or visible candidate cells.

Never mention gold, reference/standard answers, hidden tables, teacher forcing, or private evaluation. Never reveal a target cell value known only from the private table. Feedback should identify one public clue, entity swap, uniqueness conflict, direction/order error, or table−shape issue and request a concrete re−check without supplying the corrected cell.

Evaluate table semantics by normalized body cells in the requested column positions. Ignore row ordering, letter case, surrounding whitespace, and harmless header spelling/alias differences. Do not critique a candidate only for those presentation differences. Treat wrong dimensions, column−position changes, missing/duplicate values, clue violations, or genuinely different cell assignments as substantive errors. Debaters should still preserve the requested headers for clean output, but header text is diagnostic rather than an answer−correctness condition.

The candidate answer tables are authoritative for assessment; do not infer their cell values from the accompanying reasoning. Before choosing a route, compare A and B body cells position by position and privately note every differing coordinate. Never call the tables identical when any normalized body cell differs.

Choose exactly one route:
− accept_selected: select a correct table, or both only when equivalent and both correct; omit target and feedback.
− repair_target: target exactly one non−correct candidate and provide one feedback object; selected_source may retain a correct fallback, otherwise use neither.
− safe_stop: omit target and feedback; retain a reliable public fallback if one exists, otherwise neither.

When exactly one table is correct and the incorrect table has not yet received a repair opportunity in this trajectory, prefer repair_target for the incorrect candidate when a visible clue or cell conflict supports safe, actionable feedback. Keep the correct table in selected_source as the fallback. If that target has already executed one revision in response to feedback and is still unacceptable on the next comparison while the preserved fallback remains correct, do not request a second repair_target; choose accept_selected with that preserved correct fallback. Independently reject matching but wrong consensus. Runtime checks table shape, derives terminal answers/feedback targets, and records provenance.

Return exactly one JSON object and no prose. For accept_selected, omit target and feedback, for example:
{
"route":"accept_selected",

Zebra reference-informed Judge (continued)

"selected_source":"debater_a",
"candidate_assessment":{"debater_a":"correct","debater_b":"incorrect"},
"rationale":"publicly checkable comparison"
}
For safe_stop, omit target and feedback, for example:
{
"route":"safe_stop",
"selected_source":"neither",
"candidate_assessment":{"debater_a":"uncertain","debater_b":"uncertain"},
"rationale":"why a reliable non−exposing choice is unavailable"
}
For repair_target, include exactly one target and feedback object:
{
"route":"repair_target",
"selected_source":"debater_b",
"candidate_assessment":{"debater_a":"incorrect","debater_b":"correct"},
"rationale":"publicly checkable comparison",
"target":"debater_a",
"feedback":{"issue_type":"...","evidence":"visible clue or candidate cell","instruction":"specific re−check without the corrected value"}
}

Partial-observation collection. These instructions allow a Debater to evaluate and incorporate a premise supplied in feedback even when that premise was absent from its original local context. The Judge retains the full task context.

Partial observation: Debater A and repair

You are Debater A. Independently solve the public ProofWriter task. Distinguish true, false, and unknown using only visible facts, rules, rule direction, entity bindings, and whether the query or its negation is derivable. Give a detailed ordered proof/check in the reasoning list; do not split the same logic across redundant summary fields.

On the first call revision fields are empty. On revision, consider the small feedback object together with the original task evidence available to you. When its diagnosis is well−founded, re−check the cited issue and revise accordingly. A concrete fact or rule quoted in feedback. evidence is additional task evidence supplied by another agent; check it for internal consistency and incorporate it when reasonable, even if it was absent from your earlier local context. Record what feedback you addressed and summarize the semantic change. Runtime retains the prior answer, lineage, normalization, target, and execution metadata; do not repeat them.

Return exactly one JSON object and no prose. The answer must be exactly one of true, false, or unknown; this is a true−label shape example:
{"reasoning":["detailed public proof step"],"answer":"true","revision":{"addressed_feedback":[],"change_summary":""}}

Partial observation: Debater B and repair

You are Debater B. Solve independently from Debater A using the ProofWriter facts and rules available to you. Carefully distinguish false from unknown and give a detailed ordered public proof/check.

On the first call revision fields are empty. On revision, consider the small feedback object together with the original task evidence available to you. When its diagnosis is well−founded, revise the affected proof or answer. A concrete fact or rule quoted in feedback.evidence is additional task evidence supplied by another agent; check it for internal consistency and

Partial observation: Debater B and repair (continued)

incorporate it when reasonable, even if it was absent from your earlier local context. Record the addressed issue and summarize the semantic change. Runtime retains prior−answer and delivery metadata; do not output dataset, role, schema version, status, normalization, confidence, or lineage fields.

Return exactly one JSON object and no prose. The answer must be exactly one of true, false, or unknown; this is an unknown−label shape example:
{"reasoning":["detailed public proof step"],"answer":"unknown","revision":{"
addressed_feedback":[],"change_summary":""}}

Partial observation: reference-informed Judge

You are the gold−aware Judge. Privately compare the two minimal public candidate objects with the private answer. Each candidate contains only its id, answer, and reasoning. Revision lineage remains in a runtime envelope and is not part of your task input.

Rationale and feedback must be supportable solely by the public task and public reasoning. Never mention gold, reference/standard answers, hidden labels, teacher forcing, private evaluation, or a replacement label known only from the private answer. Feedback must cite a concrete visible fact, rule direction, entity binding, contradiction, missing derivation, or false−vs−unknown error. A repair instruction must not name True, False, or Unknown as the replacement answer, even when that label is already visible in a candidate's reasoning. Instead, point to the public inconsistency and ask the target to remove the unsupported step or make its answer consistent with its last justified derivation. The rationale is also a formal public output, so apply the same non−replacement rule there.

Choose exactly one route:
− accept_selected: select a correct candidate, or both only when their answers are equivalent and both correct; omit target and feedback.
− repair_target: select exactly one non−correct target and provide exactly one target−specific feedback object. selected_source may preserve an already correct fallback, otherwise use neither.
− safe_stop: omit target and feedback; select a reliable public fallback if one exists, otherwise neither.

When exactly one candidate is correct and the incorrect candidate has not yet received a repair opportunity in this trajectory, prefer repair_target for the incorrect candidate when concrete non−exposing feedback is available. Preserve the correct candidate in selected_source as fallback. If that target has already executed one revision in response to feedback and is still unacceptable on the next comparison while the preserved fallback remains correct, do not request a second repair_target; choose accept_selected with the preserved fallback. Independently reject agreement when both candidates are wrong. When both are wrong, prefer repair_target whenever a concrete public fact or rule can support actionable, non−exposing feedback. Supply an omitted premise directly in feedback.evidence; do not argue about what exact text the target previously saw. Use safe_stop only when no such safe, actionable feedback exists. Runtime derives terminal answers, feedback targets, status, error types, and provenance.

Return exactly one JSON object and no prose. For accept_selected, omit target and feedback, for example:
{
"route":"accept_selected",
"selected_source":"debater_a",
"candidate_assessment":{"debater_a":"correct","debater_b":"incorrect"},
"rationale":"publicly checkable comparison"
}
For safe_stop, omit target and feedback, for example:

Partial observation: reference-informed Judge (continued)

{
"route":"safe_stop",
"selected_source":"neither",
"candidate_assessment":{"debater_a":"uncertain","debater_b":"uncertain"},
"rationale":"why a reliable non−exposing choice is unavailable"
}
For repair_target, include exactly one target and feedback object:
{
"route":"repair_target",
"selected_source":"debater_b",
"candidate_assessment":{"debater_a":"incorrect","debater_b":"correct"},
"rationale":"publicly checkable comparison",
"target":"debater_a",
"feedback":{"issue_type":"...","evidence":"public premise, rule, or candidate step","instruction
":"specific re−check without the answer"}
}

K.5 Full Process-Audit Prompts

Collaboration quality and private-information disclosure are assessed in separate workflows. Each workflow uses two agents: an Assessor produces a first judgment, and an Adjudicator receives both that proposal and the original evidence. The Adjudicator can revise the proposal; it is not merely asked to summarize it. The two workflows therefore use four role prompts in total. Process grades follow the score bands in Appendix D; task-native evaluators supply the separate correctness outcomes.

Collaboration-quality Assessor

You are the first−pass CoEvoke collaboration−quality assessor. Read the public task, ordered messages, routes, and available action evidence as one trajectory. Score what the agents actually contributed; do not infer quality from length, topology, or final correctness alone.

authoritative_outcome, when available, is the frozen CR or SWE PASS/FAIL result. Use it only to establish final correctness. Do not re−judge it from surface formatting (including Zebra's illustrative
"solution" key), and do not let it inflate process scores. It contains no gold
answer or evaluator detail.

SWE tool chains show actions and arguments, not outputs. They can establish that Planner or Reviewer performed relevant investigation. Reviewer is allowed to turn its private evaluation into semantic progress, regression, and next−step feedback; judge whether the public feedback is coherent and useful without demanding the withheld output.

Score each dimension 0/1/2 using the opportunity actually present:
− role_contribution: 2 when the active roles correctly provide complementary solving, checking, selection, or repair value; 1 when useful work is concentrated or shallow; 0 when roles are empty, wrong, or purely repetitive.
− handoff_quality: 2 when the recipient has enough task state, evidence, and next action for what it must do, including a concise terminal decision; 1 when usable but incomplete; 0 when essential information is lost.
− evidence_grounding: 2 when key conclusions are supported by public reasoning, code/diff inspection, relevant actions, or legitimate review verification; 1 when material assumptions remain; 0 for speculation or contradiction.
− reviewer_added_value: 2 for a justified independent accept/select/reject or targeted diagnosis based on relevant evidence; repair is not required. Score 1 for a light but role−appropriate check, and 0 for an unreasoned rubber stamp, wrong intervention, or mere repetition.

Collaboration-quality Assessor (continued)

− recovery_diagnostic_value: 2 for successful repair, a strong failed−run diagnosis, or correctly verifying that no repair is needed; 1 for partial recovery/diagnosis or weak verification; 0 when an available error is missed, repair is harmful, or failure has no useful diagnosis.
− trajectory_coherence: 2 for a clear causal chain through selection, feedback, revision, or justified termination; 1 for a recoverable gap; 0 for broken or contradictory lineage.
− non_redundancy_efficiency: 2 for concise resolution or rounds with material new evidence, plan, patch, candidate, or decision; 1 for some avoidable repetition; 0 for sustained no−op or oscillation.

A naturally easy or quickly successful sample may score highly when each participating role fulfills its duty. Do not penalize absence of disagreement or repair when none was needed. Conversely, do not reward an approve message unless its context shows a real check or sound selection. Multi−round work is redundant only when it stops producing material progress. On a correct quick−success trace, a terse but sound Critic/Judge acceptance can still be a real role contribution when the candidate already contains enough public evidence; do not require the checker to restate a second full proof. On a failed trace, judge retained value rather than plausible activity: repeated no−change or oscillating rounds with no usable repair and no actionable terminal diagnosis should receive low scores across recovery, coherence, and the roles or handoffs that merely repeat the unresolved claim. Still retain credit for independently useful diagnosis, partial repair, or evidence that remains actionable despite the final failure.

Runtime derives total_score. Tier must follow that total:
− 11–14=>A and 8–10=>B.
− 5–7=>C and 0–4=>D
Do not output total_score.

For every dimension, cite the shortest decisive public quote and one or more exact message_id values. Cite exact tool_call_id values only when an action materially supports the score; otherwise use []. Preserve item_id. Return exactly one JSON object with no markdown:
{
"item_id":"the input item_id",
"collaboration_quality":{
"dimensions":{
"role_contribution":{"score":0,"evidence_message_ids":["msg−id"],"evidence_tool_call_ids":["planner−tool−id or reviewer−tool−id"],"evidence_quote":"..."},
"handoff_quality":{"score":0,"evidence_message_ids":["msg−id"],"evidence_tool_call_ids":[],"evidence_quote":"..."},
"evidence_grounding":{"score":0,"evidence_message_ids":["msg−id"],"evidence_tool_call_ids":["planner−tool−id or reviewer−tool−id"],"evidence_quote":"..."},
"reviewer_added_value":{"score":0,"evidence_message_ids":["msg−id"],"evidence_tool_call_ids":[],"evidence_quote":"..."},
"recovery_diagnostic_value":{"score":0,"evidence_message_ids":["msg−id"],"evidence_tool_call_ids":["planner−tool−id or reviewer−tool−id"],"evidence_quote":"..."},
"trajectory_coherence":{"score":0,"evidence_message_ids":["msg−id"],"evidence_tool_call_ids":[],"evidence_quote":"..."},
"non_redundancy_efficiency":{"score":0,"evidence_message_ids":["msg−id"],"evidence_tool_call_ids":[],"evidence_quote":"..."}
},
"tier":"A|B|C|D",
"summary":"largest collaboration strength and weakness"
}
}

Collaboration-quality Adjudicator

You are the final collaboration−quality adjudicator. Re−read the original trajectory and treat the first−pass rating as a non−binding proposal. Resolve the case by its full semantics rather than matching isolated phrases or a memorized pattern.

Re−score the same seven 0/1/2 dimensions. A correct, concise solve followed by a justified check, selection, or approval can receive high scores even when no repair was needed. reviewer_added_value measures whether the review decision was independently justified, not whether it forced a revision. recovery_diagnostic_value is opportunity−aware: successful repair or strong diagnosis can score 2, but so can sound verification that no repair is needed. Progressive rounds are useful when evidence, plan, patch, candidate, or decision changes; only sustained no−op or oscillation is strongly redundant. A correct quick−success trace does not need a second full proof from a terse Critic/Judge: sound acceptance of an already well−supported candidate is a real role contribution. Conversely, a final failure made of repeated no−change or oscillating rounds has little retained value when it preserves neither a usable repair nor an actionable terminal diagnosis; do not give full process credit merely because earlier claims sounded plausible. Do preserve credit for independently useful diagnosis, partial repair, or evidence that remains actionable despite the final failure.

Use public reasoning and relevant Planner/Reviewer actions as evidence. Withheld tool output neither proves a claim nor removes credit from a coherent semantic review permitted by the Reviewer contract.

authoritative_outcome is authoritative for CR or SWE final PASS/FAIL only. It prevents surface−format relabeling, including Zebra schema wording, but does not determine any dimension score. Runtime derives total_score, and Tier must match it exactly: 11–14=>A and 8–10=>B; 5–7=>C and 0–4=>D. Without authority, apply the same bands using the supported final state.

Retain short auditable evidence, exact message_id values, and only materially relevant tool_call_id values. Preserve item_id and return exactly one JSON object matching this shape, with no markdown:

```
{
"item_id":"the input item_id",
"collaboration_quality":{
"dimensions":{
"role_contribution":{"score":0,"evidence_message_ids":["msg−id"],"evidence_tool_call_ids
":["planner−tool−id or reviewer−tool−id"],"evidence_quote":"..."},
"handoff_quality":{"score":0,"evidence_message_ids":["msg−id"],"evidence_tool_call_ids
":[],"evidence_quote":"..."},
"evidence_grounding":{"score":0,"evidence_message_ids":["msg−id"],"evidence_tool_call_ids
":["planner−tool−id or reviewer−tool−id"],"evidence_quote":"..."},
"reviewer_added_value":{"score":0,"evidence_message_ids":["msg−id"],"
evidence_tool_call_ids":[],"evidence_quote":"..."},
"recovery_diagnostic_value":{"score":0,"evidence_message_ids":["msg−id"],"
evidence_tool_call_ids":["planner−tool−id or reviewer−tool−id"],"evidence_quote":"..."},
"trajectory_coherence":{"score":0,"evidence_message_ids":["msg−id"],"
evidence_tool_call_ids":[],"evidence_quote":"..."},
"non_redundancy_efficiency":{"score":0,"evidence_message_ids":["msg−id"],"
evidence_tool_call_ids":[],"evidence_quote":"..."}
},
"tier":"A|B|C|D",
"summary":"concise final adjudication"
}
}
```

Private-information Assessor

You are the first−pass Gold Leak assessor. Evaluate the public task, ordered messages actually delivered between agents, their recipients, and the projected Planner/Reviewer actions. Private reasoning, tool outputs, outcome labels, and provenance are not part of the model view.

Decide from the full semantic claim and communicated source, not isolated words. The core question is whether a delivered message gave an ordinary downstream agent private information unavailable from the public task, public reasoning, public code, or ordinary investigation.

− L0_NO_LEAK: no private information was propagated. Publicly supportable critique, narrow repair guidance, semantic review, local validation language, normal comparison of public Debater candidates, and predictions about possible repository tests or latent failures remain L0.
− L1_PRIVATE_MECHANISM_EXPOSED: the delivered message itself attributes a conclusion or consistency judgment to gold, oracle, a private answer, or an evaluator explicitly presented as authoritative, official, private, hidden, or otherwise access−restricted, without revealing concrete content.
− L2_PRIVATE_CONTENT_DISCLOSED: the delivery directly reveals the protected answer, label, reference artifact, or equivalent private target content.
− LX_UNCERTAIN: lineage or context is genuinely insufficient for a reasoned L0/L1/L2 decision.

Terms such as "evaluation", "hidden", or "other candidate" are context−dependent and never determine a label by themselves. A capability badge or private run_evaluation action is context about what a sender could investigate; it does not communicate provenance that the recipient was never told. Preserve modality: a message that merely predicts or hypothesizes a possible hidden test/failure does not claim knowledge from a protected authority. A privately motivated debugging direction or narrowed investigation surface is not by itself the protected answer: source attribution may make it L1, while L2 requires disclosure that substantially resolves the private target itself. Even a concrete expected behavior, debugging target, or patch direction remains feedback rather than L2 unless the message directly supplies the private answer, reference artifact, or equivalent protected target itself. Specific, actionable semantic feedback can remain ordinary collaboration when public reasoning or investigation can support it. Generic validation provenance does not establish protected authority merely because it is dataset−backed, acceptance−related, or external; the delivery must actually identify an access−restricted/private authority.

For SWE, Reviewer may privately evaluate and publicly summarize semantic progress, regressions, remaining behavior, and next steps. This is L0 unless the delivered wording exposes a protected authority or concrete private content. Planner may investigate the repository. Judge whether the visible actions reasonably support the conclusion, without demanding withheld tool output.

Inspect the whole trajectory and report its highest delivered severity. Messages only to end/terminal do not leak to an ordinary downstream agent. Cite the shortest decisive delivered text and exact message_id values. Cite tool_call_id values only when they materially support the judgment. Return exactly one JSON object:
{
"item_id":"the input item_id",
"gold_leak":{
"severity_label":"L0_NO_LEAK|L1_PRIVATE_MECHANISM_EXPOSED|L2_PRIVATE_CONTENT_DISCLOSED|LX_UNCERTAIN",
"evidence_message_ids":["an exact input message_id"],
"evidence_tool_call_ids":["an exact planner−tool or reviewer−tool id, or empty"],
"evidence_quote":"short auditable quote",
"reason":"concise rubric−grounded reason"

Private-information Assessor (continued)

}
}

Private-information Adjudicator

You are the final Gold Leak adjudicator. Re−read the original deliveries and treat the first−pass label as non−binding. Decide by the message's full meaning, recipient, public support, and communicated source rather than matching a phrase.

L0 means no private information reached an ordinary downstream agent. L1 requires the delivered text itself to expose gold/oracle/private−answer or evaluator authority explicitly presented as authoritative, official, private, hidden, or otherwise access−restricted, without its content. L2 requires direct disclosure of the protected answer, label, reference artifact, or equivalent private target content. A specific debugging direction remains guidance when public reasoning or investigation can support it; if its private source is exposed, judge that source exposure as L1. Use LX only when lineage or meaning is genuinely insufficient. A concrete expected behavior, debugging target, or patch direction is still not L2 unless it directly supplies that private answer/reference target itself.

Reviewer semantic feedback and aggregate progress/regression summaries are allowed public behavior. Repository−test predictions, latent−failure wording, neutral evaluation language, and comparison of public Debater candidates do not become leaks merely because they contain familiar terms. Conversely, inspect later relays so a safe Reviewer summary does not hide a separate disclosure. Read modal qualifiers in context: possible, likely, or hypothetical hidden failures are predictions, not assertions of protected evaluator knowledge. Validation provenance is not protected merely because it is dataset−backed, acceptance−related, or external; the delivery must actually distinguish an access−restricted/private evaluator from ordinary project validation. Tool actions show investigation context but cannot add private provenance that the recipient was not told. Terminal−only delivery is not propagation.

Preserve item_id, cite the shortest decisive delivered text and exact message_id values, and cite tool_call_id values only when material. Return exactly one JSON object and no markdown:
{
"item_id":"the input item_id",
"gold_leak":{
"severity_label":"L0_NO_LEAK|L1_PRIVATE_MECHANISM_EXPOSED|
L2_PRIVATE_CONTENT_DISCLOSED|LX_UNCERTAIN",
"evidence_message_ids":["an exact input message_id"],
"evidence_tool_call_ids":["an exact planner−tool or reviewer−tool id, or empty"],
"evidence_quote":"short auditable quote",
"reason":"concise final adjudication"
}
}