NODEGRAD: ATTRIBUTION-GUIDED RECURSIVE SELF-IMPROVEMENT OF MULTI-AGENT WORKFLOWS

Anonymous authors
Paper under double-blind review

ABSTRACT

Improving a multi-agent workflow requires identifying where to intervene and whether the resulting gain justifies the cost of discovery. End-to-end feedback offers little guidance, while removing a structurally necessary node can invalidate the workflow. We introduce **NodeGrad**, an attribution-guided research loop for recursive workflow improvement. Calibrated, interface-preserving interventions distinguish sensitivity to degradation from upgrade potential, and joint probes reveal conditional benefits between nodes. A language-model controller uses these signals to propose edits, while external tests select versioned updates. Attribution is recomputed after accepted edits so that search follows changing workflow bottlenecks. We further evaluate controller updates through fresh research episodes with equal budgets and fixed evaluation rules. On SWE-bench Lite, repeated attribution improved the resolved rate from 66.0% to 82.3%, and the recursive controller reached 86.0%. Independent patch tests showed that attribution helped prioritize promising nodes, while budget and transfer analyses characterized the conditions under which recursive improvement remained effective. NodeGrad thus connects local diagnosis to tested workflow and controller updates with explicit accounting of research and deployment costs.

1 INTRODUCTION

Language-model agents increasingly operate as workflows combining reasoning, retrieval, execution, verification, and repair (Yao et al., 2023; Schick et al., 2023; Wu et al., 2023; Hong et al., 2024; Qian et al., 2024). Improving a workflow requires choosing which prompt, model route, or tool policy to change. Language models can generate and test modifications, but search effort is limited: the useful question is which intervention improves the complete system.

Automated agent design and workflow search already use language models to construct executable systems (Hu et al., 2025; Zhang et al., 2025). SoL-Pi explores an RSI-inspired process for discovering agent-harness efficiency mechanisms (Liu et al., 2026). These advances motivate a complementary question: *which measurements should direct research toward the nodes and interactions most worth improving?*

End-to-end scores provide limited localization. A failed repair may originate in issue interpretation, retrieval, editing, or test feedback. Deleting a planner or verifier can break downstream execution without revealing its upgrade value. Conversely, a modest individual upgrade may become valuable when a supporting component improves. Structural necessity, sensitivity to degradation, and benefit from an upgrade are different properties.

We propose **NodeGrad**, a research-loop framework guided by *calibrated capability response*. Interface-preserving weak and strong variants measure node fragility and intervention-specific upgrade potential. Local probes check that the intended capability change occurred; selected joint interventions identify conditional benefits. These measurements guide target selection, while a fixed exploration allocation permits revisiting unpromising or unmeasured targets. Calibration does not guarantee that a new prompt or adapter edit will reproduce a model-upgrade effect.

An LLM research workflow connects diagnosis, candidate construction, and test analysis through a versioned ledger. An external runner supplies rewards, costs, and acceptance decisions. Accepted changes become new starting states, and attribution is recomputed as downstream inputs and bottlenecks change. A recursive extension also permits edits to the research workflow, retaining them only

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.

after fresh, equally budgeted episodes show improved research outcomes. Evaluation rules remain fixed; improving the task workflow alone does not establish controller improvement.

NodeGrad separates the cost of discovering an update from the cost of deploying it. Both successful and failed research actions count, including comparisons used to evolve the controller. This distinction makes it possible to evaluate whether a better workflow is economical at its intended usage scale.

Our contributions are threefold:

- A budgeted attribution formulation distinguishing fragility, calibrated response, and joint upgrade value within fixed node contracts.
- An LLM-coordinated research-loop specification connecting interventions, isolated edits, external confirmation, and state-dependent re-attribution, with a bounded recursive extension.
- An evaluation framework covering search efficiency, attribution prediction, controller transfer, and full token accounting. Our evaluation on SWE-bench Lite examines search efficiency, attribution prediction, controller transfer, and token usage.

2 Related Work

2.1 Agentic Workflows and Software Engineering

ReAct and Toolformer connect language models to actions and tools (Yao et al., 2023; Schick et al., 2023); AutoGen, MetaGPT, and ChatDev organize communication among specialized agents (Wu et al., 2023; Hong et al., 2024; Qian et al., 2024). These systems expose different choices about role specialization, shared context, and execution order. For repository-level tasks, SWE-agent, OpenHands, CodeAgent, and Agentless illustrate approaches ranging from interactive tool use to structured repair pipelines (Yang et al., 2024; Wang et al., 2024; Zhang et al., 2024; Xia et al., 2024). SWE-bench provides issue-resolution evaluation with executable outcomes (Jimenez et al., 2024). NodeGrad treats a configured workflow as an object of optimization. Its question is how to allocate improvement effort among existing components while preserving the contracts needed for comparable execution.

2.2 Automated Workflow Search and Recursive Improvement

Automated Design of Agentic Systems represents agents as executable programs and uses a meta-agent to propose new designs from an archive of earlier discoveries (Hu et al., 2025). AFlow formulates code-represented workflow optimization as a search problem, combining Monte Carlo tree search with execution feedback and program modification (Zhang et al., 2025). Both establish that language-generated edits, retained experience, and iterative evaluation can organize agent design. NodeGrad proposes a more restricted search space: graph edges and node contracts are fixed, while measured interventions guide which configurations to edit. This sacrifices topology search in exchange for controlled comparisons of node responses. Restricting a baseline to the same contracts changes that baseline's search space and must be disclosed.

SoL-Pi studies automated discovery of harness mechanisms for efficiency, separating development feedback from evaluation of a frozen harness (Liu et al., 2026). Its discussion presents recursive efficient improvement as a longer-term possibility, rather than evidence of demonstrated compounding returns. NodeGrad similarly distinguishes repeated task-workflow optimization from updating the research process itself. Its controller evaluation compared independently discovered workflows under equal episode budgets and separately accounted for the cost of developing the controller. Matched-budget comparisons isolate the contribution of attribution and assess controller transfer across task groups.

2.3 Attribution, Sensitivity, and Multi-Agent Credit

Influence Functions and TracIn estimate training-data influence (Koh & Liang, 2017; Pruthi et al., 2020); Data Shapley and SHAP relate contribution to marginal utility under a specified valuation problem (Ghorbani & Zou, 2019; Lundberg & Lee, 2017). Applying such ideas to workflow nodes requires defining what an absent component means and how the remaining system executes.

Removing a structurally necessary node can change the task being evaluated, so deletion value need not predict the benefit of improving that node.

Morris screening and Sobol indices characterize black-box responses to input variation (Morris, 1991; Sobol, 2001; Saltelli et al., 2008). This perspective is relevant to NodeGrad's intervention-specific responses and interactions, although an agent configuration is not a continuous scalar input. COMA, VDN, QMIX, and Shapley counterfactual credits address credit assignment in cooperative multi-agent learning (Foerster et al., 2018; Sunehag et al., 2018; Rashid et al., 2018; Li et al., 2021). NodeGrad instead uses deployment interventions to choose research targets without assuming access to trainable agent policies. Local calibration checks the direction of a capability change; it does not yield a unique causal contribution or guarantee comparability between unrelated probe rubrics.

2.4 Language Feedback and Cost-Aware Optimization

DSPy optimizes modular language-model programs against task objectives, while TextGrad propagates language feedback to revise variables in generative systems (Khattab et al., 2023; Yuksekgonul et al., 2025). NodeGrad can use such proposal mechanisms, but asks a separate targeting question: where should a bounded edit budget be spent? Its finite intervention responses are measured before constructing a new patch, whose effect must still be tested. Accordingly, "Grad" denotes a finite response to capability intervention, not parameter differentiation or a textual derivative. The attribution control retains the proposer and verification pipeline, isolating the relationship between diagnostic guidance and independent patch gains.

FrugalGPT and RouteLLM study cost-aware model selection (Chen et al., 2023; Ong et al., 2024). NodeGrad also permits model-route edits, alongside prompts, retrieval, memory, and adapters. Its accounting includes the cost of probing and searching for an improved configuration, not only the resulting deployment. This matters when the intended workload is too small to amortize research overhead. Utility, deployment expense, and total discovery cost therefore remain separate evaluation quantities.

3 Method

Figure 1 summarizes NodeGrad's automated research process, including its editable workflow and external execution service.

Section 3.1 defines the workflow state, objective, and budgets; Sec. 3.2 introduces the LLM research roles; and Sec. 3.3 develops calibrated attribution for target selection. Section 3.4 describes candidate construction and testing, Sec. 3.5 formulates recursive controller updates, and Sec. 3.6 specifies token accounting and final evaluation.

3.1 Workflow State, Objective, and Budgets

Let $H_z = (\mathcal{V}, \mathcal{E}, z)$ be a task workflow with node configurations $z = (z_1, \ldots, z_N)$ comprising model routes, prompts, tool policies, local memory, and adapters. Graph edges and input–output contracts remain fixed. With execution randomness ω, bounded reward $R \in [0, 1]$, and task distribution $\mathcal{D}$, define

$$J(z) = \mathbb{E}_{x\sim\mathcal{D},\omega}[R(x, H_z(x;\omega))], \qquad C_{\mathrm{dep}}(z) = \mathbb{E}_{x\sim\mathcal{D},\omega}[c(x, H_z(x;\omega))]. \tag{1}$$

Repair reward is one only when the external verifier accepts the patch; timeouts and invalid outputs receive zero. A research state is $s_t = (z_t, a_t, \mathcal{L}_t, b_t)$: task configuration, research configuration, ledger, and remaining budget. Starting from a feasible z_0, search improves J subject to deployment cap B_{dep} and research cap B_{res}, stopping at a fixed budget, round, or unsuccessful-attempt limit.

3.2 An LLM-Coordinated Research Workflow

Five roles share a ledger: the **Coordinator** selects actions, the **Diagnostician** requests probes, the **Proposer** forms hypotheses, the **Builder** constructs patches, and the **Test Analyst** interprets runner evidence. Roles may share a backbone with separate prompts. Each record links a parent snapshot, target, hypothesis, patch, budget, execution identifiers, and disposition. The analyst cannot substitute estimated scores for runner outcomes.

A fixed scheduler validates typed action requests, reserves their maximum cost, and owns the accepted-version pointer. Failed actions still consume budget. The LLM roles cannot alter evaluators, partitions,

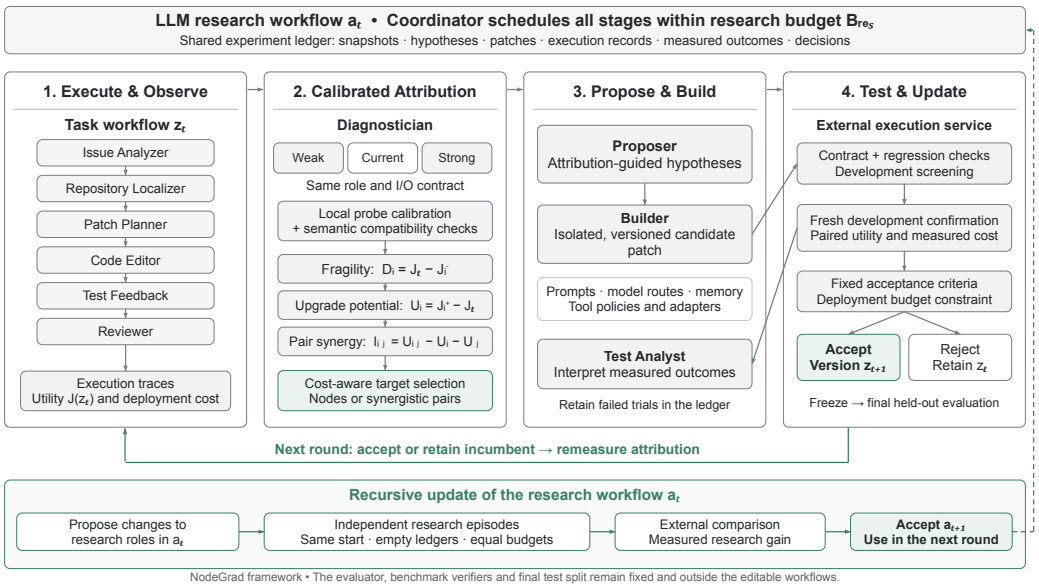

Figure 1: **NodeGrad research loop.** An LLM controller coordinates trace interpretation, attribution, candidate construction, and test analysis. Measured interventions guide target selection; external execution evidence determines version acceptance. Each accepted update triggers new diagnostics. The recursive extension applies this process to the research workflow while holding the external evaluator fixed.

or resource limits. Appendix A.1 specifies node contracts, isolated execution, cache invalidation, and failure handling.

3.3 Calibrated Attribution as Search Feedback

At z_t, weak and strong variants $z_{i,t}^-$ and $z_{i,t}^+$ replace only node i while preserving its interface. They may change model tier, retrieval depth, prompts, or tool feedback. Capability ordering is checked on local probes, not inferred from model names. Each round freezes input packets from the incumbent for paired comparison; accepted edits trigger new packets under the same rubric. With mean probe score $q_i \in [0, 1]$, define

$$\delta_{i,t}^- = q_i(z_{i,t}) - q_i(z_{i,t}^-), \qquad \delta_{i,t}^+ = q_i(z_{i,t}^+) - q_i(z_{i,t}). \tag{2}$$

Probe tasks exclude confirmation and final-test data. Only directions exceeding a fixed minimum capability change and compatibility threshold are eligible. Schema checks are executable; semantic checks use a fixed rubric. Additional proposed probes cannot retroactively change the current comparison.

On matched development tasks and repetitions, paired means estimate the population responses below. Downstream execution is rerun; upstream replay requires matching snapshot and input hashes. Matched seeds do not guarantee deterministic or identical stochastic behavior across providers.

$$D_{i,t} = J(z_t) - J(z_t[i \leftarrow z_{i,t}^-]), \tag{3}$$

$$U_{i,t} = J(z_t[i \leftarrow z_{i,t}^+]) - J(z_t), \tag{4}$$

$$\widetilde{D}_{i,t} = D_{i,t}/(\delta_{i,t}^- + \epsilon), \qquad \widetilde{U}_{i,t} = U_{i,t}/(\delta_{i,t}^+ + \epsilon). \tag{5}$$

Fragility $D_{i,t}$ measures sensitivity to the tested degradation; $U_{i,t}$ measures a particular upgrade. The slopes express response per unit of local probe change, not universal importance or equivalence between different rubrics. Ineligible directions remain unavailable rather than being assigned zero contribution. For selected pairs, define

$$I_{ij,t}^+ = J(z_t[i, j \leftarrow z_{i,t}^+, z_{j,t}^+]) - J(z_t) - U_{i,t} - U_{j,t}. \tag{6}$$

Positive values suggest a joint upgrade worth testing; negative values indicate subadditivity for the tested interventions, without establishing architectural redundancy in general.

For a screened singleton or pair S, write $z_t^{S,+}$ for its joint upgraded configuration. We use the joint measured cost rather than adding single-node costs:

$$G_t(S) = \sum_{i \in S} U_{i,t} + \sum_{i<j,\, i,j \in S} I^+_{ij,t}, \qquad |S| \le 2, \tag{7}$$

$$\Delta C_t(S) = C_{\mathrm{dep}}(z_t^{S,+}) - C_{\mathrm{dep}}(z_t), \qquad V_t(S) = \frac{G_t(S)}{\Delta C_t(S)}, \quad \Delta C_t(S) > c_{\min} > 0. \tag{8}$$

For a measured pair, $G_t(S)$ is its joint response, without higher-order extrapolation. Positive responses and calibrated slopes form a shortlist; joint tests and V_t allocate attention by absolute gain per added cost. A fixed exploration allocation also tests pairs outside the shortlist. Near-zero-cost or cost-reducing variants are handled separately by gain and feasibility. These are target scores, not predictions of an arbitrary new patch. The run manifest fixes shortlist size, exploration allocation, and probe and cost thresholds.

3.4 Candidate Construction and Automatic Testing

The Proposer receives targets, traces, attribution, and rejection history. The Builder implements a bounded edit in an isolated snapshot; paired edits identify the tested interaction they address. The runner applies contract/regression checks, development screening, then a fresh confirmation batch. Generated tests supplement development checks but never replace benchmark verifiers.

Candidate and incumbent are frozen before the confirmation batch and share tasks and execution caps. Let $F_t(c)$ denote mandatory-check success, $L_t(c)$ a one-sided lower bound on paired utility gain, and $C_t^{\mathrm{up}}(c)$ an upper bound on mean deployment cost. A quality-improving candidate is accepted when

$$A_t(c) = \mathbf{1}\{F_t(c) = 1,\ L_t(c) > \tau_J,\ C_t^{\mathrm{up}}(c) \le B_{\mathrm{dep}}\}, \tag{9}$$

where τ_J is fixed before search. Efficiency runs instead test quality noninferiority and cost reduction with declared margins. Appendix A.3 specifies a conservative bound and repeated-testing allocation. Confirmation batches are retired after use and remain development data. If budget, independent samples, or statistical power are insufficient, the candidate stays unconfirmed. Rejection retains the incumbent; acceptance creates an immutable version and triggers new diagnostics. Finite confirmation does not guarantee generalization.

3.5 Recursive Updates to the Research Workflow

Task updates with fixed a_t are iterative workflow optimization. The recursive extension also proposes edits to the five research roles. Their intervention value is evaluated through complete research episodes; local checks concern valid actions and faithful use of evidence. Comparing a_t and a' uses the same starting task snapshot, task distribution, deployment cap, and inner search budget, with separate empty ledgers.

Within each comparison the controllers are frozen and further controller updates are disabled, bounding recursion depth. If $z^{\mathrm{out}}(a, e)$ is the last accepted workflow returned by episode e, define

$$J_{\mathrm{res}}(a) = \mathbb{E}_e\left[J_{\mathcal{D}^e_{\mathrm{endpoint}}}(z^{\mathrm{out}}(a, e)) - J_{\mathcal{D}^e_{\mathrm{endpoint}}}(z^e_{\mathrm{start}})\right]. \tag{10}$$

Endpoint tasks are hidden from inner search agents. The external rule compares paired episode gains and checks the returned workflows' feasibility; an episode with no accepted edit returns its starting workflow. Accepted a' becomes the controller for subsequent research. Episodes used to select controller updates are development data, distinct from final fresh-episode evaluation. Recursion operates on the research workflow configuration while model weights remain fixed.

3.6 Evidence, Cost Accounting, and Final Evaluation

The external evaluator, immutable task snapshots, resource limits, and final test split remain outside both z and a. Development feedback may inform subsequent proposals; final test feedback cannot. We freeze both configurations before a single final evaluation, recording the selected version even if it performs poorly. Any later revision requires a new independent evaluation set.

Research cost uses disjoint event categories:

$$C_{\mathrm{res}} = \sum_t \left(C_t^{\mathrm{analysis}} + C_t^{\mathrm{build}} + C_t^{\mathrm{probe}} + C_t^{\mathrm{test}} + C_t^{\mathrm{recursive}}\right). \tag{11}$$

Each model invocation has a unique event ID and contributes its input-plus-output tokens once. Tool execution and evaluator runtime are logged separately. The recursive category contains both comparison arms and their endpoint tests; these events are excluded from other outer categories. Final reporting usage C_{eval} is recorded separately. For M future tasks, cumulative token usage is $C_{\text{res}} + MC_{\text{dep}}(z_T)$; complete accounting adds C_{eval}. We reported token usage in millions. Historical FRE and node-upgrade counts remain separate resource measures.

4 Experimental Evaluation

Section 4.1 defines the tasks, models, and comparisons; Sec. 4.2 examines workflow utility and token usage; and Sec. 4.3 isolates mechanisms and controller transfer. Section 4.4 relates attribution to patch gains, Sec. 4.5 analyzes research-budget sensitivity, and Sec. 4.6 discusses their implications for Auto Research.

4.1 Questions, Tasks, and Comparisons

Benchmarks and scoring. We evaluated repository repair on SWE-bench Lite (SWE-bench Team, n.d.b). Lite contains 300 test tasks and 23 development tasks. Search, confirmation, controller evaluation, and repository-held-out transfer use disjoint task groups specified by a frozen manifest. The test set remains hidden during search. A pinned official harness evaluates one submitted patch per task (SWE-bench Team, n.d.a); the resolved rate is $100n_{\text{resolved}}/300$. Missing or invalid patches count as unsolved, and infrastructure failures are reported separately. Per-run reports include integer solved counts, seeds, and the averaging rule. Gold patches, hidden tests, hints, and online solutions are unavailable to agents. Lite and Verified use separate denominators and reporting.

Model and execution setup. The research-controller design assigns GPT-5.6-sol (`gpt-5.6-sol`) to five research roles. Task-node variants use Gemini 2.5 Flash, Gemini 2.5 Pro, and Gemini 3 Flash as weak, base, and enhanced candidates, respectively. Local probes determine eligible capability directions at each node. Uniform strong assigns the enhanced variant to every task node. The execution manifest fixes role prompts, tools, reasoning settings, model version IDs, token caps, container revisions, and timeouts. Research-controller and task-model usage occupy separate ledger categories.

The seven comparisons in Figure 2 share the initial workflow, editable fields, proposer model, task access, execution caps, and total search budget where applicable. Unguided search retains the same roles and traces without attribution; random-target search selects nodes uniformly. One-time attribution freezes its initial ranking, whereas repeated attribution refreshes it after acceptance. Saved diagnostic tokens fund additional proposals. Fixed assignment has no search phase.

4.2 Autonomous Improvement under a Common Budget

We evaluated autonomous workflow improvement on the 300 test tasks of SWE-bench Lite. Workflow quality was measured by resolved rate. The research-token cap was 75 million, and the per-task deployment-token cap was 0.15 million. All autonomous methods began with the same workflow, so differences in final utility reflected the workflows produced by their respective research procedures.

Token accounting. We summed input and output tokens over all model invocations, including repeated context, failed attempts, and both arms of recursive controller comparisons. Cached input was counted once within input tokens, and reasoning tokens were counted once within output tokens. Research cost included controller calls and task-workflow replays. Final reporting evaluations and non-model runtime were recorded separately. Aggregate token counts measured usage across models. This accounting charged unsuccessful exploration to the method that incurred it, rather than reporting only the cost of the selected workflow.

Figure 2 reports resolved rate alongside research and deployment token usage. The unchanged workflow resolved 66.0% of tasks. Unguided LLM search and random targeting reached 74.7% and 72.7%, respectively. One-time attribution reached 79.3%; repeating attribution after workflow updates raised the resolved rate to 82.3%. The recursive controller achieved the highest resolved rate, 86.0%, an improvement of 3.7 percentage points over repeated attribution and 20.0 percentage points over the unchanged workflow. The comparison between one-time and repeated attribution indicated

that the useful intervention target changed as the workflow evolved: a ranking computed only at the initial snapshot left some later opportunities unaddressed.

These gains incurred additional research cost. Across the research run and 300 deployments, total token usage was 76.81 million for one-time attribution, 81.66 million for repeated attribution, and 97.14 million for the recursive controller. The uniform-strong reference resolved 80.0% of tasks with 33.57 million deployment tokens. Thus, the recursive controller achieved higher task utility than the uniform-strong reference while using substantially more total tokens once research was included. The separate cost components in Fig. 2(b) showed that research, rather than deployment alone, accounted for much of the difference among the autonomous methods. The uniform-strong reference also provided a distinct operating point: it avoided workflow search, but its resolved rate remained below that of repeated attribution and the recursive controller.

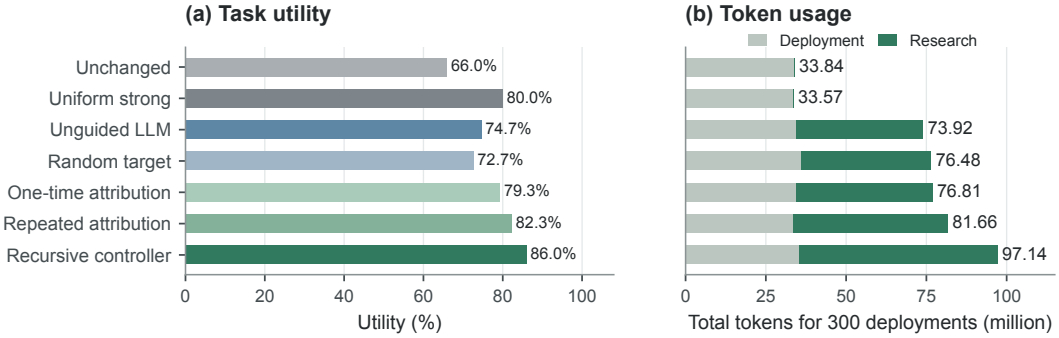

Figure 2: **Resolved rate and token usage on SWE-bench Lite.** (a) Resolved rates on 300 test tasks. (b) Research and deployment tokens for each method, with totals labeled in millions. Deployment covered the 300 test tasks; research included controller calls and task-workflow replays. Final reporting evaluations were excluded.

4.3 Mechanism Ablations and Recursive Tests

We evaluated the contributions of attribution, repeated attribution, joint proposals, and controller updates through the corresponding mechanism comparisons. For the controller test, we froze the initial controller a_0 and updated controller a_T, started them from the same workflow z_0 with empty ledgers, and evaluated their discovered workflows under equal inner budgets and fresh research episodes. We also evaluated the frozen controllers on unseen task groups. Controller performance was measured by the quality of the workflows each controller discovered; usage incurred while developing a_T was accounted for separately. Freezing the controllers separated the effect of their learned research procedures from further changes made during evaluation.

Figure 3(a) reports positive utility differences of 7.7 percentage points for attribution, 3.0 points for re-attribution, and 4.0 points for joint proposals in their respective comparisons. These differences measured distinct interventions and were not treated as additive. The joint-proposal result suggested that considering coordinated edits captured opportunities missed by proposals restricted to a single intervention target. In the fresh-episode test in Fig. 3(b), workflows discovered by a_T achieved 82.1% utility, compared with 72.6% for a_0, a gain of 9.5 points. On unseen task groups, however, a_T achieved 72.8%, compared with 77.5% for a_0, a decrease of 4.7 points. The updated controller therefore improved search in fresh episodes while performing worse in this transfer comparison. The reversal prevented us from interpreting the fresh-episode gain as evidence of a generally stronger controller across task distributions.

4.4 Does Attribution Predict Patch Gains?

We froze the workflow snapshot and compared calibrated upgrade scores with gains from independently proposed patches at six nodes. The proposer did not receive attribution scores. We measured patch gain in percentage points on the reserved diagnostic evaluation and retained negative gains for harmful patches. This protocol tested whether the ranking predicted opportunities for concrete edits, rather than merely reproducing the outcomes of the interventions used to compute attribution.

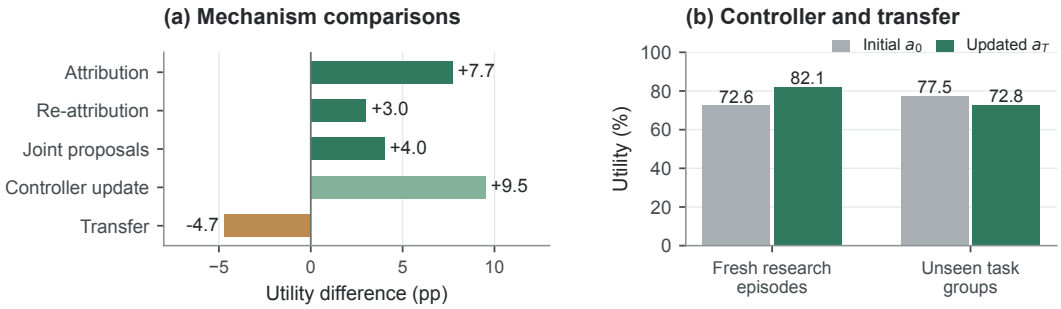

Figure 3: **Mechanism comparisons and controller transfer.** (a) Utility differences for attribution, re-attribution, joint proposals, controller update, and transfer in their respective comparisons; the bars do not represent additive effects. (b) Utility of workflows discovered by frozen a_0 and a_T in fresh research episodes and unseen task groups.

Figure 4 shows a Spearman rank correlation of 0.77 between calibrated upgrade score and observed patch gain. The two highest-scored nodes, Localizer and Editor, were also the two nodes with the largest gains, giving a top-two overlap of $2/2$. Their order differed: Localizer ranked first by score but gained 2.1 points, whereas Editor gained 3.0 points. Choosing only the top-scored node would therefore have incurred a 0.9-point regret relative to the best evaluated node. Debugger gained 1.4 points, Test gained 0.6 points, and Analyzer gained 0.3 points. Planner had a negative gain of 0.2 points despite ranking above Test and Analyzer. Attribution provided a useful ordering and shortlist, but the measured gains remained necessary for selecting a concrete patch. The six-node comparison also left substantial uncertainty about how reliably the same ranking relationship held across other workflow snapshots.

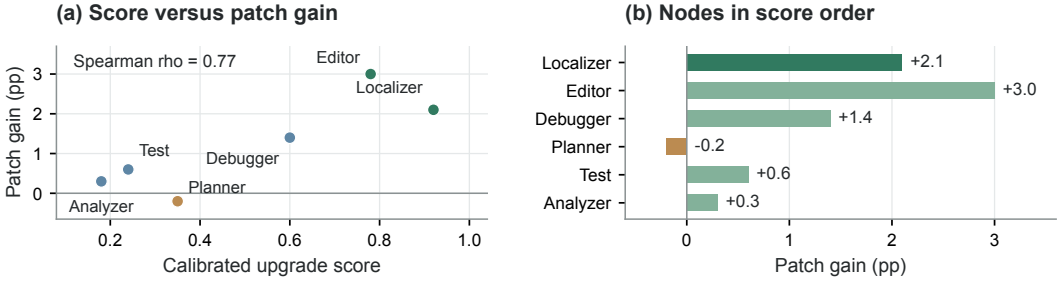

Figure 4: **Attribution scores and independent patch gains.** (a) Calibrated upgrade score versus observed patch gain at six workflow nodes; Spearman's $\rho = 0.77$. (b) The same gains in descending score order. Localizer ranked first by score, while Editor produced the largest measured gain.

4.5 Sensitivity to the Research Budget

We restarted each method from the same workflow and an empty ledger at each research-token cap, with matched seeds and deployment limits. Diagnosis, failed proposals, confirmation, and controller comparisons consumed the applicable research budget. Final-test outcomes were excluded from version selection. The zero-budget condition returned the unchanged workflow. Each plotted point represented the endpoint of a run at its specified cap, rather than a checkpoint selected retrospectively from a larger-budget run.

Figure 5(a) shows that endpoint utility increased with research budget for all four search methods, with smaller gains at the higher caps. Figure 5(b) isolates the difference between the recursive controller and repeated attribution with a fixed controller. Recursion trailed the fixed controller by 3.0 percentage points at 18.75 million research tokens and by 0.3 points at 37.5 million. At the 75-million-token cap used for the main comparison, it led by 3.7 points. Its advantage remained 3.7 points at 112.5 million and was 3.3 points at 150 million. The lower-budget results showed that controller development consumed resources that otherwise could have been spent on workflow updates; the positive differences at larger budgets showed where that investment paid off in these

runs. The slight reduction in its advantage at the largest cap also showed that a larger budget did not continually widen the gap between the two methods.

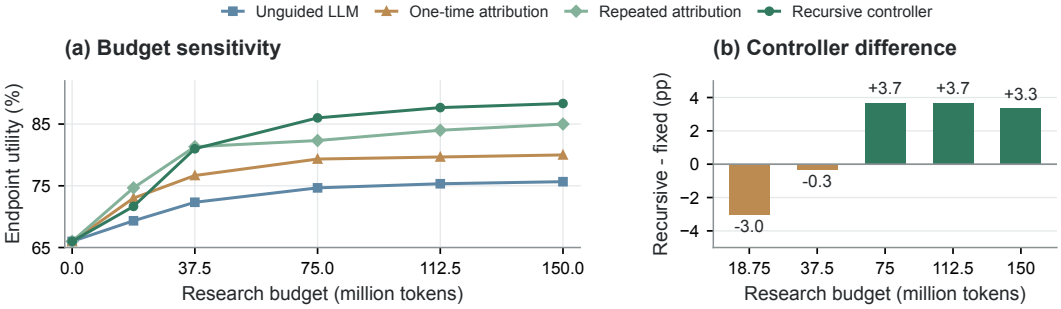

Figure 5: **Sensitivity to the research-token cap.** (a) Endpoint utility from separate runs at caps of 0, 18.75, 37.5, 75, 112.5, and 150 million tokens. Zero budget returned the unchanged workflow; connecting lines indicate trends across the measured caps. (b) Recursive-controller utility minus repeated-attribution utility with a fixed controller. The 75-million-token comparison corresponded to the main experiment.

4.6 Discussion

Attribution directed search; testing selected edits. The mechanism comparison assigned a 7.7-point utility difference to attribution and a further 3.0-point difference to re-attribution in their respective comparisons (Fig. 3). The independent patch test supported a more specific conclusion: calibrated scores correlated with patch gains and identified both of the two best nodes, but their highest-ranked node was not the best patch target (Fig. 4). The results supported using attribution to prioritize proposals and measured patch outcomes to decide which edits to retain.

Recursive improvement depended on the available budget. The recursive controller reached 86.0% resolved rate at the 75-million-token research cap, compared with 82.3% for repeated attribution, while total usage for research and 300 deployments rose from 81.66 million to 97.14 million tokens (Fig. 2). At 18.75 and 37.5 million research tokens, the recursive controller instead trailed the fixed controller (Fig. 5). These results located the benefit of controller updates within the tested budget range and showed the additional token cost associated with that benefit.

Controller improvement did not guarantee transfer. Under equal inner budgets on fresh research episodes, a_T discovered workflows with 9.5-point higher utility than a_0. On unseen task groups, its discovered workflows had 4.7-point lower utility (Fig. 3). The two evaluations measured different outcomes: improvement of the research process on fresh episodes and portability of that process to different task groups. The observed transfer loss limited the scope of the controller improvement claim.

5 Conclusion and Limitations

We developed NodeGrad, an attribution-guided research loop for improving multi-agent workflows. Calibrated node interventions identified promising targets without changing the workflow's execution contracts, while joint probes exposed opportunities that single-node measurements missed. Language-model agents translated these signals into candidate edits, and external tests determined which versions to retain. Recomputing attribution after accepted updates allowed the search to follow changing workflow bottlenecks. We further extended the loop to the research controller, evaluating controller edits through the quality of workflows discovered in fresh, equally budgeted episodes. Together, the experiments showed that attribution-guided search and recursive controller improvement increased workflow quality while making their research and deployment costs explicit.

AI USE STATEMENT

Our task-node model variants use the Gemini family: Gemini 2.5 Flash, Gemini 2.5 Pro, and Gemini 3 Flash serve as the weak, base, and enhanced variants, respectively. The research-controller design assigns GPT-5.6-sol to the controller that coordinates diagnosis, candidate construction, and test analysis. OpenAI Codex was used for LaTeX conversion and formatting and to assist with literature lookup, experimental design.

REFERENCES

Lingjiao Chen, Matei Zaharia, and James Zou. Frugalgpt: How to use large language models while reducing cost and improving performance, 2023.

Jakob N. Foerster, Gregory Farquhar, Triantafyllos Afouras, Nantas Nardelli, and Shimon Whiteson. Counterfactual multi-agent policy gradients. In *AAAI Conference on Artificial Intelligence*, 2018.

Amirata Ghorbani and James Zou. Data shapley: Equitable valuation of data for machine learning. In *International Conference on Machine Learning*, pp. 2242–2251, 2019.

Wassily Hoeffding. Probability inequalities for sums of bounded random variables. *Journal of the American Statistical Association*, 58(301):13–30, 1963. doi: 10.1080/01621459.1963.10500830.

Sirui Hong, Mingchen Zhuge, Jiaqi Chen, Xiawu Zheng, Yuheng Cheng, Ceyao Zhang, Jinlin Wang, Zili Wang, Steven Ka Shing Yau, Zijuan Lin, Liyang Zhou, Chenyu Ran, Lingfeng Xiao, Chenglin Wu, and Jürgen Schmidhuber. Metagpt: Meta programming for a multi-agent collaborative framework. In *International Conference on Learning Representations*, 2024.

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

Carlos E. Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik Narasimhan. Swe-bench: Can language models resolve real-world github issues? In *International Conference on Learning Representations*, 2024.

Omar Khattab, Arnav Singhvi, Paridhi Maheshwari, Zhiyuan Zhang, Keshav Santhanam, Sri Vardhamanan, Saiful Haq, Ashutosh Sharma, Thomas T. Joshi, Hanna Moazam, Heather Miller, Matei Zaharia, and Christopher Potts. Dspy: Compiling declarative language model calls into self-improving pipelines, 2023.

Pang Wei Koh and Percy Liang. Understanding black-box predictions via influence functions. In *International Conference on Machine Learning*, pp. 1885–1894, 2017.

Jiahui Li, Kun Kuang, Baoxiang Wang, Furui Liu, Long Chen, Fei Wu, and Jun Xiao. Shapley counterfactual credits for multi-agent reinforcement learning. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, 2021.

Haozhe Liu, Tian Ye, Sensen Gao, Qihang Cao, Yitong Li, Mingchen Zhuge, Duomin Wang, Ruihua Zhang, Ping Luo, Jiawang Bian, Lei Zhu, Ligeng Zhu, Enze Xie, and Song Han. Solpi: Recursively scaling auto-research loops for efficient agent harness, 2026. URL `https://arxiv.org/abs/2609.20519`.

Scott M. Lundberg and Su-In Lee. A unified approach to interpreting model predictions. In *Advances in Neural Information Processing Systems*, 2017.

Max D. Morris. Factorial sampling plans for preliminary computational experiments. *Technometrics*, 33(2):161–174, 1991.

Isaac Ong, Amjad Almahairi, Vincent Wu, Wei-Lin Chiang, Tianhao Wu, Joseph E. Gonzalez, M. Waleed Kadous, and Ion Stoica. Routellm: Learning to route llms with preference data, 2024.

Garima Pruthi, Frederick Liu, Satyen Kale, and Mukund Sundararajan. Estimating training data influence by tracing gradient descent. In *Advances in Neural Information Processing Systems*, 2020.

Chen Qian, Wei Liu, Hongzhang Liu, Nuo Chen, Yufan Dang, Jiahao Li, Cheng Yang, Weize Chen, Yusheng Su, Xin Cong, Juyuan Xu, Dahai Li, Zhiyuan Liu, and Maosong Sun. Chatdev: Communicative agents for software development. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics*, 2024.

Tabish Rashid, Mikayel Samvelyan, Christian Schroeder de Witt, Gregory Farquhar, Jakob Foerster, and Shimon Whiteson. Qmix: Monotonic value function factorisation for deep multi-agent reinforcement learning. In *International Conference on Machine Learning*, pp. 4295–4304, 2018.

Andrea Saltelli, Marco Ratto, Terry Andres, Francesca Campolongo, Jessica Cariboni, Debora Gatelli, Michaela Saisana, and Stefano Tarantola. *Global Sensitivity Analysis: The Primer*. John Wiley & Sons, 2008.

Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. Toolformer: Language models can teach themselves to use tools. In *Advances in Neural Information Processing Systems*, 2023.

Ilya M. Sobol. Global sensitivity indices for nonlinear mathematical models and their monte carlo estimates. *Mathematics and Computers in Simulation*, 55(1–3):271–280, 2001.

Peter Sunehag, Guy Lever, Audrunas Gruslys, Wojciech Marian Czarnecki, Vinicius Zambaldi, Max Jaderberg, Marc Lanctot, Nicolas Sonnerat, Joel Z. Leibo, Karl Tuyls, and Thore Graepel. Value-decomposition networks for cooperative multi-agent learning. In *International Conference on Autonomous Agents and Multiagent Systems*, 2018.

SWE-bench Team. Evaluation Guide, n.d.a. URL `https://www.swebench.com/SWE-bench/guides/evaluation/`. Official harness documentation. Accessed September 26, 2026.

SWE-bench Team. SWE-bench Lite. Official benchmark documentation, n.d.b. URL `https://www.swebench.com/lite`. Accessed September 26, 2026.

Xingyao Wang, Boxuan Li, Yufan Song, Frank F. Xu, Xiangru Tang, Mingchen Zhuge, Jiayi Pan, Yueqi Song, Bowen Li, Jaskirat Singh, Hoang H. Tran, Fuqiang Li, Ren Ma, Mingzhang Zheng, Bill Qian, Yanjun Shao, Niklas Muennighoff, Yizhe Zhang, Binyuan Hui, Junyang Lin, Robert Brennan, Hao Peng, Heng Ji, and Graham Neubig. Openhands: An open platform for ai software developers as generalist agents, 2024.

Qingyun Wu, Gagan Bansal, Jieyu Zhang, Yiran Wu, Beibin Li, Erkang Zhu, Li Jiang, Xiaoyun Zhang, Shaokun Zhang, Jiale Liu, Ahmed Hassan Awadallah, Ryen W. White, Doug Burger, and Chi Wang. Autogen: Enabling next-gen llm applications via multi-agent conversation, 2023.

Chunqiu Steven Xia, Yinlin Deng, Soren Dunn, and Lingming Zhang. Agentless: Demystifying llm-based software engineering agents, 2024.

John Yang, Carlos E. Jimenez, Alexander Wettig, Kilian Lieret, Shunyu Yao, Karthik Narasimhan, and Ofir Press. Swe-agent: Agent-computer interfaces enable automated software engineering. In *Advances in Neural Information Processing Systems*, 2024.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models. In *International Conference on Learning Representations*, 2023.

Mert Yuksekgonul, Federico Bianchi, Joseph Boen, Sheng Liu, Zhi Huang, Carlos Guestrin, and James Zou. Optimizing generative ai by backpropagating language model feedback. *Nature*, 2025.

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.

Kechi Zhang, Jia Li, Ge Li, Xianjie Shi, and Zhi Jin. Codeagent: Enhancing code generation with tool-integrated agent systems for real-world repo-level coding challenges. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics*, 2024.

A Research Loop and Implementation Specification

The execution specification connects six task nodes and five research roles through typed actions, immutable snapshots, and an externally controlled acceptance rule.

A.1 Execution Contract and State Transitions

The task workflow has six fixed stages: Issue Analyzer, Repository Localizer, Patch Planner, Code Editor, Test Feedback, and Reviewer/Debugger. Their contracts carry, respectively, structured requirements, file/symbol references, an edit plan, a patch artifact, runner-produced test records with a textual summary, and a final patch with provenance. Task packets include the issue identifier, starting repository hash, permitted file roots, and resource limits. A schema-valid output can still be semantically incompatible; downstream replay and regression checks test this separately. The final patch is evaluated on a clean copy of the pinned repository, not on a workspace whose state was modified by earlier candidates.

The five research roles exchange typed ledger records. Each action specifies its parent snapshot, target nodes, action type, artifact hashes, task manifest, and requested resource reservation. The external scheduler validates the request, launches an isolated job, and appends its raw outcome. Candidate states are `proposed`, `built`, `screened`, `confirmed`, and `accepted`, with explicit `rejected`, `failed`, and `unconfirmed` outcomes. Only the scheduler advances the accepted-version pointer. A stale-parent candidate is rejected or rebuilt and retested against the current parent. The baseline design evaluates candidates sequentially; parallel search would require separately specified scheduling and accounting.

The Builder may alter only allowlisted workflow fields and adapters. It cannot modify the executor, evaluator, acceptance rule, hidden task files, or budget ledger. Generated tests are supplementary development artifacts and cannot replace authoritative tests. Memory is scoped to a task or research episode; fresh episodes receive empty ledgers and caches, except for explicitly shared immutable inputs. Cache reuse within an episode requires matching input, configuration, and artifact hashes. An upstream change invalidates downstream cached outputs.

Malformed responses, timeouts, and invalid patches have explicit error records. A workflow execution that fails to produce an accepted solution receives reward zero. A candidate rejected before task execution leaves the incumbent unchanged and still consumes its build and check costs. Infrastructure outages follow a fixed retry limit; every model invocation is counted. Unresolved infrastructure failures are disclosed with missingness counts and a failure-inclusive sensitivity result, rather than silently dropped. The execution manifest specifies JSON schemas, container digests, provider retry settings, and executable commands.

A.2 Data Access and Controller Evaluation

Partition task groups before any search. Let $\mathcal{D}_{\text{search}}$ provide traces and reusable development scores, and let $\mathcal{D}_{\text{accept}}$ supply disjoint batches for task-candidate confirmation. Research-controller development episodes have their own inner search and acceptance subsets plus hidden endpoint tasks. Endpoint feedback may guide the outer controller search, so these episodes remain development data. A separate collection $\mathcal{E}_{\text{test}}$ evaluates the frozen pair (a_0, a_T) in fresh episodes; $\mathcal{D}_{\text{test}}$ evaluates the frozen task workflow. Neither final collection is exposed during selection. The attribution-prediction diagnostic uses another reserved subset or a separately frozen evaluation campaign.

Related issues, patches, repositories used as transfer groups, and task-generator seeds must respect the declared grouping policy. Repeated executions of the same task do not create new task-level independent samples. Provider seed support and nondeterminism are logged; matching seed labels alone does not establish paired stochastic realizations. Final comparisons use identical task manifests and prespecified repetitions for every method. Group-level uncertainty, task-micro averages, and repository-macro averages are reported distinctly.

For a controller comparison with E episode pairs and inner search cap B_{inner}, the scheduler reserves up to $2EB_{\text{inner}}$ plus endpoint evaluation and orchestration costs. It starts no comparison that exceeds the remaining outer budget. An inner episode freezes its controller and disables further meta-updates. Comparing final a_T with a_0 on equal fresh-episode budgets measures their conditional research ability; a total-usage comparison additionally includes the tokens of discovering a_T.

A.3 Confirmation Rule and Statistical Feasibility

The following is a conservative reference rule for independent bounded evaluation units. A unit may be a task, or a prespecified independent group whose repeated task outcomes are averaged first. Its weighting must match the declared objective. Candidate and incumbent are fixed before the batch is sampled. For attempt j, let $d_g \in [-1, 1]$ be the paired reward difference for unit g, and let $c_g \in [0, c_{\max}]$ be the candidate cost under an enforced per-unit cap. For m_j units and a fixed per-attempt error allocation α_j, Hoeffding's inequality (Hoeffding, 1963) gives

$$L_j = \bar{d} - \sqrt{\frac{2\log(2/\alpha_j)}{m_j}}, \qquad C_j^{\mathrm{up}} = \bar{c} + c_{\max}\sqrt{\frac{\log(2/\alpha_j)}{2m_j}}. \tag{12}$$

Allocate $\sum_j \alpha_j \le \alpha$ across task and controller acceptance attempts before observing their outcomes. The two bounds split α_j equally; a union bound covers the planned comparisons under the stated independence and fresh-batch conditions. Controller comparisons use paired endpoint differences on the same episode, so the common starting-workflow term cancels and the difference remains in $[-1, 1]$. Their deployment checks require their own allocation from the same error budget. An efficiency run separately bounds the paired cost decrease and quality noninferiority margin. No guarantee is asserted for adaptively reused batches or dependent units treated as independent.

This rule may be too conservative for a small repository benchmark. To make the reward-bound radius smaller than a desired gap γ, it requires $m_j > 2\log(2/\alpha_j)/\gamma^2$. Increasing repeated calls on the same few repositories does not fix insufficient independent coverage. The evaluation configuration fixes the independent unit, detectable effect, error allocation, batch size, and number of acceptance attempts. A replacement paired procedure must specify its assumptions and error control.

A.4 Attribution Cost and Reproducibility Manifest

One complete diagnostic refresh on m development tasks with r repetitions costs up to $(1 + 2N + P)mr$ workflow executions, before local probe calls, when every node has valid weak and strong directions and P pairs are tested. For six nodes this is $(13 + P)mr$; a recorded base execution may be reused only for the identical batch and snapshot. Probe failures reduce valid coverage, not the recorded cost of attempts. Exhaustive evaluation of seven nodes at four levels requires $4^7 = 16{,}384$ configurations per task/repetition grid; an oracle is thus a reference diagnostic, not an equal-cost baseline.

A run manifest records dataset revision, task/group identifiers, starting code and image digests, exact model IDs, prompts, decoding settings, allowed tools, context limits, per-task execution caps, and retry rules. It also freezes the probe rubric and minimum change, compatibility threshold, shortlist and pair budgets, exploration fraction, candidate count, acceptance schedule, and stopping rule. The ledger records immutable event IDs, raw token categories, tool time, cache hits, per-model input/output token totals, patch hashes, test outputs, and dispositions.