PROCEVO: SELF-EVOLVING MULTI-AGENT ORCHESTRATION VIA PROCESS-GUIDED ATTRIBUTION

Anonymous authors
Paper under double-blind review

ABSTRACT

LLM-based multi-agent systems increasingly rely on learnable orchestration to coordinate specialized agents for complex tasks. Self-evolving orchestration further seeks to improve the orchestration policy from accumulated execution experience, making effective credit assignment over past trajectories critical. However, a single task- or trajectory-level outcome cannot distinguish the heterogeneous effects of individual agent transitions: the same execution may contain useful, redundant, and harmful transitions. Inspecting local state changes provides finer-grained information, but a state change alone does not reveal whether a transition constitutes meaningful progress, since repetition or unsupported reasoning may also alter the execution state. Therefore, we propose **ProcEvo**, a framework for self-evolving multi-agent orchestration that performs transition-level process attribution. ProcEvo first compares the task states before and after each agent invocation to estimate its local effect. It then uses an external LLM to extract evidence for a checklist instantiated with the task, applying relevance, consistency, support, and advancement to assess the evidence. The resulting process evidence is converted into transition-level process credit and combined with terminal supervision to optimize the learnable orchestrator. Across evaluated subsets of MMLU-Pro, GSM-Hard, and GPQA Main, ProcEvo achieves an average accuracy of **72.76%**, compared with **67.47%** for the highest-averaging competing baseline. Component ablations and process-level analyses further highlight the distinct roles of progress estimation by comparing task states and checklist verification. The implementation is available in the supplementary anonymous repository: `https://anonymous.4open.science/r/ProcEvo-2EC7`.

1 INTRODUCTION

Large language models (LLMs) have enabled multi-agent systems to solve complex tasks by coordinating agents with complementary roles and capabilities. Early systems mainly relied on manually designed collaboration protocols. Recent studies increasingly treat agent orchestration as an adaptive component that can be optimized through workflow search, routing, and reinforcement learning (Zhuge et al., 2024; Zhang et al., 2025b;a; Yue et al., 2025; Dang et al., 2025; Ke et al., 2026). Building on this trend, self-evolving orchestration seeks to improve collaboration policies from accumulated execution experience. This direction is related to recursive self-improvement (RSI), which examines how AI systems turn their own experience and evaluation feedback into persistent improvements in behavior or, in more general settings, in the mechanisms that generate subsequent improvements (Chen et al., 2026a; Zhang et al., 2026b). In multi-agent orchestration, the improvement target is more specific: the orchestrator uses traces generated by its current agent-selection policy to inform future orchestration decisions.

Prior work has optimized collaboration graphs and workflows, learned adaptive routing policies, and developed process-level evaluation of intermediate decisions (Zhuge et al., 2024; Zhang et al., 2025b; Yue et al., 2025; Dang et al., 2025; Xi et al., 2026; Cheng et al., 2026). These advances provide complementary tools for designing, adapting, and assessing multi-agent collaboration.

Nevertheless, closing this policy-level self-improvement loop requires reliable feedback on how individual orchestration decisions contribute to the evolving task-solving process. Figure 1(a) illustrates this issue with a simple multi-agent reasoning example.

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

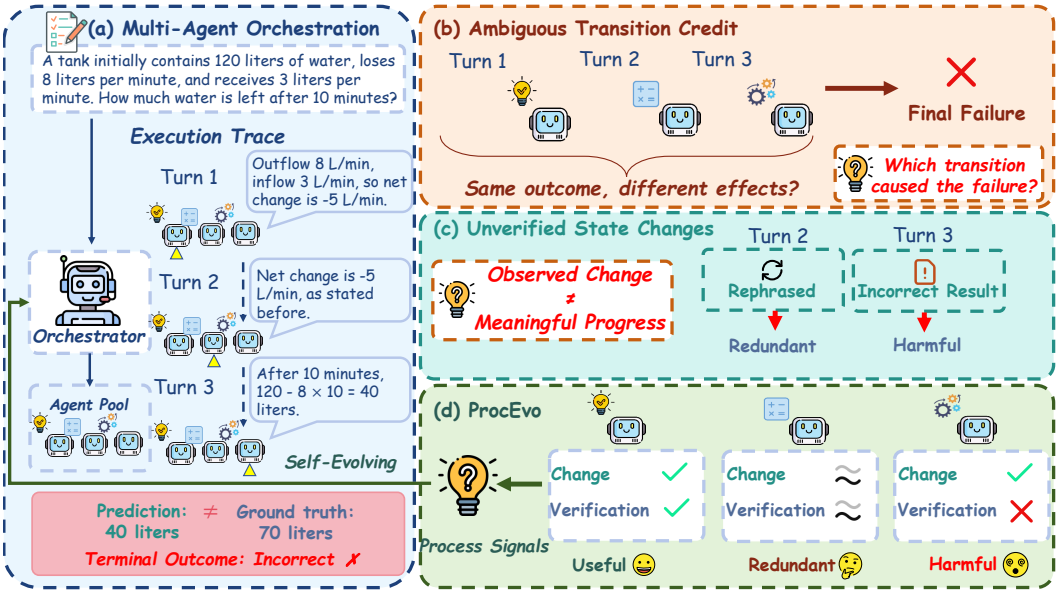

Figure 1: **From ambiguous transition effects to process-guided self-evolution in multi-agent orchestration.** (a) An orchestrator selects agents from an agent pool as the task state evolves, producing an execution trace with useful, redundant, and harmful transitions. (b) A single terminal failure does not reveal the contribution of each orchestration decision. (c) Observed state changes are insufficient to establish meaningful progress: rephrasing existing information is redundant, while introducing an incorrect result can harm the evolving solution. (d) ProcEvo evaluates individual transitions through state changes and checklist-based verification, then uses the resulting process feedback to improve the orchestrator for subsequent executions.

At each turn, the orchestrator selects an agent from the available agent pool according to the current task state. The first invocation correctly derives a net flow of -5 L/min, the second merely restates the same intermediate result, and the third ignores the inflow term and produces an incorrect final answer. Although the execution eventually fails, the selected agent transitions have different effects on the evolving solution: the first is useful, the second is redundant, and the third is harmful. As shown in Figure 1(b), coarse outcome supervision collapses these heterogeneous effects into the same terminal failure. The final outcome reveals that the execution was unsuccessful, but not which orchestration decisions advanced the solution, merely repeated existing information, or damaged it. This motivates the first challenge: *how can heterogeneous transition effects be distinguished within an execution that receives only coarse outcome supervision?*

A natural way to obtain more local information is to inspect how the task state changes after each agent invocation. Comparing consecutive task states provides evidence of how a selected agent alters the current solution, but an observed change alone does not establish meaningful progress. Figure 1(c) highlights this limitation using the same execution trace. The second transition rephrases an already established result and is therefore redundant, whereas the third introduces a new result but is harmful because its derivation ignores a task condition. Both transitions modify the execution state, yet neither advances the solution. More generally, repetition, reformulation, unsupported reasoning, and inconsistent claims may all produce observable changes without reliably advancing the solution. This motivates the second challenge: *how can we determine whether an observed state change constitutes meaningful progress rather than merely altering the execution state?*

To address these challenges, we propose **ProcEvo**, a framework for self-evolving multi-agent orchestration. ProcEvo attributes each agent transition using two complementary forms of process evidence. First, it compares the task states before and after an agent invocation to estimate the local progress induced by that transition. Second, an external LLM extracts structured evidence for a checklist instantiated with the task.

Relevance, consistency, support, and advancement guide the assessment of whether the change contributes to the solution. As illustrated in Figure 1(d), these complementary evaluations help distinguish useful contributions from redundant or harmful changes within the same execution trace.

The resulting process evidence is combined with terminal supervision to optimize the orchestrator, forming a self-evolution loop. In this way, ProcEvo turns intermediate execution information into reusable learning experience: the orchestrator learns not only from whether an execution eventually succeeds or fails, but also from how individual orchestration decisions affect the evolving solution.

Across evaluated subsets of MMLU-Pro, GSM-Hard, and GPQA Main, ProcEvo achieves mean accuracies of 80.29%, 77.20%, and 60.80%, respectively, yielding a three-benchmark average of 72.76%, 5.30 percentage points above the highest-averaging baseline. Component ablations show gains from each process-supervision component, and process analyses highlight their distinct roles in transition attribution.

Our main contributions are summarized as follows:

- We identify two limitations in learning from multi-agent execution traces: coarse outcome supervision cannot distinguish heterogeneous transition effects, while observed state changes alone do not determine whether those effects constitute meaningful progress.
- We evaluate transitions by comparing task states before and after each agent invocation and incorporating historical progress to characterize each decision's local effect.
- We introduce checklist-based evidence verification that tracks support for task requirements across agent invocations and assigns progress credit when accumulated evidence matures.
- We integrate transition-level process credit with terminal supervision to optimize a learnable orchestrator. We evaluate ProcEvo on three complementary reasoning benchmarks and use component ablations and process analyses to assess its components.

2 Related Work

2.1 Self-Evolving Multi-Agent Orchestration

LLM-based multi-agent systems increasingly automate collaboration structures and agent selection. GPTSwarm, AFlow, and MaAS search over graphs or workflows (Zhuge et al., 2024; Zhang et al., 2025b;a), whereas MasRouter and Puppeteer adapt routing or sequential agent selection (Yue et al., 2025; Dang et al., 2025). AOrchestra creates task-specific sub-agents, while MAS-Orchestra trains a holistic orchestration policy (Ruan et al., 2026; Ke et al., 2026).

Self-evolving systems also differ in what is updated. CoMAS improves participating agents using interaction rewards, whereas LEMON assigns counterfactual credit to edits of an executable orchestration specification (Xue et al., 2026; Chen et al., 2026b). ProcEvo updates a sequential agent-selection policy using feedback tied to the observed effect of each invocation on the evolving shared task state.

2.2 Process-Level Credit Assignment and Checklist-Based Evaluation

Process supervision offers intermediate feedback beyond terminal outcomes. Step verifiers and AgentPRM assess reasoning steps or agent actions (Lightman et al., 2024; Xi et al., 2026). AgenTracer localizes failures, while RewardFlow and GraphGPO assign state- or edge-level credit (Zhang et al., 2026a; Feng et al., 2026; Cheng et al., 2026).

In multi-agent settings, MASPRM scores partial transcripts for inference-time search (Yazdani et al., 2025), while Orch-RM learns orchestration-level preferences for reward-guided orchestrator training (Tsang et al., 2026). LLM judges and checklists assess generated outputs (Zheng et al., 2023; Cook et al., 2024; Viswanathan et al., 2025). In sequential agent selection, two challenges must be addressed together: a shared terminal outcome obscures individual contributions, while an apparent state change may be redundant or unsupported. ProcEvo assesses each invocation's effect and verifies task-specific evidence before using that feedback to update the selection policy.

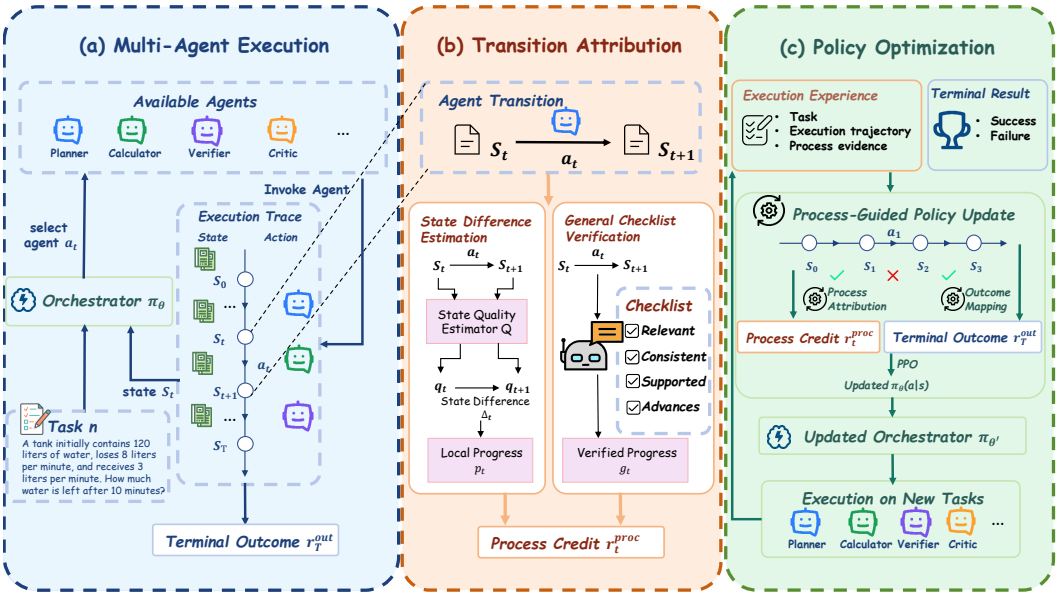

Figure 2: **Method overview of ProcEvo.** (a) A learnable orchestrator selects agents according to the evolving task state, forming a multi-agent trajectory and a terminal outcome. The displayed agent roles are illustrative; the experimental pool is detailed in Appendix A.5 and Table 5. (b) State-difference estimation and checklist verification evaluate individual transitions and produce process credit. (c) PPO uses process credit and terminal feedback to optimize the orchestrator, which is reused on subsequent tasks to form a self-evolving learning loop.

3 METHOD

3.1 OVERVIEW OF PROCEVO

ProcEvo is a framework for self-evolving multi-agent orchestration that performs process-guided attribution over individual agent transitions. Unlike trajectory-level optimization with coarse outcome feedback, ProcEvo estimates the contributions of individual transitions and converts them into process credit for orchestrator optimization. As illustrated in Figure 2, ProcEvo consists of three stages: multi-agent execution, transition-level process attribution, and orchestrator optimization.

During execution, a learnable orchestrator selects agents from an available pool according to the evolving task state, producing an execution trajectory and a terminal outcome. ProcEvo then evaluates individual transitions through state-progress estimation and checklist verification, and combines the resulting process signals with terminal feedback to optimize the orchestrator. The updated policy is reused on new tasks, forming a self-evolving orchestration loop.

3.2 ORCHESTRATION FORMULATION

Existing orchestration methods often optimize agent selection from final task outcomes, which provides limited feedback for individual decisions. ProcEvo therefore models each agent invocation as a transition whose contribution can be explicitly evaluated.

Given a task x, the system maintains a task state s_t containing the task specification and accumulated execution context. A learnable orchestrator π_θ selects an agent from the available pool:

$$a_t \sim \pi_\theta(a \mid s_t). \tag{1}$$

The selected agent operates on the current task state to produce the next state:

$$s_t \xrightarrow{a_t} s_{t+1}. \tag{2}$$

Each invocation defines an agent transition $e_t = (s_t, a_t, s_{t+1})$. Repeated invocations form an execution trajectory $\tau = (s_0, a_0, s_1, \ldots, s_T)$ and a terminal task outcome. ProcEvo attributes process supervision to individual agent selection decisions within this trajectory.

3.3 PROCESS-GUIDED TRANSITION ATTRIBUTION

Terminal outcomes do not distinguish heterogeneous transition effects. ProcEvo therefore evaluates each transition using two complementary signals: state progress measures whether an agent invocation changes the task state toward a better solution, while checklist verification determines whether the observed change is supported by task-relevant evidence. These two signals address different failure modes of transition evaluation: state-based signals may reward unsupported changes, whereas evidence-based verification alone may overlook useful intermediate improvements before sufficient evidence is accumulated.

State Progress. ProcEvo first estimates the local contribution of each agent invocation by comparing the task states before and after the transition. A frozen pretrained reward model scores each state using the task specification and accumulated execution context. Let q_t denote the resulting normalized quality score of state s_t. For transition $e_t = (s_t, a_t, s_{t+1})$, the raw state difference is

$$\Delta_t^{\text{raw}} = q_{t+1} - q_t. \tag{3}$$

A positive value indicates an estimated quality improvement, whereas a negative value indicates deterioration. To reduce sensitivity to evaluator noise, we apply a small deadband to obtain Δ_t, as detailed in Appendix A.7.

An increase relative to the immediately preceding state need not indicate new progress: after a poor transition, the system may merely recover a quality level reached earlier. We therefore distinguish local recovery from surpassing the best state reached so far. The best quality attained before a_t is

$$q^*_{\le t} = \max_{0 \le j \le t} q_j. \tag{4}$$

The resulting state-progress signal is

$$p_t = [\Delta_t]_+ - \alpha[-\Delta_t]_+ + \beta[q_{t+1} - q^*_{\le t} - \delta]_+, \tag{5}$$

where $[\cdot]_+ = \max(\cdot, 0)$, α controls the penalty for state deterioration, and the final term rewards improvements beyond the previously observed best state with margin δ.

The first two terms reward local improvement and penalize deterioration, with the latter scaled by α; the historical-best term adds credit only when the new state exceeds the previous maximum by margin δ. This provides local feedback even within unsuccessful trajectories. However, state improvement alone may reflect repetition or unsupported claims. ProcEvo therefore separately verifies whether the observed change represents meaningful progress.

Checklist Verification. State changes do not always correspond to valid progress: a transition may rephrase existing information or introduce unsupported results. ProcEvo therefore complements state progress with checklist-based verification.

Before execution, ProcEvo instantiates a fixed checklist template with the task specification, or uses a checklist provided with the task:

$$\mathcal{C} = \{(c_i, w_i)\}_{i=1}^{K}, \tag{6}$$

where c_i denotes a requirement and w_i its importance weight. The default template covers task completion, constraints, answer support, and output format. The checklist does not use the gold answer or later trajectory information. Four criteria—*relevance*, *consistency*, *support*, and *advancement*—guide the assessment of new evidence against these requirements.

For each evaluated transition, an external LLM extracts structured evidence, including new claims, changes to the candidate answer, and supporting information. A deterministic reducer accumulates this evidence and updates checklist-item maturity as TENTATIVE, SUPPORTED, or VERIFIED. A newly asserted proposition cannot verify itself through repetition or self-assertion; verification requires qualifying evidence, such as a trusted tool result or independent support from a later transition. The reducer also accounts for conflicting or invalidated evidence, allowing maturity to decrease

when previously established support is undermined. In this process, the LLM identifies evidence, while the reducer determines its contribution to checklist progress.

Let $m_{i,t}$ encode the maturity of item c_i after step t, and let $\mathcal{I}_{\text{credit}}$ denote the items eligible for verified-progress credit. The `task_answer` item does not receive process credit; its correctness is evaluated through terminal supervision. The checklist potential is

$$\Phi_t = \sum_{i \in \mathcal{I}_{\text{credit}}} w_i m_{i,t}, \tag{7}$$

and the positive increase in potential provides checklist-based credit:

$$g_t = [\Phi_{t+1} - \Phi_t]_+. \tag{8}$$

Credit is assigned to the transition that increases accumulated evidence maturity. Repeating unchanged information produces no additional credit, whereas a later invocation can earn credit by establishing support for an earlier claim. Because g_t is nonnegative, only net increases in checklist potential receive positive credit; regressions are not directly penalized by this signal. Negative state progress captures local deterioration, while terminal feedback supervises final correctness. A trajectory-level budget calibrates g_t to $\tilde{g}_t$; evidence rules and calibration appear in Appendix B.

3.4 Orchestrator Optimization

ProcEvo integrates both transition-level signals with terminal feedback to optimize the orchestrator. Sparse, evidence-grounded checklist gains enter as bounded step rewards; denser but noisier state progress enters as a trajectory-calibrated auxiliary advantage. Checklist gains mark the step at which evidence supports a requirement, making them suitable for direct local reward. State progress offers feedback before evidence is complete, but may also favor plausible yet unsupported changes. We therefore calibrate it against trajectory performance and use it to adjust relative policy preferences, reserving terminal feedback for final correctness.

Let r_t^{out} be zero at non-terminal transitions and contain the task outcome at the end of the trajectory. The reward used to compute the reward-derived advantage is

$$r_t = r_t^{\text{out}} + \lambda_v \tilde{g}_t, \tag{9}$$

where λ_v scales the calibrated checklist contribution.

After sampling multiple trajectories for a task, ProcEvo calibrates p_t by its trajectory's relative performance, based on step rewards, terminal outcomes, and trajectory length, to obtain $\tilde{p}_t$. The PPO advantage combines this signal with the GAE advantage computed from Eq. 9:

$$A_t^{\text{ProcEvo}} = \mathcal{N}(A_t^{\text{GAE}}) + \lambda_p \mathcal{N}(\tilde{p}_t), \tag{10}$$

where $\mathcal{N}$ denotes batch-wise standardization of each signal and λ_p controls the state-progress contribution. The combined advantage guides the PPO update of the orchestrator: checklist verification contributes through step rewards, and state progress guides the relative preference for agent selections. Terminal feedback informs both GAE and trajectory-relative calibration.

ProcEvo uses PPO (Schulman et al., 2017) to optimize only the learnable orchestrator; the state-quality evaluator, external evidence extractor, and deterministic checklist reducer remain fixed. The calibration and advantage construction, including their coefficients, are detailed in Appendices A.8, A.9, and A.6.

4 Experiments

4.1 Experimental Setup

We evaluate ProcEvo on three reasoning benchmarks: MMLU-Pro (Wang et al., 2024), GSM-Hard (Gao et al., 2023), and GPQA Main (Rein et al., 2024), covering broad knowledge, numerical reasoning, and expert-level scientific questions. We compare ProcEvo with representative baselines spanning single-agent reasoning, iterative refinement, static and evolving multi-agent systems, and

adaptive orchestration, including Single Agent, Self-Refine (Madaan et al., 2023), MacNet (Qian et al., 2025), EvoAgent (Yuan et al., 2025), Puppeteer (Dang et al., 2025), and AOrchestra (Ruan et al., 2026). All methods use GPT-4.1-mini for task-executing agents and are evaluated on identical instances with the same answer-scoring rules. We report mean accuracy and standard deviation over five independent runs; process diagnostics use logged transitions from the analyzed ProcEvo run. Detailed benchmark descriptions, baseline configurations, and implementation details are provided in Appendix A.

4.2 Main Results

Table 1: Main results on three reasoning benchmarks. Results are mean accuracy (%) $\pm$ standard deviation over five independent runs. Avg. is computed from unrounded benchmark means. † indicates $p < 0.05$ against both Puppeteer and AOrchestra (two-sided Welch tests over five independent runs).

Method	Paradigm	MMLU-Pro	GSM-Hard	GPQA Main	Avg.
Single Agent	Single-agent	73.71 ± 1.63	60.00 ± 1.22	55.00 ± 2.55	62.90
Self-Refine	Self-refinement	79.43 ± 1.63	63.00 ± 1.22	50.20 ± 1.30	64.21
MacNet	Static MAS	65.43 ± 1.20	58.00 ± 2.35	52.40 ± 1.67	58.61
EvoAgent	Evolving MAS	76.57 ± 1.63	66.60 ± 1.34	50.20 ± 1.48	64.46
Puppeteer	RL orchestration	78.29 ± 1.20	64.00 ± 0.71	56.80 ± 1.92	66.36
AOrchestra	Dynamic orchestration	78.00 ± 0.78	70.60 ± 1.95	53.80 ± 1.30	67.47
ProcEvo	Process-guided orchestration	$\mathbf{80.29 \pm 1.57}$	$\mathbf{77.20 \pm 0.84}^{\dagger}$	$\mathbf{60.80 \pm 0.84}^{\dagger}$	**72.76**

Table 1 reports the accuracy achieved by different methods on three reasoning benchmarks over five independent runs. ProcEvo achieves the highest accuracy on all three benchmarks, improving the average accuracy to 72.76%, compared with 67.47% for AOrchestra and 66.36% for Puppeteer. Relative to the Single Agent baseline, ProcEvo improves the average accuracy from 62.90% to 72.76%. The largest improvement is observed on GSM-Hard, where ProcEvo reaches 77.20%, outperforming AOrchestra and Puppeteer by 6.60 and 13.20 percentage points, respectively. This suggests that transition-level process feedback may be beneficial for tasks involving successive reasoning and verification steps. ProcEvo also achieves consistent gains on GPQA Main, reaching 60.80% compared with 56.80% for Puppeteer and 53.80% for AOrchestra. In contrast, the improvement on MMLU-Pro is smaller, where ProcEvo achieves 80.29% compared with 78.29% for Puppeteer and 78.00% for AOrchestra.

We further perform two-sided Welch's t-tests over five independent runs. ProcEvo achieves statistically significant improvements on GSM-Hard and GPQA Main against all compared baselines. On MMLU-Pro, ProcEvo also significantly outperforms AOrchestra and several other baselines, while comparisons with some strong baselines do not reach significance. This may reflect fewer opportunities for orchestration gains on knowledge-intensive questions requiring limited agent interaction, while tasks involving successive reasoning and verification may benefit more from process feedback. Full per-benchmark p-values are provided in Appendix A.3.

4.3 Ablation Study

We examine the two process-supervision components on GSM-Hard using a 2×2 ablation. All variants share the same task instances, agent configurations, and optimization settings, differing only in whether state-progress and checklist-derived supervision are enabled.

Table 2 shows that removing either process-supervision component decreases performance. Removing checklist verification reduces accuracy from 77.20% to 70.40%, while removing state progress results in 71.80%. Removing both components further decreases accuracy to 68.60%. These results indicate that state-progress estimation and checklist verification provide complementary supervision signals. Two-sided Welch's t-tests over five independent runs confirm that ProcEvo significantly outperforms all ablated variants ($p < 0.05$); detailed results are provided in Appendix A.4.

Table 2: Ablation of the two process-supervision components on GSM-Hard. Results are mean accuracy (%) ± standard deviation over five independent runs. † indicates statistically significant differences from ProcEvo ($p < 0.05$, two-sided Welch's t-test over five independent runs).

Variant	State Progress	Checklist Verification	Accuracy
ProcEvo	Yes	Yes	$\mathbf{77.20 \pm 0.84}$
w/o State Progress	No	Yes	$71.80 \pm 3.90^{\dagger}$
w/o Checklist Verification	Yes	No	$70.40 \pm 2.07^{\dagger}$
w/o Both Components	No	No	$68.60 \pm 0.55^{\dagger}$

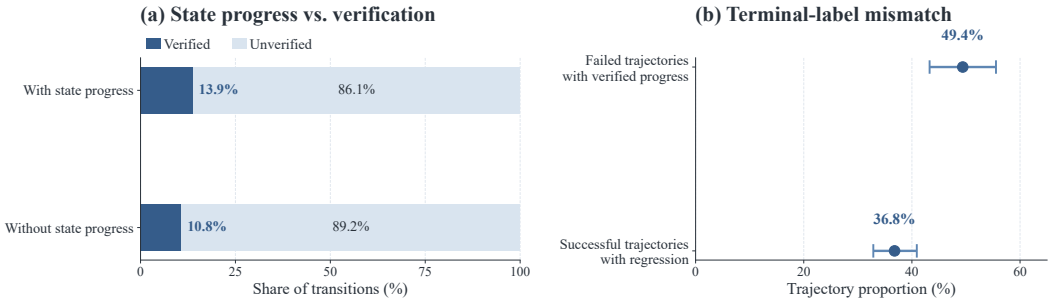

Figure 3: Summary of transition-level process credit. **(a)** State progress and checklist verification are not equivalent. **(b)** Terminal outcomes obscure heterogeneous local process quality. Error bars denote 95% Wilson confidence intervals.

Among the single-component variants, removing state progress produces greater run-to-run variability than removing checklist verification. The full model achieves the highest accuracy and is less variable than either single-component variant.

4.4 Process-Level Credit Analysis

Figure 3(a) compares state progress with checklist-verified credit. Only 13.9% of transitions with positive state progress receive verified credit, while 10.8% of transitions without positive same-step state progress nevertheless receive it. The checklist is therefore not a magnitude threshold on the state difference: it can withhold credit from apparent gains and recognize later evidence that supports previously introduced information.

Figure 3(b) shows that 49.4% of failed trajectories contain verified local progress, while 36.8% of successful trajectories contain semantic regression, such as invalidating established evidence or degrading the candidate answer. Terminal outcomes thus obscure heterogeneous local effects; full diagnostic definitions appear in Appendix C.

Viewed together, these results show that state progress and checklist verification capture complementary transition-level information. State progress measures changes in estimated solution quality, and checklist verification determines whether those changes are supported by task-relevant evidence.

4.5 Checklist Verification Analysis

Panels (a) and (b) of Figure 4 show how verification rate, suppression-event incidence, and mean verified gain vary with state-progress magnitude. Larger estimated state improvements do not automatically yield more verified credit: the checklist can suppress unsupported claims despite positive state progress, while a later transition can verify earlier information without a positive same-step quality change. Thus, verification is not equivalent to thresholding the state-progress score.

Panels (c) and (d) of Figure 4 show the composition of suppression events and the sources of verified credit. Of 4,543 suppression events, 86.5% involve self-certification. Among 759 credit-producing events, 48.6% arise from later novel support and 36.4% from trusted external-tool results.

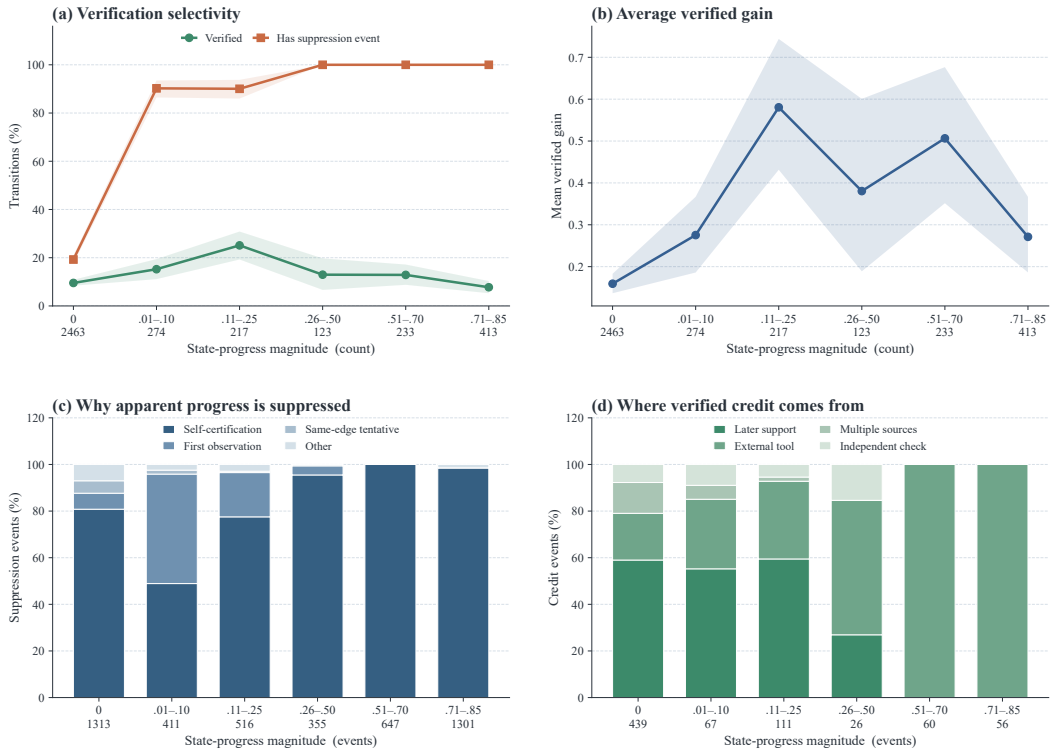

Figure 4: Process-level analysis of checklist verification. **(a)** Verification rate and suppression-event incidence across different state-progress magnitudes. **(b)** Mean verified gain within each state-progress range. **(c)** Composition of suppression events triggered by the checklist. **(d)** Evidence sources that produce verified credit. Counts beneath the horizontal-axis bins indicate the number of transitions or events in the corresponding range. Confidence bands in (a)–(b) are obtained by resampling trajectories.

These event-level counts may include multiple events per transition. Together, the analyses show how the checklist withholds credit from apparent progress and recognizes evidence that matures over subsequent invocations. Additional event counts are reported in Appendix C.4.

Qualitative Case Study. We analyze a failed MMLU-Pro trajectory to illustrate how terminal outcomes can obscure transition-level contributions. The trajectory first derives the correct intermediate result through a Python agent, which satisfies the relevant checklist requirements and receives positive process credit. However, a later Concluder agent selects an incorrect answer option despite retaining the correct intermediate calculation, causing the final execution to fail. The trajectory thus contains a useful transition followed by harmful decisions, even though its terminal outcome records only failure. The complete trace is provided in Appendix C.3.

5 Conclusion

We introduced ProcEvo, a framework for self-evolving multi-agent orchestration via process-guided attribution. State differences and checklist-based evidence verification supervise individual transitions. Together with terminal feedback, these signals train the orchestrator. ProcEvo improves mean accuracy across three reasoning benchmarks; ablations show that both process components contribute to performance, and transition analyses distinguish state change from verified progress. Future work may extend process attribution to long-horizon agentic settings.

AI USE STATEMENT

Generative AI tools were used to assist with research ideation, methodology discussions, code implementation, debugging, literature summarization, and manuscript preparation. AI tools were used to explore research directions, improve implementation efficiency, and refine technical writing. All AI-assisted outputs, including generated code and textual suggestions, were reviewed, modified, and validated by the authors. The authors take full responsibility for the final content, methodology, experimental results, and claims presented in this work.

REPRODUCIBILITY STATEMENT

We provide sufficient methodological, experimental, and implementation details to facilitate reproduction of ProcEvo. The main paper describes the transition-level attribution framework and process-guided optimization procedure, while the appendix provides detailed descriptions of the evaluation protocol, benchmark construction, baseline configurations, model and agent settings, and optimization hyperparameters. We report benchmark results over five independent runs and provide the corresponding statistical analysis and per-benchmark significance tests. The appendix further details the state-quality estimation, progress calibration, checklist construction, evidence extraction, and verified-progress computation used to generate process supervision. All compared methods are evaluated on the same benchmark instances with identical answer-scoring rules, and implementation details necessary for reproducing the reported experiments are provided in the supplementary materials. The anonymized implementation is provided in the supplementary repository.

ETHICS STATEMENT

We have reviewed and adhered to the ICLR Code of Ethics. This work does not involve human-subject studies, the collection of personal or sensitive data, or user profiling. Our experiments are conducted on established reasoning benchmarks and pretrained language models, and focus on evaluating multi-agent orchestration in controlled benchmark settings. As with other LLM-based agent systems, the resulting orchestration policies may inherit errors or biases from the underlying language models and should not be interpreted as establishing safety or reliability for deployment in high-stakes real-world applications. To the best of our knowledge, this work does not introduce additional ethical concerns beyond those generally associated with LLM-based agent systems.

REFERENCES

Mingguang Chen, Licheng Wang, and Bo Qu. Recursive self-improvement in AI: From bounded self-refinement to autonomous research loops, 2026a. URL https://arxiv.org/abs/2607.07663.

Xudong Chen, Yixin Liu, Hua Wei, and Kaize Ding. LEMON: Learning executable multi-agent orchestration via counterfactual reinforcement learning. *arXiv preprint arXiv:2605.14483*, 2026b. doi: 10.48550/arXiv.2605.14483. URL https://arxiv.org/abs/2605.14483.

Xin Cheng, Shuo He, Lang Feng, HaiYang Xu, Ming Yan, Lei Feng, and Bo An. Beyond trajectory-level attribution: Graph-based credit assignment for agentic reinforcement learning. In *Proceedings of the 43rd International Conference on Machine Learning*, 2026.

Jonathan Cook, Tim Rocktäschel, Jakob Foerster, Dennis Aumiller, and Alex Wang. TICKing all the boxes: Generated checklists improve LLM evaluation and generation. In *NeurIPS 2024 Workshop on Language Gamification (LangGame)*, 2024. URL https://openreview.net/forum?id=Q3y6QhOUnI.

Yufan Dang, Chen Qian, Xueheng Luo, Jingru Fan, Zihao Xie, Ruijie Shi, Weize Chen, Cheng Yang, Xiaoyin Che, Ye Tian, Xuantang Xiong, Lei Han, Zhiyuan Liu, and Maosong Sun. Multi-agent collaboration via evolving orchestration. In *Advances in Neural Information Processing Systems*, volume 38, 2025. doi: 10.52202/085713-5502. URL https://proceedings.neurips.cc/paper_files/paper/2025/hash/f1320d2e2842169c6fc89dcbd80e94d0-Abstract-Conference.html.

DeepSeek-AI. DeepSeek-V3.2: Pushing the frontier of open large language models, 2025. URL https://arxiv.org/abs/2512.02556.

Xiao Feng, Bo Han, Zhanke Zhou, Jiaqi Fan, Jiangchao Yao, Ka Ho Li, Dahai Yu, and Michael Kwok-Po Ng. RewardFlow: Topology-aware reward propagation on state graphs for agentic RL with large language models. *arXiv preprint arXiv:2603.18859*, 2026. doi: 10.48550/arXiv.2603.18859. URL https://arxiv.org/abs/2603.18859.

Luyu Gao, Aman Madaan, Shuyan Zhou, Uri Alon, Pengfei Liu, Yiming Yang, Jamie Callan, and Graham Neubig. PAL: Program-aided language models. In Andreas Krause, Emma Brunskill, Kyunghyun Cho, Barbara Engelhardt, Sivan Sabato, and Jonathan Scarlett (eds.), *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pp. 10764–10799. PMLR, 2023. URL https://proceedings.mlr.press/v202/gao23f.html.

Zixuan Ke, Yifei Ming, Austin Xu, Ryan Chin, Xuan-Phi Nguyen, Prathyusha Jwalapuram, Jiayu Wang, Semih Yavuz, Caiming Xiong, and Shafiq Joty. MAS-Orchestra: Understanding and improving multi-agent reasoning through holistic orchestration and controlled benchmarks. *arXiv preprint arXiv:2601.14652*, 2026. URL https://arxiv.org/abs/2601.14652.

Hunter Lightman, Vineet Kosaraju, Yuri Burda, Harrison Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. Let's verify step by step. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=v8L0pN6EOi.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, Shashank Gupta, Bodhisattwa Prasad Majumder, Katherine Hermann, Sean Welleck, Amir Yazdanbakhsh, and Peter Clark. Self-Refine: Iterative refinement with self-feedback. In *Advances in Neural Information Processing Systems*, volume 36, pp. 46534–46594, 2023. doi: 10.52202/075280-2019. URL https://proceedings.neurips.cc/paper_files/paper/2023/hash/91edff07232fb1b55a505a9e9f6c0ff3-Abstract-Conference.html.

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

David Rein, Betty Li Hou, Asa Cooper Stickland, Jackson Petty, Richard Yuanzhe Pang, Julien Dirani, Julian Michael, and Samuel R. Bowman. GPQA: A graduate-level google-proof q&a benchmark. In *First Conference on Language Modeling*, 2024. URL https://openreview.net/forum?id=Ti67584b98.

Jianhao Ruan, Zhihao Xu, Yiran Peng, Fashen Ren, Zhaoyang Yu, Xinbing Liang, Jinyu Xiang, Yongru Chen, Bang Liu, Chenglin Wu, Yuyu Luo, and Jiayi Zhang. AOrchestra: Automating sub-agent creation for agentic orchestration. *arXiv preprint arXiv:2602.03786*, 2026. URL https://arxiv.org/abs/2602.03786.

John Schulman, Philipp Moritz, Sergey Levine, Michael I. Jordan, and Pieter Abbeel. High-dimensional continuous control using generalized advantage estimation. In *International Conference on Learning Representations*, 2016. URL https://arxiv.org/abs/1506.02438.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017. URL https://arxiv.org/abs/1707.06347.

King Yeung Tsang, Zihao Zhao, Vishal Venkataramani, Haizhou Shi, Zixuan Ke, Semih Yavuz, Shafiq Joty, and Hao Wang. Reward modeling for multi-agent orchestration. *arXiv preprint arXiv:2606.13598*, 2026. doi: 10.48550/arXiv.2606.13598. URL https://arxiv.org/abs/2606.13598.

Vijay Viswanathan, Yanchao Sun, Shuang Ma, Xiang Kong, Meng Cao, Graham Neubig, and Tongshuang Wu. Checklists are better than reward models for aligning language models. In *Advances in Neural Information Processing Systems*, volume 38, 2025. doi: 10.52202/085713-3824. URL https://proceedings.neurips.cc/paper_files/paper/2025/hash/a6837c1dd021f76f1b4098e3722052a8-Abstract-Conference.html.

Yubo Wang, Xueguang Ma, Ge Zhang, Yuansheng Ni, Abhranil Chandra, Shiguang Guo, Weiming Ren, Aaran Arulraj, Xuan He, Ziyan Jiang, Tianle Li, Max Ku, Kai Wang, Alex Zhuang, Rongqi Fan, Xiang Yue, and Wenhu Chen. MMLU-Pro: A more robust and challenging multi-task language understanding benchmark. In *Advances in Neural Information Processing Systems*, volume 37, pp. 95266–95290, 2024. doi: 10.52202/079017-3018. URL https://proceedings.neurips.cc/paper_files/paper/2024/hash/ad236edc564f3e3156e1b2feafb99a24-Abstract-Datasets_and_Benchmarks_Track.html.

Zhiheng Xi, Chenyang Liao, Guanyu Li, Zhihao Zhang, Wenxiang Chen, Binghai Wang, Senjie Jin, Yuhao Zhou, Jian Guan, Wei Wu, Tao Ji, Tao Gui, Qi Zhang, and Xuanjing Huang. AgentPRM: Process reward models for LLM agents via step-wise promise and progress. In *Proceedings of the ACM Web Conference 2026*, pp. 4184–4195. Association for Computing Machinery, 2026. doi: 10.1145/3774904.3792551. URL https://doi.org/10.1145/3774904.3792551.

Xiangyuan Xue, Yifan Zhou, Guibin Zhang, Zaibin Zhang, Yijiang Li, Chen Zhang, Zhenfei Yin, Philip Torr, Wanli Ouyang, and Lei Bai. CoMAS: Co-evolving multi-agent systems via interaction rewards. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=ihwAzktmWc.

Milad Yazdani, Mahdi Mostajabdaveh, Zirui Zhou, and Ying Xiong. MASPRM: Multi-agent system process reward model. *arXiv preprint arXiv:2510.24803*, 2025. doi: 10.48550/arXiv.2510.24803. URL https://arxiv.org/abs/2510.24803.

Siyu Yuan, Kaitao Song, Jiangjie Chen, Xu Tan, Dongsheng Li, and Deqing Yang. EvoAgent: Towards automatic multi-agent generation via evolutionary algorithms. In Luis Chiruzzo, Alan Ritter, and Lu Wang (eds.), *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pp. 6192–6217, Albuquerque, New Mexico, April 2025. Association for Computational Linguistics. doi: 10.18653/v1/2025.naacl-long.315. URL https://aclanthology.org/2025.naacl-long.315/.

Yanwei Yue, Guibin Zhang, Boyang Liu, Guancheng Wan, Kun Wang, Dawei Cheng, and Yiyan Qi. MasRouter: Learning to route LLMs for multi-agent systems. In Wanxiang Che, Joyce Nabende, Ekaterina Shutova, and Mohammad Taher Pilehvar (eds.), *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 15549–15572, Vienna, Austria, July 2025. Association for Computational Linguistics. doi: 10.18653/v1/2025.acl-long.757. URL https://aclanthology.org/2025.acl-long.757/.

Guibin Zhang, Luyang Niu, Junfeng Fang, Kun Wang, Lei Bai, and Xiang Wang. Multi-agent architecture search via agentic supernet. In Aarti Singh, Maryam Fazel, Daniel Hsu, Simon Lacoste-Julien, Felix Berkenkamp, Tegan Maharaj, Kiri Wagstaff, and Jerry Zhu (eds.), *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 75834–75852. PMLR, 2025a. URL https://proceedings.mlr.press/v267/zhang25bi.html.

Guibin Zhang, Junhao Wang, Junjie Chen, Wangchunshu Zhou, Kun Wang, and Shuicheng Yan. AgenTracer: Who is inducing failure in the LLM agentic systems? In *The Fourteenth International Conference on Learning Representations*, 2026a. URL https://openreview.net/forum?id=l05DseqvuD.

Jenny Zhang, Bingchen Zhao, Wannan Yang, Jakob Foerster, Jeff Clune, Minqi Jiang, Sam Devlin, and Tatiana Shavrina. Hyperagents, 2026b. URL https://arxiv.org/abs/2603.19461.

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 *The Thirteenth International Conference on Learning Representations*, 2025b. URL https://openreview.net/forum?id=z5uVAKwmjf.

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric P. Xing, Hao Zhang, Joseph E. Gonzalez, and Ion Stoica. Judging LLM-as-a-judge with MT-bench and Chatbot Arena. In *Advances in Neural Information Processing Systems*, volume 36, pp. 46595–46623, 2023. URL https://proceedings.neurips.cc/paper_files/paper/2023/hash/91f18a1287b398d378ef22505bf41832-Abstract-Datasets_and_Benchmarks.html.

Mingchen Zhuge, Wenyi Wang, Louis Kirsch, Francesco Faccio, Dmitrii Khizbullin, and Jürgen Schmidhuber. GPTSwarm: Language agents as optimizable graphs. In Ruslan Salakhutdinov, Zico Kolter, Katherine Heller, Adrian Weller, Nuria Oliver, Jonathan Scarlett, and Felix Berkenkamp (eds.), *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 62743–62767. PMLR, 2024. URL https://proceedings.mlr.press/v235/zhuge24a.html.

A Experimental and Implementation Details

A.1 Evaluation Protocol

Benchmarks. **MMLU-Pro** contains ten-option questions across diverse subjects and emphasizes robust multi-step reasoning (Wang et al., 2024). **GSM-Hard**, introduced with PAL, modifies numerical values in GSM8K problems to stress arithmetic robustness (Gao et al., 2023). **GPQA Main** contains expert-written graduate-level questions in biology, physics, and chemistry (Rein et al., 2024).

We construct a fixed, multi-benchmark evaluation suite by randomly sampling instances from the publicly released MMLU-Pro, GSM-Hard, and GPQA Main benchmarks. The suite comprises 70, 100, and 100 instances from the respective benchmarks. For training, we sample additional subsets of the same size from each benchmark. We independently train the ProcEvo policy five times and evaluate each trained policy on the fixed evaluation subsets. All compared methods are evaluated on the same instances within each benchmark under identical answer-scoring rules. We report mean accuracy and standard deviation over five independent runs per method to characterize run-to-run variation.

For MMLU-Pro and GPQA Main, a prediction is considered correct when the selected answer choice matches the reference answer. For GSM-Hard, we extract the final numerical answer from the generated response and compare it with the reference solution.

A.2 Baseline Configurations

We compare against methods representing direct reasoning, iterative refinement, structured collaboration, and adaptive orchestration:

- **Single Agent** uses the underlying model without multi-agent collaboration.
- **Self-Refine** revises an initial solution using self-generated feedback (Madaan et al., 2023).
- **MacNet** coordinates agents in a directed acyclic network (Qian et al., 2025).
- **EvoAgent** evolves specialized agents into multi-agent configurations (Yuan et al., 2025).
- **Puppeteer** learns a centralized agent-selection policy via reinforcement learning (Dang et al., 2025).
- **AOrchestra** creates and delegates to specialized sub-agents at inference time (Ruan et al., 2026).

Different orchestration frameworks rely on different agent structures, prompts, execution procedures, and internal control mechanisms. We therefore preserve the method-specific configurations of external baselines rather than enforcing identical internal settings that may alter their original behavior. Whenever available, we follow the corresponding implementation and recommended configuration of each baseline. All methods use the same evaluation instances and answer-scoring rules; the comparison measures end-to-end systems with their respective prompts, agent pools, and calling patterns.

A.3 Statistical Significance of Main Results

We compare ProcEvo with each baseline separately on each fixed benchmark subset using two-sided Welch's t-tests over five independent runs per method. The independent run, rather than an individual question or agent transition, is the statistical unit. For MMLU-Pro, calculations use accuracies reconstructed from integer numbers of correct answers among the 70 selected questions; GSM-Hard and GPQA Main each contain 100 selected questions. The tests use sample variances from the same five independent runs summarized in Table 1. Table 3 provides all 18 comparisons, including those that do not reach the 0.05 threshold. All p-values are unadjusted for multiple comparisons. Given the small number of runs, these tests are interpreted as supplementary evidence of run-to-run performance differences.

Table 3: Two-sided Welch t-test p-values for ProcEvo versus each baseline on each benchmark ($n = 5$ runs per method). Bold values indicate $p < 0.05$.

Baseline	MMLU-Pro	GSM-Hard	GPQA Main
Single Agent	**0.00019**	**<0.0001**	**0.0051**
Self-Refine	0.4208	**<0.0001**	**<0.0001**
MacNet	**<0.0001**	**0.000012**	**0.000064**
EvoAgent	**0.0063**	**<0.0001**	**<0.0001**
Puppeteer	0.0550	**<0.0001**	**0.0066**
AOrchestra	**0.0272**	**0.00067**	**0.000024**

A.4 Statistical Significance of Ablation Results

We perform two-sided Welch's t-tests over five independent runs to evaluate whether the performance differences between ProcEvo and its ablated variants are statistically significant. Table 4 reports the corresponding p-values. All comparisons are significant at $p < 0.05$, indicating that both state-progress estimation and checklist verification contribute to the final performance.

Table 4: Statistical significance tests for ablation results on GSM-Hard.

Comparison	p-value
ProcEvo vs. w/o State Progress	0.0347
ProcEvo vs. w/o Checklist Verification	0.00085
ProcEvo vs. w/o Both Components	3.0×10^{-7}

A.5 Model and Agent Configuration

All executable agents in ProcEvo use GPT-4.1-mini as their underlying language model. The agents differ only in their role instructions and available actions. DeepSeek-V3.2 (DeepSeek-AI, 2025) serves as the external evaluator for checklist-based evidence extraction.

ProcEvo operates over a heterogeneous pool of specialized agents. At each step, the learnable orchestrator selects an agent according to the current task state rather than invoking the entire pool. Table 5 summarizes the agent pool used in our experiments.

At each decision step, the orchestrator selects an agent from a set of at most three active agents. The active set may change during execution, and a trajectory can contain up to six agent transitions.

Table 5: Agent pool used by ProcEvo. Each agent is associated with a specialized action and role instruction, while all executable agents share the same GPT-4.1-mini backbone.

Agent	Action	Backbone
FileAgent	`read_file`	GPT-4.1-mini
ArxivAgent	`search_arxiv`	GPT-4.1-mini
BingAgent	`search_bing`	GPT-4.1-mini
WebsiteAgent	`access_website`	GPT-4.1-mini
TerminatorAgent	`terminate`	GPT-4.1-mini
PythonAgent	`run_python`	GPT-4.1-mini
PlannerAgent	`planning`	GPT-4.1-mini
ReasoningAgent	`reasoning`	GPT-4.1-mini
CriticAgent	`critique`	GPT-4.1-mini
ReflectAgent	`reflect`	GPT-4.1-mini
QuestionAgent	`question`	GPT-4.1-mini
SummarizerAgent	`summarize`	GPT-4.1-mini
ConcluderAgent	`conclude`	GPT-4.1-mini
ModifierAgent	`modify`	GPT-4.1-mini

During policy optimization, four candidate trajectories are sampled for each task. These limits control the execution budget while allowing the policy to explore alternative collaboration patterns.

A.6 Optimization Details

ProcEvo optimizes the centralized orchestration policy using Proximal Policy Optimization (PPO) (Schulman et al., 2017) with Generalized Advantage Estimation (GAE) (Schulman et al., 2016). Most optimization settings are shared across benchmarks. The learning rate and a small number of optimization coefficients are adjusted slightly for individual benchmarks. Table 6 reports the representative configuration used in our experiments.

Table 6: Representative optimization configuration of ProcEvo. Learning rates and a small number of optimization coefficients are adjusted slightly across benchmarks, while the overall optimization procedure remains unchanged.

Hyperparameter	Value
Policy learning rate	1.8×10^{-4}
Value learning rate	2.5×10^{-4}
Discount factor γ	0.99
GAE λ	0.95
PPO clip ratio	0.20
PPO epochs	8
Mini-batch size	64
Trajectories per task	4
Entropy coefficient	0.0034
Maximum active agents	3
Maximum trajectory length	6
State-progress weight	1.0
State-progress deadband	0.05
Negative-progress coefficient	0.20
Best-state bonus weight	0.06
Best-state margin	0.02
Process reward weight	0.05
Positive process budget	0.012
Auxiliary process-advantage weight	0.07
Path-return weight	0.70
Terminal-score weight	0.30
Early-stop penalty coefficient	0.22

A.7 State-Quality Estimation and Progress Calibration

ProcEvo estimates the local effect of an agent invocation by comparing the quality of the task states before and after the corresponding transition. At step t, the task state s_t contains the original task description together with the accumulated execution history, including previous agent actions and their intermediate results. The gold answer and terminal correctness label are not included in the state representation.

We use Skywork-Reward-V2-Qwen3-8B as a frozen state-quality evaluator. Given this context, it provides a scalar score $Q(s_t)$, normalized as

$$q_t = \tanh\left(\frac{Q(s_t)}{5}\right). \tag{11}$$

For transition (s_t, a_t, s_{t+1}), the raw state difference is

$$\Delta_t^{\text{raw}} = q_{t+1} - q_t. \tag{12}$$

To reduce sensitivity to small fluctuations in the state-quality estimate, we apply a deadband with threshold $\epsilon = 0.05$:

$$\Delta_t = \begin{cases} 0, & |\Delta_t^{\text{raw}}| < \epsilon, \\ \Delta_t^{\text{raw}}, & \text{otherwise.} \end{cases} \tag{13}$$

Positive and negative changes are treated asymmetrically. The local state-progress signal is

$$p_t^{\text{local}} = [\Delta_t]_+ - \alpha[-\Delta_t]_+, \tag{14}$$

where $[x]_+ = \max(x, 0)$ and $\alpha = 0.20$. This down-weights temporary state deterioration relative to positive progress.

ProcEvo further rewards transitions that move the execution beyond the best state previously reached in the same trajectory. Let

$$q_{\le t}^* = \max_{0 \le j \le t} q_j. \tag{15}$$

The full state-progress signal is

$$p_t = [\Delta_t]_+ - \alpha[-\Delta_t]_+ + \beta\left[q_{t+1} - q_{\le t}^* - \delta\right]_+, \tag{16}$$

where $\beta = 0.06$ controls the best-state bonus and $\delta = 0.02$ is the corresponding improvement margin. The components play different roles. When $|\Delta_t^{\text{raw}}| < \epsilon$, the two local-difference terms contribute zero; the historical-best term is evaluated separately. When $\Delta_t < 0$, the local contribution is negative and scaled by α. When $\Delta_t > 0$ but $q_{t+1} \le q_{\le t}^* + \delta$, the transition may receive local improvement credit but no historical-best bonus. The additional β term is positive only when this margin is exceeded.

This distinction matters when a trajectory first deteriorates and then recovers: the recovery can improve on s_t without surpassing the quality reached before the deterioration. All comparisons use the evaluator's state scores rather than the gold answer or the eventual terminal label. The resulting state-progress signal is calibrated across candidate trajectories and used for the process-guided advantage described in Appendix A.9.

A.8 Process Supervision and Reward Composition

ProcEvo uses state progress and checklist verification to supervise the same agent selection decisions. The two signals enter PPO through complementary pathways. Checklist gains are scaled

and limited by a trajectory-level budget (Section B.5). The calibrated checklist signal $\tilde{g}_t$ enters the step-level reward:

$$r_t = r_t^{\text{out}} + \lambda_v \tilde{g}_t, \tag{17}$$

where r_t^{out} contains terminal feedback and is zero at non-terminal transitions. The representative checklist reward weight is $\lambda_v = 0.05$, and the positive checklist budget is 0.012 per trajectory. State progress instead supplies a calibrated advantage signal, which is combined with the reward-derived advantage as described in Appendix A.9.

Both pathways attach supervision to the decisions that produced the observed transitions. Neither intermediate evaluator receives the gold answer or terminal correctness label.

A.9 State-Progress Advantage Calibration

For each task, ProcEvo samples multiple candidate trajectories. After an execution finishes, its local state-progress signal p_t is calibrated using the relative performance of its trajectory. For candidate trajectory τ_j, the representative trajectory score is

$$S_j = 0.70 \sum_{k \in \tau_j} r_k^{\text{step}} + 0.30 r_j^{\text{terminal}} - 0.10 \frac{|\tau_j|}{L_{\max}}, \tag{18}$$

where r_k^{step} denotes the step-level reward used in the trajectory score, r_j^{terminal} denotes its terminal score, and $L_{\max}$ is the maximum trajectory length. Let z_j denote the relative standardized score of S_j among the completed candidate trajectories for the same task. State progress on that trajectory is then gated by

$$\tilde{p}_t = \sigma(z_j) p_t, \qquad t \in \tau_j, \tag{19}$$

where σ is the sigmoid function. The gated state-progress signal and the reward-derived GAE advantage are standardized within the PPO batch before being combined. The representative coefficient for the state-progress advantage is 0.07. The same calibration and optimization settings are used across component-ablation variants, with each ablated process-supervision component disabled in its corresponding pathway.

B Checklist Verification Details

B.1 Task Requirements and General Verification Criteria

ProcEvo separates *what* a solution must satisfy from *how* evidence of progress is verified.

The checklist specifies the requirements to be satisfied for the current task. ProcEvo uses a fixed template when the task does not provide an explicit checklist. The general verification criteria specify how evidence for those requirements is assessed before it earns process credit. The verification rules remain the same across tasks.

The four general verification criteria are:

- **Relevance**: whether the newly introduced information is directly related to the current task or checklist requirement;
- **Consistency**: whether the information is compatible with established evidence and does not violate known task constraints;
- **Support**: whether the information is supported by sufficient evidence, computation, tools, or independent verification rather than relying only on self-assertion;
- **Advancement**: whether the transition introduces new useful information or strengthens existing evidence toward solving the task, rather than merely repeating already established content.

B.2 Task Checklist Construction

The checklist is set once before trajectory execution using only the task specification. It does not use the selected agent, agent identity, gold answer, terminal correctness label, reward, or future trajectory information.

Table 7: Default checklist template for multiple-choice reasoning tasks. The task description is inserted before trajectory execution.

Item	Importance	Requirement
`task_answer`	Critical	Produce a substantively correct answer to the given task.
`task_constraints`	Critical	Respect every explicit condition, quantity, and constraint stated in the task.
`answer_support`	Major	Provide task-relevant evidence, computation, verification, or reasoning that supports the candidate answer.
`output_format`	Major	Select and clearly identify one of the answer choices supplied by the task.

If a task provides a checklist, requirement list, or evaluation rubric, ProcEvo uses it directly. Otherwise, ProcEvo uses a fixed template with four requirements and inserts the task specification. The checklist remains unchanged during execution: agent outputs update the evidence for its requirements, not the requirements themselves. Later candidate answers cannot redefine what counts as success for the task.

Table 7 shows the default template for multiple-choice reasoning tasks such as MMLU-Pro. Its four items cover answer content, task constraints, supporting evidence, and output format; they are not separate LLM-generated reward scores.

B.3 Transition-Level Evidence Extraction

For each evaluated transition, an external process evaluator based on DeepSeek-V3.2 extracts structured evidence from the task states before and after the agent invocation. The extracted evidence is associated with the task checklist and general verification criteria.

The evaluator receives the task specification and relevant execution context without access to the gold answer or terminal correctness label. It identifies newly introduced claims, candidate-answer changes, and supporting information while distinguishing novel evidence from repeated assertions.

The external LLM performs evidence extraction rather than direct reward assignment. Evidence aggregation, maturity updates, and verified-progress computation are handled by a deterministic reducer. Consequently, checklist credit reflects changes in accumulated evidence rather than the evaluator's direct assessment of an agent response.

B.4 Evidence Maturity and Deterministic Reduction

ProcEvo maintains an ordinal maturity state for checklist-relevant evidence:

$$\textsc{Tentative} \rightarrow \textsc{Supported} \rightarrow \textsc{Verified}. \tag{20}$$

A newly introduced proposition is initially treated as tentative unless qualifying supporting evidence is already available. Subsequent transitions may provide additional evidence and increase its maturity.

A central rule is that agent-generated text cannot verify its own proposition merely by repeating, reformulating, or reasserting it. Such self-certification is explicitly suppressed. Verification requires additional qualifying evidence, such as a trusted external-tool result, multiple independent sources, or an independent later check. A first observation can therefore remain TENTATIVE even if the proposing agent states its conclusion confidently. Similarly, an additional assertion on the same transition does not by itself constitute independent support. Subsequent evidence may promote an existing proposition without introducing an entirely new answer.

The accumulated evidence can also expose a conflict with an earlier claim or invalidate the currently supported candidate answer. Accordingly, the semantic state need not improve monotonically; checklist-related maturity may be revised when contradictory or corrective evidence appears. These changes underlie the semantic-regression events analyzed in Appendix C.2.

This design separates observable state change from evidence-supported progress. A transition may substantially modify the textual state while introducing information that remains tentative or unsupported. Conversely, a later transition may provide independent support that raises the maturity of information introduced earlier. The positive potential difference used as credit is computed only after these evidence updates have been reduced.

B.5 Verified Progress Credit

Following Eqs. 7 and 8, let $m_{i,t}$ denote the maturity associated with checklist item c_i after step t, and let w_i denote its importance. In the verified-progress reducer, importance weights are 3, 2, and 1 for Critical, Major, and Minor requirements, respectively; maturity values are 0, 0.5, and 1 for TENTATIVE, SUPPORTED, and VERIFIED. Only eligible requirements enter verified-progress credit: the `task_answer` item is excluded from the potential. Thus, $\mathcal{I}_{\text{credit}}$ indexes the remaining eligible checklist items, and the accumulated potential is their weighted sum,

$$\Phi_t = \sum_{i \in \mathcal{I}_{\text{credit}}} w_i m_{i,t}. \tag{21}$$

The raw verified progress associated with transition (s_t, a_t, s_{t+1}) is the positive increase in checklist potential,

$$g_t = [\Phi_{t+1} - \Phi_t]_+. \tag{22}$$

Positive checklist credit arises only from a *net* increase in the maturity-weighted checklist potential. To see the distinction, suppose that a transition strengthens the evidence for item c_i without changing the other items. Its potential change is $w_i(m_{i,t+1} - m_{i,t})$, which can contribute positive g_t. By contrast, merely repeating evidence that leaves all maturity states unchanged yields $g_t = 0$. If a transition strengthens one item but simultaneously invalidates another, the signed changes are aggregated *before* applying $[\cdot]_+$, so the credit need not be positive. This net-potential computation is applied after updating the maturity of the eligible checklist items.

To bound the cumulative checklist contribution, let $u_t = \eta g_t$ denote the uncalibrated checklist contribution, with $\eta = 0.002$. The positive budget is $B = 0.012$ per trajectory. For trajectory τ, the calibration factor is

$$\gamma_\tau = \begin{cases} \min\left(1, \dfrac{B}{\sum_j u_j}\right), & \sum_j u_j > 0, \\ 0, & \text{otherwise}, \end{cases} \tag{23}$$

and the calibrated checklist signal is

$$\tilde{g}_t = \gamma_\tau u_t. \tag{24}$$

For $\sum_j u_j > 0$, this calibration ensures $\sum_t \tilde{g}_t = \min(\sum_j u_j, B)$. When no positive checklist evidence is recorded, all calibrated checklist gains are zero. For a trajectory that exceeds the budget, each positive contribution is scaled by the same factor, preserving its share of that trajectory's checklist credit. The budget is therefore applied to the sequence of transition gains rather than to each LLM-extracted event independently.

B.6 Transition-Attribution Procedure

The attribution procedure links the state-based and checklist-based pathways at the same agent transition. The sequence below summarizes the operations described in Sections 3.3 and 3.4; it introduces no additional supervision source.

1. Use the task's explicit checklist or fill the fixed template with the task specification, then initialize its evidence state.
2. Select an agent a_t from $\pi_\theta(\cdot \mid s_t)$ and record the resulting state s_{t+1}.
3. Compare the two state-quality estimates, apply the deadband, and compute p_t using the local difference and historical-best term.
4. Extract checklist-related transition evidence through the external LLM evaluation pathway, and reduce the available evidence into updated maturity states and potential Φ_{t+1}.
5. Attribute the resulting local credit to a_t. After trajectory execution, calibrate positive checklist gains against the trajectory budget and combine the available process signals with terminal feedback.
6. Combine the reward-derived advantage with calibrated state-progress guidance to update the orchestrator with PPO; the evidence evaluator is not updated by this policy step.

C Additional Process Analysis

C.1 Joint Distribution of State Progress and Verification

We provide the transition-level statistics underlying Figure 3(a). Across the analyzed ProcEvo execution traces, we observe 3,723 agent transitions. Among them, 1,260 transitions exhibit positive state progress and 2,463 do not. Checklist verification assigns verified progress to 442 transitions.

Table 8 reports the joint distribution.

Table 8: Joint distribution of state-progress and checklist-verification signals in the analyzed ProcEvo traces.

	Verified	**Unverified**
Positive state progress	175	1,085
No positive state progress	267	2,196

Among transitions with positive state progress, 13.9% receive verified progress credit. Among transitions without positive same-step state progress, 10.8% nevertheless receive verified credit. These results show that checklist verification is not equivalent to thresholding the magnitude of the state-difference signal.

C.2 Local Progress and Terminal Outcomes

We define semantic regression as a transition that degrades or revises previously established semantic task state. Specifically, a transition is flagged as a semantic regression when it causes a checklist requirement to be downgraded, degrades the current candidate answer, introduces a new active conflict, invalidates a previously active claim, or replaces or retracts the current answer. A trajectory is considered to contain semantic regression if at least one of its transitions satisfies this criterion.

At the trajectory level, 49.4% of failed trajectories contain at least one transition with verified local progress. Conversely, 36.8% of successful trajectories contain at least one transition exhibiting semantic regression.

These observations show that trajectory-level success and failure labels do not uniquely determine the quality of individual transitions. Failed executions can contain locally useful decisions, while successful executions can contain locally regressive steps.

C.3 Qualitative Case Study

We present a representative MMLU-Pro execution trace involving straight-line depreciation. A bicycle was purchased for \$80 and sold for \$15 after four years. The correct annual depreciation is \$16.25, with an annual depreciation rate of approximately 20.3%, corresponding to option B. Table 9 summarizes the execution.

Table 9: A representative failed trajectory with useful intermediate progress. Verified gain denotes the raw positive checklist-potential increase before reward calibration.

Step	Agent	Observed Transition	Verified Gain
0	Python	Computes \$16.25 per year and 20.31%, matching option B.	+2
1	FileAgent	File retrieval fails; no new task evidence.	0
2	Concluder	Retains the correct calculation but incorrectly selects option H (\$16 per year, 20%).	0
3	FileAgent	Repeated file retrieval fails; no new task evidence.	0
4	Critic	Rechecks the calculation but incorrectly endorses option H.	0
5	Forced termination	Execution ends with the incorrect candidate answer.	0

The Python agent obtains the correct numerical result and receives positive verified credit. A subsequent Concluder selects option H despite retaining the correct calculation, and the Critic fails to correct this choice. The trajectory ultimately fails, although its initial computation constitutes useful local progress.

C.4 Suppression and Credit Events

Figure 4 in the main text summarizes the verification-selectivity and evidence-event analyses. We provide the underlying event counts below.

We report the event-level statistics underlying Figures 4(c)–(d). Across the analyzed run, the checklist reducer records 4,543 suppression events. Of these, 3,930 (86.5%) correspond to self-certification. The remaining events mainly arise from first observations that remain tentative, same-edge tentative evidence, and other consistency or evidence-scope constraints.

We additionally observe 759 credit-producing evidence events. Among them, 369 (48.6%) arise from later novel support, 276 (36.4%) from trusted external-tool results, 64 (8.4%) from multiple independent sources, and 50 (6.6%) from an independent later check.

A single transition may trigger multiple suppression or credit events. Therefore, event-level counts need not equal the number of unique transitions.