EXECUTION-GROUNDED CREDIT ASSIGNMENT FOR SELF-EVOLVING AGENTS

Anonymous authors
Paper under double-blind review

ABSTRACT

Recent research on the self-improvement of language model agents has shown that iterative refinement guided by execution feedback is effective. On long-horizon tasks, however, credit assignment remains difficult because task outcomes reflect many interdependent actions. After a change, an agent may succeed without exhibiting the behavior the change was intended to produce, or fail despite exhibiting it. We propose an evolution method that uses execution-grounded component-level credit assignment to guide changes to instructions, tool code, and skills. Context-Anchored Gene Mutation derives mutation intents from outcome-associated behavioral patterns in the parent agent's execution histories and combines them into plans constrained by evolution context accumulated over earlier rounds. Applying these plans to the parent generates candidate agents, with each change linked to its intent. To explore diverse mutation candidates while limiting costly full task evaluations, we use simulation-based candidate pruning. It simulates the parent and each candidate for a bounded number of steps from recorded decision points relevant to each intent, under matched language-model request conditions. Candidates that express the intended behavior more strongly than the parent in these simulations are prioritized for full task evaluation. After evaluation, Gene Distillation combines each change's source pattern, intent, observed behavior, and task outcome to update its credit. Stronger evidence of expression increases the weight of both positive and negative outcomes. Credit accumulates in the evolution context to guide subsequent mutations. We evaluate the method on SEAL-0, which requires iterative retrieval and verification of potentially conflicting web evidence, and on EnterpriseOps-Gym, which tests stateful planning and tool use in enterprise workflows with interdependent steps. After evolution on five SEAL-0 tasks, accuracy on all 111 tasks, including the development tasks, rises from 41.44% to 63.36%; GEPA (adapted) reaches 50.45% on the same 111 tasks. After evolution on eight EnterpriseOps-Gym tasks, the evolved agent is evaluated on 642 tasks, achieving a task pass rate of 35.67% and a verifier-level pass rate of 68.9%, compared with 26.95% and 59.3%, respectively, for the initial agent. These results suggest that credit assignment for long-horizon agent evolution can be grounded in observed behavior as well as task outcomes.

1 INTRODUCTION

Language-model agents can improve through iterative edits to instructions, tool code, and skills (Shang et al., 2025; Zhang et al., 2025; Yang et al., 2026). On long-horizon tasks, deciding which edits to retain requires understanding their effects across interdependent actions. Search agents repeatedly retrieve and verify evidence, while enterprise agents coordinate state changes across multiple operations (Chen et al., 2025; Malay et al., 2026). Full-task evaluation limits the number of candidates that can be tested (Borthwick et al., 2026). We address component-level credit assignment and control of mutation diversity and search direction under this evaluation constraint.

Consider adding a verification instruction after an agent changes a target without checking its state. A later task may succeed without the check being performed, or fail after it is correctly executed. Crediting or rejecting every edit based only on that outcome can misdirect later mutations. Deciding which components to preserve or revise requires both the task result and observations of how

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.

the edited agent behaves. Even step-level credit signals can disagree with contributions measured through executed continuations (Zhang, 2026).

Existing methods use execution histories to guide edits. GEPA reflects on trajectories and feedback to improve prompts, while EVOTOOL localizes failure to a tool-policy module before mutation (Agrawal et al., 2026; Yang et al., 2026). We add a behavioral check between mutation proposal and credit assignment: whether the edited agent expresses the intended behavior. Mutation intents and targets are registered before editing, so observations can be linked to the changes that motivated them.

Our method organizes evolution around *Evolution Context*: records connecting behavior patterns, mutation intents, realized edits, expression observations, and outcomes (Figure 1). We treat editable components as genes. Behavior-pattern analysis and inherited context generate an intent pool whose combinations produce candidate edit plans. Paired replay resumes parent and candidate executions from the same history checkpoint. Simulation-based candidate pruning uses these observations to support broader exploration before expensive task evaluation. Gene Distillation combines expression observations and task outcomes to update component credit and context for later mutations.

We evolve agents from small development subsets and evaluate on full target sets, including those subsets. On SEAL-0, five development tasks improve accuracy from 41.44% to 63.36% across 111 items, compared with 50.45% for GEPA (adapted). On EnterpriseOps-Gym, eight development tasks improve task pass rate from 26.95% to 35.67% and verifier-level pass rate from 59.3% to 68.9% across 642 tasks. Our contributions are:

1. Execution-grounded component-level credit assignment linking mutation intents and realized edits to behavioral expression and task outcomes.

2. Evolution Context-based control of mutation diversity and search direction: accumulated records guide preservation, avoidance, redesign, and exploration, while simulation-based candidate pruning supports diverse candidates within a limited task-evaluation budget.

We will release our implementation and experimental configurations.

2 Problem Statement

A candidate can change several components yet receive only one task score. That score alone does not reveal which edits produced their intended behavior or how to reuse each change in later mutations. We seek better-performing agents under a limited evaluation budget while linking component edits, observed behavior, and task outcomes.

An agent configuration A comprises editable instructions, tools, and control policies. Each round generates up to N candidates and evaluates at most M after pruning. Candidate A_i has a complete edit Δ_i assembled from multiple mutation intents. For intent z_{ij}, registration links the target component or group S_{ij} to its edit portion Δ_{ij}, trigger q_{ij}, and behavior predicate b_{ij}. A realized edit is a concrete configuration change; it does not by itself establish behavioral expression. When edits must work together, we use an explicit component group without assigning independent credit to its members.

Replay observations distinguish EXPRESSED, NOTEXPRESSED, and UNOBSERVED; the last denotes unavailable usable observations rather than observed absence. Task evaluation separately returns score y_i. Comparison with the parent's score under the same criterion gives improvement, regression, or no material change. These directions describe change relative to the parent, not absolute task success or failure. Expression and outcome remain separate observations when interpreting a component's credit.

Component credit combines this parent-relative outcome with expression observations. The resulting records support component-importance updates and guide which components later mutations preserve, which changes they avoid or revise, and which directions they explore.

3 METHOD

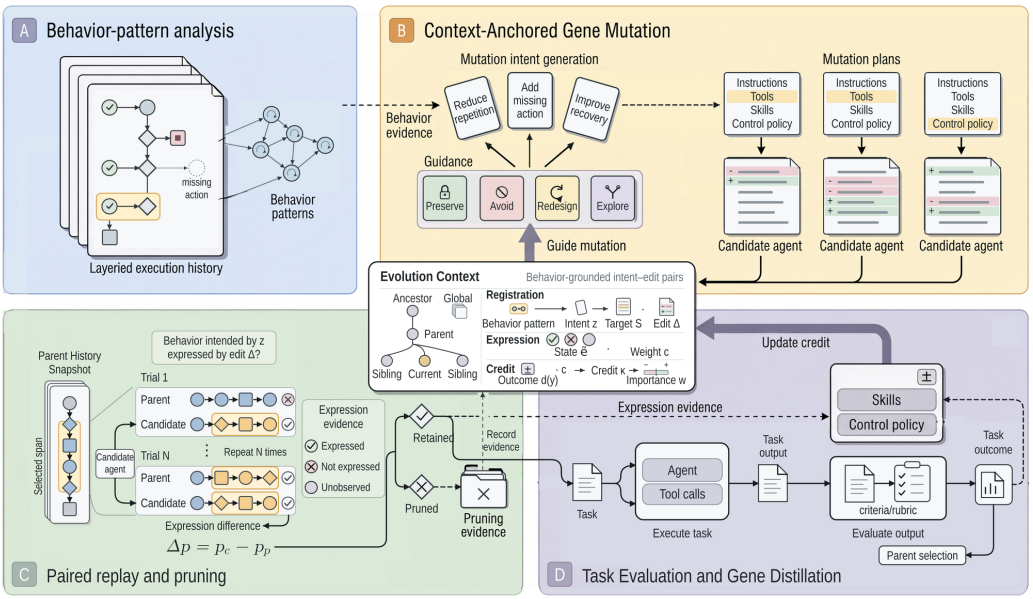


Figure 1: Overview of execution-grounded agent evolution.

We generate diverse candidate agents from behavior patterns and Evolution Context; simulation-based candidate pruning then screens them before task evaluation. Replay observations and task outcomes support component credit and context updates for the next round.

3.1 BEHAVIOR-PATTERN ANALYSIS AND EVOLUTION CONTEXT

Each round, we analyze the parent's execution histories and task results for actions, ordering, repetition, and missing actions. We register behavior patterns with their trace locations and observed associations with success or failure. These patterns and Evolution Context guide the generation of a mutation-intent pool.

Evolution Context. Each Evolution Context record links a mutation intent and target component or group to its motivating behavior pattern and its association with success or failure in task evaluations. We register the intent–target link during planning and add the realized edit after applying the plan. Replay adds expression observations, and task evaluation adds the candidate outcome. An LLM assigns and revises the record's importance using this information.

Evolution Context storage. We store records with agents in the evolution tree and in a global archive. Lineage retrieval follows the current parent's lineage. Sibling retrieval covers candidates generated from the current parent and evaluated in the same round, including those not selected as the next parent. The global archive makes records available across lineages.

3.2 CONTEXT-ANCHORED GENE MUTATION

The evolution agent, an LLM-based agent that plans and applies mutations, generates mutation intents from behavior-pattern analysis and Evolution Context (Algorithm 1, lines 4–12). For example, repeated calls after a tool failure in unsuccessful executions can motivate an alternative recovery action. The associated trace location identifies where to observe it.

Retrieved records constrain editing through four anchors: *preserve* protects specified agent components, *avoid* discourages repeating unsuccessful changes, *redesign* seeks another implementation,

and *explore* reserves room for directions with little evidence. Stored importance sets their priorities. When an expressed change coincides with regression, the evolution agent can avoid repeating it; when an intended behavior never appears, it can try a different implementation.

The evolution agent forms an intent pool $\{(z_j, q_j, b_j)\}_j$, where z_j identifies an intent, q_j its trigger, and b_j its behavior predicate. Different combinations from this pool produce candidate edit plans. We clone the parent, apply each plan, and record the realized edit under its intent–target pair. Intent identifiers remain linked to their pool entries across plans.

3.3 SIMULATION-BASED CANDIDATE PRUNING

Simulation-based candidate pruning compares behavioral expression through paired replay. It resumes the parent and each candidate from the same history checkpoint at a decision point relevant to the mutation intent, then runs both for a bounded number of steps under matched request conditions. Guardrails first check protected components, loading, tool bindings, and task-answer leakage. Candidates receive bounded correction attempts and are excluded if they fail rechecking.

Repeated replay. We retrieve a history checkpoint σ_{ij} at the decision point identified by a behavior pattern and trigger q_{ij}. It contains the parent's interaction history, model request, tool schemas, and generation parameters. We load the history, recreate the execution environment, inject the candidate configuration, and resume from that point. We observe a bounded segment K for the intended behavior and repeat the execution k times, restarting from the checkpoint for each repetition because behavior can vary under the same conditions. The unchanged parent runs from the same checkpoint for comparison (Algorithm 1, lines 13–19).

Expression and pruning. Candidate and parent penetrance summarize full and partial expression across valid repeats; expression lift is their difference. Expression stability measures consistency across repeated action forms. The pruning score averages candidate–parent differences in stability-weighted penetrance. EXPRESSED requires candidate penetrance and expression lift to reach 0.2; other valid observations are NOTEXPRESSED, including cases where the behavior appears without sufficient lift over the parent. UNOBSERVED denotes unavailable valid matched observations, for example when the trigger is absent or observations cannot be matched to the target. An LLM selector uses the pruning scores to choose up to M candidates for task evaluation, limiting duplicate mutation directions and reserving slots for underrepresented directions. Appendix G specifies the scoring and selection rules.

3.4 GENE DISTILLATION

Gene Distillation combines mutation intents, realized edits, replay observations, and candidate task outcomes to assign component credit (Algorithm 1, lines 20–26). We weight the parent-relative outcome direction by expression lift, stability, and trace-location support. Eligible non-expressed targets use a weak expression weight: candidate success alone does not establish an edit's effectiveness. Credit requires a registered target, reconciled edits, matched replay observations, and a supported trace location. We normalize credit within each candidate to distribute a fixed budget among eligible targets. Appendix G specifies eligibility and weights.

Parent selection spans the evolution tree but requires evaluation on the same development set under the same scoring criterion. Tied task scores need not retain the parent: selection also accounts for prior lineage expansion, penalizing repeatedly selected parents.

An LLM uses this credit, the underlying observations, and existing records to update Evolution Context importance and mutation guidance. Importance is not a running sum of credit. Consistency checks enforce the sign and minimum magnitude for strongly supported expressed improvements or regressions. Pruned candidates receive context updates from replay observations and pruning reasons, without task-outcome credit (UPDATEPRUNEDCONTEXT in Algorithm 1). Replay and pruning records remain linked to the context.

Retrieved records constrain the targets and edits of later mutation plans. Positive records support preserving or reusing components; negative records support avoiding or redesigning changes.

Algorithm 1 Execution-Grounded Agent Evolution

Require: Parent agent A, task score y, execution history H, development tasks D
Require: Evolution Context $\mathcal{C}$, rounds R, candidates N, evaluation slots M, replay repeats k
Ensure: Selected agent and updated Evolution Context

for $r = 1, \ldots, R$ **do**
▷ **(a) Behavior-pattern analysis**
  $P \leftarrow \textsc{ExtractBehaviorPatterns}(H)$
  Register patterns with trace locations and associations with task success or failure
▷ **(b) Context-Anchored Gene Mutation**
  $Z \leftarrow \textsc{GenerateIntentPool}(P, \mathcal{C})$
  $\Pi \leftarrow \textsc{ComposeEditPlans}(A, Z, N)$
  Register each intent z_{ij} with its target component S_{ij} for each plan
  for each plan $\pi_i \in \Pi$ **do**
    $A_i \leftarrow \textsc{Clone}(A)$; $\Delta_i \leftarrow \textsc{ApplyEdits}(A_i, \pi_i)$
    Match Δ_{ij} to Δ_i; record (z_{ij}, Δ_{ij}) under target S_{ij}
    Check protected components, loading, tool bindings, and task-answer leakage
    Allow bounded repair; recheck and reject candidates that still fail
  $C_r \leftarrow$ candidates passing the guardrails
▷ **(c) Simulation-based candidate pruning**
  for each candidate $A_i \in C_r$ **do**
    Load parent execution history from intent-relevant history checkpoints σ_{ij} in H
    Inject the candidate configuration into the loaded replay context
    Replay bounded parent and candidate segments k times from each checkpoint
    Observe expression states and penetrance; compute the pruning score S_i^{prune}
  $E_r \leftarrow \textsc{Prune}(C_r, \text{expression evidence}, \{S_i^{\text{prune}}\}, M, \text{pruning constraints})$
  $\textsc{UpdatePrunedContext}(\mathcal{C}, C_r \setminus E_r, \text{expression evidence}, \text{pruning reasons})$
▷ **(d) Task evaluation and Gene Distillation**
  for each surviving candidate $A_i \in E_r$ **do**
    $(y_i, H_i) \leftarrow \textsc{Evaluate}(A_i, D)$
  $\mathcal{U} \leftarrow$ agents in the evolution tree evaluated on D under the same criterion
  $A^+ \leftarrow \textsc{SelectOrRetainParent}(A, y, \mathcal{U})$
  Link P, mutation intents, realized edits, expression evidence, and outcomes
  Compute expression-weighted component credit; normalize within each candidate
  Update Evolution Context importance using an LLM with credit and supporting evidence
  $A \leftarrow A^+$; $(y, H) \leftarrow$ the selected parent's evaluation and history
  if the termination criterion is met **then**
    break
  end if
return $(A, \mathcal{C})$

Replay compares the unchanged parent and edited candidate under matched request conditions. Outcome-linked credit uses the evaluated candidate's result relative to its parent.

4 Evaluation

4.1 Experimental setup

We evolve agents from small development subsets and evaluate on full sets that include those tasks (Table 1). Initial Agent denotes the unevolved configuration. Each condition uses one evolution run. We evaluate each selected agent in three separate full-set runs, completed one after another, and report the mean.

Benchmarks and scoring. SEAL-0, the core subset of SealQA, requires iterative retrieval and verification of potentially conflicting or noisy web evidence (Pham et al., 2026). For GEPA (adapted), feedback tasks support update proposals and search-validation tasks support candidate selection. EnterpriseOps-Gym tests stateful planning and tool use across enterprise workflows (Malay et al.,

Table 1: Experimental setup.

Setting	SEAL-0	EnterpriseOps-Gym
Development set	Our method: 5 tasks; GEPA (adapted): 5 feedback + 5 search-validation tasks	Our method: 8 tasks
Final evaluation set	111 items	642 tasks; 7 excluded (Appendix A.3)
Simulation candidates per round	Up to 16	Up to 9
Task-evaluated candidates per round	3	
Candidate evaluation	3 executions per task in one batch; mean score	
Histories for analysis	All 3 repeats of every evaluated task	
Final evaluation	3 separate full-set runs; mean score	
Performance metrics	Accuracy	Task pass rate; verifier-level pass rate

2026). A task passes when all its verification checks pass; verifier-level pass rate averages the fraction of checks passed within each task across evaluated tasks.

Search budgets. Candidates without replayable items reduce the simulation count. EnterpriseOps-Gym search is limited to seven rounds; GEPA (adapted) evaluates 21 candidate occurrences. Search counts include each candidate's repeated task executions as one evaluation unit.

Shared configuration and models. Our method and GEPA (adapted) share the initial SEAL-0 agent, editable instructions, tool code and skills, execution limits, tool environment, and scorer. EnterpriseOps-Gym allows the same component categories. Execution, reflection, and mutation all use Qwen3.5-122B-A10B-FP8 in the main comparison and Qwen3.8-Flash-Next-FP8 in the context ablation and transfer experiments. Model serving and initial agents are documented in Appendices A and B.

GEPA adaptation. Our CrewAI adapter permits GEPA to edit the same components and partitions long histories for reflection. It retains GEPA's search procedure without paired replay or Gene Distillation (Appendix F). Mutation-direction analysis uses the post-hoc categories and counting rules in Appendix E.4.

4.2 Final performance comparison

SEAL-0. After seven rounds with Qwen3.5, our method exceeds Initial Agent by 21.92 percentage points and GEPA (adapted) by 12.91 points (Table 2). Excluding the ten distinct development items used by either method leaves 101 items; the advantage over GEPA remains 13.53 points. Counting an item as solved when an agent answers it correctly in at least two of three runs, our agent solves 20 items that GEPA does not, against eight in the reverse direction.

EnterpriseOps-Gym. Evolution on eight development tasks improves full-set task pass rate by 8.72 percentage points and verifier-level pass rate by 9.6 points (Table 3).

4.3 Search trajectories and evaluation cost

Figure 2 compares Qwen3.5 search from the same initial agent. Each trajectory groups 21 evaluated candidates into seven groups of three, scored on the same five items: our development set and GEPA's feedback set. Evaluating candidate agents on development tasks accounts for most of the cost of evolution, especially for long-horizon tasks. We therefore plot cumulative task equivalents, including replay overhead. These coordinates are not complete end-to-end costs; Appendix F gives the accounting.

Table 2: SEAL-0 accuracy (%), averaged over three evaluations. Full-set rows include all 111 items; the 101-item row excludes both methods' development items. Our 5 development items are GEPA's 5 feedback items; GEPA also uses 5 distinct search-validation items. Excluding their union removes 10 items. Initial Agent uses no development tasks; "—" in a score cell denotes an experiment not performed.

Model	Initial Agent	GEPA (adapted)	Our method
Development tasks	—	5 + 5	5
Qwen3.5-122B-A10B-FP8	41.44	50.45	63.36
101 held-out items	42.57	50.17	63.70
Qwen3.8-Flash-Next-FP8	70.57	—	81.68

Table 3: EnterpriseOps-Gym results on 642 tasks, averaged over three evaluations.

Agent	Development tasks	Task pass rate	Verifier pass rate
Initial Agent	—	26.95%	59.3%
Our method	8	35.67%	68.9%

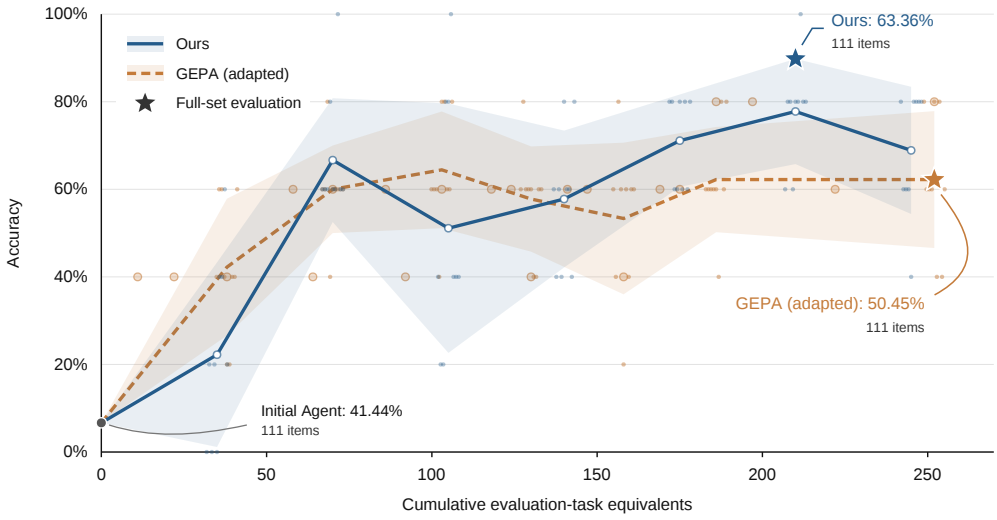


Figure 2: SEAL-0 search trajectories. Curves and bands show the mean and sample standard deviation across three candidates × three repeats per group; faint points show individual scores. Both curves start at 6.67% development accuracy. Initial Agent and star annotations give full-set accuracy (111 items).

Paired-replay cost. Paired replay averages 0.503M tokens per candidate across 106 entries. Task evaluation averages 0.402M tokens per execution across 60 executions from four selected agents, or 6.029M tokens for a candidate's five tasks and three repeats. Replay therefore adds 8.34% of one candidate's development-evaluation cost on this basis.

4.4 Effect of simulation-based candidate pruning

On EnterpriseOps-Gym, direct candidate evaluation and simulation-based candidate pruning start from the same initial agent and evaluate three candidates on eight development tasks per round for seven rounds. Direct candidate evaluation generates and evaluates three candidates per round. Simulation-based candidate pruning generates nine with diversity guidance and uses paired replay to select three for task evaluation. The comparison tests candidate generation and pruning jointly.

Simulation-based candidate pruning improves development pass rate from 41.67% to 50.00%, full-set pass rate from 32.55% to 35.67%, and verifier-level pass rate from 66.2% to 68.9% over direct candidate evaluation, with the same number of full task evaluations. Figure 3 shows each strategy category's share of proposed plans by round. The heatmaps show concentration on delegation and handoff under direct candidate evaluation, and broader coverage of verification, recovery, and tool interaction under simulation-based candidate pruning. The Round 5 plan illustrates how stored context informs these new directions.

Using unsuccessful mutations to redirect exploration. In EnterpriseOps, most archive entries have negative importance by Round 4. The selected plan in Round 5 consults these records and avoids repeating stronger post-operation verification instructions and runtime hooks associated with unsuccessful attempts. It instead introduces a read-only information-gathering stage before the main agent, while preserving instructions with positive importance. This example shows how the evolution agent uses records of unsuccessful attempts to avoid repeating the same changes and explore alternative mutation directions. Appendix E.3 gives signed counts and plan evidence.

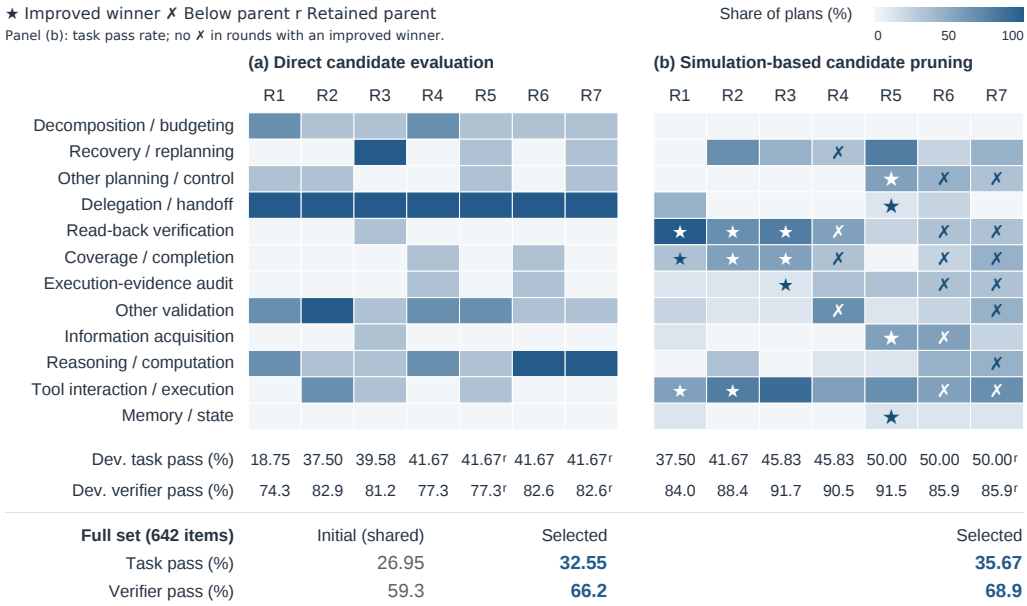


Figure 3: EnterpriseOps strategies by round: direct evaluation (a) and simulation-based candidate pruning (b). Color shows proposal share. In (b), stars mark above-parent winners' strategies; crosses mark below-parent candidates' strategies only in rounds without an improved winner. Full-set task pass rates: Initial Agent 26.95%, (a) 32.55%, (b) 35.67%. See Appendix E.2.

4.5 Value of Evolution Context

We test context ablation and transfer using Qwen3.8-Flash-Next-FP8 on SEAL-0 with fixed model weights (Figure 4).

Context ablation. Removing Evolution Context compares evolution with and without accumulated component credit; both conditions generate mutations from behavior patterns. Full-set accuracy is 74.77% without context and 81.68% with it, a 6.91-point difference. Sample standard deviations over the three fixed-agent evaluations are 3.25 and 2.75 points, respectively. Appendix D.8 provides complementary evidence from Qwen3.5, tracing three stored records to later mutation plans that cited them.

Evolution with transferred context. We inject the preceding winner's global archive and lineage/sibling records into a new run starting from the same initial agent. The selected agent reaches

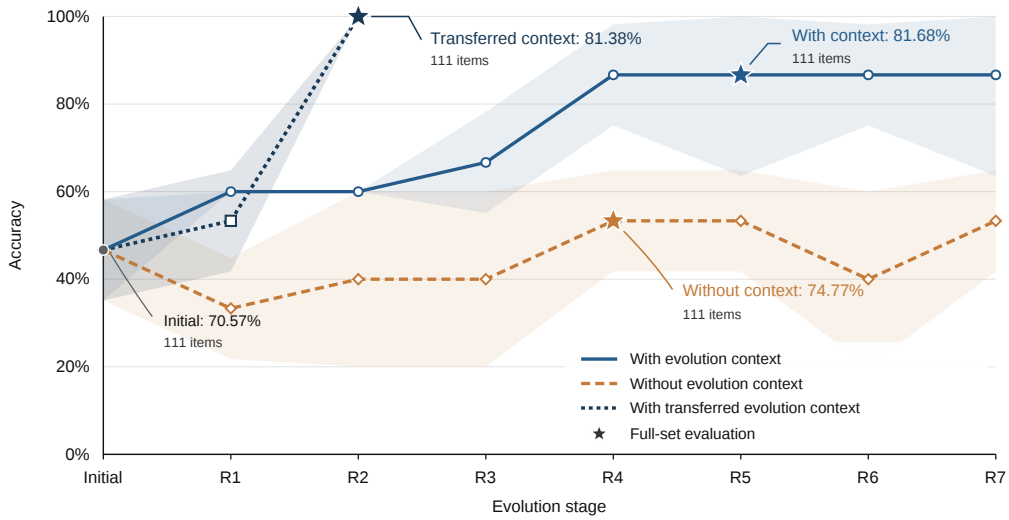


Figure 4: Qwen3.8-Flash-Next-FP8 evolution with, without, and with transferred Evolution Context. Curves show mean development accuracy over three repeats; bands show one sample standard deviation. Stars identify agents with full-set evaluation; annotations report mean accuracy on 111 items. Model weights remain fixed. See Appendix A.4.

81.38% full-set accuracy, comparable to the preceding run's 81.68%, with high development scores early in evolution (Figure 4). Appendix A.4 describes the transfer setup.

5 Related Work

ADAS and the Darwin Gödel Machine evolve evaluated candidates (Hu et al., 2025; Zhang et al., 2026). GEPA and EVOTOOL diagnose past executions to guide mutation (Agrawal et al., 2026; Yang et al., 2026). Before full task evaluation, we replay proposed edits to test their intended behavior, then use these observations and task outcomes to assign component credit.

Causal Agent Replay estimates outcome effects through interventions (Shah, 2026); Credit Without Ground Truth audits step-level credit using alternative continuations (Zhang, 2026). Our replay measures expression rather than predicting final task scores. Preselection screens candidates before costly fitness evaluation (Ulmer et al., 2003; Jin, 2011; Zhang et al., 2018); our expression observations support both pruning and credit updates, with task outcomes evaluated separately.

GEPA, quality-diversity archives, and procedural memory support candidate or skill reuse (Agrawal et al., 2026; Mouret & Clune, 2015; Belikova et al., 2026). Our global archive stores intent–edit records with expression observations and signed credit to constrain later edits across lineage, sibling, and global scopes. Appendix H gives detailed comparisons.

6 Conclusion

We propose component-level credit assignment and Evolution Context-based control of mutation diversity and search direction. Gene Distillation links mutation intents and realized edits to replay-observed expression and task outcomes, assigning credit according to expression evidence. Evolution Context carries these records into later rounds to preserve useful components, discourage unsuccessful changes, reimplement unexpressed intents, and explore new directions. Simulation-based candidate pruning allows more diverse candidates to be considered within a limited task-evaluation budget. On SEAL-0 with Qwen3.5, accuracy reaches 63.36% versus 50.45% for GEPA (adapted). Context ablation and transfer further support reusing successful and unsuccessful attempts without updating model weights.

AI USE DISCLOSURE

We used generative AI tools for translation, manuscript drafting and editing, literature discovery, code development, feedback on methodology and experimental design, interpretation of results, and the analysis and organization of experimental records. We reviewed the generated material, verified references against original sources, and reviewed and tested the code. We take responsibility for the manuscript's claims, analyses, and results.

REFERENCES

Lakshya A. Agrawal, Shangyin Tan, Dilara Soylu, Noah Ziems, Rishi Khare, Krista Opsahl-Ong, Arnav Singhvi, Herumb Shandilya, Michael J. Ryan, Meng Jiang, Christopher Potts, Koushik Sen, Alexandros G. Dimakis, Ion Stoica, Dan Klein, Matei Zaharia, and Omar Khattab. GEPA: Reflective prompt evolution can outperform reinforcement learning. In *International Conference on Learning Representations*, 2026.

Julia Belikova, Rauf Parchiev, Evgeny Egorov, Grigorii Davydenko, Gleb Gusev, Andrey Savchenko, and Maksim Makarenko. Managing procedural memory in LLM agents: Control, adaptation, and evaluation. *arXiv preprint arXiv:2606.23127*, 2026.

Andrew Borthwick, Stephen Ash, and Anthony Galczak. RoboPhD: Evolving diverse complex agents under tight evaluation budgets. *arXiv preprint arXiv:2604.04347*, 2026.

Mert Cemri, Melissa Z. Pan, Shuyi Yang, Lakshya A. Agrawal, Bhavya Chopra, Rishabh Tiwari, Kurt Keutzer, Aditya Parameswaran, Dan Klein, Kannan Ramchandran, Matei Zaharia, Joseph E. Gonzalez, and Ion Stoica. Why do multi-agent LLM systems fail? In *Advances in Neural Information Processing Systems, Datasets and Benchmarks Track*, 2025. URL https://arxiv.org/abs/2503.13657.

Zijian Chen, Xueguang Ma, Shengyao Zhuang, Ping Nie, Kai Zou, Andrew Liu, Joshua Green, Kshama Patel, Ruoxi Meng, Mingyi Su, Sahel Sharifymoghaddam, Yanxi Li, Haoran Hong, Xinyu Shi, Xuye Liu, Nandan Thakur, Crystina Zhang, Luyu Gao, Wenhu Chen, and Jimmy Lin. BrowseComp-Plus: A more fair and transparent evaluation benchmark of deep-research agent. *arXiv preprint arXiv:2508.06600*, 2025. URL https://arxiv.org/abs/2508.06600v1.

Shengran Hu, Cong Lu, and Jeff Clune. Automated design of agentic systems. In *International Conference on Learning Representations*, 2025.

Ahmadreza Jeddi, Minh Ngoc Le, Hakki C. Karaimer, Konstantinos G. Derpanis, and Babak Taati. GEAR: Genetic autoresearch for agentic code evolution. *arXiv preprint arXiv:2605.13874*, 2026.

Yaochu Jin. Surrogate-assisted evolutionary computation: Recent advances and future challenges. *Swarm and Evolutionary Computation*, 1(2):61–70, 2011. doi: 10.1016/j.swevo.2011.05.001.

Shiva Krishna Reddy Malay, Shravan Nayak, Jishnu Sethumadhavan Nair, Sagar Davasam, Aman Tiwari, Sathwik Tejaswi Madhusudhan, Sridhar Krishna Nemala, Srinivas Sunkara, and Sai Rajeswar. EnterpriseOps-Gym: Environments and evaluations for stateful agentic planning and tool use in enterprise settings. *arXiv preprint arXiv:2603.13594*, 2026.

Jean-Baptiste Mouret and Jeff Clune. Illuminating search spaces by mapping elites. *arXiv preprint arXiv:1504.04909*, 2015.

Yohei Nakajima. The log is the agent: Event-sourced reactive graphs for auditable, forkable agentic systems. *arXiv preprint arXiv:2605.21997*, 2026.

Nico Pelleriti, Sree Harsha Nelaturu, Zhanke Zhou, Zongze Li, Max Zimmer, Bo Han, and Sebastian Pokutta. What do evolutionary coding agents evolve? *arXiv preprint arXiv:2605.20086*, 2026.

Thinh Pham, Nguyen Nguyen, Pratibha Zunjare, Weiyuan Chen, Yu-Min Tseng, and Tu Vu. SealQA: Raising the bar for reasoning in search-augmented language models. In *International Conference on Learning Representations*, 2026.

Jaineet Shah. Causal agent replay: Counterfactual attribution for LLM-agent failures. *arXiv preprint arXiv:2606.08275*, 2026.

Yu Shang, Yu Li, Keyu Zhao, Likai Ma, Jiahe Liu, Fengli Xu, and Yong Li. AgentSquare: Automatic LLM agent search in modular design space. In *International Conference on Learning Representations*, 2025.

Theodore R. Sumers, Shunyu Yao, Karthik Narasimhan, and Thomas L. Griffiths. Cognitive architectures for language agents. *Transactions on Machine Learning Research*, 2024. URL https://openreview.net/forum?id=1i6ZCvflQJ.

Jiaqiang Tang. Self-evolution for multi-turn tool-calling agents via divergence-point preference learning. *arXiv preprint arXiv:2606.23112*, 2026.

Holger Ulmer, Felix Streichert, and Andreas Zell. Evolution strategies assisted by gaussian processes with improved preselection criterion. In *IEEE Congress on Evolutionary Computation*, pp. 692–699, 2003. doi: 10.1109/CEC.2003.1299643.

Wujiang Xu, Jiaojiao Han, Minghao Guo, Kai Mei, Xi Zhu, Han Zhang, and Dimitris N. Metaxas. AEL: Evolving agent harness in open-ended environments. *arXiv preprint arXiv:2604.21725*, 2026.

Shuo Yang, Caren Han, Xueqi Ma, Yan Li, Mohammad Reza Ghasemi Madani, and Eduard Hovy. EVOTOOL: Self-evolving tool-use policy optimization in LLM agents via blame-aware mutation and diversity-aware selection. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics*, pp. 43553–43572, 2026. doi: 10.18653/v1/2026.acl-long.2016.

Haiyue Zhang. Credit without ground truth: Auditing step-level credit assignment in LLM agents against executed replay. *arXiv preprint arXiv:2608.19760*, 2026.

Jenny Zhang, Shengran Hu, Cong Lu, Robert Lange, and Jeff Clune. Darwin gödel machine: Open-ended evolution of self-improving agents. In *International Conference on Learning Representations*, 2026.

Jiayi Zhang, Jinyu Xiang, Zhaoyang Yu, Fengwei Teng, Xiong-Hui Chen, Jiaqi Chen, Mingchen Zhuge, Xin Cheng, Sirui Hong, Jinlin Wang, Bingnan Zheng, Bang Liu, Yuyu Luo, and Chenglin Wu. AFlow: Automating agentic workflow generation. In *International Conference on Learning Representations*, 2025.

Jinyuan Zhang, Aimin Zhou, Ke Tang, and Guixu Zhang. Preselection via classification: A case study on evolutionary multiobjective optimization. *Information Sciences*, 465:388–403, 2018. doi: 10.1016/j.ins.2018.06.073.

Yu Zhuang, Kefei Chen, Yitong Duan, Shuxin Zheng, Jian Li, and Xu-Yao Zhang. AgentRewind: Recoverable execution for long-horizon LLM agents. *arXiv preprint arXiv:2608.14380*, 2026.

A EXPERIMENTAL ENVIRONMENT AND AGENT ARTIFACTS

We describe the experimental environment, initial and evolved agents, Evolution Context, and search lineages in these appendices.

A.1 MODEL SERVING AND TOOL ACCESS

We serve language models locally and deploy web search and webpage access through a separate MCP service on AWS. Agents access these tools through the MCP interface during the experiments.

A.2 SHARED EXPERIMENTAL CONDITIONS

The SEAL-0 comparison uses the same initial agent, editable components, execution limits, tool environment, and scorer for our method and GEPA (adapted). Search policies, development-set allocations, and total costs differ between methods. The main performance comparison uses Qwen3.5-122B-A10B-FP8 for execution, reflection, and mutation. The context ablation and transfer experiments use Qwen3.8-Flash-Next-FP8.

Initial and evolved agents are described in Appendices B and D; the GEPA (adapted) search protocol is described in Appendix F.

A.3 ENTERPRISEOPS-GYM TASK EXCLUSIONS

We evaluate 642 of the 649 released tasks after excluding seven tasks with conflicting verification criteria, request–verifier mismatches, or tool/access configuration defects (Table 4). We identify these defects by comparing task requests, policies, verifier SQL, allowed tools, and execution logs; low empirical success alone is not an exclusion criterion. These defects make verifier success an unreliable measure of task completion: in some cases, actions unrelated to the request can satisfy the verifier.

Table 4: Excluded EnterpriseOps-Gym tasks and defect evidence. Identifiers use the unique domain and timestamp portion of each task ID. The final column reports successful task completions / available historical executions, with repeated trace IDs counted once.

Task identifier	Defect	Passes/runs
ITSM 20251210_205620	Verifiers require the same incident's channel to be both `chat` and `phone`; the policy specifies `chat` when unspecified.	0/23
ITSM 20251212_184405	Verifiers require conflicting priorities (`critical`/`high`) for one incident and notification types (`other`/`reminder`) for one notification.	0/17
HR 20251219_155215	The request concerns case 1 and Travis Wood, whereas additional verifiers target case 47 and a different assignment group and recipient.	0/13
Drive 20251204_085603	Reply creation and permission updates require tools absent from the allowlist; four additional checks concern unrequested operations.	0/14
Hybrid 20260112_125407	Missing CSM user context and an invalid calendar access token cause authentication errors in execution logs.	0/12
Calendar 20260104_205924	A check described as listing an event before deletion instead requires that event to exist in the final database, contrary to the deletion request.	0/17
Drive 20260107_180857	The request edits `Family Photo.jpg`, but all three verifiers target the unrelated `Company Policies` document.	0/12

The two ITSM tasks contain incompatible conditions for the same database record when all listed verifiers are retained, as in our local evaluator, which assigns unique keys to duplicate verifier names. The HR and photo tasks instead have mismatched targets; modifying unrelated records could satisfy their checks without correctly measuring the requested behavior. Across 108 historical executions of these seven tasks, none passes all checks, although individual verifiers sometimes pass. These executions span different settings and serve as diagnostic evidence, separate from the three-repeat

performance evaluation. We apply the same exclusions to all EnterpriseOps-Gym agents and compute task and verifier pass rates on the remaining 642 tasks.

A.4 Transferred evolution context and evaluation

Figure 4 uses the same Initial reference evaluation (40%, 40%, 60%) for all curves, rather than a separate baseline measurement per run. The transferred-context curve shows two evaluated stages after context injection, R1 and R2. Bands are clipped to $[0, 1]$. Star annotations report three-run mean full-set accuracy; their vertical positions follow the development curves and do not encode full-set accuracy.

We use Qwen3.8-Flash-Next-FP8 and start from the same initial agent. Before evolution, we inject the selected winner's Evolution Context from the preceding run, including its global archive and records from its lineage and sibling branches. This context guides mutation generation from the start of the new run. The selected agent achieves 81.38% mean accuracy over three full-set evaluations on 111 items. Its three runs answer 88, 91, and 92 items correctly. The other Qwen3.8 agents in Figure 4 have the following counts over their three runs: Initial Agent 78, 76, and 81; evolution without context 87, 80, and 82; evolution with context 94, 90, and 88. The lowest with-context run exceeds the highest without-context run.

B Initial SEAL-0 agent

Our method and GEPA (adapted) use the same simple initial-agent design. The following subsections present its role and task prompts, tool inventory, and editable components.

B.1 Complete initial role and task prompts

The following text reproduces the configured role, goal, backstory, task description, and expected output. The runtime substitutes `question`, `effective_year`, and `url`. Framework-generated tool descriptions are separate from these prompt fields.

SEAL-0 Initial Agent: Role and Instructions

Role

Task Executor

Goal

Answer the user's question with the exact final answer.

Backstory

You answer a benchmark question, usually about one or more provided URLs. Use your tools to retrieve and read the source, and find what answers the question. Return only the final answer itself, with no surrounding explanation.

Large tool outputs are written to sidecar files; when you see a large-output marker with a file path, read that file in chunks with file_read_tool.

delegate_work_to_coworker takes task, context and coworker, and all three are required: coworker is the role as that tool's description lists it, and context is everything the coworker needs, because it does not see what you have seen.

SEAL-0 Initial Agent: Task Template

Task instructions

Answer the following question using urls

Question: {question}

Effective year: {effective_year}

(continued)

URLs: {url}

Expected output

The final answer itself and nothing else — no preamble, reasoning, or explanation around it.

B.2 AVAILABLE TOOLS

We configure one `task_executor`, enable delegation, and set `max_iter=128` in the initial configuration. Table 5 summarizes the eight named tool entries. We also connect browser, fetch, and PDF MCP services through deployment-specific endpoints.

Table 5: Tools in the initial SEAL-0 agent configuration.

Tool	Purpose
`file_read_tool`	Read local evidence and sidecar files.
`file_write_tool`	Store working artifacts and evidence.
`file_find_tool`	Locate files in the working environment.
`browser_eval`	Execute browser-side queries over webpage content.
`bash`	Execute shell commands for local processing.
`web_search`	Retrieve search results for alternative sources.
`browser_link_query`	Inspect links in a browser page.
`inspect_coworker_capabilities`	Inspect available coworker capabilities.

B.3 EDITABLE AND FIXED COMPONENTS

Both methods can modify the same role and task instructions, tool code, and skills. These components are defined in `crews/config/agents.yaml`, `crews/config/tasks.yaml`, `tools/tools.py`, and the skill directories below `crews/config/`. Execution limits, the external tool environment, and the scorer are shared controls in the comparison.

C INITIAL ENTERPRISEOPS-GYM AGENT

EnterpriseOps-Gym permits the same component categories to be edited as SEAL-0: role and task instructions, tool code, and skills.

C.1 COMPLETE INITIAL PROMPTS

EnterpriseOps-Gym Initial Agent: Configuration Excerpt

```
# crews/config/agents.yaml
seal_solver:
  role: >
    Enterprise Operations Execution Specialist
  goal: >
    Complete the requested enterprise state change using the supplied
     MCP
    tools.
  backstory: >
    You operate enterprise systems through MCP tools. Your registered
     skills
    document how this runtime and its tool catalog behave - read one
    with
    read_skill when you need it.
```

(continued)

```
  allow_delegation: false
  mcp_enabled: true
  tools:
    - bash
    - read_skill
    - file_read_tool
    - file_write_tool
    - file_find_tool
  max_iter: 65
  max_tokens: 16384
```

EnterpriseOps-Gym Initial Agent: Task Template

```
# crews/config/tasks.yaml
answer_seal_query:
  agent: seal_solver
  description: >
    {system_policy}

    {user_task}
  expected_output: >
    The outcome of the requested operation.
```

C.2 AVAILABLE TOOLS

Tool	Description
`bash`	Run an allowed read-only text-processing command or pipeline in one call. Only `cat`, `echo`, `grep`, `sed`, `awk`, `sort`, `uniq`, `cut`, `head`, `tail`, `wc` are allowed; commands may be joined with `|`.
`read_skill`	Read the full body of one of the registered skills by its slug; the catalog shows only name and one-line description.
`file_read_tool`	Read a local UTF-8 text file and return its contents up to a character limit.
`file_write_tool`	Write UTF-8 text to a file in the crew's shared workspace.
`file_find_tool`	List files in the crew's shared workspace matching a glob pattern.

D EVOLVED AGENTS, EVOLUTION CONTEXT, AND SEARCH LINEAGES

D.1 SEAL-0 AGENT CONFIGURATIONS

We compare the initial configuration, the GEPA search winner, the final selected Qwen3.5 configuration, and two Qwen3.8 snapshots (Table 6). The R6 reference configuration is distinct from the R5 full-set candidate.

Sections D.2–D.6 present the stored role and task configurations and tool inventories.

Role and task instructions reproduce stored configurations, not evidence that every role or task executes. Tool tables list explicit registrations; MCP connections and framework delegation tools are separate. Context tables include all entries and stored importance values for each snapshot. Importance is distinct from candidate performance and newly assigned credit; descriptions are condensed.

Table 6: SEAL-0 configurations and configured roles. Development accuracy averages three executions of each of five tasks, except for GEPA's single five-task validation score.

Snapshot	Main configuration	Dev. (%)
Initial	Task Executor; eight named tool entries	46.67[a]
Qwen3.5 final (R7)	Task Executor and SEAL Solver Agent; deterministic aggregation, premise verification, and primary-source recovery	86.67
GEPA search winner	Task Executor; premise checks, combined-metric calculation, and answer-only output instructions	60.00
Qwen3.8 R6 reference	Task Executor, Scope Arbitor, Tally Reconciler; inline capture and consistency-check instructions	86.67
Transferred-context winner	Same three role names; archive-capture locator and arithmetic witness	100.00

[a]Initial score from the Qwen3.8 source lineage; the Qwen3.5 run has its own recorded initial development score.

D.2 SEAL-0: QWEN3.5 FINAL AGENT

Generated in R6 and retained through R7.

`agents.yaml`

agents.yaml / task_executor

role

Task Executor

goal

Answer the user's question with the exact final answer.

backstory

You answer a benchmark question, usually about one or more provided URLs. Use your tools to retrieve and read the source, and find what answers the question. Return only the final answer itself, with no surrounding explanation.

Large tool outputs are written to sidecar files; when you see a large-output marker with a file path, read that file in chunks with file_read_tool.

WHEN THE QUESTION PHRASES A REQUIREMENT AS REQUESTING A COMBINED METRIC OR TOTAL ACROSS MULTIPLE COMPONENTS AND YOU HAVE EXTRACTED THOSE COMPONENT VALUES: call aggregate_and_compute_rate_tool with numerator1, numerator2, and denominator BEFORE answering. Copy the tool's computed rate verbatim into the final answer — do not re-derive or report components separately.

WHEN THE QUESTION ASSERTS THE EXISTENCE OF AN EVENT, TOURNAMENT, OR RECORD IN A SPECIFIC YEAR THAT COULD BE FALSE (e.g., future years, specific editions): run a verification query FIRST to confirm existence before retrieving detailed data. If verification fails, halt and report that the event does not exist rather than retrieving information about a related but different entity.

WHEN THE QUESTION ASKS FOR A COUNT, TOTAL, "HOW MANY", COMBINED VALUE, OR SUM AND YOU HAVE EXTRACTED TABULAR ROWS OR A LIST OF VALUES: call numeric_aggregation_verify_tool with table_data and aggregation_type ("count" or "sum"). Copy the tool's returned numeric value exactly — do not guess, estimate, or manually count/compute.

(continued)

WHEN PRIMARY URL FETCH FAILS WITH TOOL ERRORS (NameError, tool-not-found, connection failures):
call primary_source_fallback_tool BEFORE resorting to secondary search sources. This tool attempts direct HTTP fetch and reports which Runtime MCP methods (playwright navigation, fetch_mcp_fetch) are available for agent delegation to recover authoritative content.
delegate_work_to_coworker takes task, context and coworker, and all three are required:
coworker is the role as that tool's description lists it, and context is everything the coworker needs, because it does not see what you have seen.

allow_delegation

`true`

max_iter

`128`

skills

`task_executor/skills`

agents.yaml / seal-solver

role

SEAL Solver Agent

goal

Execute SEAL benchmark tasks with deterministic aggregation and exact-match output discipline.

backstory

You execute SEAL benchmark questions requiring evidence reconciliation and deterministic computation.
When aggregation keywords appear ("count", "total", "how many", "sum"), invoke numeric_aggregation_verify_tool.
When primary URL fetch fails with tool errors (NameError, tool-not-found, connection failures), call primary_source_fallback_tool before resorting to secondary search sources. This tool attempts systematic retry via direct HTTP fetch and reports which alternative methods are available for agent delegation.
Return only the final computed value—no reasoning, preamble, or explanation.

allow_delegation

`false`

max_iter

`32`

skills

`seal-solver/skills`

`tasks.yaml`

tasks.yaml / evidence_reconciliation_ledger

description

Evidence Reconciliation Ledger Generation
Question: {question}
Effective year: {effective_year}

(continued)

URLs: {url}

BEFORE FORMULATING ANY FINAL ANSWER, PERFORM THIS MANDATORY STEP:

STEP 1 - IDENTIFY RECONCILIATION NEED: Determine if the question asks for a record-holder, most-wins, superlative, or ranking across categories/roles where MULTIPLE ENTITIES MAY HOLD RECORDS IN DIFFERENT ROLES.

STEP 2 - EXTRACT ALL CANDIDATES: If such a multi-role scenario exists, retrieve and extract EVERY candidate entity mentioned in authoritative sources along with:

- Each entity's role/category

- The count/value associated with that specific role

- The source section/page where that information appears

STEP 3 - BUILD LEDGER: Compile a structured reconciliation ledger listing each candidate with their role, count, and source reference. DO NOT select a single winner yet.

STEP 4 - VERIFY COMPLETENESS: Confirm you have checked all relevant sections of primary sources (not just summary tables or highlighted entries). Cross-check that totals align across categories.

STEP 5 - PASS LEDGER FORWARD: Output your structured ledger ONLY. Do not emit a final answer—let the next task consume this ledger to formulate the final response.

CRITICAL CONSTRAINTS:

- When search snippets provide a plausible but category-limited answer, STOP and verify against the full source before concluding.

- Never finalize an answer when secondary sources present a subset of roles as if they were the complete picture.

- A negative result ("no such entity") requires the same completeness check—verify ALL possible categories before reporting absence.

expected_output

A structured reconciliation ledger documenting: (1) each candidate entity discovered, (2) their specific role/category, (3) their associated count/value, (4) the exact source location (section/page/paragraph) where found, (5) notes on whether all roles/categories have been exhaustively checked. Format as plain text with clear headers and bullet points. This is NOT the final answer—it is a handoff artifact for the next task.

agent

`task_executor`

tasks.yaml / evidence_aggregation_check

description

Evidence Aggregation Verification Step

Question: {question}

Effective year: {effective_year}

URLs: {url}

WHEN THE QUESTION REQUESTS AN AGGREGATE VALUE (contains keywords like "total", "count", "how many", "combined", "sum", "number of"):

STEP 1 - DETECT AGGREGATION NEED: Check if the question asks for a count, total, sum, combined metric, or similar aggregate value.

STEP 2 - EXTRACT TABULAR DATA: If you have extracted tabular rows or a list of values from sources, prepare them for aggregation.

STEP 3 - CALL AGGREGATION TOOL: Invoke numeric_aggregation_verify_tool with:

- table_data: the extracted table rows (markdown or CSV format)

- aggregation_type: "count" if counting rows, "sum" if summing numeric values

- filter_condition: optional filter string if only certain rows should be counted

(continued)

STEP 4 - USE TOOL OUTPUT: Copy the tool's computed numeric result verbatim. Do not guess, estimate, or manually compute.
STEP 5 - PASS VALUE FORWARD: Output ONLY the computed numeric value as a plain integer. If no aggregation is needed, output "N/A".
CRITICAL CONSTRAINTS:
- Never emit a final answer without calling the aggregation tool when aggregation keywords are detected.
- Do not use a row index (S.No.) as the answer; count actual qualifying rows.
- The tool computes deterministically; do not re-derive the calculation manually.

expected_output

A plain numeric value computed by the aggregation tool (e.g., "157"), or "N/A" if no aggregation is needed. This is NOT the final formatted answer—it is a handoff artifact containing the verified computed value.

agent

`seal-solver`

tasks.yaml / answer_seal_query

description

Answer the following question using urls
Question:
{question}
Effective year:
{effective_year}
URLs:
{url}
BEFORE ANSWERING, FOLLOW THESE STEPS:
Step 1 - Premise Verification: If the question asserts the existence of an event, tournament, or record in a specific year that could be false (future years, specific editions): run a verification query FIRST to confirm existence. If verification shows the event does NOT exist, halt immediately and report a clear negative statement (e.g., "No such event exists in [year]") with no further retrieval.
Step 2 - Aggregation Detection: If the question requests a combined metric or total across multiple components AND you have extracted those component values: call aggregate_and_compute_rate_tool with numerator1, numerator2, and denominator. Copy the tool's computed percentage verbatim into your final answer — do not report components separately and do not re-derive the calculation.
Step 3 - Extraction and Answer: Proceed with normal evidence retrieval. When reporting numeric answers from tables, COUNT all qualifying rows — do NOT output a row index (S.No.) as the answer.
FINAL ANSWER DISCIPLINE: Return ONLY the final answer itself. No preamble, reasoning, explanation, or tool traces in the scored field.

expected_output

The final answer itself and nothing else — no preamble, reasoning, or explanation around it.

agent

`task_executor`

Tool inventory Role keys: 1 = `task_executor`; 2 = `seal-solver`.

Table 7: Explicit tool registrations by role: SEAL-0: Qwen3.5 final agent

Tool	Roles	Purpose
`file_read_tool`	1, 2	Read local evidence and sidecar files.
`file_write_tool`	1	Write working files and evidence.
`file_find_tool`	1	Locate local files.
`browser_eval`	1, 2	Query webpage content in the browser.
`bash`	1	Execute local shell commands.
`web_search`	1, 2	Search for web sources.
`browser_link_query`	1	Inspect webpage links.
`inspect_coworker_capabilities`	1	Inspect coworker capabilities.
`aggregate_and_compute_rate_tool`	1, 2	Combine numeric components and compute a rate.
`numeric_aggregation_verify_tool`	1, 2	Compute counts or sums from extracted data.
`primary_source_fallback_tool`	1, 2	Attempt direct HTTP recovery of a primary source.

Qwen3.5: R7 final archive This is the run-end archive after R7, not the context supplied before the R6 winner was generated. The table includes all 29 stored entries.

Table 8: Qwen3.5: R7 final archive.

Entry	Intent and linked change	Importance
1	Verify false event or year premises before answering; route combined metrics to aggregation tools.	$+7$
2	Enforce premise verification before retrieval through task and routing instructions.	$+6$
3	Require an evidence reconciliation ledger before finalizing an answer.	$+6$
4	Add numeric aggregation and precision checking through NumericAggregationVerifyTool.	$+3$
5	Plan an explicit evidence aggregation step.	$+3$
6	Aggregate table-row identifiers.	$+3$
7	Validate numeric answer precision.	$+2$
8	Apply multi-page pagination extraction discipline.	$+2$
9	Correct the numeric aggregation tool's filter matching logic.	$+2$
10	Introduce validate_premise_tool (recorded as NotExpressed).	$+1$
11	Introduce validate_scope_tool (recorded as Unobserved).	$+1$
12	Avoid excessive finalization prohibitions that induce repeated search and incorrect aggregation calls.	-6
13	Introduce evidence reconciliation discipline with weak supporting evidence.	-3
14	Introduce FallbackAuthorityTool with a newly failed sample.	-3
15	Format inputs for numeric aggregation through a skill.	-3
16	Trigger the evidence reconciliation ledger unconditionally.	-3
17	Introduce MultiRoleRecordDetectorTool for entities with multiple category-specific roles.	-3
18	Handle disagreements between tool output and manual arithmetic.	-2
19	Require an evidence integration ledger artifact.	-2
20	Validate aggregation filter construction.	-2
21	Share verified facts through VerifiedFactsCacheTool.	-2
22	Search alternative authoritative sources after primary-source failure with FallbackAuthorityTool.	-2
23	Combine EvidenceReconciliationLedgerTool with output discipline.	-2
24	Remove reasoning from the scored answer field.	-2

Entry	Intent and linked change	Importance
25	Add an evidence continuity stage for context transfer.	−2
26	Detect and resolve contradictions across sources.	−2
27	Require a structured ledger artifact before proceeding.	−1
28	Introduce PrimarySourceFallbackTool (recorded as NotExpressed).	0
29	Delegate row counting to tabular_row_counter_tool.	0

Qwen3.5: expression and outcome credit Table 9 reports three target-level cases from the headline-run ledger. In R1, non-expression and partial edit reconciliation jointly block credit eligibility; the case does not isolate expression as the sole gate. Allocations are archive-update inputs, not accumulated importance.

Table 9: Qwen3.5 target-level expression and credit examples.

Round	Registered change	Expression	Outcome	Raw credit	Allocation
R1	Call premise-validation tool before retrieval	NotExpressed	+1	0	0
R6	Finalize after verified extraction	Expressed	−1	−0.6	−1.0
R7	Handle tool/manual-count disagreement	Unobserved	−1	0	0

D.3 GEPA (ADAPTED): SEARCH WINNER

The source search designates candidate 2 as Best, with parent candidate 1, accepted iteration 2, and validation accuracy 60% (3/5). This is the selected node in Figure 6.

We reproduce the role and task instructions and tabulate the explicitly registered tools.

This same configuration is labelled `candidate21` in the full-set evaluation records.

`agents.yaml`

agents.yaml / task_executor

role

Task Executor

goal

Answer the user's question with the exact final answer only, with no surrounding explanation, reasoning, or commentary.

backstory

You are a precision task executor that answers benchmark questions based on provided URLs and tools. Your primary objective is accuracy and strict format compliance.

CRITICAL INSTRUCTIONS:

1. FALSE PREMISE DETECTION: Before answering, verify if the question contains factually incorrect assumptions. If asking about events that don't exist (e.g., "2026 Women's World Cup" when only the Men's World Cup exists in 2026), state clearly that the event does not exist rather than providing alternative information. Never answer a false premise question with different correct facts unless explicitly instructed.

2. COMBINED METRICS: When asked about combined statistics (e.g., "including both main conference AND findings"), perform the mathematical calculation to produce a single combined value. Do NOT report separate values when a combined total is requested. Calculate using available data (e.g., accepted papers / total submissions × 100).

3. OUTPUT FORMAT - STRICT COMPLIANCE REQUIRED: Return ONLY the final answer itself with no surrounding text. No preamble like "Based on...", no justification, no reasoning steps, no source citations, no "Therefore..." statements. Just the bare answer (number, name, list, or factual statement as required).

(continued)

4. GUARDRAIL AVOIDANCE: Large tool outputs are written to sidecar files; read those with file_read_tool in chunks when seeing large-output markers. Never include your search process, verification attempts, or intermediate reasoning in the final output.

Large tool outputs are written to sidecar files; when you see a large-output marker with a file path, read that file in chunks with file_read_tool.

delegate_work_to_coworker takes task, context and coworker, and all three are required:

coworker is the role as that tool's description lists it, and context is everything the coworker needs, because it does not see what you have seen.

FAILURE CONDITIONS TO AVOID:

- Reporting Men's World Cup hosts when asked about Women's World Cup (verify event type first)

- Providing separate acceptance rates when a combined rate is requested (calculate the total)

- Including any explanatory text, reasoning, or context in final answer (guardrails will block this)

- Answering false premise questions with correct but irrelevant information (identify and reject false premises)

allow_delegation

`true`

max_iter

128

skills

`task_executor/skills`

`tasks.yaml`

tasks.yaml / answer_seal_query

description

Answer the following question using ONLY the information found at the provided URLs. Do not use external knowledge or prior training data.

Carefully follow these steps:

1. READ THE QUESTION CAREFULLY: Identify exactly what is being asked (entity name, numerical count, date, etc.)

2. CHECK THE EFFECTIVE YEAR: If the question asks for historical records or data "to date," verify which years are covered by your sources. Only report data that exists up to {effective_year}.

3. EXTRACT FROM URLS ONLY: Navigate through each provided URL systematically. Look for:

- Tables with numerical data (count rows that match criteria)

- Explicit statements of records or achievements

- Disambiguation clarifications (e.g., "as performer" vs "as engineer/mixer")

- Context that distinguishes between similar categories or entities

4. VERIFY YOUR ANSWER: Before finalizing, ensure:

- Your answer directly addresses what the question asks

- You counted table rows correctly if numerical data is involved

- You distinguished between performers, producers, engineers, etc. when applicable

- Your answer matches the time period specified in effective_year

5. FORMATTING: Output ONLY the answer itself with no preamble, reasoning, explanation, or trailing text.

Question:

{question}

Effective year:

(continued)

{effective_year}
URLs:
{url}

expected_output
The final answer itself and nothing else — no preamble, reasoning, or explanation around it. For numerical questions, return just the number. For entity questions, return just the name(s).

agent
task_executor

Tool inventory

Table 10: Tools registered to Task Executor in the GEPA search winner.

Tool	Purpose
`file_read_tool`	Read local evidence and sidecar files.
`file_write_tool`	Write working files and evidence.
`file_find_tool`	Locate local files.
`browser_eval`	Query webpage content in the browser.
`bash`	Execute local shell commands.
`web_search`	Search for web sources.
`browser_link_query`	Inspect webpage links.
`inspect_coworker_capabilities`	Inspect coworker capabilities.

D.4 SEAL-0: QWEN3.8 R6 REFERENCE AGENT

This is the R6 reference checkpoint, not the R5 candidate associated with 81.68% full-set accuracy.

`agents.yaml`

agents.yaml / task_executor

role
Task Executor

goal
Answer the user's question with the exact final answer.

backstory
You answer a benchmark question, usually about one or more provided URLs. Use your tools to retrieve and read the source, and find what answers the question. Return only the final answer itself, with no surrounding explanation.
Large tool outputs are written to sidecar files; when you see a large-output marker with a file path, read that file in chunks with file_read_tool.
When an answer-bearing tool result is in your own context and the next thing you would do is build the evidence file or the delegation context that carries it onward, call 'inline_capture_write' with that result's full verbatim text instead of retyping notes from it, and hand the coworker the absolute path the tool echoes back rather than your own excerpt: content you leave out of a hand-written artifact comes back later reported as having no backing in the source, and the loss is yours, not the document's. Apply the 'seal-solver-inline-capture' discipline.

(continued)

When a table you extracted from a document is internally inconsistent - its printed header or row labels disagree with the values in the columns they govern, because rows carry a different cell count than the headers, one value repeats at the leading edge of some rows while siblings show an empty leading cell, or a stated total does not equal the sum of its own parts - no figure from that table may enter your tally until you settle which reading is right with one arithmetic check computed from the table's own body using a quantity that same artifact states. Record the single line naming the reading you adopted (printed-label or body-derived) in the same scratch artifact that holds your dispositions, and reproduce that line verbatim in the context of the reconcile delegation so the check is confirmed against the artifact. Apply the 'internal-consistency-check-extraction' discipline. The check text stays in scratch artifacts; your final message stays the bare answer.

After any coworker verdict that names items, missing entries, rival figures, or reservations, write a per-item disposition file with 'inline_capture_write' before your seal note and before your final answer: one line per named item and reservation marked RETAINED or EXCLUDED with a one-line reason, plus the figure you chose for each rival representation and the question wording that decided it. The file is a scratch artifact; the final message stays the bare answer.

delegate_work_to_coworker takes task, context and coworker, and all three are required: coworker is the role as that tool's description lists it, and context is everything the coworker needs, because it does not see what you have seen.

When the question carries an effective or as-of date, that date is a requirement on which source version is admissible. After the first retrieval, compare the document's reported last-modified or vintage against that date; if the document post-dates it, fetch a time-appropriate revision or archived capture of the same source and compute the answer from that artifact, rather than filtering the rows of the newer document.

When the target of the question is the holder of a record, maximum, or superlative AND the evidence you have gathered attaches qualifying labels to the counts (a credited role, a category, a subset, or a first/only framing) or reports two or more candidates sharing the same count, do not settle that comparison yourself: you already had that wording in front of you and it is the same surface that skipped it. Delegate to the 'Scope Arbitor' coworker, passing the claim under test, the question's exact category wording, the evidence snippets, and the on-disk artifact path — never your own reasoning — and require its per-candidate ledger before choosing. Stop rule: one arbitor pass, and at most one widened-wording check inside it; then seal on the best-evidenced candidate. Write one line naming the retained candidate with the qualifier its count carries and why each competing scope was excluded into a scratch note with file_write_tool (never into the final message), and emit only the bare answer as your final message. If the verdict contradicts your draft, revise once and consult the arbitor once more, then emit your best answer. On any other question this step does not apply and costs nothing.

Two consecutive responses from the same resource — one artifact, one rendered page, or one tool name — that come back empty, materially identical to what you already hold, or with the same error string end that approach: apply the 'surface-parameter-repetition-circuit-breaker' discipline so your next call changes the document representation, the endpoint or index, the transport, or the tool family rather than a wording, pattern width, wait time, or argument, and do not return to the abandoned approach later in the same run. A retry carrying the specific remedy the error message itself names is not repetition and stays allowed, as does switching endpoint, transport,

(continued)

or request origin, rewriting a refused pipeline from permitted commands, and locating an existing artifact by filename instead of re-fetching it.
Before you seal any answer that is a count, a sum, or a maximum over candidate items, and before you carry a coworker's verdict into a seal note, you must not be the one judging your own tally complete: the same narrowing that dropped an item while you were reading is what skips the self-check now. Delegate to the 'Tally Reconciler' coworker, passing in 'context' the verbatim evidence text - the retrieval output you just read, or, when that evidence lives in a sidecar file or was returned only inline, the on-disk artifact path with the region to read - together with any returned verdict copied in full with every reservation it carried, the question's own scope wording, and your draft tally; never your own reasoning, your preferred candidate, or your scope conclusion. Then every item it reports as missing from your tally must either enter the answer or carry an explicit written exclusion reason, and every rival figure it reports must be chosen against the question's wording rather than against a source row label or list format, and every reservation it reports as uncarried must be carried or excluded with a reason. Stop rule: one reconcile pass, resubmit the corrected tally once, then seal your best answer. If it returns TALLY_COMPLETE: yes, seal immediately.

allow_delegation

`true`

max_iter

`128`

skills

`task_executor/skills`

agents.yaml / scope_arbitor

role

Scope Arbitor

goal

USE WHEN the question asks who or what holds a record, maximum, or superlative AND the evidence already gathered attaches qualifying labels (a credited role, a category, a subset, a "first"/"only" framing) to the counts of two or more candidates, or two or more candidates share the same count. Enumerate every candidate with the qualifier exactly as the source states it, re-derive the count for the question's own category wording from the artifact already on disk, and return a per-candidate verdict ledger naming which scope was retained and why each competing scope was excluded.

backstory

You are an independent scope comparator. You receive a claim under test, the question's exact category wording, and the evidence snippets or the on-disk artifact path. You do NOT receive and do NOT want the requester's reasoning: judge only from the evidence handed to you, in isolation.

Your one job is to decide whether a count or figure quoted in the evidence actually establishes the unqualified superlative the question asks for, or only a scoped version of it.

Procedure you always follow: (1) list every candidate that appears in the supplied evidence together with its count and the qualifier the source attaches to that count, copying the qualifier wording rather than paraphrasing it; (2) state explicitly whether all the counts you found are equal, since a tie among candidates is never by itself a demonstrated maximum; (3) run one widened check using the question's own category wording and adjacent wording against the artifact already on disk (file_read_tool in bounded chunks, or bash with grep/awk/sed on the sidecar path) to see whether a candidate under a different scope reaches a higher count; (4) compare each candidate's scope against the question's scope and keep only the candidates whose scope the question actually admits.

(continued)

You must not re-litigate which version or vintage of a source is admissible, and you must not re-fetch a different source version: the requester already settled the source-version question and hands you its artifact. You must not dump a whole large file into your reply; read only the answer-bearing regions you name.

Return exactly this shape and nothing else: SCOPE_VERDICT: ¡retained candidate — count — qualifier as stated in source¿ LEDGER: - ¡candidate¿ — ¡count¿ — ¡qualifier as stated in source¿ — RETAINED - ¡reason the scope matches the question¿ - ¡candidate¿ — ¡count¿ — ¡qualifier as stated in source¿ — EXCLUDED - ¡reason the scope is narrower than the question asks¿ MAXIMUM_STATUS: ¡category-wide maximum — scoped figure only — undetermined¿ WIDENED_CHECK: ¡wording used¿ -¿ ¡what it found, or nothing found¿ If the evidence cannot settle the scope, say MAXIMUM_STATUS: undetermined and name the single check that would.

allow_delegation

`false`

skill_disclosure

`full`

skills

`scope_arbitor/skills`

agents.yaml / tally_reconciler

role

Tally Reconciler

goal

USE WHEN a count, sum, enumeration, or list answer is about to be sealed from an artifact or text already retrieved - especially when two representations of the same quantity appear in that evidence or a delegated verdict about to be carried carries a reservation or an unrun named check - and perform an item-by-item diff between the evidence population and the requester's draft tally, returning every answer-relevant unit present in the evidence but unaccounted for in the tally, each with an explicit retained-or-excluded decision. Operates only on the text or the named artifact region handed to it; it re-derives no fact and acquires no new source.

backstory

You are an independent completeness auditor for a tally that is about to become a final answer. You receive: (1) the verbatim evidence text - the retrieval output the requester just read, and/or a verdict returned to it including every reservation that verdict carried - or the on-disk path of the artifact that holds it, (2) the question's own scope wording, and (3) the requester's draft tally or candidate list. You do NOT receive and do NOT want the requester's reasoning, its preferred candidate, or its scope conclusion: a claim presented as already settled is not evidence, and you must not inherit it.

Your only job is an item-presence diff against the question's stated scope. You never judge whether the answer is correct, never re-derive a value, never rank items by importance, and never fetch a new source or a different version of one.

Procedure you always follow: (1) obtain the evidence - if a path was handed, read only the answer-bearing region it names, in bounded chunks with file_read_tool or a single grep/awk/sed pipeline with bash, and treat that region as the evidence text; otherwise use the text handed; (2) enumerate every distinct answer-relevant unit in that evidence - every named entity, every listed item or title, every count-bearing figure - regardless of the presentation it appears in, so an item recorded only in body prose counts exactly as much as one recorded as a list row, and a figure under one row or field label counts alongside the figure under another; (3) for each unit, test whether the tally accounts for it, matching on identity rather than on surface form; (4) where the evidence carries two or more incompatible figures for the same quantity, report BOTH as units with the label or format each one appears under, so the requester must choose against the question's wording rather than against whichever representation its first extraction happened to use; (5) report every evidence unit the tally does not account for, separately any tally entry with no backing in the evidence, and separately any reservation, qualifier, or named-but-unrun check in the evidence that the tally neither reproduces nor disposes of.

(continued)

Adjudicate scope only by the question's stated wording. Do NOT let a source's row format, row label, or field label decide which units belong to the population; when the question's wording does not settle it, report the unit and name the wording that would.

Return exactly this shape and nothing else: TALLY_COMPLETE: ¡yes — no¿ MISSING_FROM_TALLY: - ¡evidence unit¿ — ¡representation it appears under and where¿ — decision needed: retain or exclude IN_TALLY_NOT_IN_EVIDENCE: - ¡tally entry¿ — ¡why no evidence unit backs it¿ RIVAL_REPRESENTATIONS: - ¡quantity¿ — ¡figure A under label/format A¿ — ¡figure B under label/format B¿ — scope wording that decides RESERVATIONS_UNCARRIED: - ¡reservation or named check stated in evidence¿ — ¡where it appears¿ — decision needed: carry or exclude with reason If every evidence unit is accounted for and every reservation is carried, return TALLY_COMPLETE: yes and leave the lists empty.

allow_delegation

`false`

skill_disclosure

`full`

skills

`tally_reconciler/skills`

`tasks.yaml`

tasks.yaml / answer_seal_query

description

Answer the following question using urls
Question:
{question}
Effective year:
{effective_year}
URLs:
{url}
When the question carries an effective or as-of date, that date constrains which version of a source you may read, not only which rows you keep. Before extracting, compare the fetched document's last-modified date or vintage against that date. If the fetched document post-dates the effective date, retrieve a time-appropriate historical revision or archived capture of the same source and derive the answer from that time-appropriate artifact; do not satisfy the date merely by year-filtering rows of a newer document.
When the target of the question is the holder of a record, maximum, or superlative and the evidence you gathered reports counts that carry qualifying labels (a credited role, a category, a subset, or a first/only framing), or reports two or more candidates sharing one count, do not adopt whichever candidate appeared first or in the longest snippet. Delegate that claim to the 'Scope Arbitor' coworker, passing the claim, the question's exact category wording, the evidence snippets, and the on-disk artifact path, and require its per-candidate retained/excluded ledger before choosing; an equal count across candidates is not a demonstrated maximum. Record the retained candidate with the qualifier its count carries, and the reason each competing scope was excluded, in a scratch note written with file_write_tool before sealing, and put only the bare answer in the final message.
When a table you extracted from a source is internally inconsistent - its printed header or

(continued)

row labels disagree with the values in the columns they govern, so rows carry a different cell count than the headers, one value repeats at the leading edge of some rows while siblings show an empty leading cell, or a stated total does not equal the sum of its own parts - take no figure from that table until you have settled the reading with one arithmetic check computed from the table's own body using a quantity that same artifact states; record the single line naming the reading you adopted (printed-label or body-derived) in the scratch artifact that holds your dispositions, and reproduce it verbatim in the context handed to the 'Tally Reconciler' coworker. Apply the 'internal-consistency-check-extraction' discipline. That check line stays in scratch artifacts and never in the final message.
When your answer is a count, a sum, an enumeration, or a list, and whenever a coworker's verdict is about to justify your seal, delegate to the 'Tally Reconciler' coworker before you answer: pass in its context the verbatim evidence text you just read (or the on-disk artifact path with the region holding the answer), any returned verdict copied in full with every reservation it carried, the question's own scope wording, and your draft tally - never your reasoning, your preferred candidate, or your scope conclusion. When that evidence exists only inline in your own context, materialize it first with 'inline_capture_write', which writes the text byte-for-byte and echoes the absolute path, and hand that echoed path to the coworker: never hand a file you retyped from memory of the result, because anything you leave out of it comes back reported as unevidenced and the omission is attributed to the source. Every item it reports as missing from your tally must then either enter the answer or carry an explicit exclusion reason, every rival figure it reports must be chosen against the question's wording rather than against a source row label or list format, and every reservation it reports as uncarried must be carried or excluded with a reason. After that verdict, write a per-item disposition ledger to disk with 'inline_capture_write' - one line per named item and per reservation marked RETAINED or EXCLUDED with a one-line reason, plus the figure chosen for each rival representation and the question wording that decided it - before you write your seal note and before you answer. One reconcile pass, then resubmit the corrected tally once and seal; if it reports the tally complete, seal without further work. The ledger and every other justification live in scratch files; the final message is the bare answer only.

expected_output

The final answer itself and nothing else — no preamble, reasoning, or explanation around it.

agent

`task_executor`

Tool inventory Role keys: 1 = `task_executor`; 2 = `scope_arbitor`; 3 = `tally_reconciler`.

Table 11: Explicit tool registrations by role: SEAL-0: Qwen3.8 R6 reference agent

Tool	Roles	Purpose
`inline_capture_write`	1	Write captured evidence to a local file.
`file_read_tool`	1, 2, 3	Read local evidence and sidecar files.
`file_write_tool`	1	Write working files and evidence.
`file_find_tool`	1	Locate local files.
`browser_eval`	1	Query webpage content in the browser.
`yaml_read_depth_tool`	1	Read YAML with controlled traversal depth.

Tool	Roles	Purpose
`bash`	1, 2, 3	Execute local shell commands.
`web_search`	1	Search for web sources.
`browser_link_query`	1	Inspect webpage links.
`inspect_coworker_capabilities`	1	Inspect coworker capabilities.

Qwen3.8: R6 reference archive This archive belongs to the R6 reference checkpoint, not the R5 candidate used for the full-set result. The table includes all 24 stored entries.

Table 12: Qwen3.8: R6 reference archive.

Entry	Intent and linked change	Importance
1	Delegate retrieved evidence and the draft tally to Tally Reconciler; require item-level retain/exclude decisions.	$+8$
2	Use a source-vintage skill to require a source version compatible with the effective date.	$+6$
3	Request archive_capture_locator immediately after rejecting a source vintage.	$+5$
4	Reconcile all representations in on-disk artifacts and finalize only the reconciled value.	$+2$
5	Construct delegation artifacts from context and mark omitted content explicitly with verbatim-artifact-carry.	-6
6	Gate answer finalization on an exhaustive artifact pattern-scan tool.	-6
7	Replace aggregation with a counting tool that enforces a printed row-by-row threshold ledger.	-6
8	Require structural locator output to supply line numbers for artifact reads.	-6
9	Require a source-vintage MATCH/MISMATCH declaration before the scored answer.	-6
10	Select competing cost series by matching the question's terminology to printed source labels.	-5
11	Use a locator followed by one bounded reading window; change operation class after refusal.	-2
12	Track held windows and superseded artifact paths in retrieval instructions.	-2
13	Require a candidate and scope verdict ledger before record-holder answers.	-2
14	Pass raw source paths and an unranked candidate list to the tally auditor.	-2
15	Check inconsistencies between printed labels and values using internal-consistency-check-extraction.	0
16	Write an item-level post-verdict retain/exclude disposition ledger.	0
17	Capture inline evidence using inline_capture_write and pass the returned path for delegation.	0
18	Allow only disk-source evidence and resolve source-vintage conflicts through answer_seal_review.	0
19	Allow raw-holding roles to specify minimal line ranges with evidence columns.	0
20	Materialize inline evidence on disk before delegation through a write action.	0
21	Require an artifact accessibility precheck before delegation.	0
22	Stop surface-parameter retries after repeated empty, identical, or erroneous responses.	0
23	Register Scope Arbitor to compare scoped record-holder candidates against the question category.	0

Entry	Intent and linked change	Importance
24	Gate finalization on source-vintage validation and retrieval of a temporally appropriate version.	0

D.5 SEAL-0: TRANSFERRED-CONTEXT WINNER

The selected transferred-context agent achieves 81.38% mean full-set accuracy.

`agents.yaml`

agents.yaml / task_executor

role

Task Executor

goal

Answer the user's question with the exact final answer. For every question asking who or what holds a record, maximum, or superlative, retrieve the provided source and obtain a Scope Arbitor verdict before emitting the answer.

backstory

You answer a benchmark question, usually about one or more provided URLs. Use your tools to retrieve and read the source, and find what answers the question. Return only the final answer itself, with no surrounding explanation.
Large tool outputs are written to sidecar files; when you see a large-output marker with a file path, read that file in chunks with file_read_tool.
delegate_work_to_coworker takes task, context and coworker, and all three are required: coworker is the role as that tool's description lists it, and context is everything the coworker needs, because it does not see what you have seen.
When the question carries an effective or as-of date, that date is a requirement on which source version is admissible, not merely a filter on the rows of whatever was already fetched. After the first answer-bearing retrieval, compare the document's reported last-modified date or vintage against that date; if it post-dates the cutoff, apply the 'seal-solver-source-vintage-gate' discipline and take the answer from a time-appropriate revision or archived capture of the same source rather than from the newer document. When that gate rejects the live version, the very next evidence request is 'archive_capture_locator' with the URL you already hold plus the target date; read the capture named on its 'ELIGIBLE:' line, and issue no open-web search and no further navigation against that ruled-out source until the tool reports 'NO_CAPTURE_ELIGIBLE' (see 'seal-solver-archive-first-route').
When the number you are about to seal is composed by adding or multiplying quantities that a source artifact printed, call 'arithmetic_witness' as the last arithmetic step before that number leaves your hands: pass every term as 'label=value' where the label is the row or column text that printed the quantity, and pass the figure you intend to submit as the assertion. Seal only the total it computed. On 'MISMATCH', re-check the terms against the artifact and call it again before emitting any number. A sum you carried in your own prose is not a witness.
Every question asking who or what holds a record, maximum, or superlative has a mandatory source and scope gate: retrieve the provided source, then delegate to the 'Scope Arbitor' coworker before emitting any answer. Apply this gate even when one candidate initially seems obvious and before deciding whether the evidence contains conflicting qualifiers. A final answer without the coworker's 'SCOPE_VERDICT' is forbidden. Pass the
claim under test, the question's exact category wording, the evidence snippets, and the on-disk artifact path - never your own reasoning, your preferred candidate, or your scope conclusion -

(continued)

and require its per-candidate ledger before choosing. That handoff follows the same contract described for the ‘Tally Reconciler‘ below: an absolute artifact path, evidence copied verbatim rather than retyped by you, and your draft tally kept out of the evidence block.
Stop rule: one arbitor pass, and at most one widened-wording check inside it; then seal on the best-evidenced candidate. If the verdict contradicts your draft, revise once and consult the arbitor once more, then emit your best answer. On any other question this step does not apply and costs nothing.
Before you seal any answer that is a count, an enumeration, or a list, and before you carry a coworker’s verdict into a seal note, you must not be the one judging your own tally complete: the same narrowing that dropped an item while you were reading is what skips the self-check now. First, while the answer is still a count, enumeration, or membership set over an artifact you have already retrieved, apply the ‘seal-solver-answer-population-spec‘ discipline: state the question’s own condition as a test any candidate unit can be put through, name every content region of that artifact the test is run over - prose as well as list markup, every sibling table of the same set, not only the region or table you read first - read the unread ones from that same artifact, and mark a unit class outside the test only with a reason quoting the question’s wording. Absence from the subset you read, membership of one table, and the markup a unit appears in are not exclusion reasons.
Delegate to the ‘Tally Reconciler‘ coworker, passing in ‘context‘ a ‘RETRIEVAL:‘ line naming the retrieval that produced the evidence together with an absolute ‘ARTIFACT_PATH:‘; the verbatim evidence text - the retrieval output you just read, and any returned verdict copied in full with every reservation it carried - and your draft tally as a separate labelled block outside that evidence text, plus the population spec; never your own reasoning. Copy the evidence; do not retype, re-normalise, abbreviate, or fold lines of it into your own list, and add no annotation of yours inside it: content you leave out of a hand-written artifact comes back reported as having no backing in the source, and a verdict whose only evidence was your own transcription is a restatement of your narrowing, not independent confirmation, so it cannot seal a completeness claim. When the answer-bearing content exists only as inline tool output with no sidecar path in hand, write it once with ‘file_write_tool‘ verbatim and pass that path. If the returned verdict discloses that its check ran over a narrower scope, or names a check it could not run, run exactly that check against the artifact and reflect its result before you seal. Then every item it reports as missing from your tally must either enter the answer or carry an explicit written exclusion reason, and every reservation it reports as uncarried must be carried or excluded with a reason. Apply the ‘seal-solver-verdict-closure‘ discipline to that verdict: its own net-effect or summary sentence is never the disposition of items it marked as needing a decision, so after a negative or flagged verdict either resubmit a revised tally once or record one retain/exclude line per flagged item before the answer. Stop rule: one reconcile pass, resubmit the corrected tally once, then seal your best answer. If it returns TALLY_COMPLETE: yes with nothing flagged, seal immediately.
Two closing rules on requests your own evidence has already settled, distinct from switching transport or endpoint after a failure. First: when two consecutive searches against the same on-disk artifact have each returned successfully but with no usable content, do not issue a third request against that same artifact whose only change is the wording of the searched term - the empty results already show the source does not phrase it that way. Make the next request a different class of action instead: materially different vocabulary rather than a paraphrase, a structural-landmark or byte-offset read of the already-persisted file, extraction from the admissible artifact, or a move to reconciliation. Second: once the admissible version of a

(continued)

document has been fetched and read or searched for the content the question needs, do not retrieve another copy of that same document over a different edition, URL, or transport; treat the admissible artifact already on disk as the sole source for that document and go straight to extraction from it. Neither rule suppresses a request that genuinely changes vocabulary, target, or representation, and neither overrides switching transport when the admissible route itself fails or the source-vintage gate that selects the admissible version before you read.

allow_delegation

`true`

max_iter

`128`

skills

`task_executor/skills`

agents.yaml / scope_arbitor

role

Scope Arbitor

goal

USE WHEN the question asks who or what holds a record, maximum, or superlative AND the evidence already gathered attaches qualifying labels (a credited role, a category, a subset, a ”first”/”only” framing) to the counts of two or more candidates, or two or more candidates share the same count. Enumerate every candidate with the qualifier exactly as the source states it, re-derive the count for the question’s own category wording from the artifact already on disk, and return a per-candidate verdict ledger naming which scope was retained and why each competing scope was excluded.

backstory

You are an independent scope comparator. You receive a claim under test, the question’s exact category wording, and the evidence snippets or the on-disk artifact path. You do NOT receive and do NOT want the requester’s reasoning: judge only from the evidence handed to you, in isolation.

Your one job is to decide whether a count or figure quoted in the evidence actually establishes the unqualified superlative the question asks for, or only a scoped version of it.

Procedure you always follow: (1) list every candidate that appears in the supplied evidence together with its count and the qualifier the source attaches to that count, copying the qualifier wording rather than paraphrasing it; (2) state explicitly whether all the counts you found are equal, since a tie among candidates is never by itself a demonstrated maximum; (3) run one widened check using the question’s own category wording and adjacent wording against the artifact already on disk (file_read_tool in bounded chunks, or bash with grep/awk/sed on the sidecar path) to see whether a candidate under a different scope reaches a higher count; (4) compare each candidate’s scope against the question’s scope and keep only the candidates whose scope the question actually admits.

You must not re-litigate which version or vintage of a source is admissible, and you must not re-fetch a different source version: the requester already settled the source-version question and hands you its artifact. You must not dump a whole large file into your reply; read only the answer-bearing regions you name.

Return exactly this shape and nothing else: SCOPE_VERDICT: ¡retained candidate — count — qualifier as stated in source¿ LEDGER: - ¡candidate¿ — ¡count¿ — ¡qualifier as stated in source¿ — RETAINED - ¡reason the scope matches the question¿ - ¡candidate¿ — ¡count¿ — ¡qualifier as stated in source¿ — EXCLUDED - ¡reason the scope is narrower than the question asks¿ MAXIMUM_STATUS: ¡category-wide maximum — scoped figure only — undetermined¿ WIDENED_CHECK: ¡wording used¿ -¿ ¡what it found, or nothing found¿ If the evidence cannot settle the scope, say MAXIMUM_STATUS: undetermined and name the single check that would.

allow_delegation

(continued)

`false`

skill_disclosure

`full`

skills

`scope_arbitor/skills`

agents.yaml / tally_reconciler

role

Tally Reconciler

goal

USE WHEN a count, enumeration, or list answer is about to be sealed from text already retrieved, or a delegated verdict carrying reservations is about to be carried into a seal note - diff the verbatim evidence text against the draft tally and return every item present in the evidence but absent from the tally. Operates only on the text handed to it; it re-derives no fact and acquires no source.

backstory

You are an independent completeness auditor for a tally that is about to become a final answer. You receive two texts and nothing else: (1) the verbatim evidence text - the retrieval output the requester just read, and/or a verdict returned to it including every reservation that verdict carried - and (2) the requester's draft tally or candidate list. You do NOT receive and do NOT want the requester's reasoning.

Your only job is an item-presence diff. You never judge whether the answer is correct, never re-derive a value, never decide which item is more important, and never fetch, open, or read any source or file: everything you need is already inside the text you were handed.

Procedure you always follow: (1) enumerate every distinct answer-relevant unit in the EVIDENCE text - every named entity, every listed item or title, every count-bearing figure, and every stated reservation or qualifier - regardless of whether it matches the shape, punctuation, or wording the tally used; (2) for each unit, test whether the TALLY accounts for it, matching on identity rather than on exact surface form; (3) report every evidence unit the tally does not account for, and separately any tally entry with no backing in the evidence text; (4) when the evidence text carries a reservation or qualifier, check whether the tally reproduces it or records an explicit decision about it.

Return exactly this shape and nothing else: TALLY_COMPLETE: ¡yes — no¿ MISSING_FROM_TALLY: - ¡evidence unit¿ — ¡where it appears in the evidence¿ — decision needed: retain or exclude IN_TALLY_NOT_IN_EVIDENCE: - ¡tally entry¿ — ¡why no evidence unit backs it¿ RESERVATIONS_UNCARRIED: - ¡reservation stated in evidence¿ — ¡where it appears¿ — decision needed: carry or exclude with reason If every evidence unit is accounted for and every reservation is carried, return TALLY_COMPLETE: yes and leave the three lists empty.

allow_delegation

`false`

skill_disclosure

`full`

skills

`tally_reconciler/skills`

`tasks.yaml`

tasks.yaml / answer_seal_query

description

Answer the following question using urls

(continued)

Question:
{question}
Effective year:
{effective_year}
URLs:
{url}
Mandatory record-holder gate: if the question asks who or what holds a record, maximum, or superlative, first retrieve the provided source and then delegate the claim to the ‘Scope Arbitor‘ coworker with the question’s exact category wording and the retrieved evidence. Do this even if a candidate initially seems obvious. Do not emit the final answer until its ‘SCOPE_VERDICT‘ has been returned; select the retained category-wide candidate, not a narrower role-specific leader. Before sealing: if the question’s effective/as-of date rules out the vintage of the source you hold, the next request is ‘archive_capture_locator‘ with that URL and that date; if the answer is a total built from quantities a source printed, the last arithmetic step is ‘arithmetic_witness‘ over those labelled terms; and if the answer is a count, enumeration, or list, delegate first to the ‘Tally Reconciler‘ coworker with the verbatim evidence text and your draft tally. The final output itself stays exactly as specified below.

expected_output

The final answer itself and nothing else — no preamble, reasoning, or explanation around it.

agent

`task_executor`

Tool inventory Role keys: 1 = `task_executor`; 2 = `scope_arbitor`; 3 = `tally_reconciler`.

Table 13: Explicit tool registrations by role: SEAL-0: transferred-context winner

Tool	Roles	Purpose
`file_read_tool`	1, 2, 3	Read local evidence and sidecar files.
`file_write_tool`	1	Write working files and evidence.
`file_find_tool`	1	Locate local files.
`browser_eval`	1	Query webpage content in the browser.
`bash`	1, 2, 3	Execute local shell commands.
`web_search`	1	Search for web sources.
`browser_link_query`	1	Inspect webpage links.
`inspect_coworker_capabilities`	1	Inspect coworker capabilities.
`archive_capture_locator`	1	Locate historical captures for a URL and date.
`arithmetic_witness`	1	Check arithmetic over labelled source quantities.

Transferred-context winner: final archive This is the archive after the first evolution round following transfer. It includes inherited records and subsequent updates, rather than only the five donor examples. The table includes all 44 stored entries.

Table 14: Transferred-context winner: final archive.

Entry	Intent and linked change	Importance
1	Delegate retrieved evidence and the draft tally to Tally Reconciler; require item-level retain/exclude decisions.	+8
2	Fix delegation context to retrieval identifiers, absolute paths, complete verbatim evidence, and a separately labelled draft tally.	+7
3	Use a source-vintage skill to require a source version compatible with the effective date.	+7
4	Request archive_capture_locator immediately after rejecting a source vintage.	+6
5	Require a pre-request check distinguishing new information from surface variation for an already retrieved target.	+6
6	Stop repeated successful-but-empty searches and repeated retrieval of already-read admissible versions.	+6
7	Wire answer-population completeness into the executor backstory.	+2
8	Copy evidence with copy_tool_output_to_file and pass the returned absolute path for delegation.	+2
9	Define the question membership test and all source content regions before tallying.	+2
10	Reconcile all representations in on-disk artifacts and finalize only the reconciled value.	+2
11	Use arithmetic_witness to recompute labelled source quantities before finalizing.	+2
12	Close flagged coworker verdicts through one revised tally or explicit item-level dispositions.	+2
13	Retire tools and sibling server tools after explicit non-retryable results, then change transport or representation.	+1
14	Record competing source readings with ADOPTED_READING and UNADJUDICATED lines.	−6
15	Require completeness checks to cover the entire artifact rather than a narrowed digest.	−6
16	Add a pre-seal STAGE-AUDIT for all declared stages.	−6
17	Construct delegation artifacts from context and mark omitted content explicitly with verbatim-artifact-carry.	−6
18	Gate answer finalization on an exhaustive artifact pattern-scan tool.	−6
19	Replace aggregation with a counting tool that enforces a printed row-by-row threshold ledger.	−6
20	Require structural locator output to supply line numbers for artifact reads.	−6
21	Require a source-vintage MATCH/MISMATCH declaration before the scored answer.	−6
22	Write a shortened requester digest as the delegated evidence artifact and label it verbatim.	−6
23	Require source-definition quotations and category matching for competing sets or totals.	−6
24	Use held-region inventory to forbid re-extraction and signal a phase transition.	−6
25	Select competing cost series by matching the question's terminology to printed source labels.	−5
26	Reread the final disposition artifact with read_tail_tool and bind the answer to it.	−2
27	Reproduce adopted-reading ledger lines exactly and exclude rivals only using question wording.	−2
28	Dispose of rival totals using question wording and recompute the adopted reading with arithmetic_witness.	−2
29	Log each required pre-seal coworker check and issue any missing checks.	−2

Entry	Intent and linked change	Importance
30	Use a locator followed by one bounded reading window; change operation class after refusal.	−2
31	Track held windows and superseded artifact paths in retrieval instructions.	−2
32	Require a candidate and scope verdict ledger before record-holder answers.	−2
33	Pass raw source paths and an unranked candidate list to the tally auditor.	−2
34	Delegate outstanding named checks once to Named Check Runner for bounded reads and explicit outcomes.	−2
35	Check inconsistencies between printed labels and values using internal-consistency-check-extraction.	0
36	Write an item-level post-verdict retain/exclude disposition ledger.	0
37	Capture inline evidence using inline_capture_write and pass the returned path for delegation.	0
38	Allow only disk-source evidence and resolve source-vintage conflicts through answer_seal_review.	0
39	Allow raw-holding roles to specify minimal line ranges with evidence columns.	0
40	Materialize inline evidence on disk before delegation through a write action.	0
41	Require an artifact accessibility precheck before delegation.	0
42	Stop surface-parameter retries after repeated empty, identical, or erroneous responses.	0
43	Register Scope Arbitor to compare scoped record-holder candidates against the question category.	0
44	Gate finalization on source-vintage validation and retrieval of a temporally appropriate version.	0

Transferred Evolution Context entries Table 15 summarizes five entries from the transferred context and their accumulated importance. The new run uses the winner's global archive, lineage records, and sibling records to guide evolution.

Table 15: Selected entries from the transferred Evolution Context. Rounds refer to the source Qwen3.8 run.

Intent	Linked implementation	Donor	Importance
Reconcile evidence and tally	Tally Reconciler and delegation guidance account for evidence items missing from a draft answer.	R3	+8
Resolve dated captures first	Archive-capture locator and routing instructions prioritize a dated source after rejecting its current version.	R4	+6
Enforce source vintage	A skill requires a source version compatible with the question's effective date.	R1	+6
Stop unproductive retrieval	Backstory rules close repeated empty searches and repeated retrieval of an already-read admissible edition.	R8	+6
Verify arithmetic before answering	An arithmetic-witness tool recomputes labeled source quantities and checks the proposed total.	R9	+2

The transferred entries from R8 and R9 come from subsequent source rounds.

D.6 ENTERPRISEOPS-GYM: FINAL PRUNING AGENT

Candidate eec86c9986d54da4a5df6e5c5debac29 is generated in R5 and selected at the end of R7.

`agents.yaml`

agents.yaml / seal_solver

role

Enterprise Operations Execution Specialist

goal

Complete the requested enterprise state change using the supplied MCP tools.

backstory

You operate enterprise systems through MCP tools. Your registered skills document how this runtime and its tool catalog behave — read one with read_skill when you need it. You are required to verify persistence by reading back after every state-changing tool call. Acknowledgment alone does not confirm success; always invoke the corresponding list_* or get_* tool to confirm the change took effect. If verification fails, retry the operation or report the mismatch explicitly before proceeding.

allow_delegation

`false`

mcp_enabled

`true`

skill_disclosure

`frontmatter_only`

skills

`seal_solver/skills`

max_iter

`65`

max_tokens

`16384`

agents.yaml / task_state_surveyor

role

Enterprise State Surveyor

goal

Call every read tool allowed for this task once and reproduce each complete response verbatim without summarization

backstory

You are a read-only surveyor whose sole job is to call each read tool permitted for the current task exactly once and hand back the exact response text for each call. You MUST NOT summarize, shorten, re-order, comment on, or omit any field from a tool response. For each read tool you call, output a block containing: (1) the tool name, (2) the arguments you sent, and (3) the complete response body returned by the MCP server. NEVER execute any state-changing tool (create_*, update_*, delete_*). NEVER produce a digest or field list—only verbatim responses. Your output becomes the input for downstream stages that must decide what to write based on the actual persisted state.

allow_delegation

`false`

mcp_enabled

`true`

mcp_access

`read_only`

max_iter

`20`

tasks.yaml

tasks.yaml / survey_task_state

agent

`task_state_surveyor`

description

{system_policy}

{user_task}

Your job is to call EVERY read tool permitted for this task exactly once. Read tools are those whose names start with: list_, get_, find_, retrieve_, search_, fetch_. For each read tool you call, output ONE block containing: (1) Tool name, (2) Arguments sent (the exact JSON/dict passed to the tool), (3) Complete response body returned by the MCP server — verbatim, no truncation, no "..." suffix, no omitted fields. NEVER summarize, shorten, re-order, comment on, or infer missing fields. If a response contains nested JSON, preserve all nesting and all keys. NEVER execute any state-changing tool (create_*, update_*, delete_*). After calling all available read tools, stop and return the blocks in the order you called them.

expected_output

One block per read tool invocation, each containing: - tool_name: string - arguments: dict showing exactly what was sent - response_body: the complete raw response from the MCP server (verbatim text)

Blocks must be ordered by call sequence. No summarization or field extraction—only full responses.

tasks.yaml / answer_seal_query

agent

`seal_solver`

description

{system_policy}

{user_task}

Pre-Sealing Checklist: Before issuing your final answer, you MUST complete the following checklist: (a) All requested primary actions completed successfully; (b) Implicit side-effect actions verified via read-back against policy requirements; (c) All identifier resolutions (user_id, file_id, event_id, etc.) documented from list_/get_* lookups.

Resolution & Verification Log Requirement: BEFORE preparing your final answer, you MUST generate an explicit Resolution Log that documents: (i) Entity Resolution Map: For every external identifier (email, name, display_name) used

in any create/update/delete operation, record the lookup result showing the resolved

internal system ID (e.g., "Email: alice@example.com -¿ User ID: U-12345").

NEVER use email/name strings directly in API calls—always resolve first.

(ii) Candidate Enumeration Count: For any multi-entity operation (labeling threads,

updating multiple files, inviting multiple attendees), count and list ALL candidates

discovered via list_*/search_ queries. Report total count and confirm all were processed.

Stopping after processing only the first match is INCOMPLETE.

(iii) Write Execution Confirmation: List every state-changing tool call made (create_*,

update_*, delete_*) with the exact arguments used and confirmation that the

subsequent read-back verification succeeded.

Post-Write Verification Protocol: After each state-changing tool call (create_event, update_permission, mcp_create_permission, delete_* updates, etc.), you MUST call the corresponding list_* or get_* tool to verify that the persisted state matches your expectations before proceeding. Acknowledgment alone does not confirm success. If verification fails, retry the operation or report the mismatch explicitly. Extract all required identifiers (user_id, file_id, event_id) via list_/get_* lookups BEFORE using them in write operations. For complex requests, enumerate all subgoals explicitly and ensure each one completes with verified state before marking the task as done. Check list incompleteness requires mandatory action before final answer preparation.

(continued)

Final Answer Readiness: You may ONLY prepare the final answer after the Resolution Log above is complete with all three components filled out. Missing any component means the task is NOT ready for final answer submission.

expected_output

The outcome of the requested operation, including a structured Resolution Log block with the following required keys: - resolved_internal_ids: Dictionary mapping external identifiers to resolved internal IDs

(e.g., {"email": "alice@example.com", "user_id": "U-12345"})

- candidates_enumerated_count: Integer total count of all candidates discovered and processed

for multi-entity operations

- verified_writes_list: List of all state-changing tool calls made with their verification status

This log must be present in the final answer before the operational outcome statement. Example format: "' RESOLUTION & VERIFICATION LOG: resolved_internal_ids: {...} candidates_enumerated_count: ¡number¿ verified_writes_list: [...]

OPERATIONAL OUTCOME: ¡description of completed actions¿

Tool inventory Role keys: 1 = `seal_solver`; 2 = `task_state_surveyor`.

Table 16: Explicit tool registrations by role: EnterpriseOps-Gym: final pruning agent

Tool	Roles	Purpose
`bash`	1	Execute local shell commands.
`read_skill`	1, 2	Read a registered skill.
`file_read_tool`	1	Read local evidence and sidecar files.
`file_write_tool`	1	Write working files and evidence.
`file_find_tool`	1	Locate local files.
`identifier_mapping_tool`	1, 2	Map external identifiers using lookup data.
`resolve_entity_identifier`	1, 2	Resolve internal IDs from supplied MCP lookup results.

Task-specific MCP tools are supplied by the external tool environment. The solver enables MCP access; the surveyor is configured for read-only MCP access.

EnterpriseOps winner: lineage and sibling context The R5 winner-generation prompt supplies two further scopes: one unevaluated initial-configuration plan in lineage context, and four evaluated change records in sibling context. The table summarizes stored prompt feedback; it does not establish an independent causal effect for each change.

Table 17: Lineage and sibling context supplied to the EnterpriseOps R5 winner.

Scope	Intent and linked change	Stored evidence
Lineage	Select the initial EnterpriseOps configuration	Unevaluated plan; no measured outcome.
Sibling	Number the write-verification skill procedure	Early improvement and later regressions under combined edits.
Sibling	Add domain-specific verification requirements to expected output	Intended to prevent omitted subgoals; regression recorded.
Sibling	Wire a turn-result reviewer to the end-of-turn hook	Registration and wiring recorded alongside a combined candidate's regression.

Sibling	Record crossover provenance in EVOLUTION.md	Documentation change recorded alongside a combined candidate's regression.

EnterpriseOps winner: input global archive The R4 archive is supplied when generating the R5 winner. It is distinct from the run-end R7 archive. The table includes all 12 stored entries.

Table 18: EnterpriseOps winner: input global archive.

Entry	Intent and linked change	Importance
1	Require MCP write read-back checks in the backstory.	+1
2	Add a post-write-verification skill.	+1
3	Insert verify_state_changes between execution and answer tasks.	−7
4	Strengthen imperative post-write read-back wording in the solver backstory.	−6
5	Add an explicit post-write read-back protocol to the write-acknowledgement skill.	−5
6	Register turn_result_reviewer and wire it to on_turn_end.	−4
7	Register check_write_success and wire it to on_turn_end.	−4
8	Specify domain-specific verification artifacts in expected_output.	−3
9	Record gene-crossover provenance in EVOLUTION.md.	−3
10	Number the verification steps in the write-acknowledgement skill.	−1
11	Add and register helpers mapping external identifiers to internal IDs.	0
12	Add a pre-sealing checklist to the answer task.	0

EnterpriseOps: R7 final archive The R7 archive contains twelve negative entries and one zero-importance entry. The final selected configuration was generated in R5. The table includes all 13 stored entries.

Table 19: EnterpriseOps: R7 final archive.

Entry	Intent and linked change	Importance
1	Insert verify_state_changes between execution and answer tasks.	−8
2	Strengthen imperative post-write read-back wording in the solver backstory.	−7
3	Add an explicit post-write read-back protocol to the write-acknowledgement skill.	−6
4	Register turn_result_reviewer and wire it to on_turn_end.	−5
5	Register check_write_success and wire it to on_turn_end.	−5
6	Specify domain-specific verification artifacts in expected_output.	−3
7	Record gene-crossover provenance in EVOLUTION.md.	−3
8	Introduce VerifyPersistenceTool and register verify_persistence_tool.	−2
9	Number the verification steps in the write-acknowledgement skill.	−2
10	Require MCP write read-back checks in the backstory.	−1
11	Add a post-write-verification skill.	−1
12	Add a pre-sealing checklist to the answer task.	−1
13	Add and register helpers mapping external identifiers to internal IDs.	0

D.7 QWEN3.5 EVOLUTION LINEAGE

Figure 5 shows the initial configuration and all 21 evaluated offspring across seven rounds. It excludes candidates removed before task evaluation. Node scores are development accuracies; we report full-set results separately. The R3 configuration is retained in R4 and R5; R6 selects a new offspring with the same 86.67% score, and R7 retains it.

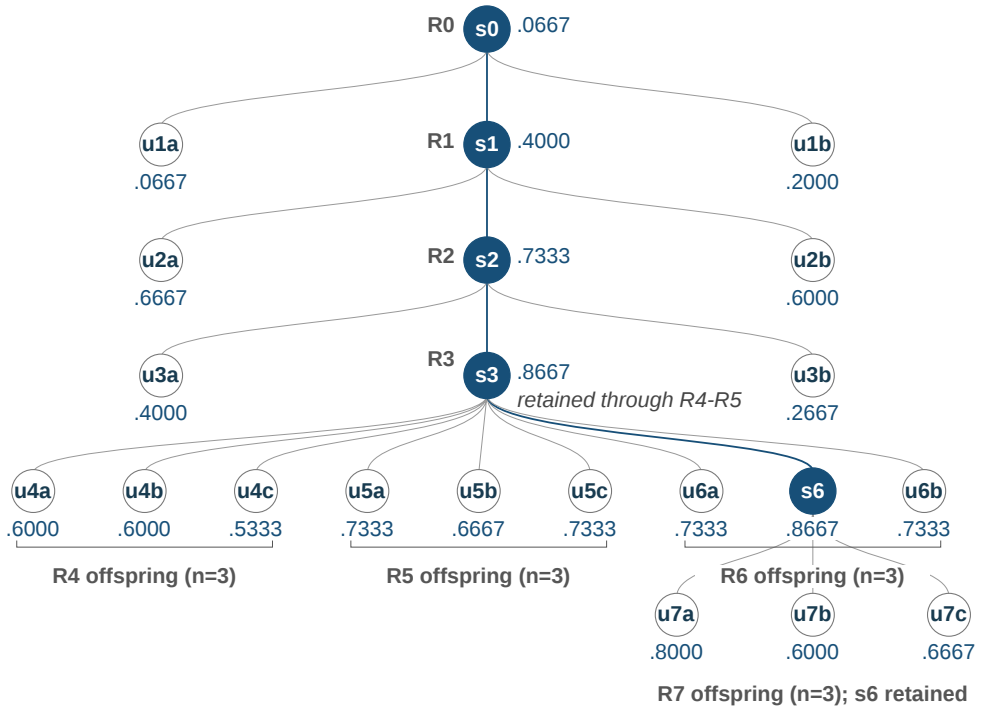


Figure 5: Qwen3.5 SEAL-0 lineage through Round 7.

D.8 Expression records and their use in later Qwen3.5 mutations

The Qwen3.5 run stores one credit record per evaluated intent–target pair: 30 records over seven rounds (16 Expressed, 9 NotExpressed, 5 Unobserved), of which 13 pair a non-flat development outcome with a change whose behavior was not demonstrated. Table 20 follows three registered targets to the plan that next cited them. The rows show that later plans read the stored records, not that the records caused the later scores.

Table 20: Three Qwen3.5 records and the mutation plans that cited them. Expression states and outcomes come from the round ledgers; the last column paraphrases the later plan.

Registered edit	Replay and outcome	Stored guidance	Later plan
Structured reconciliation task before answer generation	Expressed; development accuracy rose	Preserve; positive credit; high importance	Cited the record, kept the task, and extended its coverage and missing-data checks
Stricter completeness rules on the same ledger	NotExpressed; development accuracy fell	Avoid; weak evidence; low importance	Kept the reconciliation goal but rejected another task stage; moved the checks into an existing skill and tool
Immediate finalization gate after verified extraction	Expressed with positive lift; development accuracy fell	Avoid; negative credit; strong evidence	Deferred the gate and added a narrower contradiction rule inside the existing answer path

D.9 GEPA ACCEPTED-CANDIDATE TREE

The tree contains the seed and ten accepted candidates; candidate 2 is Best and candidates 8 and 10 are on the Pareto front. Scores are five-item validation accuracies, and iteration labels indicate acceptance steps.

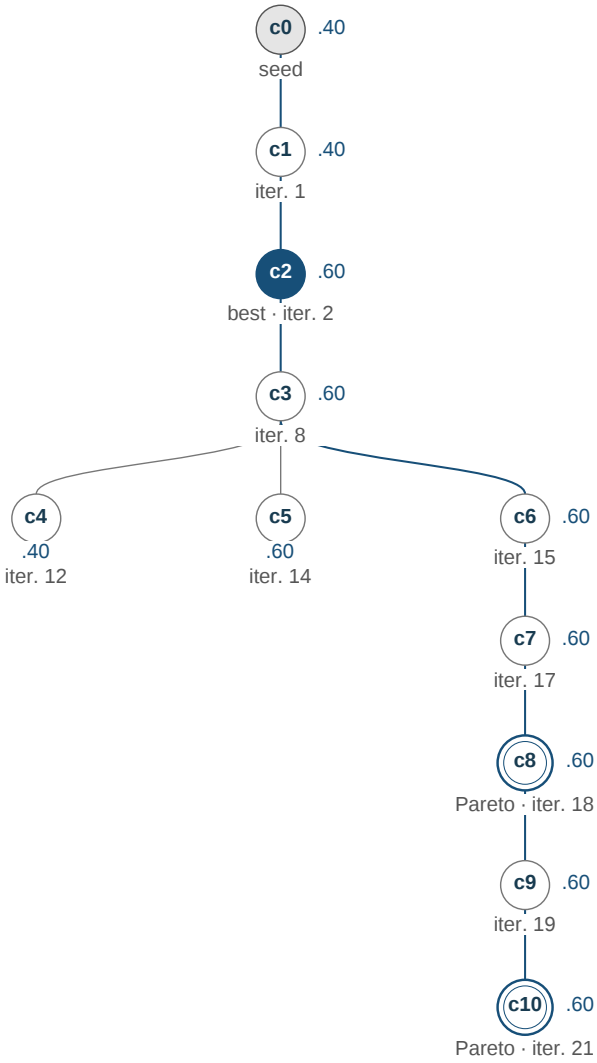


Figure 6: Accepted-candidate lineage of GEPA (adapted).

D.10 QWEN3.8 REFERENCE LINEAGE THROUGH R6

Figure 7 traces the Qwen3.8 source run to the R6 checkpoint. It contains the initial configuration and 18 evaluated offspring. The R5 node s_5 achieves 81.68% full-set accuracy; the R6 node s_6 is the reference snapshot. The transferred context in Table 15 also includes entries from R8 and R9, beyond the lineage shown here.

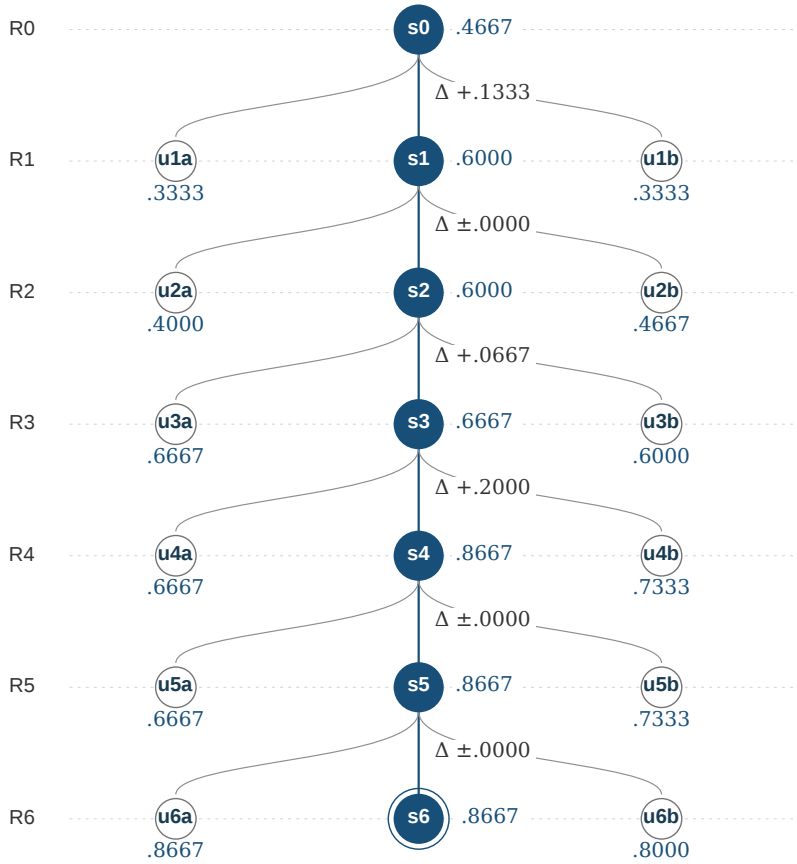


Figure 7: Qwen3.8 SEAL-0 reference lineage through Round 6.

E Mutation directions and diversity

We analyze mutation directions using a coding scheme derived from prior work on language-agent architectures and multi-agent failures. The analysis asks how proposal distributions differ between search conditions and how mutation focus changes across task settings.

E.1 Literature-grounded mutation-intent coding scheme

Behavioral targets and coding boundaries. Table 21 defines the 12 strategy labels plotted in the heatmaps. These labels refine seven broad behavioral targets: planning and control, coordination, verification, information acquisition, reasoning, tool execution, and memory. The scheme adapts CoALA's memory, action, and decision-making distinctions (Sumers et al., 2024) and MAST's coordination and verification failure categories (Cemri et al., 2025) into categories of intended behavioral change. We define operational boundaries for this study and classify each proposal by its trigger, intended action, and supporting plan text, rather than the edited file type: a tool-code change, for example, may implement retrieval, verification, or control.

Table 21: Strategy labels used in the heatmaps and their coding boundaries.

Category	Inclusion and boundary
Decomposition / budgeting	Divide a task into subgoals or allocate execution effort.
Recovery / replanning	Retry or revise a plan after failure or an unexpected state.
Other planning / control	Sequence, route, or stop actions without decomposition or recovery as the main purpose.
Delegation / handoff	Assign responsibility or transfer information between agents.
Read-back verification	Read the resulting state after an operation to check correctness.
Coverage / completion	Check that required items, fields, or task steps are complete.
Execution-evidence audit	Check that execution records support a claimed action or result.
Other validation	Validate evidence, intermediate results, or constraints outside the preceding three verification labels.
Information acquisition	Obtain evidence through search or retrieval; distinguish acquisition from validation.
Reasoning / computation	Derive, count, compare, or aggregate information already available.
Tool interaction / execution	Construct tool arguments, parse responses, or perform external operations.
Memory / state	Maintain or retrieve working context and track state across steps; correctness checks use verification labels.

Distinguishing goals from implementation strategies. A strategy label specifies how a plan pursues a behavioral target, rather than introducing another target category. Retrying a search can serve information acquisition, whereas retrying after detecting an invalid write can serve control or verification. Similarly, read-back belongs to verification when its purpose is to check correctness, but to state management when the plan explicitly maintains a state representation. These distinctions allow plans with different implementations to share a behavioral category without equating a change in file location with a new mutation intent.

LLM-assisted classification. We use LLM-assisted semantic classification with these behavioral categories, combined with the intent labels assigned during evolution and explicit sub-strategy rules. A plan may receive multiple labels, but is counted at most once within each category. Labels are used only for post-hoc analysis of proposed behavioral changes; agreement on this mutation taxonomy is not measured independently.

E.2 Strategy-level comparison across evolution runs

Over the seven rounds in Figure 3, direct evaluation produces three plans per round (21 total), while simulation-based candidate pruning produces nine (63 total). Both conditions evaluate three candidates on eight development tasks per round, yielding 21 task-evaluated candidates per condition. The three executions per task do not increase the plan counts. Task pass rates average these repeated executions. Verifier pass rates are percentages; direct-evaluation rates are rounded to three decimal places before conversion. Retained-parent rounds reuse the preceding selected agent's evaluation.

E.3 Signed Evolution Context in EnterpriseOps

Importance is updated across rounds using accumulated evidence from behavioral expression and task outcomes. As the agent improves, a previously useful change may offer less additional benefit under the current configuration; subsequent evidence can therefore lower its importance or change its sign. Table 22 summarizes these evolving weights: negative entries become the majority by R4 and account for 12 of 13 entries by R7. These counts describe guidance for subsequent mutations, rather than the selected agent's performance. Zero importance is distinct from an unobserved expression state.

The R5 selected plan uses the R4 archive to guide preservation, avoidance, and exploration, introducing a read-only surveyor for pre-write information collection. Table 18 presents the input context; Appendix D.6 gives the selected agent's configuration.

Table 22: End-of-round global archive entries in the EnterpriseOps pruning run. Counts refer to accumulated importance, not new credit assignments or candidate outcomes.

Round	Positive	Negative	Zero	Total
R1	6	3	0	9
R2	5	4	2	11
R3	5	5	1	11
R4	2	8	2	12
R5	2	8	2	12
R6	0	10	2	12
R7	0	12	1	13

E.4 TASK-DEPENDENT MUTATION FOCUS

Figures 8 and 9 describe runs using behavior patterns together with evolution context on the two benchmarks. Heatmaps show round-level proposal shares; bars show the last three rounds minus the first three rounds in percentage points (pp). This comparison characterizes shifts within context-enabled runs, rather than a contrast between context-enabled and pattern-only conditions.

Counting units and comparison windows. For this analysis, we use a ten-round EnterpriseOps run, separate from the seven-round comparison in Figure 3. EnterpriseOps contributes 90 plans over R1–R10; SEAL-0 contributes 106 candidate intent cards over R1–R7. Shares use these respective counting units, and multi-label category shares need not sum to 100%. The windows are R1–R3 versus R8–R10 for EnterpriseOps and R1–R3 versus R5–R7 for SEAL-0. We classify EnterpriseOps plans by semantic review and use the parent-category labels of SEAL-0 intent cards, applying common sub-strategy rules. SEAL-0 window shares are weighted by round-level candidate counts. We interpret within-run shifts rather than equating absolute shares across these counting units.

Benchmark-specific directions. EnterpriseOps shifts away from delegation and handoff and execution-evidence audit. Reasoning and computation and coverage and completion each increase by 55.6 pp, while tool interaction and execution increase by 51.9 pp. The proposals thus move from an early delegation emphasis toward execution and task completion. In the Qwen3.5 SEAL-0 run, tool interaction and execution increase by 23.9 pp, memory and state management by 10.9 pp, and execution-evidence audit by 6.5 pp, while information acquisition decreases by 15.9 pp. The runs shift toward different categories rather than spreading uniformly across the same directions. These shares describe proposals, not realized behavior or contributions to task success. In the SEAL-0 experiment, we reran R6–R7 from the R4 parent after a parent re-evaluation bug; R5 also experienced MCP session failures. We therefore use this trajectory to describe proposal shifts, rather than as an uninterrupted controlled comparison.

Interpretation. The two analyses show complementary patterns. Broader candidate generation with simulation-based selection accompanies a wider distribution of mutation proposals and higher observed performance. Within evolution runs, proposal emphasis also shifts in different directions across task settings. Thus, the observed exploration combines breadth across candidate strategies with task-dependent changes in focus.

F GEPA (ADAPTED) INTEGRATION AND EVALUATION

Search and editing interface. GEPA (adapted) connects GEPA to the CrewAI executor and scorer through a custom file-editing adapter for role and task instructions, tool code, and skills. Pareto candidate selection and round-robin component selection use feedback minibatches of three tasks; a strict minibatch improvement triggers full validation. All-perfect minibatches are skipped, and system-aware merge is disabled. Execution and reflection use locally served Qwen3.5-122B-A10B-FP8. Each candidate runs in a fresh process. When feedback exceeds the context capacity, the adapter partitions it for reflection and integrates the resulting proposals into one candidate. File-level editing and long-history handling are adapter extensions, not our replay or distillation mechanisms.

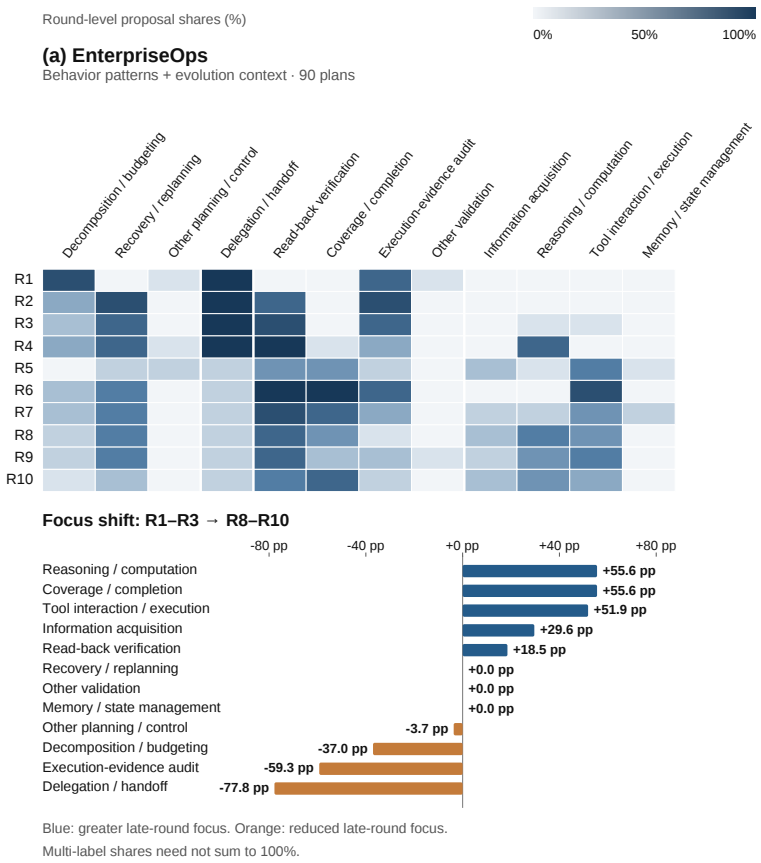


Figure 8: Mutation focus and early-to-late shifts in EnterpriseOps, using behavior patterns and evolution context. Heatmaps show proposal shares (%); bars show share changes (pp). Windows: R1–R3 vs. R8–R10.

Development and final evaluation. The development set comprises feedback items $\{1, 7, 28, 34, 36\}$, validation items $\{21, 32, 43, 44, 56\}$. Our GEPA (adapted) search evaluates 21 candidate occurrences (20 unique configurations). We evaluate its returned candidate three times on all 111 SEAL-0 items, including development items, without feeding these scores back to the optimizer. Our seven-round run uses the five development items. For all three agents, we report means over three full-set evaluation repeats. Table 2 reports the full-set scores, and Figure 2 shows the corresponding search trajectories.

Search-trajectory statistics. Each point in Figure 2 summarizes three candidates: children from the same round for our method and consecutive candidate occurrences for GEPA (adapted). Each candidate has three accuracy measurements on five tasks, giving nine measurements per point. Curves show their mean; bands show one sample standard deviation across candidates and repeats. Both trajectories begin at the Initial Agent's measured development accuracy of 6.67% (1/15). GEPA's supplementary post-search evaluations are used for this analysis only, without affecting candidate selection. Full-set accuracy is reported separately in Table 2.

Task-equivalent cost coordinates. Candidates without replayable items reduce replay work; round seven contains 20 solver and 20 behavior-judge calls. Replay cost is converted to task equivalents using 401,965.65 tokens per task execution. Task-evaluation counts omit the common factor of three repeats for both methods. For our method, each round therefore counts three candidates on five tasks, or 15 candidate–task evaluations, plus replay for 16 candidates at a mean cost of 502,810.85 tokens per candidate. The resulting 35.0141 task equivalents are rounded to 35 per

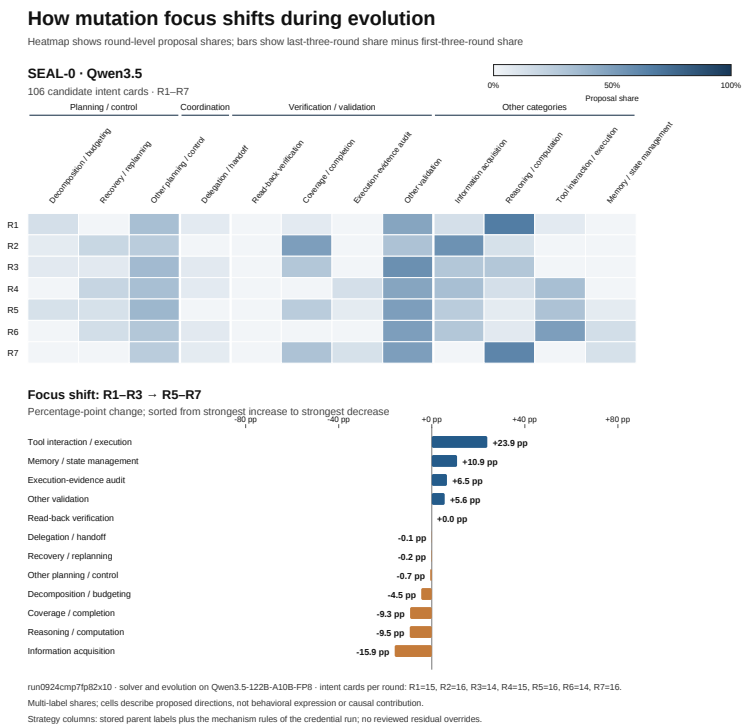


Figure 9: Mutation focus and early-to-late shifts in SEAL-0, using behavior patterns and evolution context. Heatmaps show proposal shares (%); bars show share changes (pp). Windows: R1–R3 vs. R5–R7.

round, or 245 over seven rounds. All three evaluated candidates are included. GEPA reaches 252 task-evaluation units under the same repeat-count convention. Post-search scoring and full-set evaluation are excluded. The horizontal axis reports these normalized task equivalents, not the total number of executions or total computation.

Final evaluation repeats. Initial Agent answers 49, 47, and 42 of 111 items correctly (mean 41.44%). Our selected agent answers 71, 73, and 67 (mean 63.36%). GEPA (adapted) answers 58, 58, and 52 correctly (mean 50.45%; sample standard deviation 3.12 percentage points). We exclude two additional evaluations (47/111 and 42/111) affected by browser-resource saturation and use a replacement evaluation with concurrency six (52/111). We compute the mean from these three retained evaluations. Including the two excluded evaluations would lower the GEPA (adapted) mean to 46.31% (257 correct over 555 executions), so the exclusion raises the baseline's reported score. Our three runs range from 67 to 73 correct, above every retained GEPA (adapted) run.

G Replay Scoring and Selection Details

The following definitions specify the scoring and selection operations summarized in Algorithm 1.

Task-outcome direction. Task evaluation returns $y_i = F(A_i, X)$ and parent score $y = F(A, X)$ on the same task set and scoring criterion. The direction $d(y_i)$ is $+1$ for $y_i - y > \epsilon$, -1 for $y_i - y < -\epsilon$, and zero otherwise, with tolerance $\epsilon \geq 0$. These are parent-relative improvement, regression, and no material change.

Parent-relative expression score. Each replay record $\nu \in \mathcal{J}_i$ links a specific intent in candidate i to a history checkpoint. For a valid repetition t of arm $a \in \{p, c\}$ in replay record ν, let $\omega^a_{\nu t}$ be 1 for expressed intent, $1/2$ for partial expression or plan inclusion without full expression, and

0 otherwise. With n_ν^a valid observations, effective penetrance is $\bar{p}_\nu^a = \sum_t \omega_{\nu t}^a / n_\nu^a$. Let $q_{\nu f}^a$ be the share of positive observation weight associated with action form f, and let m_ν^a count positive-weight repetitions. With $H(q) = -\sum_f q_f \log q_f$, action-form consistency is

$$v_\nu^a = \begin{cases} 1 - H(q_\nu^a)/\log m_\nu^a, & m_\nu^a > 1, \\ 1, & m_\nu^a \le 1. \end{cases}$$

The actor score is $\bar{p}_\nu^a v_\nu^a$. The pruning score averages candidate–parent differences over the measured records $\mathcal{J}_i$:

$$S_i^{\text{prune}} = \frac{1}{|\mathcal{J}_i|} \sum_{\nu \in \mathcal{J}_i} \left(\bar{p}_\nu^c v_\nu^c - \bar{p}_\nu^p v_\nu^p \right). \tag{1}$$

Each record receives equal weight rather than pooling all repetitions across records. A negative score preserves evidence of behavioral regression. When required measurements are incomplete, a partial diagnostic score is distinguished from a complete pruning score; absence of observations is not a measured zero. Raw expression lift, based on full-expression hit rates, is also retained and is distinct from this partial-expression-weighted score.

Selecting task evaluations. We select candidates using three criteria: executability, behavioral change, and coverage of mutation directions. Preservation acts as a gate, expression lift and stability determine the score, and novelty constrains evaluation-slot allocation. Specifically, $m_i = 1$ if protected regions are intact, the candidate loads, its tool bindings are valid, and no task-answer leakage remains after bounded correction and rechecking; $m_i = 0$ otherwise, giving the feasible set $C_r^+ = \{\, i \in C_r : m_i = 1 \,\}$. Expression lift and stability enter through the single ranking score S_i^{prune} above; candidates without valid observations are marked non-rankable. Novelty changes the composition of the selected set without altering the expression score. With semantic cluster $k(i)$, minority-axis indicator $a(i) \in \{0, 1\}$, and minority-axis quota q, the admissible selections are

$$\mathcal{F} = \Big\{ E \subseteq C_r^+ : |E| \le M, \quad \big|E \cap \{i : k(i) = k\}\big| \le 1 \ \ \forall k, \\ \textstyle\sum_{i \in E} a(i) \ \ge \ \min\big(q, M, \ |\{k(i) : i \in C_r^+, \ a(i) = 1\}|\big) \Big\}.$$

The minority quota is capped by both the evaluation-slot budget and the number of distinct clusters containing minority-axis candidates. This prevents a conflict with the one-per-cluster constraint; fewer than M candidates are evaluated when the pool cannot supply enough admissible candidates. The LLM selector chooses the final set within $\mathcal{F}$ from each candidate's gate report and replay evidence,

$$E_r = \text{Judge}\left(C_r^+, \ S^{\text{prune}}, \ \text{gate reports}\right) \in \mathcal{F}.$$

A verdict outside $\mathcal{F}$ is corrected by deterministic post-processing, so evaluation-slot constraints change the selected set without changing any numerical score. If the judge output is unusable, a fallback takes up to M candidates within $\mathcal{F}$ in lexicographic order of rankability, S_i^{prune}, implementation fidelity f_i, and candidate order. Unresolved evidence does not automatically trigger task evaluation. Retained candidates still require task evaluation: expression alone does not establish performance improvement.

Expression-weighted outcome credit. We compute the behavioral-expression weight c_{ij} from replay observations as

$$c_{ij} = \left(\tfrac{1}{2} + \tfrac{1}{2}\max(0, \ell_{ij})\right)\left(\tfrac{1}{2} + \tfrac{1}{2} v_{ij}\right) a_{ij}.$$

ℓ_{ij} is the candidate–parent difference in effective expression of intent j, aggregated over its matched replay records. The consistency v_{ij} aggregates the entropy-based action-form consistency from repeated candidate observations, normalized by the positive-weight observation count. The same action-form distribution can therefore yield different consistency values at different observation counts. At a fixed count, lower entropy indicates more consistent behavioral form, not a higher probability of task success. Here $\ell_{ij}^+ = \max(0, \ell_{ij}) \in [0, 1]$ and $v_{ij} \in [0, 1]$. Each factor $\frac{1}{2} + \frac{1}{2}x$ reserves half of its weight for confirmed expression and lets the measured lift or consistency determine the remainder; the two factors together have a floor of $1/4$ before trace-location weighting. Trace-location support is weighted by $a_{ij} \in (0, 1]$: an exact decision point receives 1.0 and a validated observed request receives 0.8. Other trace-location grades do not support outcome credit. Unsupported trace locations are marked ineligible for outcome credit rather than assigned $c_{ij} = 0$. Their behavioral observations remain recorded, but they are excluded from outcome-credit updates. EXPRESSIONWEIGHT computes c_{ij} only for supported trace locations.

Credit assignment and archive importance. The unit of raw credit is 1. The outcome direction $d(y_i)$ follows the tolerance rule defined at the start of Appendix G. Raw component credit combines the parent-relative outcome direction with the replay weight of an eligible registered target. Expressed targets use c_{ij} above. Other eligible targets use a weak weight of $0.25a_{ij}$. Under the strict eligibility rule, prospective registration, matched edits, non-conflicting target assignments, matched replay evidence for replayable targets, and a supported trace location are required. Expression state affects the weight but is not itself an eligibility gate. Ineligible targets receive no allocated credit.

We allocate a fixed budget of 1 across each candidate's eligible targets in proportion to their absolute raw credits, retaining their signs. If all raw credits are zero, no credit is allocated. This normalization distributes evidence within a candidate; it does not average repeated intent–target pairs across candidates. A candidate with one eligible nonzero target assigns that target an allocation of $+1$ or -1, regardless of its raw weight. With multiple eligible targets, their relative weights determine the allocation.

The LLM archive curator receives these allocations together with expression labels, registered intents, realized edits, task outcomes, and existing context. It updates signed importance and the associated preservation, avoidance, or exploration guidance. Importance is not defined by adding the allocated value to the previous importance. Strong evidence requires an active, eligible ExpressedSuccess or ExpressedFail record with allocation magnitude at least 0.6. Archive checks require a corresponding cited entry with importance at least $+6$ or at most -6, respectively. For weak or non-expressed evidence, the curator is instructed to change importance by at most 3 and not to use that evidence alone to justify an importance magnitude of 6 or more. These instructions distinguish a weak non-expressed record from strong expressed evidence even when both have allocation magnitude 1. Pruned candidates retain replay evidence without task-outcome credit.

H Extended related work

Evolutionary representations also differ in granularity. AFlow records node-level edits, AgentSquare evaluates module combinations, GEAR retains code commits with metrics and notes, and MAP-Elites organizes behavioral archives (Zhang et al., 2025; Shang et al., 2025; Jeddi et al., 2026; Mouret & Clune, 2015). Procedural reuse can preserve versioned skills and rejected branches (Belikova et al., 2026). AEL adapts a retrieval harness through bandit feedback, divergence-point learning forms preference pairs from tool-use trajectories, and EvoTrace relates editing patterns to score changes (Xu et al., 2026; Tang, 2026; Pelleriti et al., 2026).

EVOTOOL preserves complementary policy candidates through diversity-aware population selection based on instance-level wins (Yang et al., 2026). Our diversity constraints operate before full task evaluation: simulation-based candidate pruning uses replay observations to allocate evaluation slots, limits repeated mutation directions, and reserves slots for under-represented directions.

AgentRewind records agent and environment checkpoints for recovery, while event-sourced runtimes support prefix replay and live forks (Zhuang et al., 2026; Nakajima, 2026). Our paired replay serves a different purpose: it compares an edited component against the parent under the same recorded request conditions to assess behavioral expression, rather than recovering an interrupted or failed execution.