VERIEVOLVE: HARNESS EVOLUTION FOR HARDWARE VERIFICATION AGENTS

Anonymous authors
Paper under double-blind review

ABSTRACT

Functional verification is a major bottleneck in hardware development, requiring specifications to be translated into executable checks of temporal behavior. For language-model agents, this raises a broader question: can verification procedures be improved by adapting the agent harness while keeping model weights fixed? We introduce VERIEVOLVE, a framework for verification-guided harness evolution that treats resource allocation, diagnostic priorities, reaction rules, and role instructions as a reusable policy. Its multi-agent executor coordinates specification analysis, artifact construction, and experimental challenge within each task; an outer search evaluates policy changes on development tasks before freezing them for transfer. The search is structured around four verification requirements: activating relevant behavior, observing its effects, discriminating correct from faulty behavior, and preserving established checks. A fixed evidence-validation kernel constrains artifact revisions, separating task-level repair from cross-task policy adaptation. We study four verification and debugging categories in CVDP, measuring task success, executed policy changes, and inference resources under explicit input and feedback protocols. In a matched final-only evaluation on 73 tasks selected from CVDP's agentic subset, the Sonnet-based MAS achieves 26.0% success, compared with 23.3% for a single agent under identical budget ceilings. Policy-search controls retain the initial policy, so these results do not establish an evolution gain. The framework makes adaptation without weight updates explicit through executable policies, evidence constraints, and frozen transfer evaluation.

1 INTRODUCTION

Functional verification checks whether a hardware implementation satisfies its specification. It is a major bottleneck in electronic design automation (EDA): verification can consume up to 70% of design time (Siemens, 2026). Engineers translate design intent into stimuli, checkers, and assertions, then diagnose discrepancies between expected and observed behavior. Automating these activities requires agents that connect specification reasoning to executable experiments, including clock alignment, legal transactions, and regression checks.

Verification can begin from a behavioral contract before the final register-transfer-level (RTL) implementation is available. Industrial model-first workflows develop executable specifications and verification environments alongside the design (Brunham et al., 2023). Early testbench development can also use a transaction-level reference model while RTL development proceeds (Jain et al., 2014). These workflows motivate agents that construct verification artifacts from specifications without assuming access to implementation internals. The agent must decide what behavior the specification requires, which assumptions an experiment introduces, and what its observations establish.

CVDP provides stimulus generation, checker construction, assertion generation, and RTL debugging tasks (Pinckney et al., 2025). Its official input boundary hides the evaluation harness and reference answers while allowing task-specific public context, including RTL and starter code. We study both this public-context setting and a stricter *specification-only* projection that removes separate RTL and template files. The latter stresses specification-driven construction; code already present in the prompt remains visible. For debugging tasks that require a particular implementation, removing its source can also remove information needed to identify a repair. We therefore distinguish difficulty due to restricted inputs from difficulty due to the agent's reasoning or control policy.

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

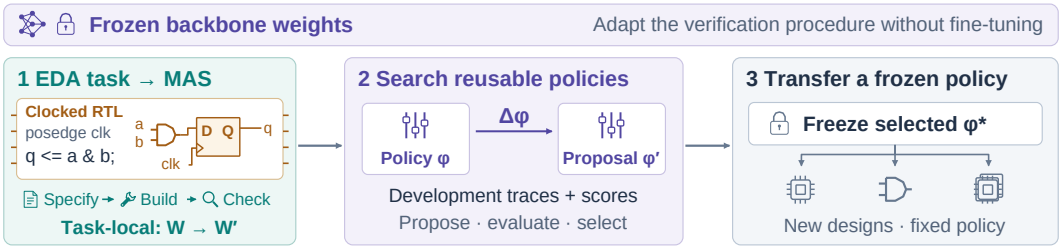


Figure 1: **Two timescales of verification-agent adaptation.** With backbone weights fixed, MAS produces and revises task artifacts. Across development designs, search proposes reusable policy changes from traces and scalar outcomes. The selected policy freezes for transfer. Figure 2 details the executor and search.

Research Challenge. How can an agent improve its verification procedure when the implementation is hidden or incomplete and local feedback only partially explains failure? A stimulus may fail to activate a behavior; an observation may occur at the wrong cycle; a checker may accept both legal and faulty traces. These failures can produce the same unsuccessful outcome while requiring different interventions. More inference steps help only if the agent allocates them to experiments that distinguish these explanations. A useful revision must also survive beyond the current design: repairing one checker and improving the procedure for constructing future checkers are different adaptation problems.

We address this challenge through *harness evolution*: search over the procedures that govern an agent's roles, resource allocation, and reactions to evidence. Model weights remain fixed, so adaptation requires no fine-tuning and works through existing model interfaces. Prior work establishes agent-design search and empirical agent self-improvement (Hu et al., 2024; Zhang et al., 2025); our hypothesis is that verification-specific failure distinctions provide a useful prior for this search. Figure 1 separates task-local artifact revision from changes to a policy reused across designs.

VERIEVOLVE implements this idea with a multi-agent system (MAS) for specification analysis, artifact construction, and experimental challenge. An outer search proposes changes to stage budgets, diagnostic priorities, reaction rules, and role instructions. A fixed execution kernel preserves tool permissions and requires reproducible evidence before accepting a repair. Development evaluations select a policy, which then freezes for transfer to other design families and backbones. Our primary evaluation uses CVDP's agentic subset across four verification and debugging tasks.

Contributions. **(1) A verification-specific adaptation problem.** We formulate harness evolution with fixed model weights and distinguish task-local artifact revision, development selection, and frozen transfer. **(2) An executable policy space grounded in verification.** We expose budgets, diagnostic priorities, and evidence-conditioned reactions within a fixed multi-agent executor. This connects policy changes to temporal reasoning, fault activation, observation, and regression preservation. **(3) An evaluation that separates performance from mechanism.** We organize published CVDP references, complete local baselines, matched architecture controls on all 73 selected tasks, and policy-search results by harness and backbone. Policy admission, executed reactions, and inference usage accompany task success. Historical Gemma results motivate the approach; the controlled study identifies which parts of executable self-improvement still require empirical validation.

2 VERIFICATION HARNESSES AS AN IMPROVEMENT SPACE

A task x provides a specification, permitted source files or templates, and required deliverable paths. A fixed language model with parameters θ executes a harness H under an episode budget B, producing an artifact $W = A_{\theta,H}(x; B)$ and trace τ. An external benchmark measures utility $U_x(W)$. The agent harness organizes this process; the verification artifact and benchmark grader have distinct roles.

Optimization at Two Timescales. Write $H = K[\phi]$, where K is a fixed execution kernel and $\phi \in \Phi_K$ is an admissible policy. Model parameters θ remain fixed throughout search and execution for each backbone; optimization requires no gradient update or access to model weights. Task

execution changes W; development search changes the reusable ϕ. For a fixed backbone, the target of search is

$$J_\theta(\phi; B) = \mathbb{E}_{x\sim\mathcal{D}_{\text{transfer}}}\mathbb{E}_\xi\left[U_x(A_{\theta,K[\phi]}(x; B, \xi))\right], \qquad \phi \in \Phi_K. \tag{1}$$

Search observes a finite development set, not this transfer distribution. It uses the empirical utility $\widehat{J}_{\text{dev}}$ to select a policy within a declared search allowance. Transfer then holds that policy fixed; cross-backbone evaluation additionally changes θ without retuning. A development maximum is therefore a selection statistic, not an estimate of Equation 1. Search cost and per-task execution cost are separate experimental quantities. Here ξ denotes model sampling and execution variability; a single rollout does not determine a policy's expected utility.

Evidence-Constrained Sequential Decisions. The controller observes $s_t = (W_t, \mathcal{O}_t, \mathcal{E}_t, \mathcal{C}_t, r_t)$: the current artifact, specification obligations, execution evidence, registered checks, and remaining resource vector. This recorded state does not reveal the task's complete functional semantics. A policy requests a role or reaction u_t, while the fixed kernel determines whether that request is executable:

$$u_t \sim \mu_{\theta,\phi}(\cdot \mid x, \tau_{\le t}), \qquad \bar{u}_t = G_K(s_t, u_t), \qquad s_{t+1} = T_K(s_t, \bar{u}_t, o_{t+1}). \tag{2}$$

Here o_{t+1} is a tool observation, and G_K enforces permissions, remaining allowances, and evidence requirements. Unsupported repair requests become rechecks or deferrals. Equation 2 is a formalization of the implemented controller, not a learned transition model or a claim that diagnoses identify the true failure cause. MAS factors μ across role contexts and prescribed handoffs; the single-agent control uses one context with the same public inputs and total allowance. This identifies role organization as an empirical intervention while keeping the information boundary explicit.

Why Verification Needs Diagnostic State. Compilation and a failing check do not identify what should change. For example, a ready/valid transaction is accepted at the specified event when both signals hold. A failed output check could reflect an unaccepted input, an early observation, or an incorrect result. These alternatives suggest different experiments before any repair: establish acceptance, align the observation with the required latency, then test the expected relation. The task specification determines the legal event and latency; the harness must not assume a universal handshake convention.

An obligation associates a source requirement with expected behavior and explicit assumptions. A diagnostic experiment connects an observation to an executed candidate and a proposed failure cause. We distinguish activation, propagation or observation, and discrimination, with separate outcomes for temporal prerequisites, preservation failures, unsupported tools, and unresolved semantics. The classes need not be independent or exhaustive. Their purpose is to select an intervention whose effect can be checked. Our implementation records source, expected behavior, and assumptions; a fully structured temporal obligation graph remains an extension.

Domain Structure Guides Proposals. A modifier samples $\phi' \sim q_m(\cdot \mid \phi, \tau_{\text{dev}}, \widehat{J}_{\text{dev}})$, where m specifies generic or EDA guidance. Both proposal conditions use the same feasible space Φ_K and selection rule. EDA guidance changes the proposal prior: it relates a diagnosed bottleneck to an experimental or control decision (Table 1). It does not supply a correct reference model, change the grader, or relax the acceptance guard. Whether this prior improves search is an empirical question.

Local Preservation and Transfer. For the checks $\mathcal{C}_t$ frozen before a repair, replay establishes only preservation on that finite set:

$$\bigwedge_{c\in\mathcal{C}_t} c(W_{t+1}) = 1 \quad \not\Rightarrow \quad U_x(W_{t+1}) = 1.$$

Generated references may share the same misunderstanding, and passing checks can miss relevant behavior. Similarly, an admissible policy with higher development utility need not improve transfer utility. The fixed kernel governs local execution; held-out evaluation measures the usefulness of the selected policy across designs.

3 VeriEvolve: Execution and Harness Evolution

Figure 2 connects three parts of the method: verification structure supplies diagnostic hypotheses (A), a mutable policy governs execution within a fixed kernel (B), and selected policies freeze for transfer

Table 1: Verification structure motivates different policy interventions. These are design hypotheses, not separately evaluated modules.

Task	Diagnostic question	Reusable intervention
Stimulus	Did a legal transaction reach the intended state?	Allocate challenge effort to acceptance and completion witnesses.
Checker	Does rejection distinguish faults from legal behavior?	Pair specification-supported legal traces with targeted faulty traces.
Assertions	Was the antecedent activated in the intended observation window?	Prioritize activation witnesses and clock/reset alignment checks.
RTL repair	Did the patch fix the trigger while retaining other behavior?	Reproduce the counterexample and replay frozen preservation checks.

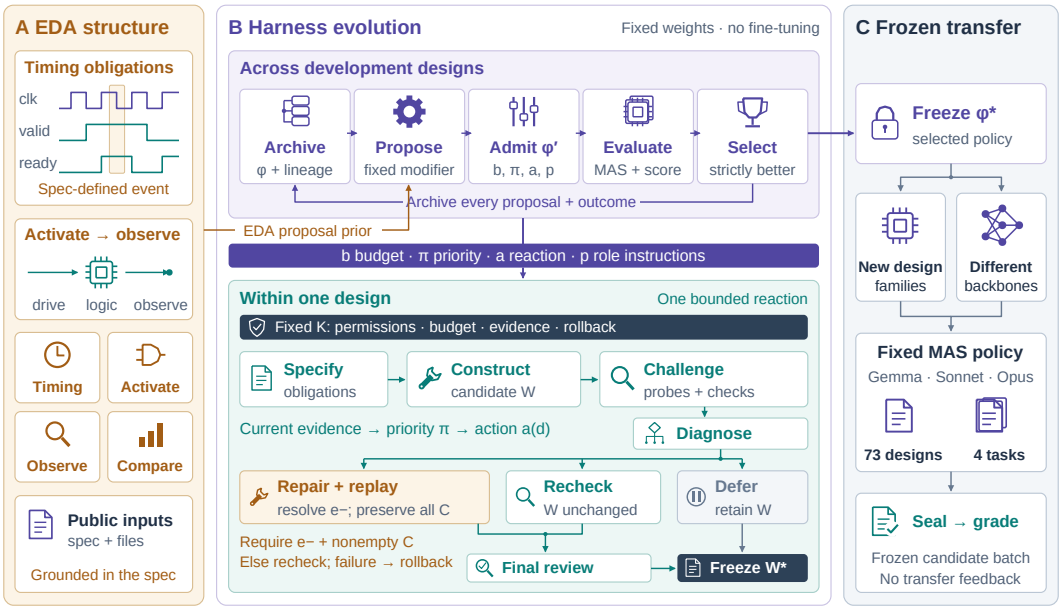


Figure 2: **EDA-guided executable policy search. (A)** Timing and circuit behavior provide a diagnostic vocabulary; the waveform illustrates a specification-defined observation event. **(B)** Development search changes budgets, priorities, reactions, and instructions while model weights remain fixed. A fixed kernel executes each policy as a MAS with one bounded reaction. Repair must resolve a reproduced failure and retain frozen passing checks; repair and recheck lead to final review. **(C)** Selected policies freeze across transfer families and backbones. Each design uses the same MAS executor.

(C). The agent receives the task specification, allowed context, and deliverable paths. Depending on the input projection, that context includes separate RTL or templates; the controller stays the same.

3.1 Executable Policies over a Fixed Kernel

The policy $\phi = (b, \pi, a, p)$ controls the middle panel of Figure 2. Five integer shares b allocate the episode budget to specification analysis, construction, challenge, reaction, and final review. Each stage ends at its cumulative allowance. The permutation π orders diagnosis classes, $a(d)$ requests repair, recheck, or defer for class d, and p supplies bounded role instructions. For example, shifting budget from construction to challenge changes how many experiments the verifier can attempt. Prioritizing temporal evidence changes which current diagnosis determines the reaction. These edits alter execution while keeping the backbone, total budget, tool permissions, and evidence guards fixed. The admissible allocation set is

$$\mathcal{B} = \{b \in \mathbb{Z}^5 : \textstyle\sum_i b_i = 100, \quad b_i \geq 5, \quad b_{\text{artifact}} \geq 20, \quad b_{\text{verifier}} \geq 15\}.$$

Together with a permutation of seven diagnosis classes, three possible reactions per class, and four bounded instruction strings, these constraints define Φ_K. Repair is excluded for tool limitations and unknown semantics.

From Specification to Diagnostic Evidence. The specification worker extracts obligations, source references, and assumptions. The construction worker receives these obligations and writes the declared deliverables. An independent challenge worker then tests the candidate in scratch space. For a checker, this can pair a legal trace with a deliberately faulty trace. Testing a timing-sensitive assertion requires checking whether reset and antecedent conditions occur at the specification's observation event, as illustrated in Figure 2A.

The challenge worker records a diagnosis only with an executed receipt and an exact quotation from the public specification. Each receipt binds an observation to the candidate's content hash, preventing evidence from an earlier artifact from silently authorizing a later change. Diagnoses distinguish temporal prerequisites, activation, observation, discrimination, preservation, tool limitations, and unknown semantics. This evidence establishes what ran and what the specification states; the worker's causal interpretation remains a hypothesis.

Evidence-Conditioned Reaction. The highest-priority current diagnosis determines the requested reaction. Repair requires a negative experiment that reproduces from frozen source files and a nonempty set $\mathcal{C}$ of separately preserved passing checks. Insufficient evidence changes the reaction to recheck; unknown semantics or tool limitations cannot authorize repair. After construction of W', the same negative experiment must pass and every preserved check must still pass:

$$\mathrm{accept}(W') = \mathrm{resolve}(e_{\mathrm{fail}}, W') \wedge (|\mathcal{C}| > 0) \wedge \bigwedge_{c \in \mathcal{C}} c(W').$$

Failure or an interrupted repair restores the complete prior candidate. This is local preservation under model-authored checks, not a correctness proof. It strengthens the earlier kernel, whose repair gate did not require resolution of the triggering negative experiment.

Evidence-Conditional Preservation. Let W^- be the captured artifact before a guarded repair and $R_{\mathcal{C}} = \{W : \bigwedge_{c \in \mathcal{C}} c(W) = 1\}$ for the frozen checks. Assuming faithful replay, unchanged check sources, and complete artifact restoration, the repair output satisfies

$$W^+ \in \{W^-\} \cup \{W \in R_{\mathcal{C}} : \mathrm{resolve}(e_{\mathrm{fail}}, W) = 1\}. \tag{3}$$

An accepted revision resolves the recorded trigger and passes the frozen checks; otherwise the kernel restores W^-. The case argument and its assumptions appear in Appendix O. This property holds independently of the proposed policy and concerns recorded experiments, not all possible hardware behaviors.

Recheck allocates one more challenge visit while leaving the candidate unchanged. Defer retains the candidate when the policy requests it, no current diagnosis is available, or the remaining reaction budget is insufficient. Repair and recheck lead to final review; defer proceeds directly to freezing, as shown in Figure 2B. Each episode permits one bounded reaction, so a changed policy affects a controlled decision rather than an unrestricted retry loop.

Runtime Boundary. All shell executions mount candidate files read-only. Scoped write/edit capabilities permit only the construction worker to change deliverables. Experiments run offline with bounded termination, and records distinguish successful exit, nonzero exit, and timeout. Registered experiment sources are replayed from preserved copies. These implemented boundaries are behavior-tested, without certifying the adequacy of generated checks.

3.2 Development Search and Controls

The upper loop in Figure 2B updates the reusable policy. A fixed modifier proposes a complete typed policy from its parent, public-input solver traces, and scalar development outcomes. Candidates are admitted only within the declared policy space, evaluated on the same development tasks, and archived with lineage and usage. Selection replaces the incumbent only for a higher development pass count; ties retain it. Repeated identical policies do not receive additional solver samples. Grader

Table 2: Success (%) by task and difficulty on the 73-design evaluation cohort: stimulus generation (cid012), checker construction (cid013), assertion generation (cid014), and RTL debugging (cid016). Gray: CVDP-based references; teal: VeriEvolve. Each Total pools all tasks in its category. Bold marks numerical column leaders within each shaded group.

		cid012				cid013				cid014				cid016			
Harness	**Backbone**	Easy	Med.	Hard	**Total**	Easy	Med.	Hard	**Total**	Easy	Med.	Hard	**Total**	Easy	Med.	Hard	**Total**
CVDP†	Haiku 4.5	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	28.6	28.6	42.9	32.1	**85.7**	33.3	100.0	72.7
	Sonnet 5	**33.3**	0.0	50.0	12.5	20.0	0.0	0.0	5.6	**71.4**	35.7	**57.1**	**50.0**	57.1	33.3	100.0	54.5
	Opus 5	**33.3**	**9.1**	**100.0**	**25.0**	**60.0**	**25.0**	0.0	**33.3**	57.1	**42.9**	28.6	42.9	71.4	**100.0**	100.0	**81.8**
VeriEvolve	Gemma	**66.7**	**18.2**	0.0	**25.0**	**80.0**	**25.0**	0.0	**38.9**	**42.9**	**57.1**	42.9	**50.0**	57.1	0.0	**100.0**	45.5
	Sonnet 5	33.3	0.0	0.0	6.3	60.0	8.3	0.0	22.2	28.6	28.6	42.9	32.1	57.1	**33.3**	0.0	45.5

† References use a local single-agent grader-stop wrapper. VeriEvolve–Gemma uses historical grader-stop evaluation; VeriEvolve–Sonnet uses final-only evaluation.

source, expected solutions, and grader logs are excluded from both solver and modifier messages. Development scores still supervise outer selection, so optimization is not grader-free.

The registered controls are fixed H_0, instruction-only search, generic executable search, and EDA-guided executable search. Generic and EDA-guided search share the editable fields and proposal allowance; the latter adds structured diagnostic guidance to the modifier. Instruction-only search can change p alone. This comparison tests a domain-informed mutation prior, not a claim that the underlying incumbent search algorithm is new.

MAS roles share 84 responses, 96,000 output tokens, 600,000 processed tokens, and 1,800 seconds per episode. Stage reservations include a report opportunity; token thresholds are checked between calls. The selected policy is subsequently fixed across transfer designs and backbones. Each evaluation task uses multi-role, multi-step execution under the same frozen policy; task difficulty does not select a different controller.

4 Experiments

4.1 Tasks and Evaluation Protocols

Our primary evaluation selects the *agentic subset* of CVDP v1.1.0 and includes every task in four categories: stimulus generation (cid012), checker construction (cid013), assertion generation (cid014), and RTL debugging (cid016). This yields 73 designs, with category sizes $(16, 18, 28, 11)$ and difficulty counts of 22 Easy, 40 Medium, and 11 Hard. Within this subset, every task is retained, including failures. All primary rows cover these same task IDs. We report observed success on this cohort; no extrapolation to unmeasured CVDP tasks enters the table.

Inputs and Feedback. Official-context runs retain the benchmark's public specification and permitted source files. Specification-only projection removes separate RTL and template files while retaining the prompt, documentation, and required output paths. In the rebuilt executor, neither solver nor modifier receives grader source, reference answers, or grader logs. The modifier receives scalar development scores after candidate sealing. Every transfer candidate freezes before the first transfer grade; no transfer outcome returns to policy search. Public RTL availability and private grader isolation are separate properties of this protocol.

Historical *grader-stop* runs use intermediate benchmark outcomes to stop generation. The earlier *final-evaluation* runs score the final artifact but retain a shared skill store curated using earlier grader success. These records provide developmental context under different feedback and budget conditions. The matched Sonnet control fixes public inputs, policy, task IDs, and budget ceilings before either architecture runs, allowing a narrower comparison of role decomposition.

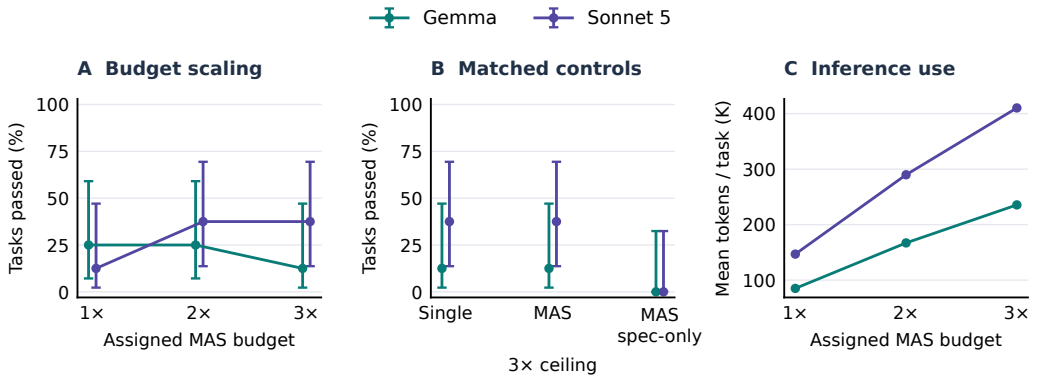


Figure 3: **Matched budget and architecture controls.** (A) Pass rate under assigned budget multipliers. (B) Three configurations at the same 3× ceiling. (C) Realized mean processed tokens under each assigned ceiling. Error bars are 95% Wilson intervals over eight development designs with one rollout each; they do not measure variation across repeated searches. Lines join measured settings only.

4.2 Performance by Task and Difficulty

Table 2 crosses the four task categories with CVDP's three difficulty labels. Each percentage is computed from the corresponding task outcomes. Aggregate scores weight designs equally; they are not averages of difficulty-cell percentages. Cell sizes matter: cid013 and cid016 each contain only one Hard design, so a 100.0% entry in either cell does not establish reliable performance on difficult designs.

Local Haiku 4.5, Sonnet 5, and Opus 5 references use a single-agent wrapper with intermediate grading and a 600-second ceiling. This is a CVDP-based wrapper, not the untouched official harness. Published CVDP references use a different release and generation protocol (Pinckney et al., 2025); Appendix C retains them separately from our cohort.

Historical Performance Context. The recorded Gemma MAS achieves 41.1%, compared with 42.5% for the local Opus reference, under differing budgets and grader-stop protocols. This is not a matched equivalence result. Appendices A and B document whole-run selection, component controls, and the Gemma/Opus 4.6 pilot. HORIZON's 100.0% cumulative completion (Yu et al., 2026) and our corresponding local records appear in Appendix I, separately from final-only success.

Task Structure Changes the Comparison. Across all difficulties, the historical Gemma MAS scores 25.0%, 38.9%, 50.0%, and 45.5% on cid012, cid013, cid014, and cid016, respectively. The local Opus reference scores 25.0%, 33.3%, 42.9%, and 81.8%. Thus, similar aggregate performance conceals a substantial repair gap. Activation sequences, discriminating checkers, temporal assertions, and RTL repair demand different evidence and tool use. This motivates the task-aware policy space, while backbone and protocol differences prevent attributing the observed pattern to any individual intervention.

4.3 Budget, Architecture, and Policy Controls

The auxiliary development matrices evaluate five configurations on the same eight designs for each of Gemma and Sonnet 5. These earlier pilot cohorts are separate from the primary 73-design comparison; their exact membership and relationship to the benchmark populations are documented in Appendix F. MAS receives proportional 1×, 2×, and 3× allowances; single-agent and specification-only controls receive the 3× allowance. Each multiplier scales model calls, token limits, and elapsed time together. All candidates seal before scoring. Table 13 gives numerical results and resource use.

Table 3: Matched Sonnet 5 architecture control on the selected 73-design cohort. Both arms receive identical public inputs and 3× budget ceilings; final-only scoring follows candidate sealing. Success is in percent; tokens are mean input plus output per task.

Architecture	Easy	Medium	Hard	Total	Tokens (K)
Single agent	36.4%	12.5%	36.4%	23.3%	402.7
VeriEvolve	45.5%	15.0%	27.3%	26.0%	431.1

Table 4: Policy-space ablation on the same eight Sonnet development tasks. Each search condition proposes once. Checks indicate admitted edits relative to the fixed policy; – for the rejected proposal means unmeasured. All evaluated policies tie, so selection retains the ancestor.

Condition	Role notes	Executable fields	Admitted	Success
Fixed policy	–	–	–	37.5%
Instruction-only	✓	–	Yes	37.5%
Generic executable	✓	✓	Yes	37.5%
EDA-guided	–	–	No	–

Does Additional Inference Improve Verification? Figure 3 shows that Sonnet's MAS improves from 12.5% at 1× to 37.5% at 2×, then plateaus at 3×. Mean processed tokens nevertheless increase from 146.9K to 289.9K and 410.4K per task. Gemma scores 25.0%, 25.0%, and 12.5% while consuming 85.1K, 167.1K, and 235.6K tokens. Greater allowance therefore does not yield uniformly higher observed success. These are individual rollouts on eight designs; the nonmonotonic Gemma result does not establish that more compute generally harms performance. A larger allowance creates opportunities for useful verification only if the resulting actions acquire relevant evidence.

Does Role Decomposition Improve Success? In the development pilot at 3×, MAS and single-agent execution tie: 12.5% for Gemma and 37.5% for Sonnet. Gemma's MAS uses more tokens than its single-agent control (235.6K versus 73.7K per task); Sonnet's means are closer (410.4K versus 396.8K). Both backbones score 0.0% with the specification-only projection. This small input ablation indicates that removing public implementation context can change the task materially; it does not establish that such context is necessary for every category.

The matched Sonnet evaluation covers all 73 designs in our selected cohort. MAS scores 26.0%, compared with 23.3% for the single agent (Table 3). There are seven MAS-only and five single-agent-only successes; the paired difference is not statistically established ($p = 0.774$, exact two-sided discordance test). Mean processed tokens are 431.1K for MAS and 402.7K for the single agent. Equal ceilings therefore do not imply equal realized cost. This control supports a modest observed difference under matched conditions, while leaving the benefit of role decomposition unresolved.

Transfer and Difficulty. The auxiliary matched Sonnet control completes sixteen transfer designs (Table 15). MAS scores 43.8%, compared with 31.3% for the single agent, using 403.2K versus 307.8K mean processed tokens. Four MAS-only and two single-agent-only successes account for the difference. The original development and transfer cohorts contain no Hard designs. A separate diagnostic covers all eleven Hard designs from the primary cohort and gives 27.3% for MAS versus 36.4% for the single agent (Table 16). These contrasting observations motivate reporting task and difficulty together rather than treating a small transfer result as evidence of uniform improvement.

Do Admitted Policy Changes Transfer? The Sonnet intervention pilot admits instruction-only and generic executable proposals. The generic policy changes stage allocation and diagnostic priority; both evaluated children score 37.5% on development, tying the ancestor. The EDA-guided proposal fails schema validation and receives no solver score. Selection therefore retains the ancestor for all conditions. Its shared frozen policy scores 43.8% on the auxiliary transfer cohort, from one deduplicated evaluation. This measures transfer of the retained policy, not an evolution gain. Appendix N documents admission outcomes and usage. Gemma's matrix is complete, but its modifier request was rejected by the relay before search completed; no Gemma search or transfer score is inferred.

Where Does the Evidence Pipeline Constrain Adaptation? Across the 73-design Sonnet MAS evaluation, every episode produces a complete candidate, but only 21 register a regression check and only one admits a diagnosis. The policy executes one recheck and no repair (Table 17). Artifact completion is therefore a weak proxy for verification progress. The executor can finish a deliverable without obtaining evidence strong enough to justify a conditional revision. This distinction matters for evaluating harness evolution: a changed policy may be valid yet receive no opportunity to exercise its repair rule.

Does a More Explicit Diagnosis Interface Help? A separate eight-design Sonnet experiment holds the backbone, fixed policy, evidence validator, and budget constant while changing how agents name specification sources. An explicit list of valid sources yields 37.5% success, compared with 25.0% for the original free-text interface. The original arm makes fourteen diagnosis requests, including two with literal specification quotations but invalid source labels; the explicit arm makes none. Neither arm admits a diagnosis or executes repair. This comparison separates an observed task-score change from its proposed mechanism: the extra success does not demonstrate better diagnosis. Appendix N records the interface and trace audit.

5 Related Work

Reflection and Workflow Optimization. Self-Refine iterates feedback and revision on a model's output, while Reflexion stores verbal feedback for subsequent trials (Madaan et al., 2023; Shinn et al., 2023). These approaches motivate learning from execution but do not make every revision a reusable controller change. DSPy optimizes declarative model pipelines; GEPA evolves prompts using trajectory reflection; AFlow searches over code-represented workflows (Khattab et al., 2023; Agrawal et al., 2025; Zhang et al., 2024). Our question concerns the search space and evidence needed in verification: which diagnostic distinctions should guide changes to resource allocation and conditional reactions, and whether those changes transfer.

Self-Modifying Agents. ADAS searches programmatic agent designs; DGM modifies agent code through empirical evaluation and an archive (Hu et al., 2024; Zhang et al., 2025). Gödel Agent explores self-referential modification, and Hyperagents make the meta-agent editable (Yin et al., 2024; Zhang et al., 2026). The distinction between model updates and scaffold updates is also central to the taxonomy of Ren et al. (2026). Our modifier and execution guards remain fixed; bounded policy search carries no optimality guarantee from a theoretical Gödel machine (Schmidhuber, 2003). SBCO studies verifier-bank and harness-policy improvement in planning (Kulkarni et al., 2026). Harness optimization itself is therefore established prior work. We focus on the EDA-specific intervention space and on separating valid proposals, exercised control changes, and transfer utility.

Hardware Verification and Repair. AssertLLM structures assertion generation around interpretation of natural language specifications and waveform diagrams (Yan et al., 2025). CorrectBench applies specification-based functional validation and correction to generated testbenches (Qiu et al., 2024). HORIZON organizes hardware work as evaluated repository evolution (Yu et al., 2026); CVDP provides our stimulus, checker, assertion, and repair populations (Pinckney et al., 2025). We study reusable execution policies across these tasks. Temporal alignment, activation witnesses, discrimination between legal and faulty behavior, and regression preservation motivate different policy choices. The published CVDP references and our newer local Claude baselines provide complementary performance context; their evaluation protocols are kept explicit.

6 Conclusion

VeriEvolve treats the verification harness as an explicit adaptation target while keeping model weights fixed. Its multi-agent executor connects specification analysis, artifact construction, and experimental challenge; its policy search separates task-local revision from reusable procedural changes. The CVDP evaluation combines task-specific outcomes, matched architecture and resource controls, and execution-level evidence. Together, these measurements provide a concrete basis for studying when harness changes produce useful verification behavior and transfer across designs without backbone fine-tuning.

Reproducibility Statement

Sections 2 and 3 specify the policy space, execution kernel, and search procedure. Section 4.1 describes the benchmark release, task categories, input projections, and feedback boundaries. The appendices document exact development and transfer tasks (Appendix F), recorded models and historical cohorts (Appendix A), candidate sealing and interrupted-run accounting (Appendix E), and resource budgets and control protocols (Appendix M). Historical results retain their original protocol qualifications rather than being treated as matched comparisons. The anonymous project repository is linked at `https://anonymous.4open.science/r/VeriEvolve/`. Re-execution with hosted models depends on access to the recorded model versions and may vary with model sampling and service conditions.

AI Use Statement

We used LLM-based assistants to improve the clarity and organization of the manuscript, identify and retrieve relevant literature, and draft selected sections. AI assistance also supported methodological brainstorming, formalization and drafting the conditional preservation argument, implementation and debugging, experiment orchestration, and interpretation of recorded results. AI tools assisted diagram design, including an image-generated prototype of Figure 2; the final diagrams use editable vector shapes. The LLMs evaluated as components of our verification system, along with their experimental roles, are described in the methodology and experiment sections. Reported empirical results are derived from saved execution and evaluation records. The authors take responsibility for the final manuscript, including the accuracy of its claims, references, and AI-assisted text and artifacts.

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. *arXiv preprint arXiv:2507.19457*, 2025.

Kalen Brunham, Anthony Moore, Tobias Rozario, Wei Jun Yeap, and Jakob Engblom. Closed-loop model-first SoC development with the Intel Simics simulator. In *Design and Verification Conference and Exhibition (DVCon)*, 2023. URL `https://dvcon-proceedings.org/wp-content/uploads/91435.pdf`.

Shengran Hu, Cong Lu, and Jeff Clune. Automated design of agentic systems. *arXiv preprint arXiv:2408.08435*, 2024.

Abhishek Jain, Hima Gupta, Sandeep Jana, and Krishna Kumar. Early development of UVM based verification environment of image signal processing designs using TLM reference model of RTL. *International Journal of Advanced Computer Science and Applications*, 5(2), 2014. URL `https://arxiv.org/abs/1408.1150`.

Omar Khattab, Arnav Singhvi, Paridhi Maheshwari, Zhiyuan Zhang, Keshav Santhanam, Sri Vardhamanan, Saiful Haq, Ashutosh Sharma, Thomas T. Joshi, Hanna Moazam, Heather Miller, Matei Zaharia, and Christopher Potts. DSPy: Compiling declarative language model calls into self-improving pipelines. *arXiv preprint arXiv:2310.03714*, 2023.

Vivek Kulkarni, Sudipta Paul, Aounon Kumar, Nicholas Tzou, and Srinivas Chappidi. SBCO: Self-supervised, verifier-grounded harness optimization for planning agents. *arXiv preprint arXiv:2608.10157*, 2026.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, Shashank Gupta, Bodhisattwa Prasad Majumder, Katherine Hermann, Sean Welleck, Amir Yazdanbakhsh, and Peter Clark. Self-refine: Iterative refinement with self-feedback. In *Advances in Neural Information Processing Systems*, 2023.

Nathaniel Pinckney, Chenhui Deng, Chia-Tung Ho, Yun-Da Tsai, Mingjie Liu, Wenfei Zhou, Brucek Khailany, and Haoxing Ren. Comprehensive Verilog design problems: A next-generation benchmark dataset for evaluating large language models and agents on RTL design and verification. *arXiv preprint arXiv:2506.14074*, 2025.

Ruidi Qiu, Grace Li Zhang, Rolf Drechsler, Ulf Schlichtmann, and Bing Li. CorrectBench: Automatic testbench generation with functional self-correction using LLMs for HDL design. *arXiv preprint arXiv:2411.08510*, 2024.

Zhe Ren, Yimeng Chen, Dandan Guo, Guowei Rong, Tonghui Li, R. B. Xiong, Qingfeng Lan, Wenyi Wang, Li Nanbo, Yibo Yang, Mingchen Zhuge, and Jürgen Schmidhuber. Self-improvements in modern agentic systems: A survey. *arXiv preprint arXiv:2607.13104*, 2026.

Jürgen Schmidhuber. Gödel machines: Self-referential universal problem solvers making provably optimal self-improvements. *arXiv preprint cs/0309048*, 2003.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. In *Advances in Neural Information Processing Systems*, 2023.

Siemens. Siemens advances self-verifying agentic AI workflows for semiconductor and PCB design. Press release, July 26, July 2026. URL `https://news.siemens.com/en-us/siemens-nvidia-dac-2026/`. Accessed September 21, 2026.

Zhiyuan Yan, Wenji Fang, Mengming Li, Min Li, Shang Liu, Zhiyao Xie, and Hongce Zhang. AssertLLM: Generating hardware verification assertions from design specifications via multi-LLMs. In *Proceedings of the 30th Asia and South Pacific Design Automation Conference*, 2025. doi: 10.1145/3658617.3697756.

Xunjian Yin, Xinyi Wang, Liangming Pan, Li Lin, Xiaojun Wan, and William Yang Wang. Gödel agent: A self-referential agent framework for recursive self-improvement. *arXiv preprint arXiv:2410.04444*, 2024.

Cunxi Yu, Chenhui Deng, Nathaniel Pinckney, and Brucek Khailany. Agentic hardware design as repository-level code evolution. *arXiv preprint arXiv:2606.28279*, 2026.

Jenny Zhang, Shengran Hu, Cong Lu, Robert Lange, and Jeff Clune. Darwin Gödel machine: Open-ended evolution of self-improving agents. *arXiv preprint arXiv:2505.22954*, 2025.

Jenny Zhang, Bingchen Zhao, Wannan Yang, Jakob Foerster, Jeff Clune, Minqi Jiang, Sam Devlin, and Tatiana Shavrina. Hyperagents. *arXiv preprint arXiv:2603.19461*, 2026.

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. *arXiv preprint arXiv:2410.10762*, 2024.

Table 5: Matched pilot pass rates on four development tasks, one per category. Rates use final native-grader outcomes; timeout is listed separately and remains in the denominator. Tokens include input and output reported for completed generation attempts.

Configuration	Pass rate	Timeout	Calls	Tokens (M)
Gemma single	25.0%	0	54	0.876
Gemma MAS	50.0%	0	81	0.656
Opus single	50.0%	1	64	1.286
Opus MAS	50.0%	0	93	1.354

A Recorded Experimental Cohorts

The earlier full-split results correspond to `mas_strict_agentic_NOORACLE`, `NOORACLE_CKPT`, and `CKPT_TSFIX` for the three final-evaluation configurations, and `FIXEDDISPATCH` and `SEED2` for grader-stop MAS. The first of these grader-stop files contains 75 rows but 73 unique task IDs; retaining the first or last record for each task yields 30 passes in both cases. Table 2 reports the higher-scoring complete Gemma run (41.1%) from these two recorded runs; the second scores 32.9%. The selection is across whole runs, not individual tasks or difficulty strata. Both runs remain in Table 6. The comparative baseline files are locally generated records on the same 73-task cohort. Claude files record `recovered`; Gemma files record `passed`. Model identifiers are `claude-haiku-4-5-20251001`, `claude-sonnet-5`, `claude-opus-5`, and `gemma-4-31b-it`. The selected Gemma row remains a historical run; its backbone endpoint and grader-stop protocol differ from the rebuilt final-only executor. Data from other CVDP populations remain in the research archive and are not pooled into the primary comparison.

The cumulative completion sources are `agentic4_cascade.jsonl` and `agentic4_escalation.jsonl`. Their ID sets were checked: the escalation set equals the initial failure set, and their union matches all 73 native agentic tasks in the four categories. The initial and escalation stages report approximately $92.00 and $85.33 respectively. No claim of identical HORIZON settings follows from these percentages.

B Historical Controls and Architecture Pilot

Under final evaluation, the earlier MAS scores 12.3% initially, 20.5% with checkpointing, and 26.0% with an additional instruction correction. These runs retain the shared-memory caveat described in Section 4.1. Checkpointing changes six paired outcomes positively and none negatively; adding both changes to the initial configuration changes twelve positively and two negatively. Exact paired tests are exploratory because the configurations were developed on the benchmark. The two grader-stop MAS runs differ on 14 task outcomes. One run is numerically close to the local Opus baseline; this is not evidence of statistical equivalence, equal-cost performance, or a general MAS advantage.

Component removal gives additional, mixed observations. Removing the verifier reduces the recorded score from 41.1% to 30.1%; removing the skill store gives 35.6%; removing diff conditioning leaves 41.1%; a local best-of-three selector gives 38.4%. On checker tasks, removing the severity contract gives 11.1% versus 38.9%. These limited repeats do not establish stable individual causal effects. In particular, deterministic lint must not be interpreted as a sound semantic verifier.

B.1 Matched Backbone and Architecture Pilot

The rebuilt kernel uses the relay identifiers `gemma-4-31b` and `claude-opus-4-6`. These are distinct from historical labels. Each backbone runs both single-context and MAS configurations with the same tools, inputs, and extended episode ceilings. Tasks come from the previously specified eight-task development selection, two distinct families per category. They are already explored development tasks, not a newly untouched test set. This earlier pilot is distinct from the specification-only version-2 study below.

On the four-task development sample, the rates in Table 5 are descriptive; neither model parity nor a stable architecture gain follows from this sample. The complementary four-task batch is incomplete

Table 6: Verification results (percent). Gray: CVDP-based references; teal: our Gemma configurations. Bold: column maxima within each harness group, including ties. Published and local rows differ in release and protocol; local agentic runs use grader-stop feedback and unequal budgets. Totals cover the four categories; – means unreported. The two historical Gemma runs are retained for provenance.

Harness	Backbone / variant	Stim.	Check.	Assert.	Repair	Total
Selected task cohort: published CVDP reference						
CVDP Official Harness	Sonnet 3.7	7.0%	0.0%	19.0%	53.0%	–
	Sonnet 3.7 + thinking	7.0%	1.0%	**28.0%**	**56.0%**	–
	Haiku 3.5	2.0%	2.0%	10.0%	55.0%	–
	GPT-4.1	4.0%	**13.0%**	13.0%	45.0%	–
	o1	**8.0%**	3.0%	10.0%	36.0%	–
	o4-mini	6.0%	7.0%	16.0%	38.0%	–
	Llama 3.1 405B	**8.0%**	6.0%	14.0%	45.0%	–
	Llama 3.1 70B	6.0%	6.0%	6.0%	45.0%	–
Selected task cohort: complete local runs						
CVDP + local agent	Haiku 4.5	0.0%	0.0%	32.1%	72.7%	23.3%
	Sonnet 5	12.5%	5.6%	**50.0%**	54.5%	31.5%
	Opus 5	**25.0%**	**33.3%**	42.9%	**81.8%**	**42.5%**
VeriEvolve	Gemma, run 1	**25.0%**	**38.9%**	**50.0%**	45.5%	**41.1%**
	Gemma, run 2	**25.0%**	**38.9%**	25.0%	**54.5%**	32.9%

after relay concurrency/rate-limit and daily-token-limit responses. It has no final benchmark score and is excluded from the table, rather than counted as solver failures. The first four-task batch was generated serially. The complementary batch initially used ten worker processes, then resumed with two after relay transport limits. Each worker has at most one in-flight model call; ten remains the overall cap. All scoring remains serial. The underlying controller and evaluator are unchanged; parallel resource contention may nevertheless affect wall-time pressure. Completed unsuccessful solver outcomes are retained; transport interruptions are archived with their known usage and restarted only before final scoring. Results are descriptive at this sample size, and local `supported` reports are not substituted for benchmark outcomes.

Mechanism activity. In the first completed batch, neither Gemma arm registers a preserved check; Opus registers three in single-context mode and two in MAS mode. None of the eight MAS episodes enters the conditional repair stage. Thus these outcomes exercise role decomposition and artifact construction, but provide no observed repair-stage gain. Improving the production of actionable, executed challenge findings is a concrete target for the next harness intervention. Merely making a repair capability available does not establish that the agent uses it.

C Published Baseline Provenance

Table 6 reproduces the four verification-category columns from Table 3 of Pinckney et al. (2025). The original paper's *Overall* column also includes other RTL generation categories, so it cannot serve as an aggregate for our four-category study. We leave unreported four-category totals blank rather than infer exact counts from rounded published percentages. The published agentic subset contains 77 tasks in these categories; the local v1.1.0 release contains 73. The published protocol uses one response per sample, with five samples estimating pass@1. The published table contains no Opus row. Our newer Claude results come from separately recorded local runs and retain their saved model identifiers.

D Development Search Reproduction

The two main development archives contain an ancestor and 24 mutated genomes each. Their saved node results were recounted to check fitness. All eight development tasks are stimulus tasks from the non-agentic split, transformed by the benchmark's own conversion. They are distinct from the eight-task, four-category selection in the rebuilt-kernel pilot. The mutable files are one shared system instruction and four category verification procedures. The modifier can read development pass/fail and selected grader-log excerpts; the inner solver's information boundary is different from the modifier's developmental feedback.

The genome was initialized with an intentionally restored problematic time-unit instruction. The two conditions differ in the modifier's falsification instruction and prompt-length constraint. Raw fitness and prompt length are reconstructed directly from archive nodes in Figure 4. No direction-of-instruction regex is used to establish the plotted result, and no single selected candidate is treated as a held-out evaluation.

E Pilot Reproduction and Accounting

Table 7: Pilot pass rates by category, with one task per category and configuration.

Configuration	cid012	cid013	cid014	cid016
Gemma single	0.0%	0.0%	0.0%	100.0%
Gemma MAS	0.0%	0.0%	100.0%	100.0%
Opus single	0.0%	0.0%	100.0%	100.0%
Opus MAS	0.0%	0.0%	100.0%	100.0%

The complete first batch contains one stimulus, checker, assertion, and RTL-repair task. The preselected complementary batch adds one other family per category when complete. All tasks in a batch finish generation before scoring. The batch seal binds manifest, public contracts, code, datasets, and candidate files. The grader is invoked in a separate process with model credentials removed. The parallel wrapper records its own hash and an operational amendment; it does not modify the hashed solver or evaluator. Each grading attempt has a persistent invocation marker so an interrupted process cannot silently trigger a second evaluation.

Successful responses from interrupted generation are kept in the accounting archive. An interrupted in-flight response may not have a returned usage record; known totals are not claimed to reconcile exactly to a provider bill. Cached and uncached input accounting can also differ across endpoints. We report provider-returned input and output counts without converting them to an unverified common dollar rate. The first batch additionally records 147,457 tokens from eleven successful responses in the interrupted September 10 attempt; those are separate from the completed-attempt totals in Table 5.

F Executable-Study Task Selection

The earlier specification-only study uses eight development tasks and sixteen transfer tasks, balanced by population and category. It removes separate RTL/templates while retaining code embedded in the prompt. Sonnet 5 proposes three policies per search condition with 16,384 output tokens per proposal and identical bounded public-trace feedback. Rejected proposals count against the search allowance; identical policies reuse evaluations. Selected policies freeze for Gemma, Sonnet 5, and Opus 5 transfer. Instruction-only development fitness improves from 0.0% to 25.0%, whereas generic and EDA-guided conditions select the ancestor because no executable child reaches evaluation. Of 96 unique transfer episodes, 85 produce complete artifacts. Reactions comprise 84 deferrals and one recheck, with no repair. Source files are removed from twelve of sixteen transfer tasks, including all repair-task RTL. Historical full-population runs preclude treating these families as globally unseen. Table 9 lists the exact development and transfer tasks. The split is stratified by source population and category. Design families are identified by normalized task-name stems and explicit aliases; this procedure does not establish independence of all underlying circuit concepts.

Table 8: Transfer pass rates with specification-only inputs: eight tasks per population, sixteen total. MAS solves both populations. † Generic and EDA search select the fixed ancestor and share its outputs. Tokens sum provider-reported input and output for completed attempts; transport overhead is separate.

Harness / policy	Backbone	Success	Tokens (M)
VeriEvolve: fixed†	Gemma	0.0%	2.48
VeriEvolve: prompt-only	Gemma	6.3%	2.25
VeriEvolve: fixed†	Sonnet 5	12.5%	5.85
VeriEvolve: prompt-only	Sonnet 5	12.5%	6.48
VeriEvolve: fixed†	Opus 5	12.5%	7.27
VeriEvolve: prompt-only	Opus 5	12.5%	6.92

Comparing the two public-input projections shows that the restricted context removes nine files from five of eight development tasks and seventeen files from twelve of sixteen transfer tasks. All six RTL-repair tasks across these splits lose supplied RTL. The original prompts and deliverable names remain unchanged, including any code embedded directly in the prompts. These facts establish an input difference; they do not isolate its contribution to the observed low scores.

G Implementation Checks

The test suite includes read-only candidate mounts, scoped edit authority, path traversal and symlink rejection, timeout handling, execution-record freshness, frozen regression replay, full-tree restoration, report reservation, and retention of completed solver failures during resume. Campaign tests check that all arms finish before grading and that model credentials are removed from grader processes. Additional process tests verify the parallel worker cap and stopping new jobs after worker failure. Twenty-nine tests passed before the earlier parallel continuation. Version-2 tests additionally exercise actual Verilog target resolution, preservation and interruption rollback, policy-dependent branching, cross-population family separation, per-provider process caps, and native signed-thinking/tool-result replay. Fifty-eight relevant tests passed before the corrected native-transport study was launched.

The initial version-2 launch stopped when the adapter rejected signed thinking blocks on eight initial responses, before any agent tool action or grader invocation. Its provider usage was not retained and is unknown, not zero. After a native roundtrip test, a separately frozen campaign restarted with the same split, initial policy, and budgets. The failed attempt remains archived and is not counted as task failure. An outer-loop pilot also exhausted its 4,096-token proposal allowance before completing policy submission. A separately recorded continuation uses shared feedback compaction and a 16,384-token allowance for every mutation condition. It reuses the sealed ancestor only after checking identical inner-solver code, models, contracts, budgets, and score hashes; the ancestor is neither regenerated nor regraded. Known pilot usage and unknown usage from an interrupted request remain separate accounting items. Additional compaction and reuse checks bring current relevant test coverage to sixty passing cases.

The completed three-proposal search selected prompt-only development fitness of 25.0%, versus the shared ancestor's 0.0%. Both passes are auxiliary tasks (stimulus and RTL repair). Generic and EDA-guided search retain the ancestor. Five executable-search proposals returned the stage shares as named dictionaries instead of the required arrays. Their numeric allocations satisfy the budget constraints, but the parser rejects the representation. The only admitted generic child changes instructions alone. No evaluated child therefore tests an executable policy change; this is a proposal-interface failure, not evidence against the EDA-guidance hypothesis. The rejected proposals remain in the archive.

Transfer generation stopped after two Gemma relay rate-limit errors; eight completed Sonnet outcomes were retained without grading. A recorded September 17 recovery preserves all completed outcomes, including failure to produce a complete candidate, and archives both interrupted attempts with their returned usage. Official-provider jobs ran at at most eight workers, followed by serial relay episodes. Neither policies nor solver budgets change, and all 96 unique transfer episodes must freeze before grading starts. Four additional recovery tests pass, checking retention, failed-attempt archival,

Table 9: Frozen task selection for the executable-policy study. This auxiliary cohort draws equally from both CVDP source populations; MAS solves every task. Development and transfer families are separated using normalized design names and the aliases recorded in the study manifest.

Category	Task ID
Development	
cid012	`cvdp_agentic_async_fifo_compute_ram_application_0004`
cid013	`cvdp_agentic_image_stego_0007`
cid014	`cvdp_agentic_crypto_0001`
cid016	`cvdp_agentic_caesar_cipher_0001`
cid012	`cvdp_copilot_afi_ptr_0004`
cid013	`cvdp_copilot_sequencial_binary_to_one_hot_decoder_0004`
cid014	`cvdp_copilot_moving_average_0025`
cid016	`cvdp_copilot_coffee_machine_0001`
Transfer	
cid012	`cvdp_agentic_direct_map_cache_0005`
cid013	`cvdp_agentic_bcd_to_excess_3_0011`
cid014	`cvdp_agentic_memory_scheduler_0005`
cid016	`cvdp_agentic_custom_fifo_0004`
cid012	`cvdp_copilot_traffic_light_controller_0007`
cid013	`cvdp_copilot_8x3_priority_encoder_0018`
cid014	`cvdp_copilot_icache_controller_0004`
cid016	`cvdp_copilot_grayscale_image_0014`
cid012	`cvdp_agentic_bcd_adder_0006`
cid013	`cvdp_agentic_64b66b_codec_0007`
cid014	`cvdp_agentic_sorter_0007`
cid016	`cvdp_agentic_AES_encryption_decryption_0005`
cid012	`cvdp_copilot_sram_fd_0024`
cid013	`cvdp_copilot_reverse_bits_0006`
cid014	`cvdp_copilot_lfsr_0073`
cid016	`cvdp_copilot_radix2_div_0001`

Table 10: Development-archive results. Each main arm evaluates 24 mutations from an ancestor scoring 12.5%. Maxima are selected development fitness, not held-out success or a replicated estimate.

Search condition	Mutations	Ancestor	Best observed
Plain DGM-style	24	12.5%	12.5%
Falsification + length cap	24	12.5%	25.0%
Earlier exploratory run	8	12.5%	12.5%

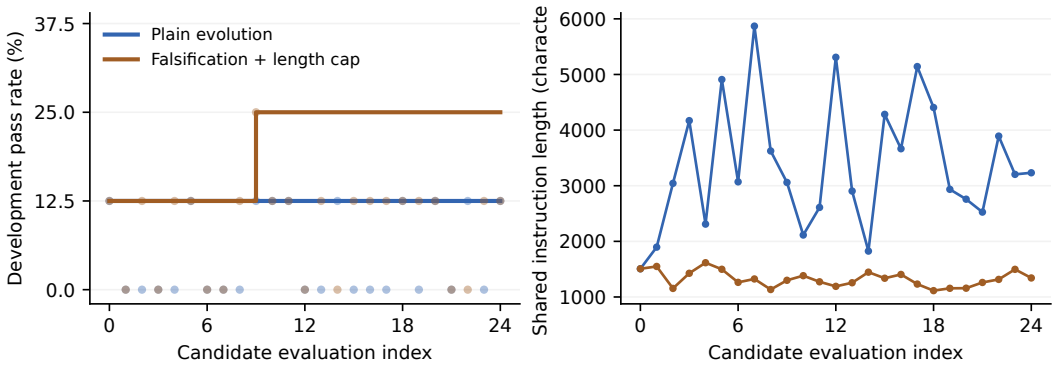


Figure 4: **Observed harness search.** Candidate fitness and best-so-far development fitness are reconstructed from the two saved archives; the horizontal axis is evaluation order, not a single parent lineage. Prompt lengths describe successive candidates. Both campaigns reuse eight development tasks, so the curves measure search behavior rather than generalization.

identity validation, and the complete batch requirement. Transport recovery does not correct the search interface; a corrected interface requires a separate search version. Subsequent September 21 recoveries archive relay read timeouts while retaining all completed outcomes. The final recovery retains 94 completed episodes and finishes the remaining two before sealing all 96 unique transfer episodes. All 96 then receive final outcomes. No successful or unsuccessful completed solver outcome is regenerated during recovery.

These tests establish behavior on the checked cases. They do not prove semantic correctness of the generated verification artifacts, adequacy of the experiment library, correctness under every tool failure, or generalization of any evolved harness.

H Instruction-Level Search Results

Both main campaigns use the same eight stimulus tasks from the non-agentic split, converted through CVDP's own agentic transformation. This is separate from the four-category pilot below. The campaigns expose a limitation of instruction/procedure search (Table 10). Plain evolution does not improve on the ancestor. The constrained variant finds a higher-scoring candidate, but the absolute gain is one development task and lacks a held-out transfer evaluation. Figure 4 shows all evaluated candidate fitness values and the selected best-so-far envelopes.

The development study was initialized with a known problematic instruction about time-unit directives. This is a diagnostic stress test of the modifier, not an unbiased sample of harness defects. Log interpretation, prompt length, and procedure changes can be inspected in the saved lineages, but these campaigns do not isolate a universal cause of self-improvement failure. They motivate testing concrete interventions on artifacts before committing a general instruction.

What changed in the selected candidate? The constrained search first reaches 25.0% at candidate 9, a child of candidate 1. Its patch prioritizes writing the required deliverable and changes the stimulus-verifier procedure to check the output path before running a reference experiment. Its saved outcomes pass the GCD and Hill cipher tasks, while the ancestor passes only the latter. The patch

also retains a blanket time-unit instruction and changes multiple rules at once. This lineage supports a concrete hypothesis about artifact-completion policy; isolating that hypothesis requires replaying the individual changes.

I HORIZON Comparison and Cumulative Completion

HORIZON reports 100.0% best-so-far completion in CVDP categories cid012, cid013, cid014, and cid016 using a fixed GPT-5.3 backbone and task-native evaluator feedback (Yu et al., 2026). Its Table 1 reports 32, 19, 1, and 13 outer iterations respectively. Its iteration-zero scores are already agent-loop outcomes, not standalone model pass@1. Table 11 distinguishes these published results from our existing local HORIZON-style implementations and cascade runs.

Table 11: HORIZON references and historical local records. Cohorts and feedback protocols differ; equal percentages do not establish a matched reproduction. – means unavailable, not a failure rate.

System	Recorded setting	Success
HORIZON (reported)	Original four CVDP categories; evaluator feedback	100.0%
HORIZON-style (local)	Ten converted assertion tasks; Opus 5	100.0%
Our historical cascade	Same ten-task subset; two API errors retained	70.0%
HORIZON-style smoke test	Two native agentic tasks; Opus 5	100.0%
Our cascade + escalation	Full local native agentic verification cohort	100.0%
HORIZON, final-only	No verified result in the located records	–

Existing HORIZON-Style Runs. The archived two-arm runner uses the same ten converted assertion tasks for a single Opus 5 agent and our historical cascade. The HORIZON-style arm allows 40 turns and 1,200 seconds per task, accepts intermediate benchmark success, and otherwise grades the final artifact. Its saved success rate is 100.0%; the cascade records 70.0%, including two API errors in the denominator. A separate native-agentic smoke test records 100.0% on two tasks; its trajectories contain three and ten grader evaluations. These are local reimplementations with different backbones and infrastructure from the published framework. Neither record is a final-only evaluation of HORIZON.

Our Cumulative Completion. The historical multi-model cascade completes 90.4% of native agentic tasks in its initial campaign. Escalation is applied to exactly the remaining seven IDs and completes all seven, yielding 100.0% cumulative completion. The winning initial tiers are Haiku for 27 tasks, single-route Sonnet for 29, and three-route Sonnet for 10. Escalation contributes four longer-budget Sonnet successes and three Opus successes. This is a meaningful engineering completion result, but not a fixed-model, single-attempt, or matched-budget score. The saved cost fields sum to $177.33 across both stages; historical harness visibility is not fully certified, and these fields are not independently reconciled billing. Thus, our historical pipeline also reaches 100.0% under a cumulative completion protocol, but identical task sets, feedback access, model, and budgets with HORIZON have not been established. The available records do not justify describing this as an identical-setting reproduction.

Strict Evaluation Boundary. In the rebuilt final-only protocol, the solver receives no hidden grader feedback during generation; the frozen final candidate is evaluated only after generation. Specification-only input projection is a separate restriction on public context. We have not located a verified HORIZON run satisfying this final-only boundary, so its score is left unavailable. Neither the 70.0% cascade pilot nor our separate restricted-input MAS results are substituted for a strict HORIZON score.

J Earlier Instruction-Level Implementation

The historical implementation adapts DGM's archive-and-mutation approach (Zhang et al., 2025). Its genome contains a shared system instruction and four category verifier procedures. Parent selection balances observed fitness with offspring count; the modifier receives development outcomes, selected

grader-log excerpts, and lineage. A search variant adds a prompt-length cap and an instruction to reverse unproductive rules. These historical campaigns change text only and have a different feedback boundary from version 2. Their results cannot be attributed to the new executable policy space or runtime guarantees.

K Isolated Proposal-Interface Diagnostic

A subsequent diagnostic preserves the completed study and its policy selection. A deterministic canonicalization converts exactly named stage, action, and role-note objects to their declared array order without changing values. All five previously rejected executable proposals satisfy the original policy constraints after this conversion. Seven tests check order preservation, missing and additional names, numeric constraints, prohibited repairs, note bounds, prompt-only restrictions, and incomplete or duplicate tool calls.

Three additional Sonnet 5 calls expose the policy as a structured tool argument rather than an encoded JSON string, using the same ancestor and development feedback. The prompt-only and EDA proposals are admitted; the EDA proposal changes executable fields. The generic proposal returns an encoded string and omits its required rationale, and is rejected. These are interface observations only: the new proposals have no task scores and were not substituted into the completed transfer study. A schema declaration alone did not eliminate malformed model outputs.

A further three-call diagnostic enables the provider's strict tool mode and uses exactly named policy fields. Runtime validation still enforces the original numerical constraints, note bounds, and repair permissions. All three proposals are admitted; generic and EDA proposals change executable fields, while the prompt-only proposal preserves them. Three additional tests check the actual request's strict-mode flag, provider-supported schema features, and prohibited repair actions. These six live calls across two interface variants are diagnostic samples, not a replicated estimate of proposal reliability or benchmark performance. The resulting policies require a separately frozen solver evaluation.

L Historical Implementation Boundaries

Reproduced counterexamples in the earlier implementation show that removing an explicit write tool did not prevent writes through the shell, lint could accept comments containing termination tokens, and rollback could retain newly added files. These counterexamples establish implementation limits, not their frequency in historical benchmark episodes. The rebuilt kernel tests read-only execution mounts, scoped edits, and full-tree restoration. Its checks do not retroactively validate earlier outcomes or certify behavior outside the tested cases.

M Resource Accounting and Budget Controls

Table 12: Recorded resources for each unique transfer configuration. Each row covers sixteen tasks. Episode minutes sum solver elapsed time; they are not parallel makespan or provider billing.

Backbone	Policy	Calls	Input (M)	Output (M)	Episode min.
Gemma	Fixed	371	2.39	0.10	33.3
Gemma	Prompt-only	283	2.09	0.16	41.3
Sonnet 5	Fixed	378	5.41	0.43	74.3
Sonnet 5	Prompt-only	389	6.00	0.47	80.9
Opus 5	Fixed	411	6.60	0.67	143.8
Opus 5	Prompt-only	368	6.29	0.63	131.5

Provider-reported input and output counts measure processed tokens. They do not directly measure monetary cost because endpoint pricing, caching, and billing conventions differ. Table 12 retains the two token components separately. Episode elapsed time includes model latency and local execution; summing it across workers does not recover parallel wall-clock duration. Known usage

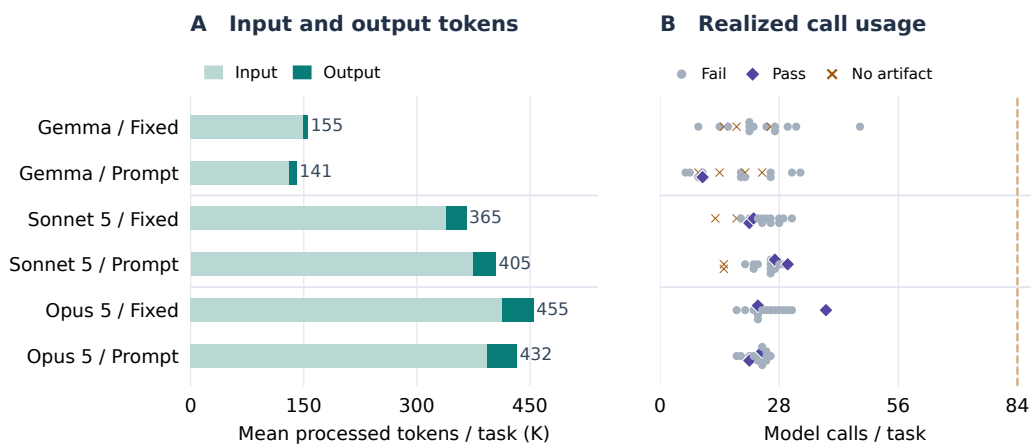


Figure 5: **Recorded inference use in the completed specification-only study.** (A) Mean input and output tokens per task. (B) Each point represents one transfer episode; crosses mark missing artifacts. The dashed line marks the 84-call ceiling. Fixed, generic, and EDA-selected policies share the same frozen outputs, which are counted once. This plot describes realized usage under a common ceiling; it is not a budget-scaling curve.

from interrupted attempts stays in a separate transport archive. An interrupted request can lack a returned usage record, so the accounting does not claim exact bill reconciliation.

Figure 5 shows variation in realized calls despite a shared 84-call allowance. Early role completion, token constraints, wall-time constraints, and missing diagnostic evidence can all limit execution. Comparing high-use and low-use tasks would confound task difficulty with compute. A scaling experiment instead assigns different ceilings to the same task set while fixing the model, input, and policy.

M.1 Registered Development Controls

The September 23 pilot uses the same eight development tasks as the restricted-input study, balanced across the two populations and four categories. The main input condition restores the original public context; a paired specification-only condition uses the earlier projection. Both conditions hide grader source, grader logs, and reference answers. Gemma and Sonnet 5 run separate registered campaigns with a fixed initial policy under $1\times$, $2\times$, and $3\times$ resource ceilings. The $1\times$ budget allows 28 responses, 32,000 output tokens, 200,000 processed tokens, and 600 seconds. The other budgets multiply all four ceilings by two or three. This measures scaling with a combined episode allowance; it does not isolate token count from elapsed time.

A single-agent control receives the same original public inputs, base file and simulation tools, backbone, and $3\times$ ceiling. It combines specification analysis, construction, and challenge in one context. The MAS comparison measures the complete role-and-control organization, rather than attributing an outcome to one isolated component. The specification-only MAS control also uses $3\times$. Generation order is shuffled with a recorded seed, and all five configurations finish generation before any matrix score. Completed failures remain in the denominator; transport interruptions stop the phase and preserve known usage. Gemma initially used two concurrent requests; recorded recovery reduced this to one after read timeouts and subsequently increased the ceiling to four. Interrupted jobs were moved behind untouched jobs, and explicit HTTP 429 rejections received bounded backoff within the original request timeout. Sonnet uses four concurrent requests. These operational differences can affect wall-time pressure and remain a limitation of the development pilot.

A subsequent intervention pilot gives instruction-only, generic, and EDA-guided modifiers one typed-policy proposal each from the common $3\times$ ancestor. All proposals use the same development traces, scalar outcomes, and output allowance. Admission checks preserve the policy's budget and evidence

constraints. A strictly higher development pass count replaces the parent; ties retain it. Identical policies reuse the same evaluation. All selected policies then freeze for the sixteen family-separated transfer tasks. Within each campaign, the three modifiers use the solver backbone. The two campaigns test replication across backbones with separate searches; they do not measure transfer of one learned policy between backbones.

Measurements and Remaining Ablations. For each budget and control, the primary quantities are final pass rate, paired outcome changes, token use, model calls, artifact completion, and executed repair/recheck decisions. A valid executable proposal must show which fields changed and which execution events exercised those changes. Follow-up controls should fix stage allocation while allowing reaction changes, and fix reaction rules while allowing allocation changes. Repeated independent searches and paired transfer runs are needed to estimate variability. A preservation-guard ablation is informative only once enough episodes enter repair; otherwise it compares inactive code paths. Pending experiments supply no numerical scaling or ablation claim.

M.2 Control-Study Results

Table 13: Matched development controls. Each row covers the same eight tasks, four per population. Tokens and median calls include all completed outcomes; interrupted-attempt costs remain separate. Budgets scale calls, tokens, and time together.

Backbone	Configuration	Pass rate	Mean tokens (K)	Median calls
Gemma	MAS, 1×	25.0%	85.1	14.5
Gemma	MAS, 2×	25.0%	167.1	22.0
Gemma	MAS, 3×	12.5%	235.6	29.5
Gemma	Single agent, 3×	12.5%	73.7	8.0
Gemma	MAS spec-only, 3×	0.0%	181.0	23.5
Sonnet 5	MAS, 1×	12.5%	146.9	13.5
Sonnet 5	MAS, 2×	37.5%	289.9	21.0
Sonnet 5	MAS, 3×	37.5%	410.4	24.5
Sonnet 5	Single agent, 3×	37.5%	396.8	17.5
Sonnet 5	MAS spec-only, 3×	0.0%	390.4	23.5

Figure 3 visualizes these controls in the main text. These controls use previously explored development tasks. Architecture differences compare role organization and control together; input differences compare public-context and specification-only projections. Repeated rollouts and frozen transfer are required before asserting a stable general advantage.

Table 14 retains the development difficulty breakdown with its stratum sizes. The eight-task development cohort contains only three Easy and five Medium designs, so these results do not establish a population-level ordering of difficulty. The auxiliary cohort is retained to document the original policy selection rather than to replace the complete primary evaluation.

N Completed Typed-Policy Intervention

The Sonnet campaign completes both proposal evaluation and frozen transfer. Table 4 in the main text summarizes the admitted instruction and executable-policy interventions. The generic proposal changes stage shares from $(10, 40, 25, 20, 5)$ to $(20, 30, 25, 20, 5)$ and reorders diagnostic priorities, placing tool limitations first. Its action map is unchanged. Instruction-only search changes role notes without changing executable fields. Both children tie the ancestor at 37.5% development success. The EDA-guided response is rejected for unknown or missing policy fields; this is an admission failure rather than a measured solver failure. Strict-improvement selection retains the ancestor in all conditions. Its 43.8% transfer score therefore represents one policy evaluated once per design.

Completed solver attempts consume 13.075M processed tokens for the development matrix, 7.193M for child evaluation, and 6.451M for transfer. Modifier calls and interrupted attempts are accounted separately. The archive retains frozen candidates, selected-policy hashes, and phase seals. The

Table 14: Auxiliary development controls by difficulty (%). Each backbone uses the same eight designs drawn equally from both CVDP source populations: three Easy and five Medium. No Hard designs occur. These pilot scores are separate from Table 2.

Architecture	Backbone	Easy	Medium	Total
Single agent	Gemma	33.3%	0.0%	12.5%
VeriEvolve	Gemma	33.3%	0.0%	12.5%
Single agent	Sonnet	33.3%	40.0%	37.5%
VeriEvolve	Sonnet	33.3%	40.0%	37.5%

Table 15: Auxiliary matched Sonnet transfer control on sixteen designs drawn equally from both CVDP source populations. Inputs and 3× budget ceilings are identical across architectures. This earlier pilot is separate from the primary 73-design evaluation.

Architecture	Success (%)	Mean tokens (K)
Single agent	31.3%	307.8
VeriEvolve	43.8%	403.2

reporting script verifies those hashes and matches each saved score to its summary before generating the table.

A follow-up single-agent evaluation was registered after inspecting these results, using the same Sonnet backbone, sixteen transfer designs, public contracts, fixed policy instructions, and 3× allowance. It changes the execution architecture without tuning the policy. Generation order uses seed 20260924 and all candidates seal before scoring. The completed control scores 31.3%, compared with the retained MAS policy's 43.8%. Four paired designs favor MAS and two favor the single agent; the exploratory two-sided exact discordance test gives $p = 0.688$. Mean processed tokens are 403.2K for MAS and 307.8K for the single agent. This is a subsequent paired control, not a new untouched test set or evidence of a statistically established general advantage.

Difficulty Coverage. The development and transfer cohorts contain three/nine Easy and five/seven Medium designs, respectively. Difficulty labels come directly from the benchmark categories and are joined to saved task IDs; no outcome-based relabeling is used. The full four-category release contains 142 Easy, 142 Medium, and 11 Hard designs. All Hard designs belong to the Agentic population. A separately registered diagnostic evaluates all eleven with the fixed Sonnet policy in MAS and single-agent modes at the same 3× ceiling. It uses four workers and seed 20260924. The completed MAS and single-agent runs score 27.3% and 36.4%, respectively. Some design families overlap earlier development, so this diagnostic is not an untouched transfer set and is reported separately from the original sixteen-task cohort.

Table 16: Separate all-Hard Sonnet diagnostic: eleven Agentic designs, equal 3× allowances, one episode per configuration and design. This post-hoc cohort can overlap earlier development families; it is not pooled with the original transfer cohort.

Architecture	Success	Mean tokens (K)
Single agent	36.4%	514.0
VERIEVOLVE	27.3%	425.5

Observed Controller Activity. Table 17 audits the completed fixed-policy cohorts. In the development, transfer, and separate Hard cohorts, every episode that reaches the reaction decision defers; one Hard MAS episode produces no complete candidate. The complete 73-design evaluation cohort admits one diagnosis and executes one recheck, but no repair. Registered regressions alone do not activate repair: a diagnosis and a reproducible failure target are also required. The conditional preservation argument below describes the guard's behavior when invoked, not an observed accuracy contribution from repair in these runs.

Table 17: Observed MAS mechanism activity (percent of episodes). Candidate means complete required deliverables; checks means at least one registered regression; diagnosis means at least one admitted diagnosis; repair means an executed repair. All rows use the fixed policy at 3×. These are activity measurements, not component-effect estimates.

Cohort	Tasks	Candidate	Checks	Diagnosis	Repair
Gemma development	8	100.0%	0.0%	0.0%	0.0%
Sonnet development	8	100.0%	37.5%	0.0%	0.0%
Sonnet transfer	16	100.0%	50.0%	0.0%	0.0%
Sonnet Hard diagnostic	11	90.9%	0.0%	0.0%	0.0%
Sonnet full Agentic	73	100.0%	28.8%	1.4%	0.0%

Across the completed Sonnet development, proposal-evaluation, and transfer campaign, agents make 24 diagnosis requests. Seven quote literal public specification text but use an unsupported source label. The tool exposes the source as an unconstrained string, whereas the validator expects `prompt` or a public documentation path. A separately registered development comparison tests explicit source choices in the tool schema, holding the evidence validator, fixed policy, backbone, tasks, and budget constant. This fresh eight-task comparison scores 25.0% with the original interface and 37.5% with explicit source choices. The original arm makes 14 diagnosis requests, including two literal quotations with invalid source labels; the explicit-source arm makes no diagnosis requests. Neither arm admits a diagnosis or executes repair. Thus, the one-task success difference does not establish improved diagnosis or evolution; this is a small post-hoc interface diagnostic.

Matched Evaluation on the Selected Cohort. The Sonnet comparison includes all 73 selected designs, with one fresh single-agent and one MAS episode per task under identical public inputs and 3× allowances. Code, policy, task order, and category blocks freeze before generation. Every block seals before its first external grade, and no score enters solver messages. These runs include previously studied tasks and are not described as untouched holdouts. Transport interruptions retain completed candidates and known usage in recorded recovery amendments. Bounded recovery retries verified transport errors, not completed unsuccessful solutions, and does not select using grades. All four selected category blocks are complete and sealed. VeriEvolve scores 26.0%, compared with 23.3% for the matched single agent. Seven paired designs favor MAS and five favor the single agent (exact two-sided discordance $p = 0.774$). Mean processed tokens per task are 431.1K for MAS and 402.7K for the single agent, excluding separately archived transport interruptions. Table 3 gives the matched architecture comparison; Table 2 includes its VeriEvolve result. The historical Sonnet Agentic baseline (31.5%) used periodic grader consultations and stopped upon a passing intermediate candidate. This selection differs from final-only scoring, even when grader files are hidden from the agent. Its difference from 23.3% cannot be attributed solely to model capability or any one protocol change.

O Evidence-Conditional Preservation Argument

Equation 3 describes the outcome of one guarded repair, conditional on three implementation assumptions: the experiment sources remain fixed during replay; the recorded pass/fail predicates faithfully reflect those executions; and rollback restores the complete captured candidate. The nonempty regression set is frozen before the repair is attempted. No assumption asserts that these checks fully characterize the specification.

Case Argument. If the repair gate accepts, the triggering negative experiment resolves and every preserved check passes by the gate's conjunction. The returned artifact is therefore in the second set of Equation 3. If the gate rejects or execution is interrupted while repair is pending, rollback restores W^-, placing the result in the first set. Subsequent final review cannot modify the candidate. These cases cover the implemented repair transition. The argument applies to any admissible policy because policy search cannot change the gate, check replay, or rollback. It does not establish that the negative experiment is a correct oracle, that the baseline is correct, or that an accepted policy improves expected transfer utility.

Selection and Generalization. Let $\widehat{J}_n(\phi)$ be the saved development pass rate used by search. Replacing an incumbent only when a child's saved score is strictly higher ensures nondecreasing selected empirical fitness. This is a property of the recorded selection rule, not of fresh rollouts or the population objective J_θ. Reusing a selected policy on separate design families addresses transfer; repeated searches are additionally needed to measure selection variability. The equal-fitness children in Table 4 illustrate why an executable mutation need not imply either a selected change or an improvement.