DRAFTING AGENTS, NOT JUST TOKENS: SPECULATIVE RECURSIVE SELF-IMPROVEMENT FOR AGENTIC SYSTEMS

Anonymous authors
Paper under double-blind review

ABSTRACT

Improving an agent through successive revisions requires balancing exploration and evaluation under a limited budget. A narrow search can miss useful sequences of edits, while fully evaluating every intermediate version can exhaust the budget before those sequences yield a verified improvement. We introduce ***Spec**ulative **R**ecursive **S**elf-**I**mprovement* (***SpecRSI***), which drafts executable successor agents before their predecessors are fully evaluated, while requiring complete verification before adoption. Verification-Aware Tree Search (VATS) allocates the budget between further edits and outstanding checks using the predicted verified return of a continuation. Group Verification Optimization (GVO) learns the improver from completed searches while keeping the real evaluator fixed. In the main scientific comparison, terminal gains exceed Janus by 33.3%–40.0%. Additional evaluations outperform Janus in all 50 scientific backbone–task pairs and exceed MetaRSI-v1 by 1.85%–2.38% in macro-average system score across twenty backbones. Coding-validation success reaches 69.4% versus HGM's 64.3% at a 20.8% lower target-cost index. These results demonstrate the efficacy of the proposed approach. Code is available in https://anonymous.4open.science/r/SpecRSI-739D/.

1 INTRODUCTION

Agent systems can be improved through successive revisions to their prompts, tools, and workflows. Each round proposes executable changes and then evaluates whether they improve task performance, and both consume time and computation. This raises a central question: **how should an agent allocate its limited budget between exploring new revisions and verifying their benefits?** Successful revisions can make later ones easier, but the evaluations that identify them are often far more expensive than proposing another edit (Zhang et al., 2026a; Wang et al., 2026b; Liu et al., 2026). The difficulty is not a shortage of candidates. It is turning exploration into verified progress before time or computation runs out.

Two limiting strategies expose the tradeoff. Following one fully evaluated lineage gives every decision fresh feedback, but it narrows the alternatives and inserts an evaluation barrier between dependent edits. Maintaining many lineages explores more, but fully evaluating them all buys evidence for branches that may never be used. Extra workers cut elapsed time without making the work free, and under a fixed budget a wider search need not leave enough computation to finish the most useful branch. The barrier, not the shortage of ideas, is what stalls progress: can we draft successors ahead of their parents' full evaluation, yet still spend verification only where it pays? For example, one coding-agent revision adds a file–test map; a successor uses it for focused intermediate retesting while retaining the final full suite. Prior systems relax part of this by separating expansion from evaluation or validating in parallel (Wang et al., 2026b; Liu et al., 2026; Hu et al., 2026b). Here we focus on jointly selecting further revisions and bundles of outstanding checks under the remaining budget.

Once successors can be drafted ahead, where should we spend the budget next? A candidate that looks strong may still need several checks before it can be adopted, while a child that repairs an unhelpful parent may need only one remaining check. A scorer that only asks How good does this version look?" therefore misses the cost of making it usable. What governs the budget is not a candidate but a *continuation*, a sequence of further edits and checks from the current state, and the

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.

operative question becomes *What verified improvement can this continuation still deliver with the remaining budget?"* This quantity must be estimated *before* the checks that would reveal it are paid, and a wrong estimate wastes irreversible budget; this pre-verification estimate under a hard budget, not the drafting of successors, is what makes the problem hard. It shifts the scoring target from candidate quality to the marginal verified return of further editing and checking.

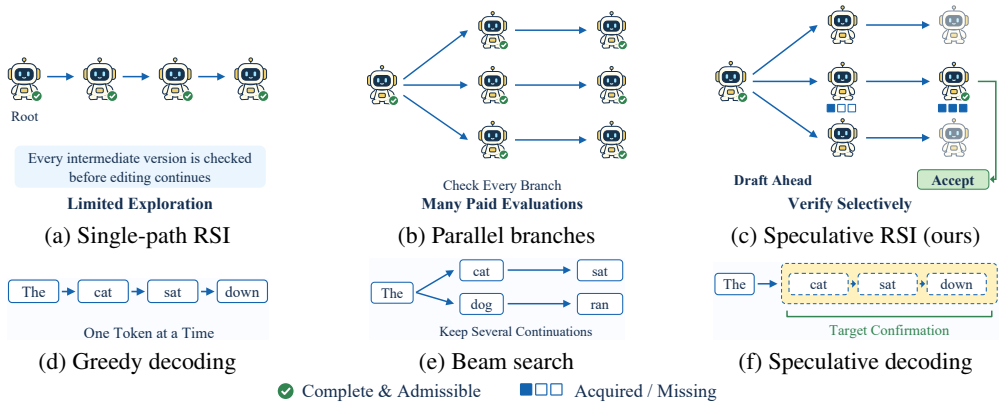


Figure 1: **Explore ahead, verify before adoption.** Agent patterns (a–c) and their token-level counterparts (d–f). Agent checks are external and paid; only the adopted version needs complete, admissible evidence.

However, the drafting–confirmation idea cannot be transferred unchanged from tokens to agent revisions. (1) Revisions are revisable programs, not accepted prefixes. A successor may repair or build on a parent that is not yet admissible. (2) Verification consists of paid, complementary checks. External evaluations are not supplied by one target-model pass: checks have heterogeneous costs and dependencies, and several may be jointly required for adoption. (3) Progress is verified improvement under a budget. Search outcomes depend on realized gains and all incurred costs, including failed and unfinished work, rather than proposal or acceptance counts.

We propose ***SpecRSI*** with the following contributions:

- **A Two-Level Framework for Speculative Self-Improvement.** To address (1), inner loop searches extend incompletely evaluated versions but adopt only fully evaluated, admissible ones. The improver remains frozen within each search and learns from complete-search feedback between rounds.
- **Verification-Aware Tree Search (VATS).** To address (2), the Solver preserves affordable completion bundles, including individually low-value checks; the Scorer predicts verified gains from further edits and checks, net of subsequent costs; and the Policy selects the next draft or verification plan under the remaining budget.
- **Group Verification Optimization (GVO).** To address (3), GVO uses independent complete searches from the same start as its unit of credit. Verified gains minus all search costs provide delayed credit to sampled edits and work choices, training the Drafter and Policy. Realized suffix returns and paid same-state comparisons train the Scorer.

2 Preliminaries and Related Work

2.1 Recursive Self-Improvement

An agent version c comprises executable code and persistent prompts, tools, and workflows. Recursive self-improvement (RSI) revises these artifacts to support later improvement rounds. With retained versions $\mathcal{V}_t$ and evaluation history H_t, an improver $\mathrm{Propose}_{\Theta_t}$ selects and revises a version:

$$\begin{aligned} \widetilde{c}_t &\sim \mathrm{Propose}_{\Theta_t}(\cdot \mid \mathcal{V}_t, H_t), \quad z_t = f_{\mathrm{real}}(\widetilde{c}_t), \\ (\mathcal{V}_{t+1}, H_{t+1}) &= \mathrm{Update}(\mathcal{V}_t, H_t, \widetilde{c}_t, z_t), \end{aligned} \tag{1}$$

where update records observed feedback z_t; a useful stepping stone need not immediately outperform the incumbent. Existing methods search over agents, workflows, or prompts (Zhang et al., 2026a;

Robeyns et al., 2025; Hu et al., 2025; Zhang et al., 2025b; Agrawal et al., 2026). HGM separates expansion from evaluation (Wang et al., 2026b), while Hyperagents and MetaRSI adapt improvement generation or scheduling (Zhang et al., 2026b; Tan et al., 2026).

2.2 Evaluators and Verified Improvement

Following Janus (Liu et al., 2026), f_{real} executes the task's assessment, returning higher-is-better utility $J(c) \in \mathbb{R}$ and validity flag $v(c) \in \{0, 1\}$ for constraint satisfaction. After n real-evaluation attempts, the best observed valid candidate satisfies

$$f_{\text{real}}(c) = (J(c), v(c)), \qquad c_n^\star \in \underset{(c,J,v)\in H_n:\, v=1}{\arg\max} J, \tag{2}$$

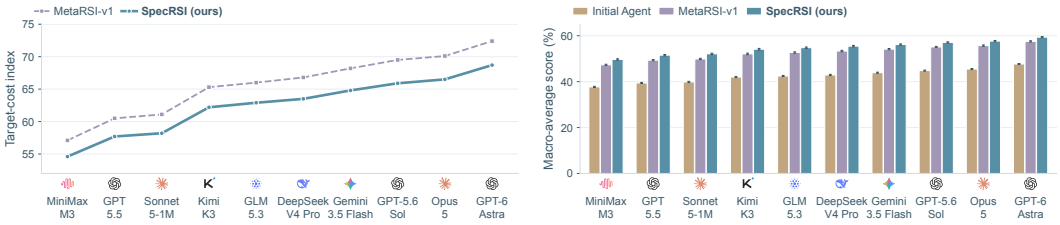
where n counts attempts, not computation or monetary cost. A *proxy evaluator* cheaply prioritizes real evaluations; Janus admits only real-valid candidates to its population or incumbent. SpecRSI preserves this separation between prediction and validation, but allocates individual checks and scores further edits and checks by their eventual verified return. Section 3.2 defines its state and evidence; Appendix A provides extended comparisons.

3 Method

3.1 Motivation

Accurate benefit evaluation is central to guiding RSI toward better agents. It informs which revisions and checks deserve further work, guiding the direction of improvement. Inaccurate evaluation can overlook useful revisions or waste resources on candidates that cannot be adopted. Therefore, evaluation should reflect the verified improvement that further work can deliver.

However, accurate benefit evaluation is non-trivial: a candidate's current quality does not determine the value of continuing from it. This difficulty stems from two key challenges. ❶ **Benefit evaluation should reflect downstream verified improvement.** Intermediate revisions may enable later gains, but establishing them requires subsequent edits and real checks. ❷ **Benefit evaluation should account for completion costs.** Complementary checks, planning, and failed attempts consume the same limited budget; even promising gains may be unaffordable to verify. Together, these challenges create a performance–cost trade-off: a useful continuation must promise sufficient verified improvement while remaining affordable to complete. Figure 2 illustrates the resulting trade-off empirically.



(a) Target-cost index (b) Macro-average score

Figure 2: **Performance across ten backbones.** Target-cost indices and macro-average scores on Terminal-Bench 2.1 (TB2.1), SWE-Pro, GPQA-hard100, and AIME (mean $\pm$ 1.0 $\times$ sample SD across five training seeds). Panels follow Initial Agent means; lines connect model categories. On all ten backbones, SpecRSI (ours) scores higher than MetaRSI-v1 at lower target cost but 0.83–0.90% higher total-cost indices.

3.2 Two-Level Speculative RSI Loop

SpecRSI tracks individual evaluation checks. A complete admissible root r anchors tree $\mathcal{T} = (\mathcal{C}, \mathcal{A})$ of immutable agent versions and executed edit edges. Versions freeze prompts, tools, workflows, and permitted persistent state, including memory; only D/H/M permits model-parameter edits. The uneditable verification contract Γ specifies finite required-item set $\mathcal{I}$, execution/admission rules, constraints, and higher-is-better score J_Γ, instantiating task utility J. State $S = (\mathcal{T}, \mathcal{E}, H, B, N, w)$

records admitted evidence $\mathcal{E}$, history H, remaining expense $B \geq 0$, attempts $N \geq 0$, and workers $w \geq 1$. For version c and item i, $m_{ci} \in \{0, 1\}$ marks a valid outcome's presence, not whether it passed. The complete admissible set and its best observed gain are

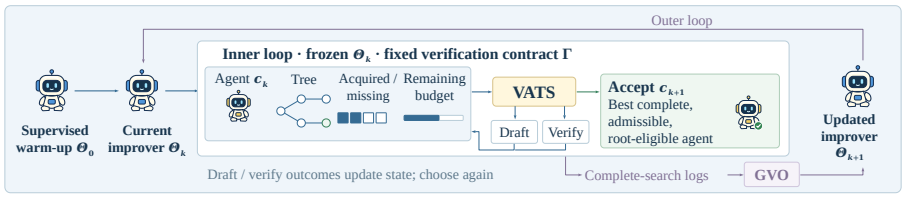
$$\begin{aligned} \mathcal{C}_{\text{full}}(S) &= \{c \in \mathcal{C} : m_{ci} = 1 \; \forall i \in \mathcal{I}, \; c \text{ admissible under } \Gamma\}, \\ F(S) &= \max_{c \in \mathcal{C}_{\text{full}}(S)} [J_\Gamma(c) - J_\Gamma(r)], \end{aligned} \tag{3}$$

where F is observed adoption-ready gain, not a learned value; the eligible root ensures $F \geq 0$. The fixed contract Γ and execution interface Acquire_Γ form the *real evaluator*: Γ specifies valid evidence; VERIFY invokes Acquire_Γ to run selected real checks. Complete admissible evidence determines $J_\Gamma(c)$ and ACCEPT eligibility. The learned Scorer $\widehat{Q}_\phi(S, \omega)$ is the *score evaluator*; it predicts expected verified gain net of subsequent expense and elapsed-time costs (Equation (6)), guiding expenditure without supplying evidence. DRAFT generates successor edits and assigns no evaluation score.

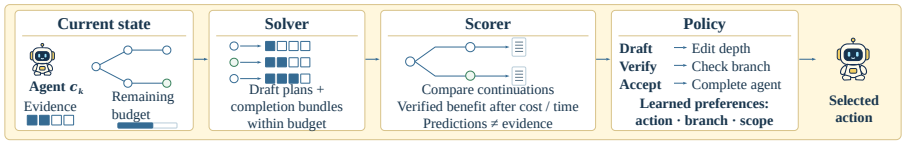
At round k, c_k is the agent and $\Theta_k = (\theta_k, \psi_k, \phi_k, \nu)$ the improver: drafter, work-selection policy, continuation-value scorer, and fixed menu settings. Collection freezes all four, with scorer snapshot $\bar{\phi}_k = \phi_k$. Independent searches from the same root supply agents and logs for the outer update:

$$\begin{aligned} &(c_{k+1}^{(g)}, \mathcal{D}_k^{(g)}) = \text{Search}_{\Theta_k}(c_k), \quad g = 1, \ldots, G \quad \text{(inner loop searches)}, \\ &\mathcal{D}_k = \bigcup_{g=1}^{G} \mathcal{D}_k^{(g)}, \qquad \Theta_{k+1} = \text{GVO}(\Theta_k, \mathcal{D}_k) \qquad \text{(outer loop update)}, \end{aligned} \tag{4}$$

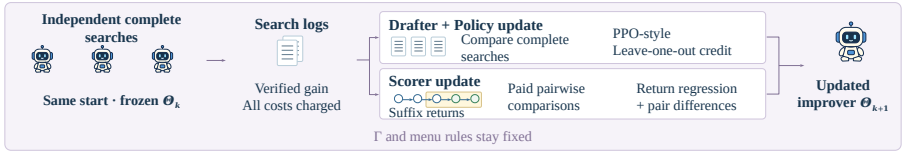
where $G \geq 2$ counts independent searches, not shared-tree leaves. Figure 3 connects VATS inner searches and GVO outer updates. Carry/reset selects the next root separately from within-search acceptance, isolating learning from artifact retention. Contract Γ and the outer update rule remain fixed (Appendix F.1).



(a) Two-level Speculative RSI Loop



(b) Verification-Aware Tree Search (VATS)



(c) Group Verification Optimization (GVO)

Figure 3: **Two-level Speculative RSI Loop.** VATS allocates drafts and checks under a frozen improver; Accept returns the best complete admissible version. GVO updates the improver between searches; Γ stays fixed.

Table S1 in Appendix A compares roles with Janus (Liu et al., 2026): its proxies screen candidate programs; our Scorer values state–work pairs, including later edits and checks.

3.3 Inner-loop Search with VATS

Verification-Aware Tree Search (VATS) selects bounded work ω: $\text{DRAFT}(P, h)$ or $\text{VERIFY}(\mathcal{J})$, where $P = (P_a)_{a=1}^{|P|}$ tags existing parents, $h \geq 1$ bounds edit depth, and $\mathcal{J}$ lists existing-version

jobs. Expense and attempt caps are enforced. Either action may repeat; ACCEPT terminates search. Complementary checks require more than token-confidence ranking: Solver constructs completion options, Scorer refines scope, and Policy selects work (Algorithm 2). Execution updates state before replanning under the frozen improver (Algorithm 1, Appendix F).

3.3.1 SOLVER

The Solver forms completion bundles that fit the remaining budget. Missing items are $U_c(S) = \{i \in \mathcal{I} : m_{ci} = 0\}$, and outstanding outcomes are $K_c(S) = \{(c, i) : i \in U_c(S)\}$. The relation reader $\mathcal{R}(S) = \text{Remaining}(S)$ reads masks, dependencies, shared setups, and source relations from existing records. It acquires no new outcomes; forecasts cannot fill missing evidence (Appendix F.2). Bundles cover selected missing-item sets even when individual checks have little value:

$$\Omega_t = \mathcal{D}_t \cup \{\text{VERIFY}(A) : A \in \mathcal{M}_t\} \cup \{\text{ACCEPT}\}, \tag{5}$$

where draft plans $\mathcal{D}_t$ use existing, possibly incomplete parents; $\mathcal{M}_t$ contains feasible verification bundles. A MILP covers selected K_c sets and cross-branch unions subject to prerequisites, shared setups, conflicts, and reserved budgets. Original completion seeds remain eligible during refinement. Feasibility guarantees neither successful outcomes nor exhaustive coverage.

3.3.2 SCORER

The Scorer $\widehat{Q}_\phi(S, \omega)$ values state–work pairs under the frozen procedure π, including later edits, checks, and branch switches:

$$Q^\pi(S_t, \omega) = \mathbb{E}_\pi[F(S_{\text{end}}) - F(S_t) - \lambda C_{t:\text{end}} - \mu L_{t:\text{end}} \mid S_t, \omega], \tag{6}$$

where S_t^- precedes preparation; S_t follows its payment; S_{end} is terminal. Subsequent expense $C_{t:\text{end}}$ and elapsed time $L_{t:\text{end}}$ include later preparation and failures, excluding current sunk preparation. Fixed $\lambda, \mu \geq 0$ convert costs to score units ($\mu = 0$ for cost-only evaluation). π includes edit generation and work selection with fixed menu rules. Predictions never replace observed F or evidence; $Q^\pi(S, \text{ACCEPT}) = 0$.

For feasible bundle A, let D_j contain job j and its prerequisites. The frozen snapshot $\widehat{Q}_{\bar{\phi}}$ refines bundles by

$$\Delta_{\bar{\phi}}(j \mid A, \widetilde{S}) = \widehat{Q}_{\bar{\phi}}(\widetilde{S}, \text{VERIFY}(A \cup D_j)) - \widehat{Q}_{\bar{\phi}}(\widetilde{S}, \text{VERIFY}(A)), \tag{7}$$

where both terms use the same conservative state $\widetilde{S}$ reserving the preparation cap. Whole-bundle comparisons preserve interactions without summing isolated check scores. Shared jobs/setups count once. Query and extension caps bound overhead, while original feasible seeds remain eligible (Appendix F.3).

3.3.3 POLICY

After paying preparation, the Policy samples from nonempty Ω_t using observed relations $\mathcal{R}_t = \text{Remaining}(S_t)$:

$$\pi_\psi(\omega \mid S_t, \mathcal{R}_t, \Omega_t) = (1 - \epsilon)\frac{\exp(z_\psi(S_t, \mathcal{R}_t, \omega)/T_{\text{pol}})}{\sum_{v \in \Omega_t} \exp(z_\psi(S_t, \mathcal{R}_t, v)/T_{\text{pol}})} + \frac{\epsilon}{|\Omega_t|}, \tag{8}$$

where preference logits z_ψ differ from value estimates: selection need not maximize $\widehat{Q}$, $T_{\text{pol}} > 0$, $\epsilon \in [0, 1)$, and the ACCEPT logit is zero. Collection samples; deployment maximizes logits (training supports: Appendix F.5).

3.3.4 EXECUTION AND ACCEPTANCE

DRAFT. For draft plan (P, h), the drafter starts at $c_{a,0} = P_a$ and samples using only context $H_{a,d}$ observed before editing:

$$e_{a,d} \sim \pi_\theta(\cdot \mid c_{a,d-1}, H_{a,d}), \qquad c_{a,d} = \text{Freeze}(\text{Apply}(c_{a,d-1}, e_{a,d})), \quad d = 1, \ldots, h, \tag{9}$$

where Apply executes the edit; Freeze saves its immutable result. $\mathrm{Generate}_\theta(P, h, S)$ runs the bounded plan on actual artifacts: incomplete parents may expand, but each edit waits for its parent. Invalid edits are charged and end that path, preserving its valid prefix.

VERIFY. For requested jobs $\mathrm{Jobs}(\omega)$, $\mathrm{Acquire}_\Gamma$ reads valid identical records or executes real checks. The core is direct-only; checked reuse imports results after scoped equality checks through the backend shared with compatible controls. Forecasts never create evidence. VATS selects branches; the executor obtains records and updates state before replanning (Appendix F.4). Acquired records alone authorize adoption.

ACCEPT. Records must satisfy identity, scope, fields, and provenance requirements, including when reporting failure. At termination,

$$\mathrm{ACCEPT}(S) = \mathrm{TieArgMax}_{c \in \mathcal{C}_{\mathrm{full}}(S)} J_\Gamma(c), \tag{10}$$

where ties prefer the root only if tied for best, then use a fixed version order. Only the selected version needs complete admissible evidence. The reference rule adds no hidden completion/adoption margin.

3.4 Outer-loop Learning with GVO

Group Verification Optimization (GVO) updates the learned components between search rounds while the verification contract stays fixed.

Supervised warm-up initializes Drafter θ from executed legal edits, Policy ψ from legal plan choices, and Scorer ϕ from observed returns, producing Θ_0. Unexecuted alternatives have no labels.

Solver feasibility rules and menu settings ν are not learned. Solver outputs and environment results have no policy likelihood.

The Scorer fits realized, undiscounted suffix returns from executed plan starts:

$$\begin{aligned} G_t &= F(S_{\mathrm{end}}) - F(S_t) - \lambda C_{t:\mathrm{end}} - \mu L_{t:\mathrm{end}}, \\ \mathcal{L}_Q(\phi) &= \mathbb{E}_{(S_t, \omega_t, G_t) \sim \mathcal{D}_Q} \left[(\widehat{Q}_\phi(S_t, \omega_t) - G_t)^2 \right], \end{aligned} \tag{11}$$

where Q^π is the conditional expectation of G_t. $\mathcal{D}_Q$ records plan starts under the frozen procedure, including paid alternatives; current preparation is sunk, later preparation charged. For legal plans a, b from the same saved paid state S, let $\overline{G}_a, \overline{G}_b$ be mean returns of paid repetitions under the same continuation policy:

$$\widehat{\delta}_{ab} = \overline{G}_a - \overline{G}_b, \quad \mathcal{L}_{\mathrm{pair}}(\phi) = \mathbb{E}_{\mathcal{D}_{\mathrm{pair}}} \left[(\widehat{Q}_\phi(S, a) - \widehat{Q}_\phi(S, b) - \widehat{\delta}_{ab})^2 \right]. \tag{12}$$

Regression anchors the mean (Gneiting, 2011). Difference-based training targets relative values (Bertsekas, 1997); here, pair differences alone leave the common offset undetermined. Both train one Scorer through $\mathcal{L}_Q + \beta_{\mathrm{pair}} \mathcal{L}_{\mathrm{pair}}$, $\beta_{\mathrm{pair}} \geq 0$. Forced continuations train only the Scorer, never actor likelihood; copies and rollout costs remain charged.

Collect $G \geq 2$ independent searches from S_0^- under frozen Θ_k, matching root, contract, budgets, and workers. "Complete" means terminated with output and costs recorded, not every visited version fully evaluated. Expense C_g and elapsed time L_g charge initial preparation and failed or unfinished work:

$$R_g = F(S_{\mathrm{end}}^g) - F(S_0^-) - \lambda C_g - \mu L_g, \qquad A_g = R_g - \frac{1}{G-1} \sum_{\substack{u=1 \\ u \neq g}}^{G} R_u. \tag{13}$$

No new complete improvement earns no gain, but costs remain charged. Intermediate edits receive delayed credit through verified descendants without themselves qualifying for adoption. A_g compares whole searches, without a learned baseline. Only sampled edit tokens and plan choices update θ, ψ through the clipped objective (Equation (20); Schulman et al., 2017; Ahmadian et al., 2024).

After collection, GVO updates Θ_{k+1} and freezes it for the next round. Actor updates change the continuation law, requiring fresh searches to recalibrate the Scorer (Algorithm 3). These updates guarantee neither per-edit causal attribution nor monotonic improvement; Γ remains unchanged.

4 EXPERIMENTAL RESULTS

4.1 TASKS, MODELS, AND BASELINES.

Evaluation setup. Tasks, models, and baselines. We evaluate five scientific domains (PyBaMM, AQM, FDTD, Reactor, and IRP), four system tasks (Terminal-Bench 2.1 (TB2.1), SWE-Pro, GPQA-hard100, and AIME), and separate coding, prompt, and workflow profiles. Qwen3.5-35B-A3B serves as the reference backbone for mechanism studies and the primary matched system comparison. For cross-backbone evaluation on the scientific domains, we additionally test GPT-6-Astra, Claude-Opus-5, Gemini-3.5-Flash, DeepSeek-V4-Pro, GLM-5.3, Kimi-K3, MiniMax-M3, GPT-5.6-Sol, Claude-Sonnet-5-1M, and GPT-5.5, covering 50 backbone–task pairs in total. We further evaluate the four system tasks across twenty model backbones to test whether the observed gains persist across different starting models. Baselines include Janus, HGM, MetaRSI, FlashEvolve, and task-specific optimizers (Liu et al., 2026; Wang et al., 2026b; Tan et al., 2026; Hu et al., 2026b). D/H/M permits data, harness, and model edits, whereas D+H excludes model edits. Comparisons remain within matched permission bands; SpecRSI (ours) uses direct checks.

Measurements. Scientific gain and call-axis AUBC retain native units; task scores use percentages, absolute gains percentage points (pp). Target-cost, training-cost, and lifecycle indices are not pooled. Validation normally uses five training seeds with three nested searches each.

4.2 OVERALL PERFORMANCE

RQ1: Does SpecRSI produce better successors within comparable evaluation settings?

Scientific progress. SpecRSI (ours) leads terminal gain and call-axis AUBC in all five domains (Table 1A), exceeding Janus by 33.3%–40.0% and 30.4%–33.7%, respectively. Scientific cost indices are 158.7 versus 148.1; call-budget efficiency and lifecycle expense are distinct quantities.

System performance. Under matched D/H/M access, gains over MetaRSI-v1 are 1.6–2.9 pp (Table 1B). Matching D+H access retains a 2.20 pp average advantage, versus 2.25 pp with D/H/M, showing that the gains persist without model-weight updates.

Table 1: **Performance across scientific discovery and agent improvement.** A: terminal gain (upper entry) and call-axis AUBC (lower entry), in native units. B: task scores (%) within matched edit permissions. Entries show mean $\pm 1.0\times$ SD; B uses SD. Bold denotes the highest mean within each comparable group.

A Scientific gain / AUBC

Method	PyBaMM	AQM	FDTD	Reactor	IRP
	0.12±0.01	0.09±0.01	0.14±0.01	0.11±0.01	0.10±0.01
Alg.-Only	7.3±0.4	5.1±0.3	8.5±0.5	6.7±0.3	5.8±0.2
	0.21±0.01	0.17±0.01	0.24±0.02	0.19±0.01	0.18±0.01
Direct-LLM	19.2±0.6	14.3±0.5	22.6±0.7	16.2±0.4	13.7±0.4
	0.30±0.02	0.25±0.01	0.33±0.02	0.27±0.01	0.25±0.01
Embed-GP	30.5±0.8	24.8±0.7	36.2±0.9	27.4±0.6	23.1±0.5
	0.36±0.02	0.30±0.02	0.39±0.02	0.32±0.02	0.30±0.01
Janus	42.8±1.0	34.7±0.8	51.3±1.1	36.9±0.8	31.5±0.7
SpecRSI	**0.48**±0.02	**0.41**±0.02	**0.52**±0.02	**0.44**±0.02	**0.42**±0.02
(ours)	**56.7**±1.2	**45.8**±1.0	**66.9**±1.3	**49.3**±1.0	**42.1**±0.9

B System scores (%, ↑)

Method	TB2.1	SWE-Pro	GPQA	AIME
D/H/M: data, harness, model				
Initial	23.6±0.7	10.3±0.5	71.2±0.8	55.0±0.6
Data-RSI	27.4±0.8	12.4±0.6	73.8±0.9	57.7±0.7
Harness-RSI	29.4±0.9	14.9±0.7	78.8±1.0	63.3±0.8
Model-RSI	27.0±0.8	14.0±0.6	75.4±0.9	59.3±0.7
Fixed D→M→H	30.3±0.9	15.3±0.7	79.4±1.0	64.3±0.8
Static router	30.8±1.0	14.9±0.7	79.6±1.1	64.0±0.9
MetaRSI-v1	31.9±1.0	19.5±0.8	83.8±1.1	68.3±0.9
SpecRSI (ours)	**33.5**±1.1	**21.7**±0.9	**86.1**±1.2	**71.2**±1.0
D+H: no model update				
MetaRSI-H	28.7±0.8	14.2±0.6	77.5±0.9	61.8±0.7
SpecRSI-H (ours)	**30.1**±0.9	**16.3**±0.7	**79.9**±1.0	**64.7**±0.8

Backbone consistency. Figure S2 in Appendix B.3 covers ten scientific and twenty system backbones. Across all 50 scientific backbone–task pairs, terminal-gain and AUBC uplifts over Janus are 17.8%–27.6% and 15.7%–19.8%. SpecRSI exceeds MetaRSI-v1 on every system backbone by 1.85–2.38 pp in four-task macro-average score. Five-seed summaries cover weaker and stronger starting agents in the tested set.

Code, prompt, and workflow. SWE-Pro coding validation gives direct-check SpecRSI (ours) $69.4 \pm 1.2\%$ versus HGM's $64.3 \pm 1.1\%$, with target-cost indices 60.8 versus 76.8 (20.8% lower; Figure S3 in

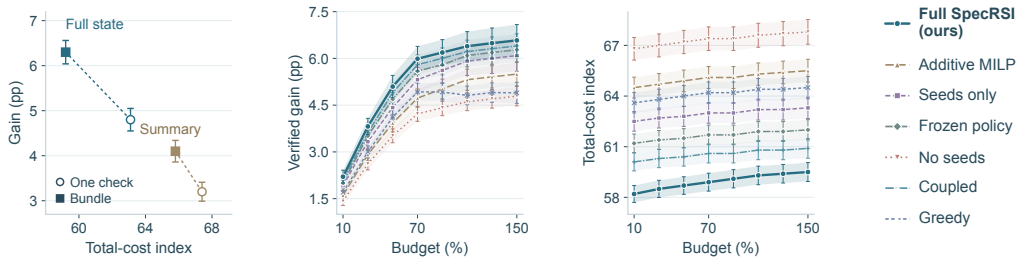
Appendix B.4). Separate profiles give GPQA-hard100 scores of 82.4% versus GEPA+Merge's 78.5%, and PlanBench success of 67.8% versus EMAS's 62.7%, with 9,290 versus 10,210 task-inference tokens. Starting agents and costs remain profile-specific; full comparisons and the separate frozen audit are in Appendix B.

4.3 Verification-Aware Search

RQ2: Does verification allocation help beyond more information or exploration?

Work scope. At fixed full-state information, bundles raise gain from 4.8 to 6.3 pp, lower cost from 63.1 to 59.2, and reduce completion failure from 9.8% to 6.5% (Figure 4a; Table S9 in Appendix C.1).

Planning ablations. Across eight budget caps from 10% to 150%, full SpecRSI has the highest reported mean gain and lowest total-cost index of seven conditions (Figure 4b,c). At 70%, full versus additive planning raises gain from 4.73 ± 0.29 to 5.99 ± 0.40 pp: planning takes 201 versus 141 ms, but total cost falls from 65.1 to 58.9. Removing completion seeds reduces gain by 0.77–1.79 pp across the reported paired-difference sweep (Figure S4). Appendix C further separates the effects of evidence quality and work scope, showing that the gain is not explained by simply executing more or deeper work. Appendix G.5 provides an exact matched-scope analysis showing how bundled verification can reduce dispatch cost under fixed information.



(a) Verification scope (b) Verified gain (c) Total cost

Figure 4: **Verification scope and planning ablations.** (a) Gain versus total-cost index for one check or bundles at fixed Summary or Full-state information; bars show reported SD. (b) Verified gain and (c) total-cost index across eight budget caps. In (b), means average five reported seed means; SD averages within-seed SDs. Lines join means; bands/bars show ± 1 SD.

4.4 Improver Learning

RQ3: Is improvement learned by the improver, or merely retained in a stronger task agent?

Artifact controls. Improver updates raise reset/carry gains from 4.1 ± 0.18 to 5.2 ± 0.21 pp and from 4.8 to 6.2 pp, respectively. Original/updated improvers yield 5.7/6.9 pp from the same released artifact (Table 2(a)), , showing gains beyond simply retaining a stronger task agent.

Roles and supervision. Scorer-only updates yield science/code/agent gains of 5.2/5.4/5.3 pp, versus 4.7/4.9/4.8 for the drafter and 5.0/5.2/5.1 for the policy. Joint updates are stronger with unequal training costs (Table 2(b)). Adding pair loss to the reported same acquired arms raises gain from 5.4 ± 0.23 to 6.3 ± 0.25 pp and lowers RMSE from 1.05 to 0.85, while training cost rises from 23.7 to 28.4.

Repeated rounds. Updated SpecRSI (ours) reaches 21.7 pp by round 5, exceeding updated Meta by 3.9 pp (17.8 pp) and original-improver SpecRSI by 7.9 pp (13.8 pp; Table 2(c)). All reported rounds continue to yield positive incremental gains, from 8.5 pp in round 2 to 2.1 pp in round 5.

Table 2: **Learning the improver.** Gains (pp). (a) Reported uncertainty; compare within each starting condition. (b) Updated roles: D, drafter; Sc, scorer; P, policy; Sh, shared backbone. Sci.: Science. Last two rows have unmatched training costs. (c) Incremental (Inc.)/cumulative (Cum.) gains for original (Orig.)/updated (Upd.) improvers. All carry the agent; round 1 initializes. Full controls: Appendix D.

Start	Update	Gain (pp)
Reset	Frozen	4.1 ± 0.18
	SFT only	4.6 ± 0.19
	GVO only	5.0 ± 0.20
	All roles	$\mathbf{5.2 \pm 0.21}$
Carry	Frozen	4.8 ± 0.20
	SFT only	5.3 ± 0.21
	GVO only	5.8 ± 0.22
	All roles	$\mathbf{6.2 \pm 0.24}$
Same artifact	Original	5.7 ± 0.22
	Updated	$\mathbf{6.9 \pm 0.25}$

(a) Artifact controls

Updated roles	Gain (pp)		
	Sci.	Code	Agent
None	4.0	4.2	4.1
D	4.7	4.9	4.8
Sc	5.2	5.4	5.3
P	5.0	5.2	5.1
D+Sc	5.4	5.6	5.5
D+P	5.3	5.5	5.4
Sc+P	5.6	5.8	5.7
D+Sc+P	**6.1**	**6.3**	**6.2**
Sh	4.5	4.7	4.6
D+Sc+P+Sh	**6.4**	**6.6**	**6.5**

(b) Updated roles

		MetaRSI		SpecRSI (ours)	
	Rd.	Orig.	Upd.	Orig.	Upd.
Inc.	2	4.9	7.2	5.8	8.5
	3	3.6	5.4	4.2	6.4
	4	2.1	3.8	2.7	4.7
	5	0.7	1.4	1.1	2.1
Cum.	2	4.9	7.2	5.8	8.5
	3	8.5	12.6	10.0	14.9
	4	10.6	16.4	12.7	19.6
	5	11.3	17.8	13.8	21.7

(c) Repeated rounds (pp)

4.5 Resource Trade-offs and Limits

RQ4: When are more planning and parallel work worth their cost?

Planning and parallelism. Doubling plans from 32 to 64 raises gain from 6.3 to 6.5 pp but lowers net return from 4.7 to 4.5 (Table 3(a)). Increasing workers from one to eight reduces elapsed time from 1,287 to 192 s, raising worker time from 1,287 to 1,403 s and gain from 6.3 to 6.5 pp (Table S17). Thus, planning and parallelism expose different resource trade-offs: the former has a finite useful range, while the latter primarily exchanges additional total work for substantially lower latency.

Table 3: **Planning capacity and lifecycle cost.** (a) MENU varies the offered-plan cap; gain is in pp with reported SD, and Missing is the percentage of omitted good options. Overhead and Net retain their reported definitions; Net is not recomputed as Gain minus Overhead. (b) Reported lifecycle indices; overlapping components are not independently additive.

Plans	Gain (pp)	Overhead	Missing (%)	Net
8	5.2 ± 0.19	**1.8**	8.6	3.9
16	5.8 ± 0.21	2.7	5.2	4.4
32	6.3 ± 0.25	3.9	3.1	**4.7**
64	$\mathbf{6.5} \pm 0.24$	5.6	**2.0**	4.5

(a) Offered-plan cap

Category	MetaRSI-v1	SpecRSI (ours)
Acquisition	14.5	**14.2**
Training	38.6	**34.8**
Search	37.2	**36.2**
Reuse checks	5.1	**4.8**
Copies	6.3	**6.1**
Failed work	6.8	**6.3**
Planning	4.2	**3.9**
Total	112.7	**106.3**

(b) Lifecycle ledger

Verification cost. Batching lowers verification cost in three regimes and raises it in three (Table S16). The separate lifecycle ledger gives SpecRSI (ours) a 5.7% lower total-cost index: 106.3 versus MetaRSI-v1's 112.7 (Table 3(b)). This lifecycle result complements Figure 2's ten-backbone target-cost analysis, which measures a different cost endpoint. Appendix E further separates planning, batching, parallelism, and objective sensitivity, clarifying when additional computation reduces cost or latency.

5 Conclusion

SpecRSI's two-level loop extends incompletely evaluated parents but adopts only complete, admissible versions. VATS allocates edits and complementary checks by predicted net verified return; GVO updates the improver from completed searches under a fixed verification contract. SpecRSI outperforms Janus in all 50 scientific backbone–task pairs and MetaRSI-v1 by 1.85–2.38 pp in four-task macro-average score across twenty backbones. Full planning achieves the highest mean verified gain and lowest total-cost index among tested planning conditions at all eight budget caps. Starting-artifact controls support gains from improver updates beyond retaining stronger task agents. Learned allocation thus turns speculation into verified agent improvement under limited budgets.

AI USE STATEMENT

Generative AI tools assisted with manuscript revision and polishing. The authors take responsibility for the final text, figures, results, and claims.

ETHICS STATEMENT

This work studies computational methods for recursive agent self-improvement on scientific-discovery and agent-system benchmarks. It does not involve human-subject experiments. Data, model, and tool use are subject to the terms of their original providers.

REPRODUCIBILITY STATEMENT

Section 3 and Appendix F describe the search and training algorithms. Section 4.1 summarizes the tasks, models, baselines, metrics, and repetition scheme. Appendices B–E provide additional comparisons and ablations, while Appendix G presents the theoretical assumptions and proofs.

REFERENCES

Lakshya A. Agrawal, Shangyin Tan, Dilara Soylu, Noah Ziems, Rishi Khare, Krista Opsahl-Ong, Arnav Singhvi, Herumb Shandilya, Michael J. Ryan, Meng Jiang, Christopher Potts, Koushik Sen, Alexandros G. Dimakis, Ion Stoica, Dan Klein, Matei Zaharia, and Omar Khattab. GEPA: Reflective Prompt Evolution Can Outperform Reinforcement Learning. In *International Conference on Learning Representations*, pp. 8479–8565, 2026. URL https://proceedings.iclr.cc/paper_files/paper/2026/hash/0e9e708b6f48e14fd0ac29e167413f76-Abstract-Conference.html.

Arash Ahmadian, Chris Cremer, Matthias Gallé, Marzieh Fadaee, Julia Kreutzer, Olivier Pietquin, Ahmet Üstün, and Sara Hooker. Back to basics: Revisiting REINFORCE-style optimization for learning from human feedback in LLMs. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 12248–12267, 2024. doi: 10.18653/v1/2024.acl-long.662. URL https://aclanthology.org/2024.acl-long.662/.

Dimitri P. Bertsekas. Differential training of rollout policies. In *Proceedings of the 35th Allerton Conference on Communication, Control, and Computing*, 1997. URL https://faculty.engineering.asu.edu/bertsekas/papers-reports.

Shrenik Bhansali and Larry Heck. Draft, verify, and improve: Toward training-aware speculative decoding. *arXiv preprint arXiv:2510.05421*, 2025. URL https://arxiv.org/abs/2510.05421.

Zhuoming Chen, Avner May, Ruslan Svirschevski, Yuhsun Huang, Max Ryabinin, Zhihao Jia, and Beidi Chen. Sequoia: Scalable and Robust Speculative Decoding. In *Advances in Neural Information Processing Systems*, volume 37, pp. 129531–129563, 2024. doi: 10.52202/079017-4116. URL https://proceedings.neurips.cc/paper_files/paper/2024/hash/ea1f5f0878d43ff4fb8bf64ef4a2326c-Abstract-Conference.html.

Amol Deshpande, Lisa Hellerstein, and Devorah Kletenik. Approximation algorithms for stochastic boolean function evaluation and stochastic submodular set cover. In *Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms*, pp. 1453–1466. Society for Industrial and Applied Mathematics, 2014. doi: 10.1137/1.9781611973402.107. URL https://epubs.siam.org/doi/10.1137/1.9781611973402.107.

Rohan Ghuge, Jai Moondra, and Mohit Singh. Stochastic function certification with correlations. *arXiv preprint arXiv:2604.02611*, 2026. URL https://arxiv.org/abs/2604.02611v1.

Fabian Gloeckle, Badr Youbi Idrissi, Baptiste Roziere, David Lopez-Paz, and Gabriel Synnaeve. Better & Faster Large Language Models via Multi-token Prediction. In *Proceedings of the 41st*

International Conference on Machine Learning, volume 235 of *Proceedings of Machine Learning Research*, pp. 15706–15734. PMLR, 2024. URL https://proceedings.mlr.press/v235/gloeckle24a.html.

Tilmann Gneiting. Making and evaluating point forecasts. *Journal of the American Statistical Association*, 106(494):746–762, 2011. doi: 10.1198/jasa.2011.r10138. URL https://doi.org/10.1198/jasa.2011.r10138.

Yilin Guan, Qingfeng Lan, Fei Sun, Dujian Ding, Devang Acharya, Chi Wang, William Yang Wang, and Wenyue Hua. Dynamic speculative agent planning. In *International Conference on Learning Representations*, pp. 7695–7717, 2026. URL https://openreview.net/forum?id=YZ5k2dVj6O.

Lisa Hellerstein, Devorah Kletenik, Naifeng Liu, and R. Teal Witter. Adaptivity gaps for the stochastic boolean function evaluation problem. In *Approximation and Online Algorithms*, volume 13538 of *Lecture Notes in Computer Science*, pp. 190–210. Springer, 2022. doi: 10.1007/978-3-031-18367-6_10. URL https://link.springer.com/chapter/10.1007/978-3-031-18367-6_10.

Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. LoRA: Low-Rank Adaptation of Large Language Models. In *International Conference on Learning Representations*, 2022. URL https://openreview.net/forum?id=nZeVKeeFYf9.

Shengran Hu, Cong Lu, and Jeff Clune. Automated Design of Agentic Systems. In *International Conference on Learning Representations*, pp. 21344–21377, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/hash/36b7acf6f6010652b3f2a433774a66fe-Abstract-Conference.html.

Shijing Hu, Jingyang Li, Zhihui Lu, and Pan Zhou. Bridging draft policy misalignment: Group tree optimization for speculative decoding. In *The Fourteenth International Conference on Learning Representations*, 2026a. URL https://openreview.net/forum?id=dwPdYFqVWO.

Zhengding Hu, Mingge Lu, Zhen Wang, Jixuan Ruan, Chang Chen, Zaifeng Pan, Yue Guan, Ruiyi Wang, Zhongkai Yu, Chao Zhang, and Yufei Ding. FlashEvolve: Accelerating agent self-evolution with asynchronous stage orchestration. *arXiv preprint arXiv:2605.08520*, 2026b. URL https://arxiv.org/abs/2605.08520.

Sungkyun Kim, Jaemin Kim, Dogyeong Yun, Jiho Shin, Junyeol Lee, and Jiwon Seo. Speculative Verification: Exploiting Information Gain for Speculative Decoding. In *Findings of the Association for Computational Linguistics: ACL 2026*, pp. 36290–36307. Association for Computational Linguistics, 2026. doi: 10.18653/v1/2026.findings-acl.1809. URL https://aclanthology.org/2026.findings-acl.1809/.

Juho Lee, Yoonho Lee, Jungtaek Kim, Adam Kosiorek, Seungjin Choi, and Yee Whye Teh. Set transformer: A framework for attention-based permutation-invariant neural networks. In *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pp. 3744–3753, 2019. URL https://proceedings.mlr.press/v97/lee19d.html.

Yaniv Leviathan, Matan Kalman, and Yossi Matias. Fast Inference from Transformers via Speculative Decoding. In *International Conference on Machine Learning*, pp. 19274–19286, 2023. URL https://proceedings.mlr.press/v202/leviathan23a.html.

Yuhui Li, Fangyun Wei, Chao Zhang, and Hongyang Zhang. EAGLE-2: Faster Inference of Language Models with Dynamic Draft Trees. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pp. 7421–7432. Association for Computational Linguistics, 2024. doi: 10.18653/v1/2024.emnlp-main.422. URL https://aclanthology.org/2024.emnlp-main.422/.

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 *International Conference on Learning Representations*, pp. 39578–39601, 2024. URL https://proceedings.iclr.cc/paper_files/paper/2024/hash/aca97732e30bcf1303bc22ac3924fd16-Abstract-Conference.html.

Ximeng Liu, Qianlong Wang, Yingming Mao, Annan Li, Yatao Li, Shizhen Zhao, Jianmin Wu, Dawei Yin, and Dou Shen. Janus: An Algorithm-Evaluator Co-Evolution Framework for LLM-Driven Discovery under Expensive Evaluation Budgets. *arXiv preprint arXiv:2608.08189*, 2026. URL https://arxiv.org/abs/2608.08189.

Xupeng Miao, Gabriele Oliaro, Zhihao Zhang, Xinhao Cheng, Zeyu Wang, Zhengxin Zhang, Rae Ying Yee Wong, Alan Zhu, Lijie Yang, Xiaoxiang Shi, Chunan Shi, Zhuoming Chen, Daiyaan Arfeen, Reyna Abhyankar, and Zhihao Jia. SpecInfer: Accelerating Large Language Model Serving with Tree-based Speculative Inference and Verification. In *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 3*, pp. 932–949. Association for Computing Machinery, 2024. doi: 10.1145/3620666.3651335. URL https://doi.org/10.1145/3620666.3651335.

Maxime Robeyns, Martin Szummer, and Laurence Aitchison. A Self-Improving Coding Agent. *arXiv preprint arXiv:2504.15228*, 2025. URL https://arxiv.org/abs/2504.15228. Presented at the ICLR 2025 Workshop on Scaling Self-Improving Foundation Models (Oral).

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.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, Y. K. Li, Y. Wu, and Daya Guo. DeepSeekMath: Pushing the Limits of Mathematical Reasoning in Open Language Models. *arXiv preprint arXiv:2402.03300*, 2024. URL https://arxiv.org/abs/2402.03300.

Charlie Snell, Jaehoon Lee, Kelvin Xu, and Aviral Kumar. Scaling LLM Test-Time Compute Optimally Can be More Effective than Scaling Parameters for Reasoning. In *International Conference on Learning Representations*, pp. 10131–10165, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/hash/1b623663fd9b874366f3ce019fdfdd44-Abstract-Conference.html.

Zihan Tan, Leixin Sun, Zitong Shi, Yitao Liu, Jiajun Wu, Nathaniel Brooks, Jiaru Qian, Xiaoran Shang, Suyuan Huang, Yi Ding, Yangxu Liao, Mukai Li, Qiushi Sun, Shudong Liu, Xuankun Rong, Xiaohang Yu, Zhuo Chen, Hejia Geng, Chenxin Li, Aozhou Wang, Zengji Tu, Robert Tang, Yuxin Zhan, Eric Jiang, Yuxin Wu, Jianqing Zhang, Xiao Liang, Fang Wu, Haochi Zhang, Alexander Marlow, and Guancheng Wan. MetaRSI / RSI^2: A meta-recursive self-improving system for recursive self-improving systems themselves. *arXiv preprint arXiv:2609.06396*, 2026. URL https://arxiv.org/abs/2609.06396v2. Version 2, 9 September 2026.

Kun Wang, Guibin Zhang, ManKit Ye, Xinyu Deng, Dongxia Wang, Xiaobin Hu, Jinyang Guo, Yang Liu, and Yufei Guo. MAS^2: Self-generative, self-configuring, self-rectifying multi-agent systems. In *International Conference on Learning Representations*, 2026a. URL https://openreview.net/forum?id=qumy27hMDY.

Wenyi Wang, Piotr Piękos, Li Nanbo, Firas Laakom, Yimeng Chen, Mateusz Ostaszewski, Mingchen Zhuge, and Jürgen Schmidhuber. Huxley-Gödel Machine: Human-Level Coding Agent Development by an Approximation of the Optimal Self-Improving Machine. In *International Conference on Learning Representations*, pp. 80356–80386, 2026b. URL https://proceedings.iclr.cc/paper_files/paper/2026/hash/821d20219c2f14850af1b5220f0ed13f-Abstract-Conference.html.

Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Thomas L. Griffiths, Yuan Cao, and Karthik Narasimhan. Tree of Thoughts: Deliberate Problem Solving with Large Language Models. In *Advances in Neural Information Processing Systems*, volume 36, pp. 11809–11822, 2023. doi: 10.52202/075280-0517. URL https://proceedings.neurips.cc/paper_files/paper/2023/hash/271db9922b8d1f4dd7aaef84ed5ac703-Abstract-Conference.html.

Naimeng Ye, Arnav Ahuja, Georgios Liargkovas, Yunan Lu, Kostis Kaffes, and Tianyi Peng. Speculative Actions: A Lossless Framework for Faster AI Agents. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=P0GOk5wslg.

Guibin Zhang, Haotian Ren, Chong Zhan, Zhenhong Zhou, Junhao Wang, He Zhu, Wangchunshu Zhou, and Shuicheng Yan. MemEvolve: Meta-evolution of agent memory systems. *arXiv preprint arXiv:2512.18746*, 2025a. URL https://arxiv.org/abs/2512.18746. Preprint version; also presented at ICML 2026.

Jenny Zhang, Shengran Hu, Cong Lu, Robert Lange, and Jeff Clune. Darwin Gödel Machine: Open-Ended Evolution of Self-Improving Agents. In *International Conference on Learning Representations*, pp. 104223–104294, 2026a. URL https://proceedings.iclr.cc/paper_files/paper/2026/hash/aa5f5e6eb6f613ec412f1d948dfa21a5-Abstract-Conference.html.

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

Jiayi Zhang, Jinyu Xiang, Zhaoyang Yu, Fengwei Teng, Xionghui 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*, pp. 34040–34077, 2025b. URL https://proceedings.iclr.cc/paper_files/paper/2025/hash/5492ecbce4439401798dcd2c90be94cd-Abstract-Conference.html.

Yifei Zhou, Andrea Zanette, Jiayi Pan, Sergey Levine, and Aviral Kumar. ArCHer: Training language model agents via hierarchical multi-turn RL. In *International Conference on Machine Learning*, pp. 62178–62209, 2024. URL https://proceedings.mlr.press/v235/zhou24t.html.

CONTENTS FOR APPENDIX

A Extended Related Work **15**

B Overall Performance: Complete Comparisons **16**

B.1 Scientific Performance 16

B.2 Matched-Permission Results 16

B.3 Backbone Comparisons 17

B.4 Coding and Audit 17

B.5 Prompts and Workflows 19

C Why Verification-Aware Search Helps **19**

C.1 Information and Work Scope 19

C.2 Planning Ablations 20

C.3 Evidence and Relations 20

C.4 Prediction and Verification 21

D Learning the Improver Across Rounds **22**

D.1 Starting-Artifact Controls 22

D.2 Role Updates 23

D.3 Supervision and Cost 23

D.4 Repeated Rounds 24

E Resource Trade-offs and Limits **25**

E.1 Planning Capacity 25

E.2 Verification Blocks 26

E.3 Time and Total Work 27

E.4 Cost Sensitivity 27

F Algorithms and Training **28**

F.1 RSI State 28

F.2 Notation and Evaluation 29

F.3 Tree Search and Solver 30

F.4 Verification and Execution 33

F.5 Two-Stage Training 33

G Theoretical Analysis **37**

G.1 Validity and Termination 37

G.2 Completion and Equivalence 38

G.3 Feedback Costs 39

G.4 Evidence and Scope 40

G.5 Matched Scope Control 43
G.6 Menu Approximation 43
G.7 Learning Identities 44

H Nomenclature **45**

READING GUIDE.

At the outset of this appendix, we clarify the aim of the manuscript: to develop a budget-aware search procedure and evaluate its performance, verification strategies, and learning across rounds. To help readers navigate the manuscript, Figure S1 maps the main sections to their supporting appendices.

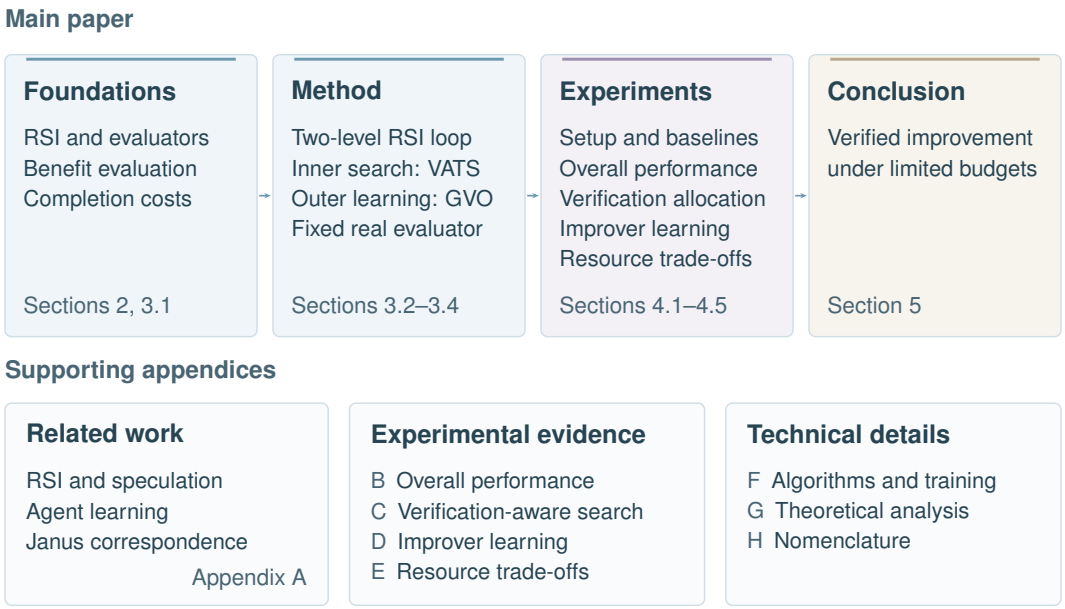


Figure S1: Reading map of the manuscript. Arrows indicate the order of exposition; appendix labels locate related work, supporting comparisons, analyses, and terminology.

A EXTENDED RELATED WORK

Recursive self-improvement. Unlike within-task reasoning search or adaptive inference compute (Yao et al., 2023; Snell et al., 2025), agent improvement persistently revises task solvers and their improvers. DGM, SICA, ADAS, AFlow and GEPA iteratively revise and evaluate code, workflows, or prompts (Zhang et al., 2026a; Robeyns et al., 2025; Hu et al., 2025; Zhang et al., 2025b; Agrawal et al., 2026); MemEvolve co-evolves experience and memory programs, while Hyperagents and MetaRSI adapt improvement generation or scheduling (Zhang et al., 2025a; 2026b; Tan et al., 2026). HGM separates lineage-metaproductivity-guided expansion from agent–task evaluation (Wang et al., 2026b); Janus co-evolves programs and proxies but retains real promotion checks (Liu et al., 2026). SpecRSI (ours) uses VATS to connect exploration to evaluation completion, jointly selecting branches and outstanding-check bundles by attainable verified improvement under the remaining budget.

Speculative execution. Speculative decoding decouples cheap drafting from target verification without changing the target distribution (Leviathan et al., 2023). SpecInfer and EAGLE-2 use token trees; Sequoia adapts their size and depth to hardware costs (Miao et al., 2024; Li et al., 2024; Chen et al., 2024). Multi-token prediction learns future-token heads for speculative inference (Gloeckle et al., 2024); Speculative Verification adapts verification length to predicted draft–target alignment (Kim

et al., 2026). Dynamic Speculative Agent Planning and Speculative Actions speculate on planning or executing actions within a task (Guan et al., 2026; Ye et al., 2026); FlashEvolve asynchronously overlaps evolution stages, optionally starting downstream work from partially evaluated artifacts (Hu et al., 2026b). SpecRSI (ours) separates executable-successor proposals from residual-check confirmation: incomplete parents may expand, but adoption requires complete, admissible outcomes for the selected version, not every ancestor. The analogy concerns verification expenditure, not exact token sampling or one-forward-pass agent evaluation.

Learning for Agent Improvement. Process supervision learns step evaluators; hierarchical agent RL learns long-horizon values and policies (Lightman et al., 2024; Zhou et al., 2024). GRPO estimates relative advantages for same-prompt responses (Shao et al., 2024); DVI trains token drafters from verifier feedback; GTO targets expected tree acceptance length (Bhansali & Heck, 2025; Hu et al., 2026a); MAS^2 trains meta-agents with value-weighted preferences from cost-sensitive terminal outcomes (Wang et al., 2026a). GVO instead credits independent complete successor searches from identical starting conditions, not responses or shared-tree leaves. Only complete admissible versions earn improvement; computation, elapsed time, failed and unfinished work remain charged. Established leave-one-out baselines and clipped updates train the drafter and work-selection policy (Ahmadian et al., 2024; Schulman et al., 2017); observed-return regression trains the scorer, with paid alternative continuations supplying scorer-only labels. These updates improve later searches while the evaluator stays fixed. The contribution is this verification-grounded, cost-sensitive learning unit integrated with successor search, not a new policy-gradient estimator.

Table S1 summarizes the functional correspondence between Janus and SpecRSI (ours) (Liu et al., 2026).

Table S1: **Functional correspondence between Janus and SpecRSI (ours).** The Scorer predicts continuation value; the real evaluator supplies evidence for adoption.

Janus	SpecRSI (ours)
LLM program generator	Drafter / DRAFT: propose executable edits
Target program / offspring	Immutable successor agent c'
Proxy evaluator	Scorer: learned score evaluator $\widehat{Q}_\phi(S, \omega)$
Promotion / acquisition rule	VATS Solver + Policy: allocate draft and check work
Real evaluator f_{real}	Γ + Acquire_Γ; invoked by VERIFY
Real-validated incumbent	ACCEPT: best complete admissible version
Proxy-evaluator adaptation	GVO Scorer update from acquired outcomes
Program / evaluator co-evolution	Agent / improver two-level loop

B Overall Performance: Complete Comparisons

This appendix expands Section 4.2. The question is whether the method returns better successors within each comparable profile, not whether unlike task scores can be combined into a single ranking. Table 1 and Figure S3 contain the scientific, system, and coding comparisons; the additional tables below supply their costs and the prompt/workflow profiles.

B.1 Scientific Performance

The five columns of Table 1A are five native objectives. SpecRSI (ours) has the largest reported terminal gain and AUBC in each. For example, PyBaMM terminal gain changes from 0.36 for Janus to 0.48, a 33.3% relative increase, not a 33.3 pp task-score increase.

Table S2 keeps evaluation, proposal, proxy-training, and planning costs visible. Its total is 158.7 for SpecRSI (ours) and 148.1 for Janus: a 7.2% premium.

B.2 Matched-Permission Results

The system comparison controls what each method is allowed to modify. D, H, and M denote data, harness, and model edits. D+H excludes model updates; it is not a harness-only setting. Within

Table S2: **Scientific cost breakdown.** Normalized source indices, not monetary lifecycle totals; no component SD was supplied.

Category	Alg.-Only	Direct-LLM	Embed-GP	Janus	SpecRSI (ours)
Evaluation	**100.0**	**100.0**	**100.0**	**100.0**	**100.0**
Proposals	**0**	12.4	18.7	22.3	29.5
Proxy training	**0**	**0**	15.2	18.6	20.7
Planning	**0**	**0**	**0**	7.2	8.5
Total	**100.0**	112.4	133.9	148.1	158.7

D/H/M, SpecRSI (ours) exceeds MetaRSI-v1 by 1.6, 2.2, 2.3, and 2.9 pp on TB2.1, SWE-Pro, GPQA-hard100, and AIME. The corresponding D+H differences are 1.4, 2.1, 2.4, and 2.9 pp. Both permission bands remain in Table 1B alongside the single-surface, fixed-schedule, and static-router controls.

The separate operator-combination study in Table S3 asks a narrower question about allowed combinations and their costs. Its full SpecRSI row has a 7.2% cost premium over the full MetaRSI row.

Table S3: **Operator combinations.** Prior deployment-profile study; absolute gains and cost ratios. Ranks compare listed gains (↑) and costs (↓) descriptively; restricted subsets lack a matched factorial.

Operators	Gain	Relative gain	Relative cost
D only	2.8	21.4%	**32.5%**
H only	6.6	50.4%	58.7%
M only	3.9	29.8%	65.2%
D+H	7.7	58.8%	67.3%
D+M	5.2	39.7%	72.1%
H+M	8.9	67.9%	84.6%
D+H+M (MetaRSI-v1)	10.9	83.2%	100.0%
D+H+M / SpecRSI (ours)	**13.1**	**100.0%**	107.2%

B.3 Backbone Comparisons

Figure S2 provides the scientific and system backbone comparisons summarized in Section 4.2.

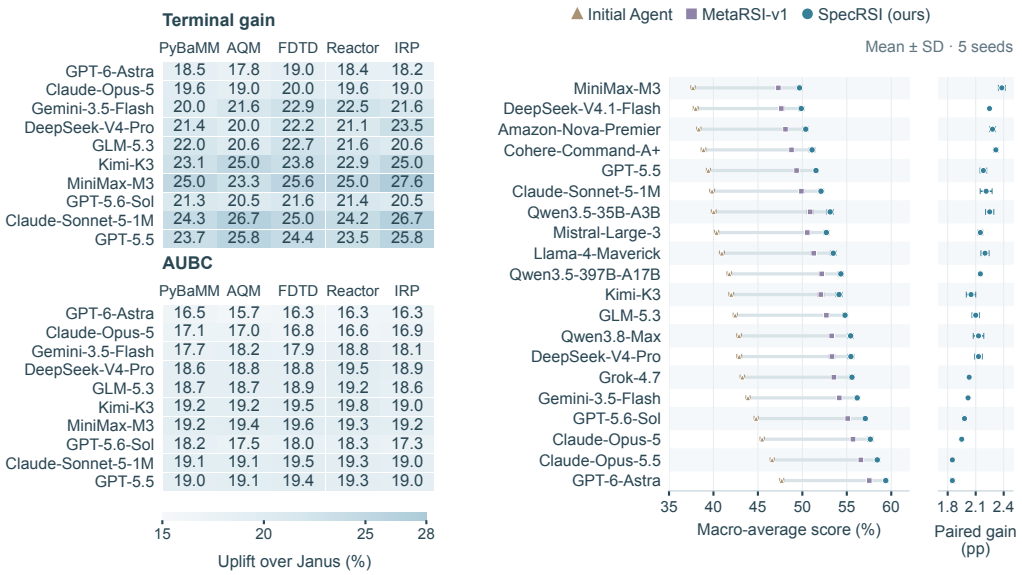
B.4 Coding and Audit

Coding validation measures the resulting agent's success, while the asynchronous completion profile measures whether the relevant execution completes. They are different endpoints, even when method names coincide. Figure S3 reports coding-validation means and target costs. The corresponding SDs are 1.0, 1.1, 1.1, 1.0, 1.1, 1.2, and 1.2 pp for DGM, SICA, HGM, Beam-Spec, FlashEvolve async, FlashEvolve optional, and SpecRSI, respectively.

Table S4: **Asynchronous completion.** Completion rate supplements the coding-validation scores; it is not another estimate of coding success. The two native FlashEvolve modes remain separate.

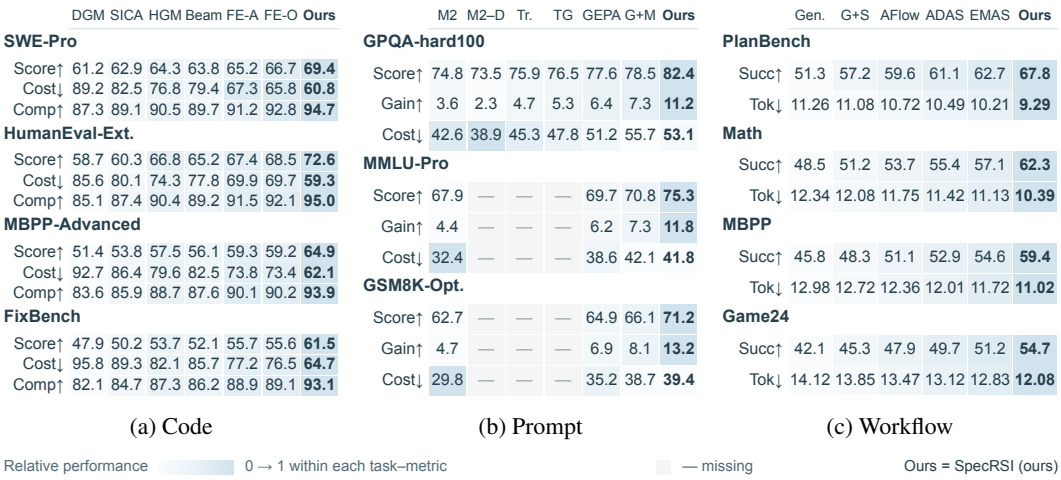
Method	Complete (%)
native FlashEvolve async-stages	91.2
native FlashEvolve optional-spec	92.8
SpecRSI (ours)	**94.7**

The frozen-successor audit assesses a selected, frozen artifact on a separately declared untouched split. It reports a SpecRSI (ours) score of $69.4 \pm 1.3\%$ and 23/25 target hits (Table S5).



(a) Scientific tasks (b) System tasks

Figure S2: **Performance across backbones.** (a) Relative terminal-gain and AUBC uplift over Janus across ten backbones and five scientific tasks. (b) Macro-average scores over TB2.1, SWE-Pro, GPQA-hard100, and AIME across twenty backbones, ordered by Initial Agent mean. Within-backbone segments connect the three methods; markers and horizontal error bars show means and ± 1 sample SD over five training seeds. The right-hand difference panel shows seed-paired SpecRSI (ours) − MetaRSI-v1 differences in percentage points with the same uncertainty convention.



(a) Code (b) Prompt (c) Workflow

Figure S3: **Performance across code, prompt, and workflow tasks.** Reported means: Score, Succ., and Comp. in percent; Gain in pp; Tok. in thousands. Darker cells indicate better within-metric performance; dashes mark unreported values. Code completion uses a separate profile, and costs retain task-specific definitions. Beam: Beam-Spec; FE-A/FE-O: FlashEvolve async/optional; M2/M2–D: MIPROv2 with/without demos; Tr.: Trace; TG: TextGrad; G+M: GEPA+Merge; Gen./G+S: Genesis/Genesis+SOP.

Table S5: **Separately reported frozen-successor audit.** These are not coding-validation replicates. Declared isolation requires the original frozen artifacts and split records.

Method	Audit score (%)	Target hits
SpecRSI (ours)	69.4 ± 1.3	23/25

B.5 Prompts and Workflows

These profiles test whether the result is restricted to agent-code editing. Each retains its own initial agent and benchmark. For GPQA-hard100 prompt optimization, SpecRSI (ours) reaches 82.4%, compared with 78.5% for GEPA+Merge (Table S6).

Table S6: **Prompt optimization.** GPQA-hard100 means; no reported SD. Gain is an absolute change on the % score scale.

Method	Score (%)	Gain (%, abs.)
MIPROv2	74.8	+3.6
MIPROv2 (no demos)	73.5	+2.3
Trace	75.9	+4.7
TextGrad	76.5	+5.3
GEPA	77.6	+6.4
GEPA+Merge	78.5	+7.3
SpecRSI (ours)	**82.4**	**11.2**

On PlanBench, SpecRSI (ours) reaches 67.8% success while using 9,290 task-inference tokens, versus 62.7% and 10,210 for EMAS (Table S7). This is a serving-token comparison, not a total-search-cost comparison. Table S8 separately reports SpecRSI scores and token use on Math, MBPP, and Game24.

Table S7: **Workflow optimization on PlanBench.** Success is a task score; tokens count task inference only, excluding search and training. Both columns retain the supplied means.

Method	Success (%)	Task tokens
Genesis	51.3	11260
Genesis+SOP	57.2	11080
AFlow	59.6	10720
ADAS	61.1	10490
EMAS	62.7	10210
SpecRSI (ours)	**67.8**	**9290**

Table S8: **Additional workflow tasks.** Success (%) / serving tokens; PlanBench is already in Table S7.

Method	Math	MBPP	Game24
SpecRSI (ours)	62.3 / 10390	59.4 / 11020	54.7 / 12080

C Why Verification-Aware Search Helps

This appendix expands Section 4.3. It separates three explanations that would otherwise be conflated: seeing more information, offering completion-aware work, and choosing how much work to execute before observing feedback. The latter's resource regimes are examined separately in Appendix E.2.

C.1 Information and Work Scope

The information–scope study crosses two information levels (summary or full state) with two work scopes (one check or a bundle). The relevant comparison is between rows that hold one factor fixed, not just the weakest and strongest rows. With full state, gain rises from 4.8 to 6.3 pp and total cost falls from 63.1 to 59.2 when bundles are permitted (Table S9). Failure falls from 9.8% to 6.5%, while hit rate rises from 84.0% to 92.0%.

With summaries only, the corresponding gain contrast is 4.1 minus 3.2 pp. The difference of these two contrasts is $(6.3 - 4.8) - (4.1 - 3.2) = 0.6$ pp. This is a descriptive interaction, not a statistical test. It supports studying information and work scope separately; it does not show that every bundle

is useful or that the full-state singleton controller is optimal. The finite output-matched analysis in Appendix G.5 addresses a distinct, exactly specified decision problem.

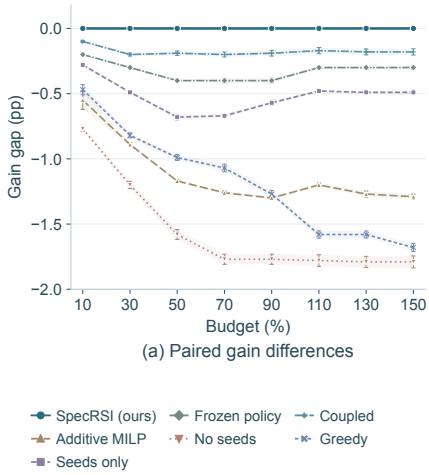
Table S9: **Verification scope with information held fixed.** Gain is in percentage points (pp), with reported SD. Cost is the study's total-cost index; failure and hit rates retain the study's definitions. The two full-state rows isolate work scope at a fixed information level.

Information	Work scope	Gain (pp)↑	Cost↓	Fail (%)↓	Hit (%)↑
Summary	One check	3.2 ± 0.21	67.4	15.2	72.0
Summary	Bundle	4.1 ± 0.24	65.8	12.7	78.0
Full state	One check	4.8 ± 0.25	63.1	9.8	84.0
Full state	Bundle	$\mathbf{6.3} \pm 0.26$	**59.2**	**6.5**	**92.0**

C.2 Planning Ablations

The planning ablation changes how the menu is built and selected, rather than changing the success criterion. Additive MILP uses the additive planning control; completion seeds retain candidate-completing check packages; refinement uses whole-plan continuation predictions. Frozen-policy, coupled scorer/policy, and greedy-selection rows then separate refinement from the final selection rule. The figure's "scorer" is the table's "value" model.

At the standard budget, removing seeds lowers gain from 6.3 to 4.5 pp, and greedy value selection reaches 4.8 pp. At the 70% cap, the corresponding values are 6.0, 4.2, and 4.9 pp. These are independent budget cohorts with their own seed pools, not extra replicates of the same condition. Figure S5 makes the trade-off explicit: full planning spends more time preparing work yet has a lower total-cost index.



(a) Paired gain differences

	10	30	50	70	90	110	130	150
Full (ours)	**199 ±5.5**	**200 ±5.6**	**200 ±5.6**	**201 ±5.7**	**202 ±5.8**	**203 ±5.9**	**204 ±6.0**	**205 ±6.1**
Additive	139 ±4.2	140 ±4.3	140 ±4.3	141 ±4.4	141 ±4.4	143 ±4.5	143 ±4.5	144 ±4.6
Seeds only	166 ±4.8	167 ±4.9	167 ±4.9	168 ±5.0	168 ±5.0	169 ±5.1	170 ±5.1	170 ±5.2
Frozen policy	184 ±5.1	185 ±5.2	186 ±5.2	186 ±5.3	187 ±5.3	188 ±5.4	188 ±5.4	189 ±5.5
No seeds	132 ±3.9	133 ±4.0	134 ±4.0	135 ±4.1	135 ±4.1	136 ±4.2	137 ±4.2	137 ±4.3
Coupled	192 ±5.3	193 ±5.4	193 ±5.4	194 ±5.5	194 ±5.5	196 ±5.6	196 ±5.6	197 ±5.7
Greedy	115 ±3.5	116 ±3.6	116 ±3.6	117 ±3.7	118 ±3.7	119 ±3.8	119 ±3.8	120 ±3.9

Budget (%)

(b) Planning time

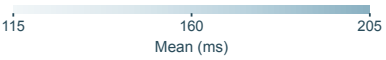


Figure S4: **Paired gain differences and planning time.** (a) Method minus Full SpecRSI (ours) gain uses the separately reported paired-seed difference means and SDs, not differences recomputed from Figure 4b. Full relative to itself is zero. Lines join means; bars/bands show ±1 SD, not SE or confidence intervals. (b) Color encodes planning-time mean; cells report mean ± SD in milliseconds. Methods and eight budget caps match Figure 4; Full (ours) denotes Full SpecRSI (ours). Seeds only: completion seeds only; No seeds: no completion seeds; Frozen policy: refinement/frozen policy; Coupled: coupled scorer/policy; Greedy: greedy scorer selection.

C.3 Evidence and Relations

The evidence-input study asks what the controller needs to know about work already done. "Allocator blind" withholds the evidence features; "Mask only" provides missingness; "No relations" removes relation information; "Restored valid fields" reinstates valid observed fields. Full evidence is the

	70% gain	100% gain
Additive MILP	4.7	5.2
Completion seeds only	5.3	5.8
Refinement / frozen policy	5.6	6.0
No completion seeds	4.2	4.5
Coupled scorer / policy	5.8	6.1
Greedy scorer selection	4.9	4.8
Full SpecRSI (ours)	6.0	6.3

(a) Gain at 70% and 100% budgets

	70% Δg	100% Δg
Additive MILP	−1.3	−1.1
Completion seeds only	−0.7	−0.5
Refinement / frozen policy	−0.4	−0.3
No completion seeds	−1.8	−1.8
Coupled scorer / policy	−0.2	−0.2
Greedy scorer selection	−1.1	−1.5
Full SpecRSI (ours)	0.0	0.0

(b) Difference from the same-cohort full method

	70%		100%	
	ms	cost	ms	cost
Additive MILP	141	65.1	142	65.2
Completion seeds only	168	63.0	169	63.1
Refinement / frozen policy	186	61.7	187	61.8
No completion seeds	135	67.4	136	67.5
Coupled scorer / policy	194	60.6	195	60.7
Greedy scorer selection	117	64.2	118	64.3
Full SpecRSI (ours)	201	58.9	203	59.2

(c) Planning time and total cost

	70%				100%			
	fin.	chk.	dep.	cand.	fin.	chk.	dep.	cand.
Additive MILP	78.2	2.0	1.5	3.0	78.8	2.1	1.6	3.1
Completion seeds only	82.9	2.3	1.9	3.5	83.7	2.4	2.0	3.6
Refinement / frozen policy	84.5	2.5	2.1	3.8	85.3	2.6	2.2	3.9
No completion seeds	76.1	1.9	1.4	2.8	76.8	2.0	1.5	2.9
Coupled scorer / policy	86.2	2.6	2.2	3.9	87.0	2.7	2.3	4.0
Greedy scorer selection	79.0	2.1	1.6	3.1	79.7	2.2	1.7	3.2
Full SpecRSI (ours)	88.9	2.7	2.5	4.1	90.3	2.8	2.6	4.3

(d) Completion and executed work

Figure S5: **What changes when planning components are removed?** These are the original reduced- and standard-budget panels, not four experiments. A/B show raw gains and each cohort's difference from its own full method; C shows planning milliseconds and total-cost indices. In D, fin. is finalist completion (%), chk. checks per block, dep. draft depth, and cand. candidates. A larger work count is not intrinsically better. Except for B, shading is within-column scaling; darker does not always mean better. The expanded eight-budget gain/cost curves and SDs are in Figure 4; paired differences and planning times are in Figure S4.

reference. These interventions affect the controller's inputs, not which predictions are accepted as real outcomes.

Table S10 reports gain and the observed draft depth/checks per block. Full evidence reaches 6.3 pp, versus 3.2 for the blind controller and 5.8 after valid fields are restored. Figure S6 adds costs, hits, failures, and coverage. More checks per block and greater depth are descriptive consequences of the selected work; maximizing either would be a different objective. The normalized heatmap is only another view of the same summaries.

Table S10: **Evidence and selected work.** Gain ± SD; depth and checks per block are observed means, not optimization targets.

Condition	Gain (%, abs.)	Draft depth	Checks/block
Allocator blind	3.2 ± 0.17	1.1	1.1
Mask only	4.7 ± 0.20	1.6	1.6
No relations	3.8 ± 0.18	1.4	1.3
Restored valid fields	5.8 ± 0.23	2.3	2.5
Full evidence	$\mathbf{6.3} \pm 0.25$	2.6	2.8

C.4 Prediction and Verification

The predictor study asks whether the model ranks candidate outcomes accurately and at what fitting cost. It is not the fixed-information work-scope experiment above, and it does not certify a candidate. In Table S11, MAE is prediction error, NDCG@3 summarizes the top-three ranking, and Regret@3 is the supplied top-three regret metric. Their exact study normalizations must be retained.

The source calls one predictive row "SpecRSI evolved evaluator." This is a historical predictor label, not a claim that the Verification Contract Γ is learned or editable. That row has the lowest reported MAE and regret and the highest NDCG, but not the smallest fitting cost. Its quality diagnostics

	gain	cost	hit	fail
Allocator blind	3.2	67.8	71.5	15.7
Mask only	4.7	64.3	82.1	10.9
No relations	3.8	66.2	77.3	12.8
Restored valid fields	5.8	60.9	87.8	7.6
Full evidence	6.3	59.2	92.0	6.5

(a) Gain, cost, target hits, and failures

	dep.	chk.	blocks	cand.
Allocator blind	1.1	1.1	1.0	2.0
Mask only	1.6	1.6	1.5	2.9
No relations	1.4	1.3	1.3	2.5
Restored valid fields	2.3	2.5	2.2	3.7
Full evidence	2.6	2.8	2.6	4.2

(b) Depth, checks, blocks, and candidates

	valid	rel.	blocks
Allocator blind	0.00	0.00	1.0
Mask only	0.35	0.20	1.5
No relations	0.45	0.00	1.3
Restored valid fields	0.85	0.60	2.2
Full evidence	1.00	1.00	2.6

(c) Available fields and relations

	gain	hit	−cost	dep.	scaled chk.	valid	rel.	cand.
Allocator blind	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
Mask only	0.48	0.52	0.41	0.33	0.29	0.35	0.20	0.41
No relations	0.19	0.28	0.19	0.20	0.12	0.45	0.00	0.23
Restored valid fields	0.84	0.80	0.80	0.80	0.82	0.85	0.60	0.77
Full evidence	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00

(d) The same columns on normalized scales

Figure S6: **What information changes verification allocation?** All panels describe the evidence-input study. C reports valid-field fraction, relation coverage, and observed blocks. D min–max rescales the same summaries to $[0, 1]$, reversing cost only; it is not another evaluation. Dep. means draft depth, chk. checks per block, and cand. candidates. Read gain and completion alongside work counts, not counts as objectives. Source gain SDs remain in Table S10; no additional uncertainty is inferred.

must not be conflated with the continuation scorer's same-data-supervision RMSE or with acquired pass/fail records.

Table S11: **Predictive-model diagnostics.** Lower MAE, regret and fitting cost are preferred; higher NDCG is preferred. The historical row name "evolved evaluator" denotes a candidate predictor, not the protected Verification Contract.

Predictor	MAE	NDCG @3	Regret @3	Fit cost
GP	0.087	0.782	0.124	12.3
RF	0.102	0.735	0.156	8.7
MLP	0.095	0.758	0.141	10.5
Quadratic	0.114	0.703	0.172	7.2
RBF-SVR	0.091	0.771	0.133	13.8
Direct LLM	0.136	0.654	0.208	5.3
LLM arithmetic proxy	0.158	0.612	0.247	**3.9**
SpecRSI evolved evaluator (ours)	**0.076**	**0.835**	**0.098**	11.7

D Learning the Improver Across Rounds

This appendix expands Section 4.4. The experiments progress from controlling the starting artifact, to changing the learned roles and supervision, to running repeated improvement rounds. They test whether the search procedure improves, in addition to the task artifact.

D.1 Starting-Artifact Controls

Reset controls return the task agent to the registered starting version while changing the improver. Carry controls retain the stronger task agent. The released-artifact fork is a third comparison: both continuations start from the same already released version, but use original or updated improvers. Comparing across these three starting conditions without holding the starting artifact fixed would conflate learning with artifact retention.

In the reset condition, frozen and all-role-updated improvers obtain 4.1 ± 0.18 and 5.2 ± 0.21 pp; in the carry condition, they obtain 4.8 ± 0.20 and 6.2 ± 0.24 pp (Figure S7a). The difference of update effects is 0.3 pp, but paired runs would be needed for an interaction test.

Starting from the same released artifact, updating the improver raises the mean gain from 5.7 to 6.9 pp and lowers the reported cost index from 54.2 to 50.8 (Table S12). This is stronger evidence for improver transfer than merely carrying a higher-scoring task agent forward, but it does not establish improvement on every possible task family.

	Gain ↑	Cost Regr. ↓	Hit ↓	Fail ↑
Reset				
Frozen	4.1 ± 0.18	0.3		
SFT only	4.6 ± 0.19	0.2		
GVO only	5.0 ± 0.20	0.1		
All roles	**5.2 ± 0.21**	0.1		
Carry				
Frozen	4.8 ± 0.20	0.2		
SFT only	5.3 ± 0.21	0.1		
GVO only	5.8 ± 0.22	0.1		
All roles	**6.2 ± 0.24**	0.0		
Same artifact				
Original	5.7 ± 0.22	0.1		
Updated	**6.9 ± 0.25**	0.0		

(a) Artifact controls

Updated roles				Gain (pp)		
D	Sc	P	Sh	Sci.	Code	Agent
				4.0	4.2	4.1
■				4.7	4.9	4.8
	■			5.2	5.4	5.3
		■		5.0	5.2	5.1
■	■			5.4	5.6	5.5
■		■		5.3	5.5	5.4
	■	■		5.6	5.8	5.7
■	■	■		**6.1**	**6.3**	**6.2**
			■	4.5	4.7	4.6
■	■	■	■	**6.4**	**6.6**	**6.5**

(b) Updated roles

Figure S7: **Starting-artifact controls and updated roles.** (a) Reset, carry, and same-artifact controls; gain is in pp. Reported uncertainty, regression values, and cost/hit/failure shading are retained. Compare rows within each starting condition. (b) Filled switches mark updated roles: D, drafter; Sc, scorer; P, policy; Sh, shared backbone. Columns report Science (Sci.), Code, and Agent gains (pp). The final two rows use a separate shared-backbone extension; training costs are unmatched.

Table S12: **Continuation from the same released artifact.** Released-fork means and costs. The corresponding gain means reappear in the retained-artifact figure; this is not a second replication.

Improver	New-task gain (pp)	Cost index	Regressions
Original	5.7	54.2	0.1
Updated	**6.9**	**50.8**	**0.0**

D.2 Role Updates

Figure S7b shows the role study, which changes the updated parts of the improver: drafter (D), scorer (Sc), policy (P), and the separately permitted shared-backbone extension (Sh). In the supplied science/code/agent aggregates, scorer-only updating has the largest single-role means, 5.2/5.4/5.3 pp; drafter-only gives 4.7/4.9/4.8 and policy-only gives 5.0/5.2/5.1. Updating all three gives 6.1/6.3/6.2.

D.3 Supervision and Cost

The supervision-and-acquisition study asks two different questions. First, holding the acquired arms fixed, does adding pairwise return supervision help the scorer? Second, at the same acquisition expense, is buying matched alternative continuations preferable to ordinary-search acquisition? Table S13 keeps these contrasts separate. Adding the pair term increases the reported Gain from 5.4 ± 0.23 to 6.3 ± 0.25 pp and lowers RMSE from 1.05 to 0.85, while total training cost increases from 23.7 to 28.4 (19.8%). The equal-acquisition ordinary-search row has 5.1 ± 0.22 pp gain, not the same data or the same total fitting expense.

Table S13: **Learning signal versus data acquisition.** The first two rows use the reported same acquired arms and differ in loss. The third uses ordinary-search acquisition at the reported equal acquisition cost. Gain follows the source Gain field; total training cost is not matched.

Condition	Gain (pp)↑	RMSE↓	Training cost↓
Same arms / means	5.4 ± 0.23	1.05	23.7
Same arms / means + pair	$\mathbf{6.3} \pm 0.25$	**0.85**	28.4
Equal-acquisition-cost search	5.1 ± 0.22	1.17	25.3

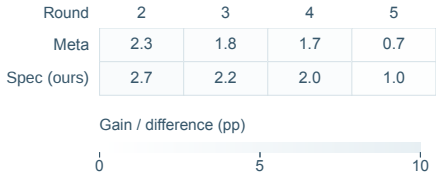
The earlier acquisition-control record reports reset gain, not the supervision study's Gain endpoint (Table S14). The same-arm means and means-plus-pair rows give 4.8 and 5.3 pp; the extra ordinary-search row gives 4.5 pp.

Table S14: **Continuation acquisition.** Paid runs are reported averages; equal-data and equal-cost controls answer different questions.

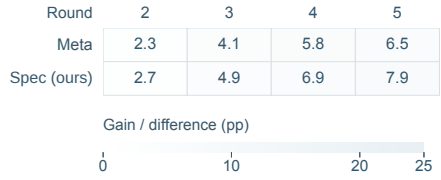
Acquisition	Paid runs	Data cost	Reset gain	State relation
Ordinary searches	24.6	1.0x	4.2	Independent
Extra ordinary equal-cost	24.6	1.0x	4.5	Independent
Same arms / means	24.6	1.0x	4.8	Paired
Same arms / means + pair	24.6	1.0x	**5.3**	Paired
Different policy snapshot	24.6	1.0x	4.0	Mismatched

D.4 Repeated Rounds

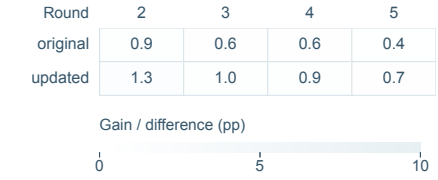
The multi-round study starts at round 1 and reports rounds 2–5. All four method/improver conditions carry the agent forward; "original" versus "updated" refers to the improver, not whether the task agent is reset. Table 2(c) gives the raw increments and cumulative gains. For updated SpecRSI (ours), the increments 8.5, 6.4, 4.7, and 2.1 pp accumulate to 21.7 pp. Updated Meta accumulates 17.8 pp, while original-improver SpecRSI (ours) accumulates 13.8 pp.



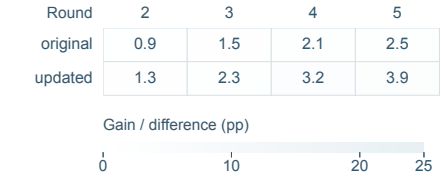
(a) Updating the improver: per-round difference



(b) Updating the improver: cumulative difference



(c) Spec versus Meta: per-round difference



(d) Spec versus Meta: cumulative difference

Figure S8: **Which contrast is being read in the recursive-round study?** A/B subtract original from updated within a method; C/D subtract Meta from Spec (ours) at the same improver status. All values are differences of Table 2(c)'s means, not new runs. Units are percentage points. The supplied color scales are retained; no paired uncertainty is reconstructed.

	Seed	r2	r3	r4	r5
Meta / original	1	4.7	3.5	2.3	0.6
	2	5.0	3.8	2.0	0.8
	3	4.8	3.4	2.2	0.7
	4	5.1	3.7	1.9	0.8
	5	4.9	3.6	2.1	0.6
Meta / updated	1	7.0	5.2	3.9	1.3
	2	7.4	5.6	3.7	1.5
	3	7.1	5.3	4.0	1.4
	4	7.3	5.5	3.6	1.5
	5	7.2	5.4	3.8	1.3
Spec (ours) / original	1	5.6	4.0	2.8	1.0
	2	5.9	4.4	2.6	1.2
	3	5.7	4.1	2.9	1.1
	4	6.0	4.3	2.5	1.2
	5	5.8	4.2	2.7	1.0
Spec (ours) / updated	1	8.3	6.2	4.9	2.0
	2	8.7	6.6	4.5	2.3
	3	8.4	6.3	4.8	2.2
	4	8.6	6.5	4.6	2.1
	5	8.5	6.4	4.7	2.0

Seed-mean increment (pp)

0 5 10

(a) Supplied per-seed mean increments

	Meta orig.	Meta upd.	Spec (ours) orig.	Spec (ours) upd.
Carry root	Yes	Yes	Yes	Yes
Best admissible	Yes	Yes	Yes	Yes
Evaluator fixed	Yes	Yes	Yes	Yes
GVO updates	No	Yes	No	Yes
Rounds	2–5	2–5	2–5	2–5
Baseline round	1	1	1	1
Unit	pp	pp	pp	pp
Training seeds	5	5	5	5
Searches / seed	3	3	3	3

(b) Declared repeated-round protocol

Figure S9: **Repetition and protocol for the recursive-round results.** Each row in A is a supplied seed mean, not an individual search. B declares carried roots in all four conditions, initialization at round 1, evaluated rounds 2–5, five training seeds, and three searches per seed. These records are not converted into unreported confidence intervals or treated as independent of Table 2(c).

E Resource Trade-offs and Limits

This appendix expands Section 4.5. Each intervention changes a different resource: planning effort, committed verification work, or worker count. These are kept separate from repeated improver updates in Appendix D.

E.1 Planning Capacity

The menu-cap study asks whether considering more options is worth the expense of generating and scoring them. Between 32 and 64 plans, gross gain rises from 6.3 to 6.5 pp and the missing-option percentage falls from 3.1% to 2.0%, yet reported net return falls from 4.7 to 4.5 (Table 3).

Table S15: **How should a menu be constructed?** This study varies construction, not just the cap. Forecast and Final are differently defined source fields. Their difference is not a calibrated prediction error or a recovered expense breakdown.

Menu	Forecast	Plan	Work	Final
Singleton only	3.8	**1.2**	54.7	3.7
Seeds	5.2	2.8	55.3	5.1
Seeds + growth	6.1	3.9	54.8	5.9
Larger menu	6.4	5.6	55.1	**6.0**
Exhaustive	**6.7**	8.9	**54.6**	5.8

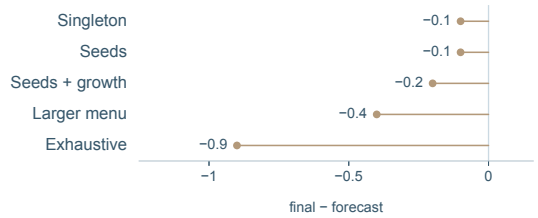
The construction study changes how the menu is assembled, from singleton-only to seeds, growth, larger menus, and exhaustive construction. Its largest Forecast is 6.7, but its Final field is 5.8, below the larger-menu row's 6.0 (Table S15).

cap	gain	SD	over	plan	exec	miss	net	comp	cand	chk	dep	hit	fail	n
8	5.2	0.19	1.8	1.2	0.6	8.6	3.9	76.3	2.1	1.3	1.2	72.5	14.2	15
16	5.8	0.21	2.7	1.9	0.8	5.2	4.4	82.7	3.5	2.1	1.7	78.3	11.6	15
32	6.3	0.25	3.9	2.8	1.1	3.1	4.7	88.5	5.2	3.0	2.2	83.1	8.7	15
64	6.5	0.24	5.6	4.1	1.5	2.0	4.5	91.2	7.8	4.2	2.6	85.4	7.3	15

(a) Offered-plan cap: quality, expense, and executed work

	fore	SD.f	plan	work	final	SD	plans	width	dep	verify	comp	n
Singleton	3.8	0.17	1.2	54.7	3.7	0.16	8	1.0	1.1	0.5	75.2	15
Seeds	5.2	0.20	2.8	55.3	5.1	0.19	16	1.4	1.5	0.8	82.6	15
Seeds + growth	6.1	0.22	3.9	54.8	5.9	0.21	24	1.8	2.0	1.1	87.3	15
Larger menu	6.4	0.23	5.6	55.1	6.0	0.22	32	2.2	2.4	1.4	89.5	15
Exhaustive	6.7	0.24	8.9	54.6	5.8	0.23	60	2.7	2.9	1.9	92.1	14

(b) Construction strategy: forecast, execution, and final result



(c) Final minus Forecast in the construction record

Figure S10: **More planning is useful only if its options repay its cost.** Menu-cap and menu-construction results are separate studies. In A, over is reported overhead, miss the missing-option percentage, and net the reported net return; the outline marks the largest tested net. In B, fore and final are different source fields. C subtracts them without interpreting the gap as a calibrated error or a cost identity. Comp. denotes completion, cand. candidates, chk. checks, dep. depth, and n retained searches. Marginal SDs are retained, not converted into paired intervals. Darker is a within-column larger value, not uniformly a better outcome.

E.2 Verification Blocks

Batching can share dispatch work, but it can also commit checks that would become unnecessary after an early failure. The batch-commitment study compares batch and serial cost in six named regimes (Table S16).

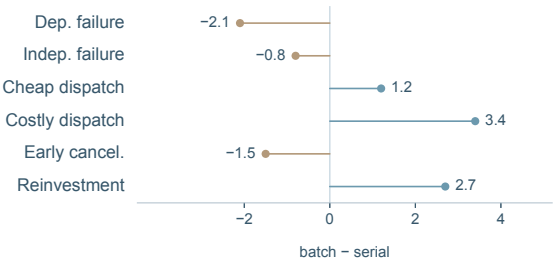
Table S16: **Batch versus serial verification.** Δ is batch cost minus serial cost. Negative values favor batching. Regime names are retained literally; they do not specify all other execution settings.

Regime	Batch	Serial	Δ
Dependent fail	**10.2**	12.3	-2.1
Independent fail	**11.5**	12.3	-0.8
Cheap dispatch	13.2	**12.0**	+1.2
Costly dispatch	15.4	**12.0**	+3.4
Early cancellation	**10.8**	12.3	-1.5
Reinvestment	14.7	**12.0**	+2.7

Figure S11 includes the supplied first-check failure probabilities, cost components, and marginal SDs. These summaries do not identify every joint-survival probability or all coupled execution settings. The finite analysis in Appendix G states the conditions under which a cost crossover follows. The empirical regime names alone are insufficient to test that identity, and the signed differences do not have paired uncertainty supplied.

	batch	SD.b	serial	SD.s	Δ	Δ%	plan	exec	p.fail	disp	chk2	can.	reinv.	net	n
Dep. failure	10.2	0.62	12.3	0.58	−2.1	−17.1	2.2	8.0	0.28	1.2	0.9	—	—	2.7	15
Indep. failure	11.5	0.59	12.3	0.60	−0.8	−6.5	2.0	9.5	0.22	1.2	0.9	—	—	3.4	15
Cheap dispatch	13.2	0.61	12.0	0.63	1.2	10.0	1.7	11.5	0.25	0.8	0.9	—	—	4.1	15
Costly dispatch	15.4	0.65	12.0	0.57	3.4	28.3	1.9	13.5	0.25	2.5	0.9	—	—	3.8	15
Early cancel.	10.8	0.60	12.3	0.62	−1.5	−12.2	2.1	8.7	0.31	1.2	0.9	1.8	—	3.0	14
Reinvestment	14.7	0.63	12.0	0.59	2.7	22.5	2.0	12.7	0.24	1.2	0.9	—	1.6	4.5	15

(a) Reported cost components and execution conditions



(b) Batch minus serial cost

Figure S11: **Batch commitment can save or waste work.** Negative differences favor batching; positive differences favor serial execution. SD.b/SD.s are the supplied marginal SDs, not the SD of the difference. The component fields retain planning, execution, first-check failure probability, dispatch, checks, cancellation, reinvestment, net and search-count diagnostics. Dashes mean not supplied. Stems in B run from zero to the mean difference; they are not error bars. Labels such as "costly dispatch" do not hold every other setting fixed.

E.3 Time and Total Work

The worker-scaling study changes the number of workers, not only the allocator's predictions. One worker uses 1,287 seconds of both wall-clock and worker time. Eight workers reduce wall-clock time to 192 seconds but consume 1,403 worker-seconds, while reported gain changes from 6.3 ± 0.25 to 6.5 ± 0.25 pp (Table S17). More parallelism shortens the wait in these configurations but does not make the work free. The comparison is not equal-work strong scaling, and the fastest configuration cannot be called the cheapest on that basis alone.

Table S17: **Worker configurations.** Time is elapsed wall-clock time; Work is total worker time. Quality and total work both change, so this is not an equal-work strong-scaling experiment.

w	Gain (%)	Time (s)	Work (s)
1	6.3 ± 0.25	1287	**1287**
2	6.4 ± 0.24	672	1312
4	6.4 ± 0.26	354	1348
8	$\mathbf{6.5} \pm 0.25$	**192**	1403

E.4 Cost Sensitivity

The lifecycle ledger in Table 3 keeps the reported acquisition, training, search, reuse-checking, copying, failure, and planning categories visible. Their associations are not sufficient to verify that every category is disjoint. The reported total is retained, but no new sum, amortization threshold, or lifecycle-saving percentage is inferred from it.

Table S18 varies the objective's regression penalty. A heavier penalty lowers the reported gain and cost means, but it does not change whether verification evidence is valid.

Table S18: **Regression-penalty sensitivity.** Lower and Upper retain the supplied interval endpoints. The penalty changes the objective, not whether a contracted outcome is valid.

Condition	ΔJ	Lower	Upper	Cost
Baseline	0.0	-	-	1.00
No penalty	**+2.4**	+1.8	+3.0	1.03
Standard	+2.1	+1.6	+2.6	1.00
Heavy	+1.7	+1.2	+2.2	**0.98**

F Algorithms and Training

GVO calls the complete-search procedure, which invokes VATS whenever it needs a new work plan. Parameters change only after collection; versions, evidence and budgets change during search.

Algorithm 1 SpecRSI: Speculative Recursive Self-Improvement

Require: Complete admissible r, fixed Γ, budgets B_0, N_0, workers w; frozen $\theta, \psi, \bar{\phi}, \nu$; mode.
Ensure: Best complete admissible version and execution log $\mathcal{D}$; no within-search parameter updates.

$\mathcal{T}_0 \leftarrow (\{r\}, \varnothing)$; $S_0^- \leftarrow (\mathcal{T}_0, \mathcal{E}_r, \varnothing, B_0, N_0, w)$; $S_{\text{end}} \leftarrow S_0^-$; $\mathcal{D} \leftarrow \varnothing$; $t \leftarrow 0$.
while preparation cap fits B_t^- and $N_t > 0$ **do**
  $(S_t, \Omega_t, \omega_t, o_t, \ell_t^m, \mathcal{P}_t) \leftarrow \text{VATS}(S_t^-; \bar{\phi}, \psi, \nu, \text{mode})$. → Algorithm 2
  $S_{\text{end}} \leftarrow S_t$; $r_t^m \leftarrow -\lambda o_t - \mu \ell_t^m$; append $(S_t^-, S_t, \mathcal{P}_t, r_t^m)$ to $\mathcal{D}$.
  if $\omega_t = \text{ACCEPT}$ **then**
    $S_{t+1}^- \leftarrow S_t$; record a zero-cost terminal transition with $r_t^w = 0$; $t \leftarrow t+1$.
    break
  end if
  if $\omega_t = \text{DRAFT}(P, h)$ **then**
    $Z_t \leftarrow \text{Generate}_\theta(P, h, S_t)$ ▷ grow actual parent–child paths
  else if $\omega_t = \text{VERIFY}(\mathcal{J})$ **then**
    $Z_t \leftarrow \text{Acquire}_\Gamma(\text{Jobs}(\omega_t), S_t)$ ▷ check existing nodes; no new edges
  end if
  $(\Delta\mathcal{T}_t, \Delta\mathcal{E}_t, \kappa_t, \ell_t^w, n_t, \mathcal{B}_t) \leftarrow Z_t$ ▷ $\Delta\mathcal{T}_t = (\Delta\mathcal{C}_t, \Delta\mathcal{A}_t)$
  $(\mathcal{C}_{t+1}, \mathcal{A}_{t+1}) \leftarrow (\mathcal{C}_t \cup \Delta\mathcal{C}_t, \mathcal{A}_t \cup \Delta\mathcal{A}_t)$; $\mathcal{T}_{t+1} \leftarrow (\mathcal{C}_{t+1}, \mathcal{A}_{t+1})$.
  $\mathcal{E}_{t+1} \leftarrow \mathcal{E}_t \cup \{e \in \Delta\mathcal{E}_t : \text{Valid}_\Gamma(e) = 1\}$ ▷ admit observed outcomes
  $(B_{t+1}^-, N_{t+1}, H_{t+1}) \leftarrow (B_t - \kappa_t, N_t - n_t, H_t \| (\omega_t, Z_t))$.
  $S_{t+1}^- \leftarrow \text{Update}(\mathcal{T}_{t+1}, \mathcal{E}_{t+1}, H_{t+1}, B_{t+1}^-, N_{t+1}, w)$.
  $r_t^w \leftarrow F(S_{t+1}^-) - F(S_t) - \lambda\kappa_t - \mu\ell_t^w$; append $(S_t, \omega_t, S_{t+1}^-, Z_t, r_t^w)$ to $\mathcal{D}$.
  $S_{\text{end}} \leftarrow S_{t+1}^-$; $t \leftarrow t+1$.
end while
$T \leftarrow t$; store T, S_{end} in $\mathcal{D}$.
return $\text{ACCEPT}(S_{\text{end}})$ and $\mathcal{D}$ to the caller.
If invoked for training: return records to the caller. → Algorithm 3

Table S19: **Two timescales, three procedures.** Only the outer training procedure updates the improver's parameters.

Procedure	Returns	Changes and calls
Algorithm 1: search	Complete agent and log	Updates versions, evidence and budget; calls Algorithm 2
Algorithm 2: VATS	Feasible selected plan	Settles preparation; fixed feasibility rules and frozen scorer/policy
Algorithm 3: GVO	Updated improver	Calls Algorithm 1; updates only after collection

F.1 RSI State

Equation (4) separates the per-search agent outputs from the group-level parameter update. Algorithm 3 collects independent complete searches before each improver update. Table S20 specifies the permitted writes; the verification contract Γ and outer update rule remain protected.

Table S20: **What changes, and when?** A legal search need not edit every task surface in one round.

Object	Permitted change	Update boundary
Task agent c_k	Prompts, tool code, workflow and scoped persistent memory	Executed edit creates an immutable successor
Task-model snapshot	Weights only when M is allowed	D/H/M profile; absent from direct fixed-weight core
Improver Θ_k	Drafter, continuation value, plan-selection parameters	GVO after complete searches; frozen inside each search
Verification contract Γ	None	Fixed tasks, admission and scoring rules
Outer update rule	None	GVO algorithm fixed; learned parameters may change

Reset controls fix the starting task agent while changing the improver checkpoint. They test transfer beyond artifact retention, not improvement of every individual surface or modification of the verification contract Γ.

F.2 Notation and Evaluation

Table S21: **The state separates versions, observations, and predictions.** These objects are shared by all three algorithms.

Symbol	Meaning
$\Gamma, \mathcal{I}, J_\Gamma$	Fixed verification contract, required items, and score aggregation
$\mathcal{T} = (\mathcal{C}, \mathcal{A})$	Actual immutable versions and executed parent–child edit edges
$\mathcal{E}, H$	Admitted outcome records and observed execution history
m_{ci}, U_c, K_c	Outcome completeness bit, missing items, and candidate–item target keys
$\mathcal{C}_{\text{full}}, F$	Complete admissible versions and their best improvement over the initial root
B, N, w	Remaining expense, primitive-attempt allowance, and worker count
$S^-, S, \widetilde{S}$	Pre-menu state, paid choice state, and conservative proposal state
ω, Ω	Executable work plan and bounded offered plan set
$\mathcal{R}(S)$	Known candidate–job, dependency, setup, and source relations
$\widehat{Q}_\phi, z_\psi$	Single predicted action value (scorer/critic) and a separate policy logit
θ, ϕ, ψ, ν	Drafter, single value/scorer, preference parameters, and fixed menu-generator settings
$\mathcal{D}$	Execution log; not the draft-plan catalog $\mathcal{D}_t$ or $\mathcal{D}_{\text{draft}}$

An agent action solves a task; an improvement action modifies the agent that will solve later tasks. A version-tree node is therefore not a step in one task rollout. Each node freezes prompts, tools, code, and runtime dependencies; a permitted task-weight update also creates a new immutable node. Old artifacts are not mutated when the improver's parameters change.

A record is keyed by candidate identity, task item, checker/executor version, relevant environment state, and required output fields. A successful-looking scalar without those fields is not a complete record. Conversely, a valid record may state that a hard constraint failed; it is admitted to the ledger but does not make the candidate admissible. Aggregation can be nonlinear and may require all outcomes. Incompleteness and failure therefore have different meanings.

Speculative correspondence. A token proposal becomes an executable edit; a draft continuation becomes a path of actual revisions; a verification batch becomes a set of required checks; acceptance becomes selection of a complete admissible agent. These are role correspondences, not a bijection of probability laws. An unlikely edit may be valuable, a failed parent may be repairable, and many external checks need many real executions. A target-model distribution guarantee does not follow from this mapping. What is retained is the separation between inexpensive exploration and authoritative validation.

Revision example. This supplementary illustration instantiates the design principles in Section 3.1; it changes the coding agent's test strategy, not the application it repairs. Version r runs the full test set after each edit. Version c_1 adds a file–test dependency map but keeps that behavior. Version c_2 uses

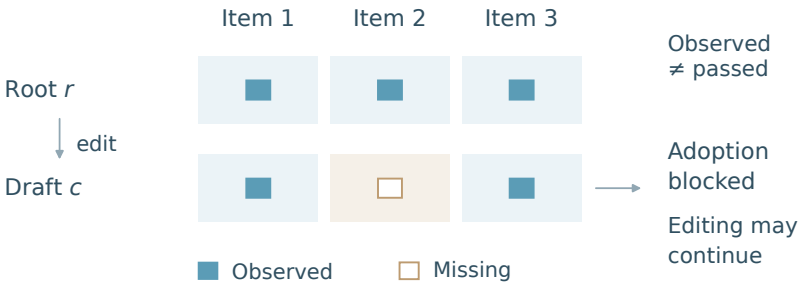


Figure S12: **Observed results, planned work, and predictions have different roles.** Planning may use missingness and source proposals; only contracted direct execution or sound scoped equality adds a valid result. Observed does not mean passed; a missing item blocks adoption but not further editing. The diagram specifies interfaces, not measured data.

the map for intermediate feedback and still runs the full suite before returning a repair. Thus c_1 can enable c_2 without itself showing a score increase. A separate cached-pass branch b_1 must invalidate entries after the corresponding code changes. A stale-pass counterexample is evidence against b_1, not evidence against every child it could later produce. The verification contract Γ remains unchanged and requires separate checks of cross-module behavior and fresh-checkout behavior. Both records are needed for adoption under the illustrative contract. Their outcomes are unknown until executed. The deterministic fixture in `motivation_example/dependency_retest.py` checks these interfaces; it is not a sampled benchmark trajectory or a timing experiment. The earlier parser fixture is archived separately from this example.

F.3 Tree Search and Solver

Search progress. A draft descriptor names existing parents and a bounded path shape. Each dependent edit waits for its actual parent, while independent branches may run in isolation. Invalid edits are charged and end their path; valid prefixes remain searchable. Parents may be internal nodes, incomplete nodes, or previously failed but editable nodes. A verification descriptor names jobs on existing nodes. It adds no version edge. Formally,

$$\begin{aligned} &\text{DRAFT}: \quad \mathcal{C}^+ = \mathcal{C} \cup \Delta\mathcal{C}, \quad \mathcal{A}^+ = \mathcal{A} \cup \Delta\mathcal{A}, \\ &\text{VERIFY}: \quad \mathcal{T}^+ = \mathcal{T}, \quad \mathcal{E}^+ = \mathcal{E} \cup \{e \in \Delta\mathcal{E} : \mathrm{Valid}_\Gamma(e) = 1\}. \end{aligned}$$

Here a superscript $+$ denotes the post-execution object and Δ denotes newly produced versions, edges or records; $\mathrm{Valid}_\Gamma(e) \in \{0, 1\}$ checks a record's identity, required fields and provenance, not whether the task passed. Both operations update history and budgets. The next decision can return to another branch; there is no requirement to expand the most recent leaf. Dependencies between checks belong to a typed job graph, not the parent–child version tree. No visit-count or UCT backup is assumed.

Feasible work. Let $\mathcal{J}_{\mathrm{avail}}(\widetilde{S})$ be the finite available-job catalog and $\mathcal{G}_{\mathrm{setup}}$ its setup groups. In this subsection, sums over j and g range over those sets, not training searches. For job j, $K_j \subseteq \mathcal{C} \times \mathcal{I}$ is its requested candidate–item key set, D_j includes itself and all prerequisites, $\bar{c}_j \geq 0$ caps job expense excluding shared setup, $\bar{n}_j \geq 0$ caps primitive attempts, and $g(j)$ names its setup group. A setup group g costs $s_g \geq 0$ once. A nonempty target K is one candidate's missing keys K_c, or a selected union of such sets. Empty targets require no work and are excluded from seed construction. Binary variables x_j, y_g select jobs and activated setups. The incompatible-job pairs form $\mathcal{X}_{\mathrm{conflict}}$. The

solver minimizes the enforceable cost cap:

$$\begin{aligned}
\min_{x,y} \quad & \sum_j \bar{c}_j x_j + \sum_g s_g y_g \\
\text{s.t.} \quad & \sum_{j:k\in K_j} x_j \geq 1 && (k \in K), \\
& x_j \leq x_i && (i \in D_j \setminus \{j\}), \\
& x_j + x_{j'} \leq 1 && ((j,j') \in \mathcal{X}_{\text{conflict}}), \\
& x_j \leq y_{g(j)}, \quad y_g \leq \sum_{j:g(j)=g} x_j, \\
& \sum_j \bar{c}_j x_j + \sum_g s_g y_g \leq B_{\text{plan}}, \quad \sum_j \bar{n}_j x_j \leq N, \quad x_j, y_g \in \{0,1\}.
\end{aligned} \tag{14}$$

The expense budget is $B_{\text{plan}} = B^- - \bar{o}$, where B^- is the pre-menu remainder and $\bar{o}$ is the enforceable preparation cap. Only nonnegative B_{plan} is considered; N is the current attempt allowance. Conflict and prerequisite sets are known implementation constraints, not guesses about future success. After solving, check integrality, closure, coverage, setup activation, and resource limits independently. A feasible timeout incumbent may be used but must not be called optimal. Coverage requests outcomes; it does not promise that attempts will return complete fields.

For a job set A, define $K(A) = \bigcup_{j\in A} K_j$, $D(A) = \bigcup_{j\in A} D_j$, $\overline{n}(A) = \sum_{j\in A} \bar{n}_j$, and

$$\overline{C}(A) = \sum_{j\in A} \bar{c}_j + \sum_{g\in\{g(j):j\in A\}} s_g.$$

The second sum counts each shared setup once. Let $\mathcal{X}_\Gamma(\widetilde{S})$ denote the nonempty job sets legal under the fixed dependency, conflict, and resource rules at $\widetilde{S}$. A returned completion seed must satisfy

$$A_K \in \left\{ A \in \mathcal{X}_\Gamma(\widetilde{S}) : K \subseteq K(A),\ D(A) = A,\ \overline{C}(A) \leq B_{\text{plan}},\ \overline{n}(A) \leq N \right\}. \tag{15}$$

Equation (14) specifies the minimizing objective; Equation (15) specifies the required feasibility of every returned seed, including a timeout incumbent. An infeasible target returns no seed, not an unverified optimum.

Seeds and refinement. The reference catalog retains up to four candidate targets, four pairwise target unions, and four singleton closures. Round-robin order and an independently seeded reservoir choose target candidates before new outcomes are acquired. All original feasible completion seeds remain eligible even when a single check has negative predicted marginal value. The frozen model then refines them with Equation (7). A positive extension is retained with the original seed; a nonpositive best increment ends that seed's extension. The reference defaults are four extensions per seed, 96 whole-plan queries, and 32 retained plans including non-verification alternatives. They are declared design settings, not claimed universal optima.

Table S22: **What the solver must preserve at one decision.** These are executable constraints, not learned predictions of correctness.

Candidate work	Required information	Role
One edit or check	Existing parent or item; expense and attempt caps	Feasible fallback
Completion bundle	All missing keys; dependency and setup closure	Preserve joint options
Refined bundle	Same proposal state and frozen value model	Compare continuations
Selected plan	Paid state, canonical menu, policy preference	Execute; then replan

Pseudocode conventions. Algorithm 2 uses $\mathcal{K}$ for the bounded family of selected target-key sets, $\mathcal{M}_0$ for feasible initial bundles, and $\mathcal{M}_{\text{singleton}}$ for feasible one-job closures. $W(A) = \widehat{Q}_{\bar{\phi}}(\widetilde{S}, \text{VERIFY}(A))$ is a cached prediction. q counts attempted uncached value queries, s counts extensions of the current seed, and $q_{\max}, s_{\max}$ are their limits. K_Ω caps the final menu. $\mathcal{U}(A)$ is the feasible extension family, $\widehat{\mathcal{U}}(A)$ its query-limited subset, and $\mathcal{U}^+(A)$ contains extensions with both

values available and a positive difference. A missing query result is not zero. Targets selects target sets; DraftPlans constructs bounded parent/path descriptors under ν; Retain keeps the registered action families within K_Ω. Required seeds and fallback slots must fit the cap; an inconsistent cap configuration is rejected rather than silently discarding promised support. $\mathcal{P}$ records preparation events, snapshots, the menu, and the selected probability.

Algorithm 2 Verification-Aware Tree Search (VATS)

Require: Pre-planning state S^-; frozen $\bar{\phi}, \psi, \nu$; mode; overhead, query, extension, and menu caps.
Ensure: Paid state S, menu Ω, chosen plan ω, and metered overhead/log $(o, \ell^m, \mathcal{P})$.

Solver: construct feasible completion seeds.

$\mathcal{R} \leftarrow \mathrm{Remaining}(S^-)$; $\widetilde{S} \leftarrow S^-[B \leftarrow B^- - \bar{o}]$; $q \leftarrow 0$; initialize cache.
$\mathcal{K} \leftarrow \mathrm{Targets}(\{K_c(\widetilde{S}) : c \in \mathcal{C},\ K_c(\widetilde{S}) \neq \varnothing\}, \mathcal{R})$.
$\mathcal{M}_0 \leftarrow \{A_K : K \in \mathcal{K},\ A_K \text{ satisfies (15)}\} \cup \mathcal{M}_{\mathrm{singleton}}$.

Scorer: estimate continuation values and refine bundles.

$\mathcal{M} \leftarrow \mathcal{M}_0$; $W(A) = \widehat{Q}_{\bar{\phi}}(\widetilde{S}, \mathrm{VERIFY}(A))$ when queried.
for $A_0 \in \mathcal{M}_0$ in the registered order **do**
  $A \leftarrow A_0$; $s \leftarrow 0$; query/cache $W(A)$ if caps permit; meter an attempted cache miss in q, o.
  while $s < s_{\max}$, $q < q_{\max}$, and metered overhead permits **do**
    $\mathcal{U}(A) \leftarrow \{A \cup D_j : A \cup D_j \in \mathcal{X}_\Gamma(\widetilde{S})\}$ ▷ solver feasibility filter
    Query a capped $\widehat{\mathcal{U}}(A) \subseteq \mathcal{U}(A)$; meter every attempted cache miss in q, o.
    $\mathcal{U}^+(A) \leftarrow \{U \in \widehat{\mathcal{U}}(A) : W(U), W(A) \text{ available},\ W(U) > W(A)\}$.
    if $\mathcal{U}^+(A) = \varnothing$ **then**
      break
    end if
    $A \leftarrow \arg\max_{U \in \mathcal{U}^+(A)} [W(U) - W(A)]$; $\mathcal{M} \leftarrow \mathcal{M} \cup \{A\}$; $s \leftarrow s + 1$.
  end while
end for

Solver: assemble and validate the final work menu.

$\mathcal{D}_{\mathrm{draft}} \leftarrow \mathrm{DraftPlans}(\mathcal{T}, \mathcal{R}, \widetilde{S}, \nu)$ ▷ parent nodes and bounded path shapes
$\Omega \leftarrow \mathrm{Retain}_{K_\Omega}(\{\mathrm{ACCEPT}\} \cup \mathcal{D}_{\mathrm{draft}} \cup \{\mathrm{VERIFY}(A) : A \in \mathcal{M}\})$.
Validate caps and canonicalize duplicates; retain feasible seeds and promised action families.

Policy: select the next action, branch, and work scope.

Settle preparation, including the reserved final policy call; $S \leftarrow \mathrm{Debit}(S^-, o)$; append preparation history. ▷ single debit
$p_\psi(\omega) \leftarrow \pi_\psi(\omega \mid S, \mathcal{R}, \Omega)$ using (8).
$\omega \sim p_\psi$ in collection; $\omega \leftarrow \arg\max_{u \in \Omega} z_\psi(S, \mathcal{R}, u)$ in deployment.
Finalize metered time ℓ^m; record S, Ω, ω, actual choice probability, mode, snapshots, and preparation events in $\mathcal{P}$.
return $(S, \Omega, \omega, o, \ell^m, \mathcal{P})$ → Algorithm 1

Policy support. Every attempted value query, MILP call, and final policy evaluation consumes metered overhead. The final policy call uses a contracted expense known before its logits are evaluated, included in the preparation debit; its measured elapsed time is finalized before return. This avoids evaluating at one budget and later relabeling the likelihood as conditioned on another. Cache reuse requires an identical state, model snapshot, and plan-semantics version. Failed queries leave a missing value feature, not an invented zero. ACCEPT, a feasible single edit, a feasible singleton check, and an affordable completion batch retain reserved menu slots when available. Higher-order plans can still be omitted; the policy cannot recover them later.

Algorithm 2 settles preparation once and returns the paid state, canonical menu, selected plan, and its actual behavior probability. Algorithm 1 does not debit that preparation again. During training, recompute policy likelihoods on the same logged menu; changing the menu changes the support. Preparation records are appended to observed history without importing task outcomes. A deterministic MILP output is not a sampled policy action and has no policy-gradient likelihood. The reference backend is SciPy MILP with HiGHS; a different modeling interface is not silently substituted in the experimental claims.

F.4 Verification and Execution

Direct execution. Requested checks run the frozen version under Γ or read an already admitted identical record. A valid record is keyed to its candidate, item, fields, executor/checker version, randomness semantics, and relevant environment state. A record with a different key does not satisfy the requested check; fresh evaluation is required.

A timeout or partial result leaves missing fields missing. A hard failure is a valid observation; it can disqualify the current version while still informing a repairable descendant. Predictions choose which verification work to buy, not which results to admit.

Execution safeguards. Draft and verification branches execute in isolated snapshots. Changes to shared files, irreversible messages, financial actions, or external services cannot be rolled back merely because a branch was speculative. Such effects require a separately enforced execution contract. A copied state must include the relevant tool and environment state, not only a prompt string.

A work cap bounds attempts and expense, not success. Every failed edit, failed check, setup, cancellation, copy, and preparation call has an explicit account. Within a synchronous reference block, feedback changes the next decision only after the block ends. An asynchronous adapter must log launches, in-flight commitments, cancellation fees, stale results, and completion times. The fastest observed configuration cannot be declared the cheapest when it also increases total worker time. Matched asynchronous comparisons cross the allocation rule with the execution scheduler, instead of attributing all speedup to the policy.

F.5 Two-Stage Training

Learned roles. The inherited open-model reference uses frozen text representations, a rank-16 drafter adapter, and separate trainable value and preference branches. Each branch processes parent–child context, typed candidate–job relations, and plan features through message passing and set attention; the proposed width is 256 with four heads. These are reference settings, not authenticated per-study model snapshots (Hu et al., 2022; Lee et al., 2019). Numeric budget, primitive allowance, current complete score, missing masks, known outcomes, and proposed work enter explicitly. Draft features cannot contain embeddings of ungenerated children.

The *scorer*, *value model*, and *critic* all denote the single action-value model $\widehat{Q}_\phi(S, \omega)$. Its input is a search state and proposed work plan; its scalar output predicts future verified improvement minus resource costs. It is not a model of an agent's current task score. The policy's scalar logit $z_\psi(S, \mathcal{R}, \omega)$ instead defines a choice distribution. These are separate value and preference branches, not two value models.

During collection, $\bar{\phi}_k = \phi_k$ is an immutable copy of the current scorer parameters. After collection, ϕ is fitted to observed returns and becomes ϕ_{k+1}; the next collection refreshes $\bar{\phi}_{k+1} = \phi_{k+1}$. The bar marks a collection snapshot, not an independently trained second critic or a bootstrapped target network. No separate learned state-value $V(S)$, twin-Q minimum, temporal-difference target, or generalized advantage estimator is specified by this reference. The group advantage in Equation (13) uses other searches' realized returns, not a learned value baseline. The score aggregation J_Γ prescribed by the verification contract Γ produces task scores and is neither the scorer nor the policy.

Frozen shared text inputs do not imply shared trainable parameters; actor gradients do not train ϕ. The draft-menu settings ν have no learning rule in the supplied formulation and remain fixed here. Updating θ, ψ, ϕ between epochs changes future decisions, not immutable artifacts or acquired evidence. The value/preference architecture therefore contains one return model and one choice model, not two critics.

Return accounting. Supervised examples initialize legal edit generation and plan selection. The exact supervised losses are stated below; they do not label unexecuted alternatives. For reinforcement learning, use two transitions per visited decision t: preparation changes S_t^- to S_t, then the selected work changes S_t to S_{t+1}^-. $\mathcal{B}_t$ contains sampled drafter-token records (context, support, token and actual probability); verification returns $\mathcal{B}_t = \varnothing$. Together with the sampled plan record in $\mathcal{P}_t$, these logs supply actor likelihoods. Let o_t, ℓ_t^m be metered preparation expense and elapsed time; let κ_t, ℓ_t^w be work expense and elapsed time. The superscripts m, w label menu preparation and work, not

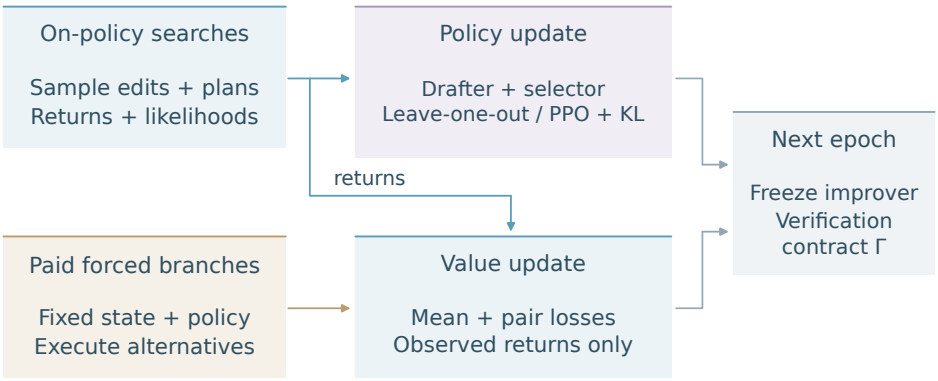


Figure S13: **Two data paths train different quantities.** Ordinary complete searches provide actor likelihoods and return labels. Paid forced continuations provide critic labels only. The next epoch freezes the updated improver; the verification contract Γ stays unchanged.

powers. n_t is the number of charged primitive attempts. Every actual charge belongs to exactly one transition:

$$\begin{aligned} &B_t = B_t^- - o_t, \qquad B_{t+1}^- = B_t - \kappa_t, \qquad N_{t+1} = N_t - n_t, \\ &r_t^m = -\lambda o_t - \mu \ell_t^m, \qquad r_t^w = F(S_{t+1}^-) - F(S_t) - \lambda \kappa_t - \mu \ell_t^w. \end{aligned} \tag{16}$$

Preparation changes resources and bookkeeping but acquires no outcomes, so $F(S_t) = F(S_t^-)$. Define T as the number of visited paid choices, including a sampled ACCEPT when present. For ACCEPT, set $S_{t+1}^- = S_t$ and $\kappa_t = \ell_t^w = n_t = r_t^w = 0$; this is bookkeeping, not extra work. If no further preparation fits, the last pre-menu state is terminal. Thus $S_{\text{end}} = S_T^-$, also for $T = 0$.

At a saved post-menu choice, current preparation is sunk. Its observed return is

$$\begin{aligned} G_t &= r_t^w + \sum_{u=t+1}^{T-1} (r_u^m + r_u^w), \\ C_{t:\text{end}} &= \kappa_t + \sum_{u=t+1}^{T-1} (o_u + \kappa_u), \qquad L_{t:\text{end}} = \ell_t^w + \sum_{u=t+1}^{T-1} (\ell_u^m + \ell_u^w). \end{aligned} \tag{17}$$

This telescopes to Equation (11), including any later terminal preparation fee. In contrast, whole-search totals are $C_g = \sum_{t=0}^{T_g-1} (o_t^g + \kappa_t^g)$ and $L_g = \sum_{t=0}^{T_g-1} (\ell_t^{m,g} + \ell_t^{w,g})$. Hence $R_g = \sum_{t=0}^{T_g-1} (r_t^{m,g} + r_t^{w,g})$. Here T_g is the decision count of search g, G is the number of searches, and G_t is a return; they are different quantities. Elapsed-time intervals follow one search clock, not a sum of concurrent worker durations. All formulas in this reference use discount one.

Supervised objectives. Let $\mathcal{D}_{\text{edit}}^{\text{sup}}$ contain executed legal edits e and their generation contexts H (including the actual parent), and let $\mathcal{D}_{\text{plan}}^{\text{sup}}$ contain demonstrated legal plan choices with their paid states and menus. Warm-up minimizes

$$\begin{aligned} \mathcal{L}_{\text{SFT}}(\theta, \psi) = &- \mathbb{E}_{(H,e) \sim \mathcal{D}_{\text{edit}}^{\text{sup}}} \log \pi_\theta(e \mid H) \\ &- \mathbb{E}_{(S,\mathcal{R},\Omega,\omega) \sim \mathcal{D}_{\text{plan}}^{\text{sup}}} \log \pi_\psi(\omega \mid S, \mathcal{R}, \Omega). \end{aligned} \tag{18}$$

The edit likelihood is the product of its autoregressive token probabilities, including a sampled end token. Its logarithm is their sum, not an additional second likelihood. Each expectation denotes an empirical mean over the specified dataset. Equation (11) separately initializes the scorer from executed returns. These losses formalize the stated warm-up roles; they do not create missing demonstration or rollout records.

Actor update. GVO groups independent searches conditional on the same root, contract, budgets, workers, and collection snapshot. A leaf inside a shared tree is not a group member. Set $\vartheta = (\theta, \psi)$;

this parameter vector excludes ϕ and ν. Index genuinely sampled controlled choices in search g by $j = 1, \ldots, J_g$, where J_g is their count, $y_{g,j}$ is the sampled token or plan, $H_{g,j}$ is its logged conditioning context, and $\mathcal{Y}_{g,j}$ is its sampling support. In this paragraph j indexes model choices, not verification jobs. Write their typed probabilities as

$$p_\vartheta(y_{g,j} \mid H_{g,j}, \mathcal{Y}_{g,j}) = \begin{cases} \pi_\psi(\omega_t \mid S_t, \mathcal{R}_t, \Omega_t), & \text{sampled plan,} \\ \pi_\theta(y_{g,j} \mid H_{g,j}, \mathcal{Y}_{g,j}), & \text{sampled edit token.} \end{cases} \tag{19}$$

For a plan, $\mathcal{Y}_{g,j} = \Omega_t$ and $y_{g,j} = \omega_t$. For a token, the context includes the real parent, observed history, and generated prefix, and the support is the allowed token set. Use the actual post-temperature, post-mask probabilities and, for plans, the exploration mixture in Equation (8). Record invalid sampled edits as well as successful ones; neither failed work nor its likelihood disappears. A deterministic choice is not assigned a fictitious sampling probability.

Keeping the collection probabilities and advantages fixed, define

$$\begin{aligned} \rho_{g,j}(\vartheta) &= \frac{p_\vartheta(y_{g,j} \mid H_{g,j}, \mathcal{Y}_{g,j})}{p_{\vartheta_k}(y_{g,j} \mid H_{g,j}, \mathcal{Y}_{g,j})}, \\ \mathcal{L}_{\mathrm{GVO}}(\vartheta) &= -\frac{1}{G} \sum_{g=1}^{G} \sum_{j=1}^{J_g} \min\{\rho_{g,j}(\vartheta) A_g, \mathrm{clip}(\rho_{g,j}(\vartheta), 1-\epsilon_c, 1+\epsilon_c) A_g\}. \end{aligned} \tag{20}$$

Here $\epsilon_c \in (0, 1)$ is the clipping radius, distinct from the exploration probability ϵ, and $\mathrm{clip}(x, l, u) = \min(\max(x, l), u)$. A_g in Equation (13) is a detached whole-search advantage, not a temporal-difference residual or a critic prediction. Each sampled choice uses it once. The denominator is the stored behavior probability, not a recomputed probability after changing the menu. Keep the recorded token support, plan menu, frozen scorer, and generator rules fixed during this update; changed supports require fresh collection. The context may change across decisions, but the same recorded context is used in each numerator–denominator pair.

Only sampled plan choices and edit tokens enter the actor sum. Solver outputs, environment results, deterministic fallbacks, and entire forced continuations are excluded. Keeping the sum over each trajectory avoids reweighting searches by J_g; a search with no sampled choices contributes zero. This is a PPO-style surrogate with a leave-one-out baseline, not a new gradient theorem or monotonic-improvement guarantee (Schulman et al., 2017; Ahmadian et al., 2024).

KL regularization. To make the optional penalty explicit, let $p^{\mathrm{ref}}_{g,j}$ be a registered frozen reference distribution on the same support $\mathcal{Y}_{g,j}$. One current-to-reference convention is

$$\begin{aligned} \mathcal{L}_{\mathrm{KL}}(\vartheta) &= \frac{1}{G} \sum_{g=1}^{G} \sum_{j=1}^{J_g} D_{\mathrm{KL}}\big(p_\vartheta(\cdot \mid H_{g,j}, \mathcal{Y}_{g,j}) \,\|\, p^{\mathrm{ref}}_{g,j}\big), \\ D_{\mathrm{KL}}(p\|q) &= \sum_{y \in \mathcal{Y}_{g,j}} p(y) \log \frac{p(y)}{q(y)}. \end{aligned} \tag{21}$$

The reference has positive probability wherever the current distribution is positive; $0 \log(0/q)$ is zero. Its checkpoint, direction, coefficient $\beta_{\mathrm{KL}} \geq 0$, and any estimator used instead of the full sum must be registered. $\beta_{\mathrm{KL}} = 0$ disables this term. The input manuscript does not identify these settings for the reported deployment; Equation (21) defines the optional convention, not an inferred experimental configuration. A frozen reference policy is not a second value model. The environment return R_g excludes this optimization penalty, which is applied only once in the actor loss.

Same-state continuations. On a prespecified development subset, restore identical paid states S. Let $\mathcal{W}_S$ be the finite set of legal first plans to compare, drawn from that saved menu. Each arm $a \in \mathcal{W}_S$ uses a prospectively fixed number $n_a \geq 1$ of independently seeded suffix rollouts under the same frozen Θ_k. Define $G_{a,s}$ as the realized suffix return of repetition $s = 1, \ldots, n_a$, from forced plan a to termination, excluding the already paid initial menu. Every later menu and failed action is charged. Unsuccessful but fully logged rollouts stay in the average; absent logs cannot be replaced by invented zeros.

$$\overline{G}_a = \frac{1}{n_a} \sum_{s=1}^{n_a} G_{a,s}, \qquad \widehat{\delta}_{ab} = \overline{G}_a - \overline{G}_b, \qquad \mathrm{Var}(\widehat{\delta}_{ab} \mid S) = \frac{\sigma_a^2}{n_a} + \frac{\sigma_b^2}{n_b}. \tag{22}$$

Here $\sigma_a^2 = \mathrm{Var}(G_{a,s} \mid S, a, \Theta_k)$; the last equality assumes independent repetitions and independent arms conditional on the state and snapshot. For coupled arms, subtract $2\,\mathrm{Cov}(\overline{G}_a, \overline{G}_b \mid S)$ instead. Matching the initial state alone does not guarantee smaller variance. Copies and all branch rollouts are charged to acquisition, without subtracting clone acquisition a second time from the suffix's counterfactual decision budget.

The empirical dataset $\mathcal{D}_Q$ contains fully logged tuples (S, a, G) from ordinary plan starts and paid forced rollouts with a matched continuation snapshot. $\mathcal{D}_{\mathrm{pair}}$ contains $(S, a, b, \overline{G}_a, \overline{G}_b)$ from the same arm records. Labels collected under different continuation policies must not silently be treated as observations of one Q^π; policy IDs are retained. At a fixed state write $e_a = \widehat{Q}_\phi(S, a) - \overline{G}_a$. The pair penalty is $(e_a - e_b)^2$: it reweights differences but cannot identify a common offset. Thus the single scorer minimizes

$$\mathcal{L}_{\mathrm{value}}(\phi) = \mathcal{L}_Q(\phi) + \beta_{\mathrm{pair}}\mathcal{L}_{\mathrm{pair}}(\phi), \qquad \beta_{\mathrm{pair}} \geq 0.$$

Both targets are observed Monte Carlo returns, with no bootstrapped Q or learned V target. The loss comparison separates same-data loss weighting from equal-acquisition-cost sampling; neither comparison implies equal fitting cost.

Algorithm 3 Group Verification Optimization (GVO)

Require: $\Theta_k = (\theta_k, \psi_k, \phi_k, \nu)$; complete root r, fixed Γ, budgets B_0, N_0, workers w, group size $G \geq 2$; registered loss/update settings.
Ensure: Full updated snapshot Θ_{k+1} and acquisition/training ledger; forced rollouts remain scorer-only.

Set $\bar{\phi} \leftarrow \phi_k$; freeze $\theta_k, \psi_k, \bar{\phi}, \nu$ during all collection below.
for $g = 1, \ldots, G$ independently **do**
    $(c_{k+1}^{(g)}, \mathcal{D}_g) \leftarrow \mathrm{Search}_{\Theta_k}(r, \mathrm{collection})$. → Algorithm 1
end for
Compute R_g, A_g by Equation (13), using each search's terminal state and full expense/time.
$\mathcal{D}_{\mathrm{actor}} \leftarrow \bigcup_g \mathcal{D}_g^{\mathrm{sampled}}$; $\mathcal{D}_Q \leftarrow \bigcup_g \mathcal{D}_g^{\text{plan starts}}$ with suffix labels from (11); $\mathcal{D}_{\mathrm{pair}} \leftarrow \varnothing$.
$\mathcal{Z} \leftarrow \mathrm{SelectStates}(\{\mathcal{D}_g\})$ using the prespecified development rule.
for saved paid state $S \in \mathcal{Z}$, legal arm $a \in \mathcal{W}_S$, repetition $s = 1, \ldots, n_a$ **do**
    Clone S, including its tree, records, budgets, root reference and environment state; log clone acquisition separately.
    Set the suffix log empty and local clock zero; execute $\omega_0 = a$ at the restored post-menu entry, then continue under Θ_k. → Algorithm 1
    Compute $G_{a,s}$ by Equation (11); add fully logged plan starts to $\mathcal{D}_Q$, never to $\mathcal{D}_{\mathrm{actor}}$.
    Log snapshot/state/arm IDs, repetitions, missing records, and all acquisition charges.
end for
Form arm means and matched tuples in $\mathcal{D}_{\mathrm{pair}}$ using Equation (22).
Initialize live $\phi \leftarrow \phi_k$, $\vartheta \leftarrow (\theta_k, \psi_k)$; keep collection probabilities and A_g detached.
Update ϕ on $\mathcal{L}_{\mathrm{value}}$ and ϑ on $\mathcal{L}_{\mathrm{GVO}} + \beta_{\mathrm{KL}}\mathcal{L}_{\mathrm{KL}}$ as in Equation (23).
Refresh $\bar{\phi}_{k+1} \leftarrow \phi_{k+1}$ for the next collection; keep Γ and ν unchanged.
return $\Theta_{k+1} = (\theta_{k+1}, \psi_{k+1}, \phi_{k+1}, \nu)$ and the ledger; checkpoint choice uses development data only.
Next collection: run new searches with the refreshed snapshot. → Algorithm 1

Update schedule. An ordinary Algorithm 1 call initializes the root. A matched continuation instead restores the complete paid state, sets its suffix log empty and clock to zero, fixes $\omega_0 = a$, and enters after the solver. It does not reset the tree, reacquire an already paid menu, or resample the forced first choice. Later decisions use normal paid preparation. Entire forced continuations are excluded from actor updates; they supply critic targets only. The return goes back to the external training caller.

For one update step with learning rates $\eta_\phi, \eta_\pi > 0$, the parameter assignments are

$$\begin{aligned}
\phi_{k+1} &= \phi_k - \eta_\phi \, \nabla_\phi \mathcal{L}_{\mathrm{value}}(\phi)\big|_{\phi=\phi_k}, \\
\vartheta_{k+1} &= \vartheta_k - \eta_\pi \, \nabla_\vartheta [\mathcal{L}_{\mathrm{GVO}}(\vartheta) + \beta_{\mathrm{KL}}\mathcal{L}_{\mathrm{KL}}(\vartheta)]\big|_{\vartheta=\vartheta_k}, \\
\Theta_{k+1} &= (\theta_{k+1}, \psi_{k+1}, \phi_{k+1}, \nu), \qquad \bar{\phi}_{k+1} = \phi_{k+1}.
\end{aligned} \tag{23}$$

∇ denotes differentiation with respect to the subscripted parameters; all returns, advantages, logged contexts, old probabilities, and reference distributions are constants in these derivatives. Registered minibatches may repeat update steps over the frozen collection. The fitted ϕ_{k+1} uses returns under the collection procedure π_k. Updating the actors changes the continuation law to π_{k+1}, so the refreshed scorer is an approximation there, not an exact new-policy value obtained merely by copying parameters. Fresh collection supplies calibration for the next round. Parameters refresh only for subsequent collection. Deployment chooses the largest preference rather than sampling, so its calibration and quality need direct evaluation. Resetting the task agent while holding allowed histories and budgets fixed tests whether the updated improver transfers. Task-family and model-backbone transfer are separate settings. No reporting audit is used to select a training epoch, and no indefinite recursive acceleration follows from a finite number of rounds.

G Theoretical Analysis

This appendix proves the conditional claims underlying Sections 3.2–3.3. Read Appendix G.4.2 for the central three-check result; the remaining sections establish accounting, planning, and learning properties. These are finite analytical statements, not new agent benchmarks. Each proposition states the assumptions needed for its conclusion.

Table S23: **What each formal result establishes.** The assumptions belong to the reference model; these results are not neural benchmark measurements.

Reader question	Result to read	Essential condition
Can predictions corrupt a completed result?	Outcome consistency (Proposition 1)	Sound records; fixed contract
Does grouping enlarge every policy class?	Singleton simulation (Proposition 4)	Free serial decision boundaries
When does commitment save expense?	Feedback threshold (Proposition 5)	Identical output and charged blocks
What do success summaries omit?	Scope reversal and matched control (Propositions 7, 8)	Equal marginals/all-pass rate; distinct joint laws
How can an approximate menu lose value?	Menu bound (Proposition 9)	Fixed continuation; bounded value error

G.1 Validity and Termination

G.1.1 Outcome Consistency

Proposition 1. *Fix immutable candidates and a deterministic finite contract. Assume direct records equal contracted executions and inherited records have sound equalities to valid earlier sources. Then every completely covered candidate has the same aggregate and hard-constraint outcomes as full direct execution under that contract.*

Proof. Order records by their well-founded provenance relation. Direct records are correct by assumption. Each inherited record equals an earlier correct source under a checked equality, hence is correct by induction. The complete outcome vector and unchanged aggregation therefore match a full execution. No additive-score assumption is needed. □

The result guarantees consistency of completed finite outcomes under the stated execution assumptions. Stochastic services need justified replay semantics or fresh evaluation. Prediction errors can harm allocation even when admitted records remain correct; population generalization is a separate question.

G.1.2 Reward Conservation

Proposition 2. *Use the preparation/work clock in Appendix F.5, with T visited choices and disjoint metered charges. Then*

$$\sum_{t=0}^{T-1}(r_t^m + r_t^w) = F(S_T^-) - F(S_0^-) - \lambda \sum_{t=0}^{T-1}(o_t + \kappa_t) - \mu \sum_{t=0}^{T-1}(\ell_t^m + \ell_t^w).$$

Proof. Preparation leaves F unchanged: $F(S_t) = F(S_t^-)$. Substituting Equation (16) telescopes the progress terms and sums disjoint resource charges. The zero-cost terminal transition preserves the identity; empty sums cover $T = 0$. □

Regrouping the same realization into different work blocks preserves the undiscounted objective, not necessarily the realization produced by another policy. For $\mu = 0$, a serial and parallel schedule with identical results and billed work has identical reward. A block-count discount changes the objective and is not introduced implicitly. Already paid menu cost belongs to the episode but not a selected plan's remaining return.

G.1.3 Finite Termination

Admit overhead and each plan only when their enforceable reservations fit the remaining cap. Settle actual cost no greater than the reservation before releasing unused resources. Induction prevents overspending; a predicted mean expense cannot replace the cap. Every legal nonempty plan consumes at least one registered primitive attempt, including invalid edits and failed checks. Empty verification plans are excluded; empty targets need no work. No-feasible-work terminates, so zero-cost diagnostics cannot loop indefinitely. These statements require the backend to enforce its caps; violations must be reported.

G.2 Completion and Equivalence

G.2.1 Completion Complementarity

Proposition 3. *Immediate selectable gain need not be additive or submodular in obtained checks.*

Proof. Let a positively improving candidate require two remaining records a, b. For a subset $A \subseteq \{a, b\}$, let $v(A)$ be the newly selectable verified gain after obtaining exactly A, with other evidence fixed. Then $v(\varnothing) = v(\{a\}) = v(\{b\}) = 0$ while $v(\{a, b\}) = g > 0$. Additivity would imply $g = 0$; submodularity would require the marginal of b at the empty set to be at least its positive marginal after a. □

This rules out identifying individual immediate gains with batch utility, not all singleton policies or additive proposal heuristics. A correct long-horizon policy can take a anticipating b. Shared setup creates another interaction; multiple complete candidates can substitute because only the best is deployed. All strong controls receive the best complete score to avoid a trivial information disadvantage.

G.2.2 Full-State Simulation

Proposition 4. *Under serial execution, zero replanning/dispatch cost, no shared setup discounts, identical primitive edits/checks, and a finite primitive allowance, every history-dependent bounded-plan policy can be simulated by a history-dependent singleton policy with identical output and cost distribution.*

Proof. At each plan boundary, sample the same descriptor and store it. Execute its primitives one by one using the same random streams and actual-edit conditioning, deliberately ignoring intermediate feedback until the original block would return. Artifacts, observations and primitive costs match inductively. Boundary cost and discounts are absent, so final output and cost match. The singleton class can also react earlier, hence its optimal value is at least that of the bounded class under these assumptions. □

This is a policy-class comparison, not a claim that a trained singleton learner finds its optimum. It excludes real parallel latency, nonzero model calls, setup sharing and function-approximation difficulty. Grouping $n \geq 1$ identical serial jobs, with dispatch price $d \geq 0$ per launched block, can save $(n-1)d$ of dispatch expense before batch-construction cost, but may buy unnecessary jobs after a revealing first failure. Regime experiments test both effects.

G.2.3 Repairable Versions

An executable two-stage adapter supplies a minimal example. An old tool emits an unstructured string consumed by an old parser. Edit e_1 changes the output to a structured record; the unchanged parser fails the contract. Edit e_2 updates the parser to consume that record. It can then satisfy the contract. Thus a version $c \notin \mathcal{C}_{\text{full}}(S)$ or a failed hard requirement does not logically imply every editable child must fail. This example uses actual transformations and direct executions; it is not a simulated successful observation fed to the drafter. Nor does it show such two-step improvements are common. It explains the distinction between editing permission and adoption permission.

G.3 Feedback Costs

G.3.1 Delayed Feedback

Proposition 5. *Consider two prescribed serial checks with costs $c_1, c_2 \geq 0$. The first reveals a hard failure with probability $p \in [0, 1]$; after failure, the second cannot change the final eligible output. Otherwise both checks are required. Dispatch costs $d \geq 0$ per launched block. A blind two-check batch has extra construction cost $a \geq 0$. Assume both policies can afford their maximal path, have identical results and output, and end this comparison without reinvesting savings. A sequential policy stops checking after first-check failure. Then*

$$C_{\text{batch}} = d + a + c_1 + c_2, \qquad \mathbb{E}C_{\text{sequential}} = d + c_1 + (1 - p)(d + c_2),$$
$$C_{\text{batch}} - \mathbb{E}C_{\text{sequential}} = a + pc_2 - (1 - p)d. \tag{24}$$

In the cost-only objective with $\lambda > 0$, the batch has higher return exactly when $(1 - p)d > a + pc_2$.

Proof. Both policies pay for the first dispatch and check. The sequential policy pays the second dispatch and check exactly on the probability-$1 - p$ branch. The batch always pays the second check and its extra construction cost. Subtracting gives the result; identical output makes terminal utility cancel. □

This crossover explains a limited mechanism, not a universal scheduler. With $p = 1/2$, $c_2 = 1$, and $a = 0$, batching is cheaper exactly when $d > 1$. These are analytical values. Allowing a batch to cancel work on early feedback, changing shared setup, parallelizing elapsed time, repairing a failed candidate, or reinvesting unused budget changes the model. Such settings require new accounting rather than reusing the threshold. First-check failure excludes only the tested version from this output comparison; it does not rule out useful descendants.

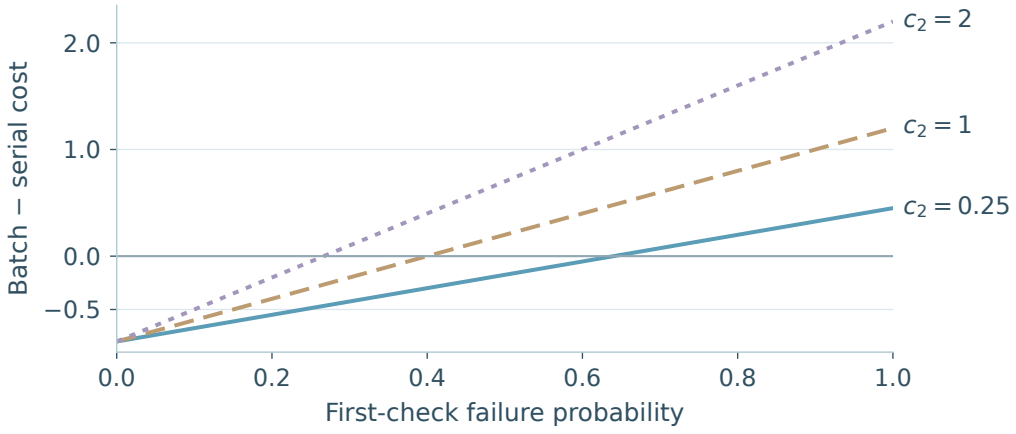


Figure S14: **Early feedback changes the cost of committing work.** Exact sensitivity of Equation (24), with dispatch 1 and planning 0.2. These curves do not fit the batch-versus-serial study or assert that batching always wins.

G.3.2 Menu Overhead

Additional proposals can remove good execution options by consuming their budget. To isolate this effect, consider two menu builders on the same pre-overhead state with deterministic settled costs $o_1 \geq o_0$. They reveal no additional task evidence and differ only in their current proposal computation and offered plans; future generator and execution rules are fixed. Write S_0, S_1 for the corresponding post-overhead states. The cost-only objective uses $\mu = 0$. An ideal continuation policy can ignore extra budget, so its feasible policy classes are nested. This assumption is about an ideal class, not a learned network's behavior.

Let Ω_0 be the cheap menu, Ω_0' its still-affordable subset at S_1, and $\Omega_1 \supseteq \Omega_0'$ the richer menu. Each includes ACCEPT. For this comparison, $Q^*(S, \omega)$ is the supremum of expected net returns over the stipulated feasible continuation-policy class after first work ω. It is an analytical oracle quantity, not another trained network. Let V_i be the optimal value before paying builder i's overhead, for $i \in \{0, 1\}$. Define

$$M_0^0 = \max_{\Omega_0} Q^*(S_0, \cdot), \quad M_0^1 = \max_{\Omega_0'} Q^*(S_1, \cdot), \quad M_1^1 = \max_{\Omega_1} Q^*(S_1, \cdot).$$

The best pre-overhead values for the two builders satisfy the accounting identity

$$V_1 - V_0 = \underbrace{M_1^1 - M_0^1}_{\text{new offered-work value}} - \lambda(o_1 - o_0) - \underbrace{(M_0^0 - M_0^1)}_{\text{lost execution opportunity}}. \tag{25}$$

In M_j^i, superscript i identifies the paid state and subscript j the menu family. The scalars V_0, V_1 are used only in this comparison; they are not learned state-value models. Both bracketed differences are nonnegative under the stated inclusion and nested-policy assumptions. The first follows from menu inclusion at S_1; the last follows by simulating any lower-budget continuation at the higher budget. Substituting $V_i = -\lambda o_i + M_i^i$ proves the identity. A richer menu is worthwhile only if its new options pay for both its direct cost and its lost execution opportunity. This is not an optimal menu-generation algorithm, nor a bound for approximate Q or an actor that changes with budget. The menu-size study in Table S15 reports pre-overhead and final gains, but does not identify them as matched-policy values in this proposition. The default controller still uses the bounded menu of Appendix F.3; no oracle is added.

G.4 Evidence and Scope

G.4.1 Incomplete Summaries

The two-check model extends without assuming independent failures. Fix an ordered list of $n \geq 1$ serial checks, nonnegative costs c_i, common dispatch cost d, and extra blind-batch construction cost a. A hard failure at any prefix makes later checks irrelevant to the final eligible output in this comparison. Let s_{i-1} be the probability that the first $i-1$ checks contain no such failure, with $s_0 = 1$. Assume identical potential check outcomes, no early cancellation inside the blind batch, maximal-path affordability, and no repair or reinvestment after termination.

Proposition 6. *Under these conditions, for arbitrary dependence between check outcomes,*

$$C_{\text{batch}} = d + a + \sum_{i=1}^{n} c_i, \tag{26}$$

$$\mathbb{E}C_{\text{sequential}} = \sum_{i=1}^{n} s_{i-1}(d + c_i), \tag{27}$$

$$C_{\text{batch}} - \mathbb{E}C_{\text{sequential}} = a + \sum_{i=2}^{n} \big[(1 - s_{i-1})c_i - s_{i-1}d\big]. \tag{28}$$

Proof. The sequential policy launches check i exactly when the preceding checks survive. Multiply its launch and check costs by this indicator and take expectations. The blind batch always executes the full list and pays one dispatch. Subtraction cancels $d + c_1$ because $s_0 = 1$. Dependence changes the survival probabilities, not the identity. □

For $n = 2$, $s_1 = 1 - p$ recovers Equation (24). Estimating survival by a product of marginal pass rates would add an independence assumption that the result does not require. These probabilities are diagnostic inputs, not secretly available to the deployed controller. The finite test enumerates joint outcomes; Table S16 reports costs but not the survival probabilities and coupled regime settings needed for a numerical test of this identity. This calculation predicts a cost crossover for a fixed output, not that longer speculative drafts always help.

G.4.2 Ordering-Robust Reversal

Finite setting. A candidate is adoptable exactly when three binary hard checks all pass; otherwise the incumbent is returned. Each check costs one and each dispatched nonempty block costs $d = 3/2$. A launched block completes all its checks before feedback can affect the next choice. Policies must stop at an observed failure and finish all checks on a surviving path. They may choose any unchecked subset at each boundary. Both contexts have a budget of at least $15/2$ and at least three primitive attempts. There is no within-block cancellation, repair, or reinvestment after termination. These are *output-equivalent verification policies*, not unrestricted agent-search policies allowed to stop without completing a promising version.

Prior legal evidence identifies the conditional law, not the realized future outcome. Let $X \in \{0,1\}^3$ be the potential outcome vector, with 1 denoting pass. In context A, the four atoms $111, 100, 010, 001$ each have probability $1/4$. In context B, atom 111 has probability $1/4$, each of $110, 101, 011$ has probability $1/8$, and 000 has probability $3/8$. All other probabilities are zero. Both laws are exchangeable: every permutation of the checks leaves the law unchanged. Both have $\Pr(X_i = 1) = 1/2$ and $\Pr(X = 111) = 1/4$. The same candidate description, missing-item mask, resources, marginal predictions, and final-success prediction therefore do not identify the context. Observing its joint evidence does not reveal any future check result.

Proposition 7 (Ordering-robust scope reversal)**.** *In this setting, the unique optimal first-block size is two in* A *and one in* B*, even when subsequent subset choices are adaptive and ordering is unrestricted. Optimal costs are* $33/8$ *and* $17/4$*, respectively. A selector whose first decision receives identical compressed input in the two contexts has worst-context expected excess cost at least* $3/40$*, even if its subsequent decisions are optimal. This bound is tight. In the cost-only objective it gives net-return loss at least* $3\lambda/40$ *for* $\lambda > 0$*.*

Proof. Before a failure, every observed bit equals one. Exchangeability makes all remaining subsets of the same size equivalent. Thus every deterministic surviving-path policy is represented by one of the four ordered block-size compositions $1 \to 1 \to 1$, $1 \to 2$, $2 \to 1$, or 3. Randomization is a mixture over such boundary choices and cannot improve a linear expected cost.

Let $q = \Pr(X_i = X_j = 1)$ for distinct items. Its values are $1/4$ in A and $3/8$ in B. Costs for the four compositions are, respectively,

$$\frac{5}{2}\left(\frac{3}{2} + q\right), \qquad \frac{17}{4}, \qquad \frac{7}{2} + \frac{5}{2}q, \qquad \frac{9}{2}.$$

For example, the two-then-one plan always pays $2 + 3/2$ for its first block, and pays $1 + 3/2$ again only with probability q. Substitution gives the four costs in Table S24. Conditioning on a first block of size one, the optimal continuation batches the two remaining checks in both contexts. The best achievable regrets conditional on first-block sizes $1, 2, 3$ are therefore

$$\mathsf{A}: \ (1/8, 0, 3/8), \qquad \mathsf{B}: \ (0, 3/16, 1/4).$$

These already grant optimal subsequent information use. For any first-block mixture, averaging its regret with context weights $(3/5, 2/5)$ is at least $3/40$: the three pure-size averages are $3/40$, $3/40$, and $13/40$. Its worst-context regret is no smaller. Choosing first size one with probability $3/5$ and size two with probability $2/5$ attains $3/40$ in both contexts. Identical realized adoption outcomes make terminal utility cancel, yielding the stated return loss. □

Scope of the result. Reordering checks does not remove the reversal. The result isolates information missing from compressed summaries; it does not lower-bound a named baseline given full evidence. Learned joint features may distinguish the contexts, but incorrect features can still select poorly. Cost-sensitive information acquisition is studied in stochastic function evaluation (Deshpande et al.,

2014; Hellerstein et al., 2022); correlated observations are studied in stochastic function certification (Ghuge et al., 2026). The example isolates the consequence of correlation for paid, feedback-delaying blocks with unchanged success summaries.

Table S24: **Exact laws and complete scheduling costs.** Each bit is a required check outcome. The left panel lists all probability atoms; the right lists all surviving-path block compositions at $c = 1$, $d = 3/2$. Both contexts have marginal pass probability $1/2$ and all-pass probability $1/4$. This is one finite construction, not two experimental datasets.

Outcome	A	B
000	0	3/8
001	1/4	0
010	1/4	0
011	0	1/8
100	1/4	0
101	0	1/8
110	0	1/8
111	1/4	1/4

Schedule	Cost A	Cost B
$1 \to 1 \to 1$	4.3750	4.6875
$1 \to 2$	4.2500	**4.2500**
$2 \to 1$	**4.1250**	4.4375
3	4.5000	4.5000

A compressed first-block choice has regret
$L_A(p) = p/8, \quad L_B(p) = 3(1-p)/16.$
Here p is the probability of choosing first-block size one (otherwise two), not a check-failure probability. They meet at $p = 3/5$, with loss $3/40$.

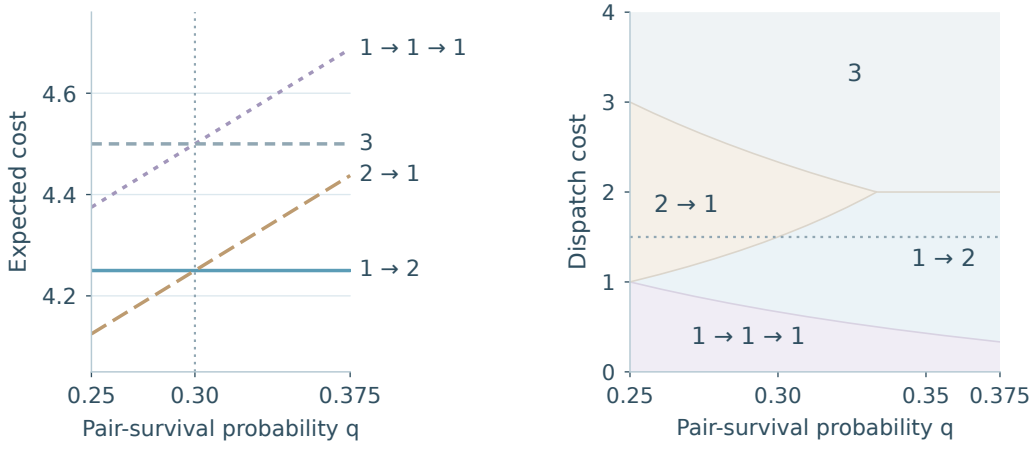
Continuous family. For pair-survival probability $q \in [1/4, 3/8]$, let $x \in \{0,1\}^3$ denote an outcome atom and $|x| = x_1 + x_2 + x_3$ its number of passes. Assign the probability of each atom according to its Hamming weight:

$$\Pr_q(X = x) = \begin{cases} 3q - 3/4, & |x| = 0, \\ 3/4 - 2q, & |x| = 1, \\ q - 1/4, & |x| = 2, \\ 1/4, & |x| = 3. \end{cases}$$

The singleton and doubleton lines apply to *each* of their three atoms. Probabilities are nonnegative and sum to one. Every marginal remains $1/2$, all-pass remains $1/4$, and every pair-survival probability is q. With unit per-check cost and arbitrary dispatch $d \geq 0$, the costs for schedules $1 \to 1 \to 1$, $1 \to 2$, $2 \to 1$, and 3, in that order, are

$$(1+d)(3/2+q), \quad 2+3d/2, \quad 2+d+q(1+d), \quad 3+d. \tag{29}$$

At $d = 3/2$, the optimum is $\min\{17/4, 7/2 + (5/2)q\}$, with a tie at $q = 3/10$. This is an exact sensitivity analysis, not a fitted empirical law.



(a) Expected cost at fixed dispatch cost

(b) Optimal scope across commitment costs

Figure S15: **The optimal scope depends on commitment costs.** Both plots evaluate the stated analytical costs, not fitted outcomes. The one-dimensional slice fixes dispatch at $3/2$; the phase map varies dispatch and pair-survival probability. Boundaries and ties follow the declared formulas.

Executable checks. The accompanying rational-arithmetic module enumerates every joint atom and solves the finite decision problem by dynamic programming over all nonempty unchecked subsets. Eleven tests check normalization, equal summaries, exchangeability, all six permutations, realized output equivalence, budget sufficiency, global optimality, the continuous switch, and the tight minimax certificate. Earlier fixed-order reference code is retained for provenance but is not used to establish this stronger proposition. These are finite mathematical tests, not neural or environment experiments.

G.5 Matched Scope Control

The preceding reversal can also be read as an exact two-factor comparison. This separates the benefit of a different permitted work scope from a weaker controller. Draw contexts A and B with probability $1/2$ each. Both arms see the same candidate, initial missing mask, budget, and check outcomes as they are acquired. Only the full-context row observes which joint law applies before acting. Both work-scope columns pay the same $3/2$ per dispatch and one per check, with the same final output for each realized vector. A singleton policy may remember all observed history and plan nonmyopically; it is not forced to optimize immediate reward.

Proposition 8 (Exact information–scope comparison)**.** *Under the three-check setting in Appendix G.4.2, optimal expected costs, averaging the two contexts, are*

	singleton only	*blocks permitted*
compressed summaries	$145/32$	$17/4$
full joint context	$145/32$	$67/16$

.

At full information, permitting blocks reduces expected cost by $11/32$*. With blocks permitted, context information reduces expected cost by* $1/16$*. These are dispatch-cost differences, not improved final output probabilities.*

Proof. For singleton execution, all check orders are equivalent by exchangeability, and every surviving path must finish. Its costs are $35/8$ in A and $75/16$ in B, averaging $145/32$. Knowing the context cannot improve check order. Without the context, the four fixed surviving-path block compositions have average costs $145/32, 17/4, 137/32, 9/2$; their minimum is $17/4$. Adaptation does not add a fifth surviving-path composition because every observed bit before termination is one. With full context, choose $2 \to 1$ in A and $1 \to 2$ in B, averaging $(33/8 + 17/4)/2 = 67/16$. Subtracting gives the two differences. □

The matrix above retains all four matched conditions; the zero-dispatch boundary separates the effect of information from the cost of committing a work block.

The zero-dispatch boundary is essential: when dispatch is free, singleton costs are $7/4$ and $15/8$, and allowing blocks cannot improve them. This agrees with Proposition 4. Our reference calculation exhausts all orders and nonempty unchecked subsets and checks identical realized output. Thus the comparison establishes a real cost mechanism without giving the singleton arm less information or a shorter planning horizon. It does not show that a learned network attains these optimal policies on real agent tasks. A matched empirical factorial is still needed.

G.6 Menu Approximation

G.6.1 Approximation Error

Proposition 9. *At one paid state S and fixed continuation policy π, let finite nonempty catalogs satisfy $\Omega \subseteq \Omega_\star$, where $\Omega_\star$ is the comparison catalog. Suppress the common state argument in Q^π and $\widehat{Q} = \widehat{Q}_{\tilde{\phi}}$. Suppose $|\widehat{Q}(\omega) - Q^\pi(\omega)| \le \varepsilon$ for every $\omega \in \Omega$, where $\varepsilon \ge 0$ is a uniform error bound, not the exploration rate. For selected $\omega_a \in \Omega$, write*

$$\Delta_{\mathrm{menu}} = \max_{\Omega_\star} Q^\pi - \max_{\Omega} Q^\pi, \qquad \delta_a = \max_{\Omega} \widehat{Q} - \widehat{Q}(\omega_a).$$

Here Δ_{menu} is omitted-menu loss and δ_a is the scorer gap of the chosen plan. Each maximum ranges over its displayed catalog. Then $\max_{\Omega_\star} Q^\pi - Q^\pi(\omega_a) \le \Delta_{\mathrm{menu}} + 2\varepsilon + \delta_a$.

Proof. Split at $\max_\Omega Q^\pi$, upper-bound this maximum by $\max_\Omega \widehat{Q} + \varepsilon$, and lower-bound the chosen value by $\widehat{Q}(\omega_a) - \varepsilon$. □

Observed RMSE is not a uniform error guarantee. Full catalog values require paid continuation experiments on bounded cases; unrestricted alternatives are unknown. The inequality diagnoses one decision and does not establish global regret or policy improvement. Solver gap under its additive surrogate is a separate computational quantity.

G.6.2 Seeded Refinement

For $m \geq 1$ extensions forming feasible job sets $A_0 \subset \cdots \subset A_m$, set $W(A) = \widehat{Q}_{\bar{\phi}}(\widetilde{S}, \text{VERIFY}(A))$ at one fixed proposal state. Then

$$\sum_{t=1}^{m} [W(A_t) - W(A_{t-1})] = W(A_m) - W(A_0).$$

The equality holds without additivity or submodularity. Unlike summing $W(D_j)$, it does not repeatedly add the same base continuation. However, positive single-job increments can still miss bundles: with two jobs of unit cost, gain $g > 2\lambda$, and a fixed continuation that stops immediately after the offered work, $W(\varnothing) = 0$, $W(\{a\}) = W(\{b\}) = -\lambda$, and $W(\{a, b\}) = g - 2\lambda > 0$. Empty-seed positive-gain greedy stops; an affordable complete-candidate seed presents the pair to the actor. This verifies the purpose of the seed family, not a general approximation ratio. Higher-order cross-candidate interactions and inaccurate W remain limitations.

G.7 Learning Identities

G.7.1 Policy-Gradient Baseline

Fix the initial state, verification contract Γ, menu settings ν, scorer snapshot $\bar{\phi}_k$, and support rules. Let $\vartheta = (\theta, \psi)$ contain only the sampled edit/plan parameters, τ a full search trajectory, P_ϑ its induced law, and $R(\tau)$ its realized return. With the typed probabilities of Equation (19) and the usual differentiability, integrability, and fixed-support conditions,

$$\nabla_\vartheta \mathbb{E}_{\tau \sim P_\vartheta} R(\tau) = \mathbb{E}\left[R_g \sum_{j=1}^{J_g} \nabla_\vartheta \log p_\vartheta(y_{g,j} \mid H_{g,j}, \mathcal{Y}_{g,j}) \right].$$

A deterministic menu generator may depend on observed history; its parameters are held fixed here and it contributes no sampled likelihood. Random environment outcomes likewise have no actor likelihood. For G conditionally independent searches, the baseline $b_g = (G-1)^{-1} \sum_{u \neq g} R_u$ is independent of trajectory g conditional on the common start and snapshot. The expectation of b_g times that trajectory's score function is zero. Consequently, at ϑ_k, the expected unclipped group estimator using $A_g = R_g - b_g$ equals the displayed gradient, with b_g detached. It is not an identity for gradients through scorer fitting, generator changes, or KL regularization. Clipping remains a surrogate, not a monotonicity guarantee (Ahmadian et al., 2024; Schulman et al., 2017).

Each sampled token and plan contributes once; a deterministic solver contributes none. Dividing each search by J_g, reusing shared-tree leaves as independent group members, or differentiating through A_g changes this estimator. Frozen-menu optimization is the local update in Appendix F.5; refreshed menus require fresh collection. This does not establish global policy improvement or an unbiased off-policy gradient after arbitrary optimization steps.

G.7.2 Contrast Identifiability

At a fixed state and continuation policy, each prospectively forced arm has an ordinary expected return. Independence and fixed sample sizes make its sample mean unbiased. Their difference therefore identifies the conditional contrast in Equation (12). Minimizing population squared difference error identifies that contrast, but is invariant to a common offset added to both predictions. The ordinary return loss in Equation (11) is consequently required to anchor mean calibration. Neither identity establishes data efficiency, actor improvement, or causality of an individual edit. These are separately measured claims.

H NOMENCLATURE

Full forms of abbreviations and method names in Section 2 and its cited work are collected below. Named systems without an initialism expansion are listed by their published title or a brief definition.

Table S25: Abbreviations and terminology.

Term	Full form or published name
Recursive self-improvement	
RSI	Recursive Self-Improvement.
SpecRSI (ours)	Speculative Recursive Self-Improvement (this work; Section 3).
VATS	Verification-Aware Tree Search (this work; Section 3.3).
GVO	Group Verification Optimization (this work; Section 3.4).
DGM	Darwin Gödel Machine (Zhang et al., 2026a).
SICA	Self-Improving Coding Agent (Robeyns et al., 2025).
ADAS	Automated Design of Agentic Systems (Hu et al., 2025).
AFlow	Automating Agentic Workflow Generation (Zhang et al., 2025b).
GEPA	Genetic-Pareto (Agrawal et al., 2026).
MemEvolve	Meta-Evolution of Agent Memory Systems (Zhang et al., 2025a).
Hyperagents	Self-referential agents combining a task agent and an editable meta agent in one program (Zhang et al., 2026b).
MetaRSI / RSI^2	A Meta-Recursive Self-Improving System for Recursive Self-Improving Systems Themselves (Tan et al., 2026).
HGM	Huxley-Gödel Machine (Wang et al., 2026b).
Janus	An Algorithm-Evaluator Co-Evolution Framework for LLM-Driven Discovery under Expensive Evaluation Budgets (Liu et al., 2026).
Speculative execution	
SpecInfer	Accelerating Large Language Model Serving with Tree-based Speculative Inference and Verification (Miao et al., 2024).
EAGLE-2	Extrapolation Algorithm for Greater Language-model Efficiency; the suffix denotes the dynamic-draft-tree variant (Li et al., 2024).
Sequoia	Scalable and Robust Speculative Decoding (Chen et al., 2024).
MTP	Multi-token Prediction (Gloeckle et al., 2024).
SV	Speculative Verification (Kim et al., 2026).
DSP	Dynamic Speculative Planning, the method introduced in *Dynamic Speculative Agent Planning* (Guan et al., 2026).
Speculative Actions	A Lossless Framework for Faster AI Agents (Ye et al., 2026).
FlashEvolve	Accelerating Agent Self-Evolution with Asynchronous Stage Orchestration (Hu et al., 2026b).
Learning and general terminology	
RL	Reinforcement Learning (Zhou et al., 2024).
LLM	Large Language Model (Leviathan et al., 2023).
ArCHer	Actor-Critic Framework with a Hierarchical Structure (Zhou et al., 2024).
GRPO	Group Relative Policy Optimization (Shao et al., 2024).
DVI	Draft, Verify, & Improve (Bhansali & Heck, 2025).
GTO	Group Tree Optimization (Hu et al., 2026a).
MAS	Multi-Agent System (Wang et al., 2026a).
MAS^2	Self-Generative, Self-Configuring, Self-Rectifying Multi-Agent Systems (Wang et al., 2026a).
LOO	Leave-One-Out (Ahmadian et al., 2024).
PPO	Proximal Policy Optimization (Schulman et al., 2017).
ToT	Tree of Thoughts (Yao et al., 2023).