MEASURED BEHAVIOR, RECORDED ZERO: MEASUREMENT FAILURES IN SELF-IMPROVING CODING AGENTS

Anonymous authors
Paper under double-blind review

ABSTRACT

Self-improving coding agents evolve a scaffold around a frozen language model and select variants using benchmark performance. We study whether that selection signal reliably reflects what the agent and the search actually did. On the 59-task Polyglot subset used by the Huxley–Gödel Machine (HGM), the default self-hosted setup records zero tool use at 7B, 14B, and 32B, yet the 14B transcripts contain dispatchable tool calls on 30/59 tasks. Recovering those calls changes tool use from 0/59 to 32/59 while resolve rate remains 0/59. In two evolutionary runs, all 14 proposed children are invalid Python and are archived with the same utility (0.0) that a valid child solving no tasks would receive. Replacing the proposal model with a frontier hosted model changes validity from 0/14 to 14/14 while preserving the same destructive edit pattern. Across ten configurations, resolve rates remain between 0/59 and 4/59. These results do not show that scaffold evolution fails. They show that, in this deployment regime, the recorded objective can collapse distinct behavioral states and can therefore obscure whether a search step acted, failed, or produced an executable candidate. We provide a network-level measurement layer and a set of inexpensive controls for evaluating self-improving agents.

1 INTRODUCTION

Self-improving coding agents search over programs that orchestrate a frozen foundation model. Systems such as the Darwin–Gödel Machine (DGM) (Zhang et al., 2026a), the Huxley–Gödel Machine (HGM) (Wang et al., 2025), and Hyperagents (Zhang et al., 2026b) propose scaffold changes and retain variants according to benchmark performance. Reported gains are substantial, but interpreting them requires more than a final pass rate: the evaluation must reveal whether a candidate actually executed, whether its tools and feedback were available, and whether a change improved the frozen model or merely elicited behavior that was already reachable.

We examine this measurement problem directly. Our starting point is simple: a scalar benchmark score can only support selection to the extent that it is sensitive to the states that differ across candidates. We find two concrete cases in which it is not.

First, the HGM task agent produces no structured tool calls under the released default policy when run with self-hosted quantized Qwen models. At 14B, however, the transcripts contain valid tool-call JSON in the message body on 47/59 tasks, including 30/59 calls that name a real tool and provide all required arguments. Recovering these calls changes tool use from 0/59 to 32/59 tasks, but the benchmark score remains 0/59. The intervention therefore changes observed behavior without changing the quantity used for evaluation.

Second, the evolutionary path itself can produce candidates that are not executable. In two runs from the same seed, all 14 proposed children produce invalid Python. The archive nonetheless assigns every child `mean_utility: 0.0`, the same value assigned to a valid candidate that runs and solves no tasks. When we replace only the proposal model with a frontier hosted model, validity changes from 0/14 to 14/14 while the model still uses the same whole-file editor operation. The archive assigns the same utility in both cases.

These observations motivate a narrower claim than "self-improvement does not work." We show that the quantity used for selection can be uninformative about behavior in the regime we measure, and that failures in the execution and archival layers can be hidden by the same zero score. 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.

protocol failure is specific to the self-hosted, quantized deployment we tested; the archive ambiguity is a property of the bookkeeping path and does not depend on model scale.

Contributions. (1) We introduce a network-level measurement layer that attributes tool use and token consumption from intercepted model traffic rather than agent self-report. (2) We document a protocol failure in which a model produces dispatchable tool calls outside the structured field consumed by the harness; repairing the parser changes tool use from 0/59 to 32/59 without changing resolve rate. (3) We show that the evolutionary archive records invalid candidates with the same utility as valid zero-solving candidates, and verify that the validity failure is sharply reduced when the proposal model is replaced while the edit operation is held fixed. (4) Across ten configurations, we find no separation in resolve rate on the benchmark used for the main study. (5) We translate these findings into concrete evaluation controls.

2 Related Work

Self-improving coding agents such as DGM (Zhang et al., 2026a), HGM (Wang et al., 2025), Hyperagents (Zhang et al., 2026b), and SICA (Robeyns et al., 2025) evolve scaffolds around frozen models. These systems evaluate candidate scaffolds using task-level performance. HGM also compares against a human-engineered scaffold under matched budgets; our concern is different: whether the reported score is sufficient to attribute a gain to the evolving scaffold rather than to repeated sampling or changes in elicitation.

Recent work has examined self-evolution from complementary perspectives. EvoAgentBench (Gao et al., 2026) evaluates transfer of capabilities across model families, while SEA-Eval (Jiang et al., 2026) considers performance relative to token cost. Iscan (Iscan, 2026) decomposes the source of gains in a fixed self-repair mechanism. Work on tool-use failures, including ToolScan (Kokane et al., 2024), already identifies malformed or misplaced tool calls as a failure mode. Our contribution is to study the consequence of such errors inside a self-improving loop: a behavior that is materially different at the trajectory level can be collapsed to the same selection statistic.

This question is also related to elicitation. Scaffold choice can substantially change measured capability without changing model weights (Starace, 2026), and capability estimates generally depend on how many opportunities a model receives to act (Kwa et al., 2025). Our measurement layer is aimed at the preceding question: before attributing an improvement, can the evaluation pipeline observe the behavior that produced the score?

3 Measurement and Setup

Task and grading. We use the 59 unique instances in HGM's Polyglot baseline subset, spanning C++, Go, Java, JavaScript, Python, and Rust. We use HGM's own grading harness: hidden tests are applied only after the agent stops, and a task is resolved only when the hidden tests pass.

Models and serving. The main experiments use Qwen2.5-Coder-7B-, 14B-, and 32B-Instruct plus Mistral-Small-24B-Instruct. Models are self-hosted with 4-bit AWQ quantization and served by vLLM. The 7B and 14B arms use a 16,384-token context; the 32B arm uses 6,144 tokens because the model does not fit the same memory budget at 16k. The evolutionary runs use a 32,768-token context because HGM's diagnosis prompt does not fit 16k.

Measurement layer. A reverse proxy between the agent container and model server records every request and response, tool-call fields, and token counts. We classify operations from wire-level traffic rather than instrumentation inside the agent. We also retain the raw trajectories, patches, and per-arm provenance so that headline counts can be recomputed.

Baseline and controls. The main comparison uses HGM's released default `tool_choice="auto"`. A paired `required` arm constrains decoding to the tool-call grammar and is used only as a diagnostic. A lenient-parser arm recovers tool calls from message text. We additionally test model scale, model family, a second benchmark, a second self-improving system, a matched-compute best-of-3 baseline, and two evolutionary runs from a repaired seed.

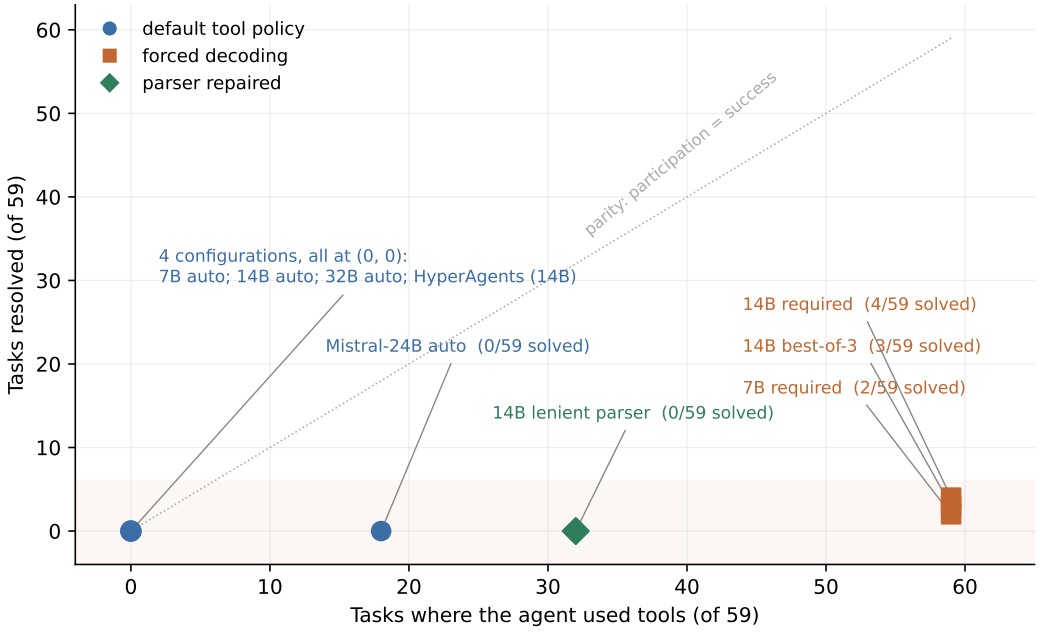


Figure 1: Task-level tool use versus resolution across the configurations in the main study. The interventions move tool participation over a wide range, while resolve rate remains close to the floor.

4 Results

Default evaluation collapses distinct behaviors. Under HGM's unmodified default policy, the 7B and 14B agents resolve 0/59 tasks, issue 0 structured tool calls, and return empty patches on every task. Each task uses one model generation and then terminates. Viewed only through these outputs, the two model sizes appear identical.

The stored trajectories tell a different story. At 7B, only 5/59 transcripts contain anything resembling a tool call, and just one is well formed. The 14B model, however, emits syntactically valid JSON in the assistant message body on 47/59 tasks. In 38 cases the JSON parses, and in 30 cases it names a real tool and supplies every required argument. None of these calls appears in the structured `tool_calls` field consumed by the harness. Thus the same recorded state, (resolve rate, tool use) $= (0, 0)$, contains at least two qualitatively different behaviors: a model that usually does not attempt the operation, and a model that often attempts it but uses a representation the harness ignores.

This distinction matters because the selection loop never observes the discarded behavior. If two scaffold variants induce these two trajectories, a search based only on the recorded benchmark score receives the same feedback from both. The problem is not that the benchmark is noisy in the ordinary statistical sense. The relevant information is absent from the statistic being optimized.

Checks on the protocol failure. Before interpreting the missing calls as an artifact, we tested whether the model had actually been told how to call tools. The model-specific chat template renders the tool schemas and includes the requested `<tool_call>` convention. In a single-request contrast, the same prompt under `tool_choice="auto"` yields tool-call JSON in the message body while `tool_choice="required"` yields a structured call carrying the same tool and arguments. Across the 59 stored auto-policy transcripts, no response contains `<tool_call>` tags, so the zero structured-call rate is not explained by a parser stripping that tag after generation.

We then varied the requested wrapper while keeping the task and tool schemas fixed. The 14B model follows alternative wrappers such as `<json>` tags, bare JSON, markdown fences, and `<invoke>` tags in 12/12 trials, but does not follow the designated `<tool_call>` wrapper in 0/3 trials. Disabling special-token skipping also produces no `<tool_call>` token. These probes

support a narrow mechanism: the model can produce a valid tool invocation, but in this deployment it does not reliably emit the particular serialization expected by the harness.

The setting is deployment dependent. All main protocol experiments use self-hosted, 4-bit quantized models. When we substitute a hosted frontier model while holding the 59 task descriptions, solve-path instruction, and tool schemas fixed, the model produces structured calls on 118/118 requests. We therefore do not interpret the protocol result as a general property of tool-calling models. The contribution is instead that this kind of deployment mismatch can become invisible to a self-improvement objective when the objective sees only downstream task scores.

Repairing the wire format changes the trajectory but not the score. To measure what the discarded calls would have produced, we add a lenient parser that recovers a message-body call only when the tool name is recognized and the arguments form a valid mapping. The model, prompt, schemas, task set, and grading procedure are otherwise unchanged. Reading stored transcripts predicts 30/59 dispatchable calls; executing the recovered calls yields tool use on 32/59 tasks, 107 tool calls, and 23 non-empty patches. The default arm has none of these events.

The difference is not a change in the benchmark outcome. Both arms resolve 0/59 tasks. The change in tool participation is large (32/59 versus 0/59; Fisher exact $p = 1.3 \times 10^{-12}$), yet the change in resolve rate is exactly zero ($p = 1.0$). The intervention therefore gives a concrete example in which the system's behavior changes substantially while the quantity used for scaffold selection does not.

This result should not be read as evidence that the parser repair improves the agent. The recovered calls often target the wrong artifact, and only 7/23 non-empty patches reach a pass/fail verdict; the remaining 16 terminate as wrong-artifact failures before grading. The purpose of the intervention is narrower: it demonstrates that the benchmark score can remain unchanged after a causal change to the trajectory that precedes it.

Controls across model scale, model family, and task setting. The floor is not confined to a single 14B run. Under the default policy, 7B, 14B, and 32B Qwen models all resolve 0/59 tasks and all issue 0 structured tool calls. Yet the transcript-level attempt rate rises sharply with scale: 5/59 tasks at 7B, 47/59 at 14B, and 51/57 at 32B after excluding two truncated generations (Table 3). At 14B, 30/59 attempted calls are dispatchable; at 32B, 14/57 are. The comparison therefore reveals that scale can change the behavior the evaluator cares about without moving the reported score, or even the recorded tool-use count.

Changing model family gives a similar separation. Mistral-Small-24B uses tools on 18/59 tasks under the same default policy, compared with 0/59 structured calls for Qwen-14B, but both models resolve 0/59. This rules out the interpretation that the phenomenon is simply a property of one model's inability to act. The precise failure mechanism is family dependent; the non-identifiability of the recorded zero is not.

We also remove the agent framework entirely. Sixteen direct requests using the same tool schemas produce 0/16 structured calls from Qwen-14B, although 10/16 responses contain a well-formed call inside a markdown fence. A short-prompt probe likewise reproduces the formatting mismatch, indicating that the effect is not solely a consequence of the long agent prompt or multi-turn context.

As a separate systems control, we run Hyperagents with the same frozen 14B model and the same 59 tasks. Its task loop uses a different message-level parsing strategy, yet it also records 0/59 resolved tasks, 0/59 tool calls, and 59/59 empty patches. The reason is different: the task-agent prompt asks for a `<json>` response schema rather than a tool-call response, so the parser does not find a tool invocation. We use this result not as replication of the exact serialization bug, but as another demonstration that identical summary statistics can arise from different underlying causes.

The sampling baseline and the selection statistic. A second question is whether the floor merely reflects an unusually weak one-shot baseline. We therefore compare against a fixed scaffold with repeated sampling and execution feedback at matched model-call budget. On the 59-task set, this baseline resolves 3/59 tasks, while the single-trajectory default resolves 0/59. An oracle-assisted pass@3 calculation gives 5/59 as an upper bound for the same frozen model under three independent samples. These numbers remain small, but the purpose of the comparison is methodological: a search

Table 1: The ten main configurations on the 59-task Polyglot subset. "Used tools" counts tasks on which a structured tool call reached the harness, measured from intercepted trajectories; "Resolved" uses the benchmark's hidden tests. The `auto` arms record zero structured calls at every scale even where most transcripts contain an attempted call in the message body; those attempt counts are reported separately in Table 3 and must not be read from this column. The best-of-3 arm runs under `required`, which constrains decoding to the tool-call grammar, so its 59/59 is definitional rather than behavioural. The dash marks the one arm for which no task-level tool-use count was retained.

Arm	Used tools	Resolved
7B `auto`	0/59	0/59
14B `auto`	0/59	0/59
32B `auto`	0/59	0/59
Mistral-24B `auto`	18/59	0/59
Hyperagents (14B)	0/59	0/59
14B lenient parser	32/59	0/59
7B `required`	59/59	2/59
14B `required`	59/59	4/59
S_* best-of-3 (14B)	59/59	3/59
Evolutionary seed	–	4/59

procedure should not be credited for gains that a fixed scaffold can obtain simply by spending more calls on the same model.

Across the ten main configurations we ran (Table 1), resolve counts range only from 0 to 4 tasks. Forty-five pairwise Fisher exact tests yield a smallest p-value of 0.12. We do not interpret this as evidence of equivalence; with 59 tasks and so few positive outcomes, the experiment has limited power to resolve small differences. Instead, the table should be read as evidence that the selection variable is operating near a boundary where many substantively different configurations collapse to the same score. This is the regime in which auxiliary measurements are most important.

A pre-registered prediction illustrates the same failure. Before running the larger model arms, we registered the prediction that 14B under `auto` would show non-zero tool use, under the assumption that the 7B result reflected a capability threshold. Using structured `tool_calls` as the registered measure, the prediction was falsified: both 7B and 14B scored 0/59, with identical one-generation and empty-patch distributions. Reading the trajectories after the registered comparison showed that attempted tool use had in fact risen from 5/59 to 47/59. We retain both statements. The prediction failed under the pre-registered measurement, while the underlying behavioral hypothesis was supported by a different measurement. The episode is a direct example of how a formally specified metric can still miss the phenomenon it was intended to measure.

5 The Selection Signal in the Evolutionary Loop

The second failure occurs after the task agent has produced a score and the meta-agent begins proposing scaffold variants. Here the issue is not malformed tool output but the representation of candidate validity in the archive.

Two evolutionary runs, no valid child. Starting from the repaired seed, whose full 59-task evaluation resolves 4/59 tasks, we run two independent evolutionary searches with a 25-evaluation budget each. Each search proposes seven children. All fourteen children modify the agent source, and all fourteen produce Python that fails to parse after the patch is applied. Thirteen are whole-file replacements containing placeholders such as `<new-content>`, `<new-file-content>`, or `Your updated code here`. Ten overwrite the 262-line agent module and three overwrite a 173-line shell tool. The remaining child is a lossy rewrite: it retains only 124 of the original 262 lines and contains a syntax error in a quoted string.

Every child is nevertheless archived with `mean_utility: 0.0`. A syntactically invalid candidate that never starts is therefore represented exactly like a valid candidate that executes and happens to solve no tasks. The distinction is not merely conceptual: a single `ast.parse` check rejects every

Table 2: Evolutionary proposals from the same repaired seed. Validity is determined by parsing the reconstructed post-image of the modified source.

Outcome	Run 1	Run 2
Children proposed	7	7
Whole-file placeholder overwrite	7	6
Lossy rewrite with syntax error	0	1
Valid children	**0**	**0**

one of the fourteen children before container creation or benchmark evaluation. The released system does not perform this check before inserting candidates into the archive, so the invalid candidates can consume evaluation budget and remain eligible for subsequent selection.

The effect is not explained by the edit operation alone. The proposal model's tool call is destructive because the editor's `edit` command replaces the entire file with the supplied `file_text`. That operation is not itself sufficient to explain the invalidity. We replay the same fourteen proposal tasks using a hosted frontier model while holding the task descriptions, seed source, and tool schemas fixed. The Qwen-14B proposal produces invalid Python on 14/14 children; the frontier model produces 0/14 invalid children. Both models still issue the same whole-file editor operation, but the frontier model supplies full Python content rather than placeholder text. Fisher's exact test gives $p = 5.0 \times 10^{-8}$ for the validity difference. With only 14 trials we treat this as "not observed in 14 attempts," not as evidence that invalid children are impossible under the frontier model.

The archive assigns the same utility to both sets of children. That is the key observation. The frontier model changes a property of the candidate that is directly relevant to whether the search can continue—executability—without changing the scalar value that the archive uses to compare candidates. This is independent of whether the frontier children would actually improve benchmark performance.

What the released code tells us. The validity distinction is also visible in the audited implementations. DGM defines a predicate that checks the performance record and requires a non-empty patch before a candidate enters the archive. HGM retains the same predicate and its import, but the released selection path never calls it. Hyperagents uses a different representation: ungradeable results are propagated as null rather than zero, and parent-selection code excludes nodes without complete scores. Thus the HGM observation is a concrete wiring defect in a released implementation, while the broader methodological point is about the information available to the selector.

The audit is deliberately narrow. Two of the three systems we inspected already enforce an invalid-versus-unhelpful distinction, so we do not claim that this particular bug characterizes the field. Instead, it shows how a bookkeeping choice can turn a failed proposal step into an apparently evaluated, zero-scoring candidate. A paper reporting only final resolve rates cannot tell those cases apart.

Why this is a measurement result, not a failure-of-evolution result. Neither evolutionary run produced a valid child, so there is no empirical estimate here of the probability that the search discovers a useful modification. The runs therefore cannot establish that scaffold evolution is ineffective. They establish something different: a search trajectory can spend its entire proposal budget on non-executable candidates while the archive stores them in the same score space used for successful evaluations. Any conclusion about evolutionary progress based on that archive would be ambiguous unless candidate validity were logged separately.

6 What Can a Selection Signal Identify?

The experiments can be viewed as failures of identification rather than failures of optimization. A scaffold search observes a candidate through an evaluation pipeline. Let z denote the state of that pipeline before the final score is produced, including whether the candidate is executable, whether the task agent issued operations, whether those operations were consumed by the harness, and whether grading reached a valid verdict. Let $y = f(z)$ be the scalar value stored by the archive. Selection

can distinguish two candidates only when the aspects of their states that matter to selection remain distinguishable under f.

This distinction is useful because both main interventions change z without changing y. In the task-agent experiment, the default and lenient-parser arms have different execution traces: the latter dispatches tools on 32/59 tasks while the former dispatches none. Yet both have resolve rate 0/59. In the evolutionary experiment, an invalid child that never reaches a valid execution path and a valid child that executes but solves no tasks are both assigned `mean_utility = 0.0`. The score is therefore not merely noisy; in these cases it omits information that exists elsewhere in the pipeline.

This does not make benchmark pass rate an inappropriate objective. It means that pass rate is insufficient as the only observable when a search can fail before grading. Candidate validity and gradeability are preconditions for interpreting a numerical score. A useful archive should therefore retain a distinction between (i) an executable candidate that was evaluated and scored, (ii) an executable candidate that was evaluated and scored poorly, and (iii) a candidate that was not executable or not gradeable. Only the first two states provide numerical evidence about task performance.

The task-agent intervention illustrates why this separation matters. Recovering a tool call is not the same as recovering a successful action. Of the 23 non-empty patches produced after parsing message-body calls, only seven reach a pass/fail verdict; sixteen terminate as wrong-artifact failures. Thus the parser repair provides a clean intervention on the observation layer without establishing a corresponding improvement in capability. The fact that the benchmark remains at 0/59 is not a weakness of the intervention. It is the evidence that a large change in the trajectory can disappear before the selected metric is computed.

The same logic applies to the evolutionary archive. The candidate-validity check in our experiment is computationally cheap: `ast.parse` rejects every one of the fourteen children before benchmark execution. Treating the resulting state as numerical zero creates an ambiguity that cannot be repaired by additional analysis of the final score, because the information has already been discarded. A selector can only rank candidates using distinctions represented in its input.

Score resolution near the floor. There is a second source of difficulty. Across the ten principal configurations, resolve counts range from 0/59 to 4/59. Forty-five pairwise Fisher exact tests produce a smallest p-value of 0.12. We do not interpret this as evidence that the arms are equivalent: with so few positive outcomes, the experiment has limited power to resolve small differences. Instead, the result shows that the scalar objective has little dynamic range in the regime tested. When many candidates occupy the same part of the score range, trajectory- and validity-level measurements become more important for understanding why they do so.

The fixed best-of-3 baseline provides an independent check on this interpretation. The single-trajectory baseline resolves 0/59, while a matched-compute fixed scaffold with repeated sampling and execution feedback resolves 3/59. An oracle-assisted pass@3 calculation gives 5/59 as an upper bound for the same frozen model under three samples. These numbers do not establish a complete attribution of any published self-improvement result, but they make explicit a control that is otherwise easy to omit: before assigning a gain to scaffold evolution, measure what the unchanged model can achieve when given comparable opportunities to act.

A practical criterion. For a search statistic to support attribution, three properties are useful. First, it should be defined only after the candidate is known to be executable and the outcome is gradeable. Second, intermediate operations that materially affect the trajectory should be observable independently of the final task score. Third, the score should have enough resolution, relative to the evaluation budget, to distinguish the candidate differences that the search is intended to optimize. None of these requirements changes the benchmark itself. They determine whether the benchmark can also function as feedback for a program-search procedure.

7 Implications for Evaluation

The minimum instrumentation suggested by these experiments is modest. Candidate validity should be checked before expensive execution. Per-trajectory operation counts should be retained alongside the final task score. Missing or ungradeable outcomes should remain distinct from numerical zero,

and parent selection should exclude candidates for which no valid score was produced. Finally, a fixed scaffold at matched compute should be included whenever the scientific question is whether evolution adds capability beyond repeated sampling and elicitation.

These controls complement rather than replace benchmark evaluation. The benchmark remains the endpoint measure of task success; the additional variables make that endpoint interpretable. In the present study, they reveal two different information-loss points: the structured tool interface can discard an otherwise dispatchable action, and the evolutionary archive can turn an unexecutable child into a numerical zero. In both cases the final score alone is insufficient to reconstruct the path that produced it.

The controls are inexpensive compared with model inference. The validity check in the evolutionary experiment is a single parse operation. The network interception layer records the model interface without modifying the frozen weights or requiring cooperation from the agent. The matched-compute baseline reuses the same model and benchmark. These properties make the proposed measurements practical for future evaluations of self-improving agents rather than a replacement for the experimental systems themselves.

More generally, the longer the causal path between a scaffold modification and its reported benchmark score, the greater the need to preserve intermediate state. A self-improvement result is ultimately a claim about that path: a proposed modification changed the agent, the modified agent executed, the execution produced a measurable outcome, and the outcome improved over an appropriate control. Recording only the final scalar leaves open several alternative explanations that the present experiments show can occur in practice.

8 LIMITATIONS

The main protocol result is specific to self-hosted, 4-bit quantized deployments. A hosted frontier model follows the structured tool protocol in 118/118 matched requests, so we do not generalize the serialization failure to tool-calling models broadly. The mechanistic probes also use a limited number of direct requests and cannot determine how often the same mismatch occurs under other serving stacks or templates.

The evolutionary evidence is narrower still. We ran two searches on one seed, and neither produced a valid child. The diagnosis prompt had to be clipped to fit the available self-hosted context, so these runs are not a faithful reproduction of a frontier-scale evolutionary deployment. The reported archive ambiguity, however, depends only on how the resulting candidate is represented in the bookkeeping layer.

The principal quantitative comparisons use HGM's 59-task Polyglot subset. We include additional model, benchmark, and system controls, but the study is not large enough to support claims of equivalence among the many configurations that remain near zero. In particular, the absence of significant pairwise differences should not be interpreted as proof that the configurations have identical expected performance. The stronger and better-supported conclusion is that, in this regime, the selection statistic provides too little resolution to explain the behavioral differences we can observe elsewhere in the pipeline.

Finally, the paper does not resolve the motivating question of whether reported gains from self-improving coding agents represent expanded capability or improved elicitation. The fixed best-of-3 comparison is evidence about one possible source of gain, not a complete decomposition. Establishing the causal contribution of scaffold evolution itself requires successful non-degenerate evolutionary runs with the additional instrumentation proposed here.

9 CONCLUSION

We studied whether the measurements used by self-improving coding agents preserve the distinctions needed to interpret and select scaffold variants. In the task agent, recovering tool calls discarded by the structured interface changes tool use from 0/59 to 32/59 while resolve rate remains 0/59. In the evolutionary loop, all fourteen proposed children in two runs are invalid Python and are archived

with utility 0.0; replacing the proposal model with a frontier model makes all fourteen valid in the matched proposal test without changing the destructive edit operation or the archived utility.

These results do not show that self-improvement fails. They show that evaluation can become non-informative when execution failures, protocol mismatches, and ungradeable candidates are collapsed into the same scalar outcome. For a search procedure whose behavior depends on that scalar, this is a methodological problem: the evaluator must preserve enough intermediate state to distinguish "did not act," "acted but the action was discarded," and "produced no executable candidate" from an ordinary zero score.

The proposed remedies are inexpensive. Syntax checks, explicit validity states, trajectory-level operation counts, matched-compute fixed baselines, and direct interception of the model interface can all be added without changing the frozen model. These measurements make it possible to ask a more basic question before interpreting a claimed improvement: did the evaluation record the behavior that was supposed to produce it?

REPRODUCIBILITY

We release the measurement instrument, per-arm result files, raw transcripts, applied patches, and the scripts that regenerate every table and figure from those files. The interception path is validated against scripted traces of known ground truth, for completeness (every call that crossed the proxy is recorded) and fidelity (the reconstructed trace equals the known truth); child-validity counts are recomputed from saved diffs. Superseded runs are retained and labeled rather than deleted.

AI USE STATEMENT

Generative AI tools (Claude Code, an agentic coding assistant) were used substantially in this work. AI assistance was used to write the measurement instrument and the analysis and plotting scripts; to propose and refine experimental designs, including the ablation arms and the frontier proposal-model control; to interpret results and to argue for the framing this paper adopts; and as a research aid in reading the three third-party repositories audited in Appendix J. It was *not* used to generate, simulate, impute, or adjust any measurement: every number reported here originates in a logged run against a real model and a real container, and each table and figure is regenerated from those result files by a released script rather than transcribed.

ETHICS STATEMENT

This work reports a defect in the released code of a named, published system. Every claim about third-party code is stated with a file, a line, and the commit at which we read it, so that a reader can check it directly rather than take our word; each is a statement about what a released artifact does, not about the competence or intent of its authors. No human subjects, no personal data, and no dual-use capability are involved. The benchmarks used are public and were used as published.

REFERENCES

Xingze Gao, Chuanrui Hu, Hongda Chen, Pengfei Yao, Zhao Wang, Yi Bai, Zhengwei Wu, Yunyun Han, Xiaofeng Cong, Jie Gui, Yafeng Deng, and Teng Li. EvoAgentBench: Benchmarking agent self-evolution via ability transfer. *arXiv preprint arXiv:2607.05202*, 2026.

Bingguang Hao, ZengZhuang Xu, Maolin Wang, Yuntao Wen, Yicheng Chen, Cunyin Peng, Long Chen, Dong Wang, Xiangyu Zhao, Jinjie Gu, Chenyi Zhuang, and Ji Zhang. FunReason: Enhancing large language models' function calling via self-refinement multiscale loss and automated data refinement. *arXiv preprint arXiv:2505.20192*, 2025.

Mehmet Iscan. Falsification, not exposure: An internally preregistered placebo-controlled decomposition of self-repair feedback in frozen small code models. *arXiv preprint arXiv:2606.31511*, 2026.

Sihang Jiang, Lipeng Ma, Zhonghua Hong, Keyi Wang, Zhiyu Lu, Tengfei Wang, Shisong Chen, Jinghao Zhang, Tianjun Pan, Weijia Li, Jiaqing Liang, and Yanghua Xiao. SEA-Eval: A benchmark for evaluating self-evolving agents beyond episodic assessment. *arXiv preprint arXiv:2604.08988*, 2026.

Shirley Kokane, Ming Zhu, Tulika Awalgaonkar, Jianguo Zhang, Thai Hoang, Akshara Prabhakar, Zuxin Liu, Tian Lan, Liangwei Yang, Juntao Tan, Rithesh Murthy, Weiran Yao, Zhiwei Liu, Juan Carlos Niebles, Huan Wang, Shelby Heinecke, Caiming Xiong, and Silvio Savarese. ToolScan: A benchmark for characterizing errors in tool-use LLMs, 2024. Characterises seven tool-use error patterns, including invalid output formatting.

Thomas Kwa, Ben West, Joel Becker, Amy Deng, Katharyn Garcia, Max Hasin, Sami Jawhar, Megan Kinniment, Nate Rush, Sydney von Arx, Ryan Bloom, Thomas Broadley, Haoxing Du, Brian Goodrich, Nikola Jurkovic, Luke Harold Miles, Seraphina Nix, Tao Lin, Chris Painter, Neev Parikh, David Rein, Lucas Jun Koba Sato, Hjalmar Wijk, Daniel M. Ziegler, Elizabeth Barnes, and Lawrence Chan. Measuring AI ability to complete long software tasks. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2025. arXiv:2503.14499.

OpenAI. SWE-bench Verified, 2024. `https://openai.com/index/introducing-swe-bench-verified/`.

Maxime Robeyns, Martin Szummer, and Laurence Aitchison. A self-improving coding agent. *arXiv preprint arXiv:2504.15228*, 2025.

Jason Starace. Scaffold effects on GAIA: A controlled comparison. *arXiv preprint arXiv:2606.08529*, 2026.

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. *arXiv preprint arXiv:2510.21614*, 2025.

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 (ICLR)*, 2026a. arXiv:2505.22954.

Jenny Zhang, Bingchen Zhao, Wannan Yang, Jakob Foerster, Jeff Clune, Minqi Jiang, Sam Devlin, and Tatiana Shavrina. Hyperagents: Self-referential self-improving agents. *arXiv preprint arXiv:2603.19461*, 2026b.

A THE INSTRUMENT IN FULL

Scaffold ladder. S_0 is the minimal scaffold (one trajectory, no retries, no selection), and by construction *defines* the frontier against which everything else is measured. S_* is a strong, fixed, hand-authored baseline (best-of-N, execution feedback against public tests only, tool use, self-consistency). Its role is to answer the elicitation question directly: how much of any observed gain is reachable *without evolution at all*, at matched compute. S_{evo} is a forked, independently-implemented evolving scaffold.

The ladder is deliberately ordered by what each rung licenses. A gain over S_0 shows only that the scaffold does something; a gain over S_* at matched compute is the first result that cannot be explained by sampling the same frozen model harder. We treat the second comparison, not the first, as the meaningful one, and report both.

Attribution by interception, not self-report. An evolved scaffold's code is produced by a process optimising "solve more tasks," not "report your own operations honestly"; a variant that omitted tagging an expanding operation would face no pressure against doing so. We therefore never rely on instrumentation *inside* the agent. Because forked agents run as separate OS processes inside Docker, we intercept at the only remaining boundary, the network, using a transparent reverse proxy between container and model server that records every request/response pair. Operations are then classified post hoc from what was observed.

Two properties of this design matter for what the paper can claim. First, the proxy observes the wire, so it sees exactly what the harness sees and nothing more: this is what let us establish that the `auto` arms issue one generation per task, and equally what prevented us from noticing, until we read trajectories, that a majority of those generations *contained* a tool call. An interception layer is an honest witness to the protocol, not to intent, and §4 is in part a report of that distinction mattering. Second, non-200 responses are recorded but excluded from generation counts, so a backend error cannot inflate a scaffold's apparent work; we found this necessary after an early version counted failed calls as legitimate generations.

Each intercepted operation is labelled *support-preserving* (selection, retries, self-consistency: replicable by sampling the frozen model harder) or *support-expanding* (tool output conditioning later generations, execution feedback reshaping a prompt). The partition is the load-bearing abstraction of the instrument: support-preserving operations cannot move the frozen model's reachable set by construction, so a gain attributable entirely to them is elicitation regardless of how large it is. Classification is behavioural—whether a later request contains a substantial fragment of an earlier response or of execution output—and we treat it as evidence rather than proof (§8).

Compute accounting. Every model call is charged to a budget accountant that records prompt and completion tokens, so "matched compute" is a measured quantity rather than a nominal setting. This is what makes the elicitation control meaningful: a scaffold that wins by issuing sixty times more generations has not demonstrated anything about the model. For containerised trajectories the charge is necessarily post hoc, since a trajectory's cost is not known until it has run, so the accountant surfaces overruns rather than preventing them.

Causal ablation. To test whether a support-expanding operation is *necessary* for a solve, we withhold the information channel itself rather than instructing the agent not to use it—an ablated agent is denied the capability regardless of what its own code attempts.

Frontier estimation. Per-task success rates use Clopper–Pearson exact intervals; coverage of the unseen difficulty distribution uses Good–Turing and Chao1. A central discipline: the estimator cannot express $p = 0$. Every "beyond frontier" record carries an explicit budget-relative qualifier.

B SETUP DETAILS

Tasks and grading. We run S_0 on the 59-task Polyglot subset that HGM itself uses as its baseline evaluation set,[1] spanning six languages (C++, Go, Java, JavaScript, Python, Rust); the full list is given in Appendix L. Choosing the system's own subset is deliberate: a null result on a set we selected would invite the objection that we chose unfavourable tasks. Grading uses HGM's real harness rather than a reimplementation: a per-task Docker container in which the hidden tests are revealed, by resetting to a test commit, only after the agent has stopped. The agent therefore never sees the tests it is graded on, and we inherit the harness's own definitions of `resolved`, `unresolved`, and `empty_patch`.

Model and serving. The frozen models are `Qwen2.5-Coder-7B-Instruct` and `Qwen2.5-Coder-14B-Instruct-AWQ` (4-bit weight quantisation, used so the larger model fits the same single 24 GB accelerator and the scale comparison costs no additional hardware). Both are self-hosted under vLLM 0.26 with a 16,384-token context, automatic tool-choice parsing enabled and the model family's own tool-call parser selected. Self-hosting is load-bearing: it makes frozen-ness verifiable rather than promised, keeps contamination questions inspectable, and matters because frontier estimation is sample-hungry. It is also what surfaces the result in §4—a hosted endpoint that silently repaired malformed tool calls would have hidden the failure entirely.

The `required` arms differ from the `auto` arms in exactly two source lines, both in the agent's own model-call wrapper, applied to an isolated copy so the measured default agent is never mutated. All other configuration is byte-identical, including sampling parameters, and each result record carries the model identifier and agent-variant name so that no two arms can be merged by accident.

[1] HGM's own subset definition lists 60 entries, of which 59 are unique; one instance identifier appears in both of its constituent files.

Table 3: Tool-call behaviour across three model scales under the unmodified `auto` policy. "Structured" is what the harness reads, and is the column Table 1 reports. "Runnable" calls name a real tool with every required argument. All three arms were evaluated on all 59 tasks; the 32B row's Emitted, Parses and Runnable columns exclude the two tasks whose generation reached the serving ceiling of 6144 total tokens and was cut off mid-response (`javascript__triangle` and `rust__doubly-linked-list`; no other task exceeds 4679), leaving $n = 57$ transcripts analysable for call shape. Structured and Resolved are 0 across all 59 evaluated tasks in every arm.

Model	n	Emitted	Parses	Runnable	Structured	Resolved
7B	59	5 (8%)	2	1 (2%)	**0**	0
14B	59	47 (80%)	38	30 (51%)	**0**	0
32B	57	51 (89%)	23	14 (25%)	**0**	0

Deviation for the self-improvement runs. The runs in §5 required a 32,768-token context, because HGM's diagnosis prompt does not fit a 16k window. Those runs therefore do *not* share a serving configuration with the arms reported in Table 1, and we do not compare their numbers directly; each re-evaluates its own seed. We state this rather than quietly pool the two.

We manipulate one variable: the agent's *tool policy*, set by the `tool_choice` parameter of the OpenAI-compatible chat API. The parameter takes three named values: `"none"`, under which the model may not call a tool; `"auto"`, under which the model decides for itself whether to call one; and `"required"`, under which it must emit at least one call, enforced by constraining decoding to the tool-call grammar. HGM's released agent uses `"auto"`. We write these values literally rather than paraphrasing them, because the distinction between "the model chose not to call a tool" and "the model could not emit a call in the required form" is what Section 4 turns on, and a gloss such as *optional* would blur it.

Throughout, we label the two conditions the **`auto` arm** and the **`required` arm**. The `required` variant differs from HGM's unmodified default agent in **exactly two lines**, both the `tool_choice` flag; we verify this by direct diff rather than assertion. All other scaffold code, the model, the task set, and the grading harness are identical.

C Robustness controls in full

Scale does not repair the protocol failure. The result so far invites a specific objection: HGM and DGM were evaluated with frontier hosted models, so a protocol failure at 7B and 14B might be a small-model artifact with no bearing on these systems as deployed. This is the strongest available criticism of the finding, and it requires an experiment rather than an argument. We therefore ran a third scale, `Qwen2.5-Coder-32B-Instruct-AWQ`, under the same unmodified default agent and the same 59 tasks.

Three observations follow (Table 3), and we are careful to separate those the data supports from those it does not.

Structured tool calls are zero at every scale we can self-host. The failure is not confined to the smallest model: it is unchanged at 32B. It is, however, specific to this deployment regime. Run against a frontier hosted model (`claude-opus-5`) on the same 59 tasks, with the same solve-path instruction and the same two tool schemas taken verbatim from the agent, the structured-call rate is **118/118** across two replicates (§I). The objection that this is an artifact of small, quantised, self-hosted models is therefore correct, and we report it as such rather than leaving it as an open question.

The attempt rate rises with scale, but not monotonically in any sense we can defend. The increase from 7B to 32B is decisive ($5/59 \to 51/57$, Fisher exact $p = 5.6 \times 10^{-20}$, non-overlapping intervals $[0.028, 0.187]$ vs $[0.785, 0.960]$). The further increase from 14B to 32B—80% to 89%—is **not significant** ($p = 0.20$), and we do not claim it. What the data supports is a large gap between the smallest model and the two larger ones, not a smooth trend.

Well-formedness falls significantly at 32B, from $30/59$ runnable at 14B to $14/57$ ($p = 0.004$, intervals $[0.375, 0.641]$ vs $[0.141, 0.378]$). This is the one genuinely surprising result of the comparison: the largest model attempts most and serialises worst, and the difference is not attributable to noise.

We do not offer a mechanism for the non-monotonicity. Appendix D establishes what the failure is at 14B, but we did not repeat that analysis at 32B, and we decline to speculate without it.

A second benchmark reproduces the result. Every measurement to this point comes from one benchmark. If the recorded zero reflected a convention of Polyglot's harness rather than the model's behaviour, a different benchmark should not reproduce it. We therefore ran the same unmodified default agent on **SWE-bench Verified** (OpenAI, 2024), using that benchmark's own harness and verification path rather than a reimplementation.

Instances are stratified by repository rather than truncated from dataset order, because the first N instances in order are approximately 88% `astropy` and would make the sample a statement about one project. Taking the first two instances from each repository yields 23 instances across 12 repositories (`astropy`, `django`, `matplotlib`, `mwaskom`, `pallets`, `psf`, `pydata`, `pylint-dev`, `pytest-dev`, `scikit-learn`, `sphinx-doc`, `sympy`).

	Polyglot	SWE-bench Verified
Instances	59	23 (12 repos)
Model calls per task	1	**1**
Tasks using tools	0	**0**
Resolved	0	**0**
Infrastructure failures	0	**0**

The signature is identical: exactly 23 model calls across 23 instances—one generation each, then termination—with a mean of 556 completion tokens and no tool call anywhere. Different repositories, different task format, different verification machinery, same result.

We note one artifact rather than smoothing it over. A single instance (`psf__requests-1142`) reports a produced diff of 871,670 characters despite making one generation and zero tool calls. An agent that never acted cannot have written that; the diff is environment noise inherited from the harness's own container state, the same effect documented when this pipeline was first validated. It is reported here because a large patch length alongside zero tool calls is exactly the kind of internal contradiction that should be examined rather than averaged into a summary statistic.

The zero reproduces outside the harness. A natural objection is that 59 tasks share one system prompt, so the zero may be a property of that prompt rather than of the model. We therefore measured the rate directly, with no agent framework involved: sixteen requests (eight distinct task descriptions $\times$ two temperatures), tools passed to the API, `tool_choice="auto"`.

Outcome	Count
Structured tool call	**0/16**
`<tool_call>` tags present but unparsed	0/16
Well-formed call inside a markdown fence	10/16
No call attempted	6/16

Zero structured calls at both temperature 0.0 and 0.7, across varied prompts, outside any harness. Two further points follow. First, the absence of `<tool_call>` tags in the raw text rules out a parser fault: the serving stack is not failing to extract calls the model made, because the model never produces that format. We confirmed the same over all 59 stored transcripts (0/59 contain the tags). Second, both failure modes documented in §4 appear here in proportion—most responses emit a well-formed call in a markdown fence, a minority narrate intent without emitting anything.

A second system: the same recorded zero, produced by a different cause. Every result measured *inside* a self-improving system to this point comes from one fork. If the recorded zero is an artifact of HGM's design, the scope of the finding collapses. Hyperagents is the natural control: an independently-implemented self-improving agent whose task loop recovers tool calls by *regular*

Table 4: Two families under the unmodified `auto` policy, same agent and tasks. Tool use differs decisively; the measured outcome does not differ at all.

	Qwen-14B	Mistral-24B	
Tasks using tools	0/59	**18/59 (31%)**	$p = 1.7 \times 10^{-6}$
95% CI	[0.000, 0.061]	[0.192, 0.439]	non-overlapping
Total tool calls	0	21	
Empty patches	59	58	
Resolved	**0/59**	**0/59**	$p = 1.0$

expression over the message text rather than from the structured API field—the opposite design choice, and one that ought to be immune to the serialisation failure of §4. Run with the same frozen 14B on the same 59 tasks, it records **0 resolved, 0 tool calls, and 59/59 empty patches, with exactly one turn per task**—HGM's numbers exactly.

The transcripts show the agreement is a coincidence of *outcome*, not of mechanism, and the difference matters more than the match. Hyperagents' task-agent prompt closes by specifying its required reply format: *"Respond in JSON format with the following schema: <json>{"response": ...}</json>"*. That schema is not the tool-call schema. The model complies with it exactly—59/59 replies are well-formed `<json>` objects carrying a `response` field, and 0/59 contain the `tool_name` or `tool_input` keys the loop's regex requires. The model is not failing to serialise a call it selected, as under HGM; it is correctly answering a question that was never a request for a tool call. The loop finds no match and terminates after one turn.

Two properties of the shipped system make this hard to see from its own outputs, and both must be stated. First, the task agent calls its chat function without `tools_available`, which defaults to no tools—so stock Hyperagents' task agent cannot use tools at all, for a reason independent of any prompt contract. We enabled tools with a one-line change matching the configuration Hyperagents' *own* DGM baseline and meta-agent already use, and the numbers above are from that patched run; they are not stock Hyperagents. The patch's effect is itself the clearest evidence for the diagnosis: enabling tools changed *no* graded quantity. Both runs record 0 resolved, 59/59 empty patches, one turn per task, and 0/59 transcripts containing a tool call. The only difference anywhere in the two sets of outputs is that 4/59 patched transcripts mention a tool in prose against 0/59 stock. Tool availability was never the binding constraint; the reply schema was. Second, its logger records the user message but sends system prompt plus user message, so the tool definitions the model actually received appear in no transcript. We initially misread that gap as evidence our patch had not applied.

The honest reading is therefore narrower than a replication and, for this paper's argument, more damaging. This is not independent confirmation that models emit calls in the wrong channel; under Hyperagents the model was never asked for a call. What replicates is the *uninformativeness of the recorded zero*. Two systems, opposite parsing strategies, produce identical summary statistics—0 resolved, 0 tool calls, $59/59$ empty patches—from causes that share nothing: a discarded well-formed call in one, a prompt that forecloses tool use in the other. Neither zero is a measurement of what the model can do, and no quantity either system logs distinguishes them.

A second model family: the failure is family-dependent, and still yields zero. Scale within one family is not the only alternative explanation; the phenomenon might be specific to Qwen. We therefore ran a fourth arm with `Mistral-Small-24B-Instruct`, chosen deliberately *larger* than the 14B so that a low rate could not be attributed to model size.

The result splits cleanly, and we are careful with it. **The protocol failure is family-dependent**: Mistral emits structured calls on 31% of tasks where Qwen emits none, a decisive difference ($p = 1.7 \times 10^{-6}$, non-overlapping intervals). The claim "models cannot follow this protocol" would therefore be false, and we do not make it. **But the measured outcome is unchanged**: 0/59 resolved in both, 58/59 empty patches for Mistral. A 31% compliance rate is still catastrophic for a loop requiring a call at every turn. Neither scaling within a family nor changing family repairs the result the selection signal sees.

(a) 7B, `python__dominoes` — the model addresses a human reader.

> *"To integrate this into your project, you can create a new Python file in the `/testbed/` directory, say `domino_solver.py`, and add the above code to it. Then, you can call this function from your main program… If you need further assistance or have additional requirements, feel free to let me know!"*

(b) 14B, `cpp__all-your-base` — the model acts, in the wrong channel.

> *"Now, let's apply these changes to the code files in the `/testbed/` directory. We'll use the `editor` tool to edit the `base_converter.py` file."*

```
{"name": "editor", "arguments": {"command": "edit",
"path": "/testbed/base_converter.py", "file_text":
"..."}}
```

Figure 2: Assistant output from the two `auto` arms; both responses carry `tool_calls=None` and are scored identically as `empty_patch`. In (b) the call names a real tool with valid arguments and is emitted as fenced JSON rather than inside `<tool_call>` tags. Note also that (b) targets a Python file for a C++ task, the wrong-artifact mode of §4.

D MECHANISM AND ITS CHECKS

A methodological precondition, and a near-miss. Measuring whether a model follows a tool-call protocol presupposes it was *told* the protocol. We therefore verify, for every model, that its chat template renders the tool schemas before running any arm. This is not pedantry: the community quantisation of Mistral-Small-24B ships a stripped template that silently ignores the tools argument, rendering an identical prompt with and without tools. Run as-is it would have scored zero because it was never offered tools, and we would have reported a generalisation that was an artifact of our own configuration. It was caught by the precondition check and fixed by loading the quantised weights with the official tokenizer. A second candidate, `DeepSeek-Coder-V2-Lite`, was excluded outright: its *official* template ignores tools, so protocol compliance is undefined for it. We report this because the check is cheap, it changed our conclusions twice, and no paper in this area that we are aware of states it.

What the failure actually is. The results above characterise the failure's *extent*. Its *mechanism* can be measured directly, and doing so refutes the explanation we initially favoured.

Holding the task fixed and varying only the requested output wrapper, with tools supplied through the API so the chat template renders its tool-calling section, the 14B model complies with **every** format asked of it except one:

Requested wrapper	Complied	Structured call
`<tool_call>` tags (its own protocol)	**0/3**	0/3
`<json>` tags	3/3	0/3
Bare JSON, no fence	3/3	0/3
Markdown code fence	3/3	0/3
`<invoke>` tags	3/3	0/3

This refutes a hypothesis we had advanced. An earlier version of this work attributed the behaviour to a pretraining-frequency prior: fenced JSON is far more common in text than either tag convention, so the model might default to it. That is wrong. The model produces a fence only when a fence is requested and follows arbitrary alternative wrappers at 12/12. There is no general prior to appeal to.

It also refutes a serving-stack explanation. In this tokenizer `<tool_call>` is a single dedicated token (id 151657), raising the possibility that the model emits it and the serving stack strips it before the parser sees it. Re-running with special-token skipping disabled returns byte-identical output containing no such token. The model does not emit it.

What the model does instead is emit a *correct* call—right tool name, right arguments—inside an improvised wrapper. Across probes we observed `<call>`, `<function-calls>`, and markdown fences. The characterisation that survives is narrow and specific: the model is format-compliant in

general, and fails to produce the single token its own tool protocol requires, substituting a near-miss no parser is watching for.

One limitation bounds this mechanistic claim. All probes use the 4-bit quantised build, because the unquantised 14B does not fit the accelerator used throughout; quantisation can affect low-frequency token generation and we cannot exclude that an unquantised model behaves differently. This bounds the mechanism, not the measurements, which describe the deployment a self-hosting practitioner would actually run.

Reconciling this with function-calling benchmarks. An obvious objection is that the Qwen family posts strong scores on function-calling benchmarks such as BFCL, which appears to contradict a $0/59$ structured-call rate. Three points reconcile these.

First, the highest reported figures for this family are for variants *fine-tuned specifically for function calling* (Hao et al., 2025); we evaluate the released instruct models, which is what the self-improving systems under study actually load.

Second, such benchmarks score short, single-turn invocations against a clean schema, whereas the setting here is a multi-turn agentic loop with a long system prompt, a real repository, and tool output fed back as conversation. Success on the former does not entail the latter.

Third, and most directly: *we tested the short-prompt condition and the failure persists*. A single request carrying the same tool schemas and a two-sentence task description—BFCL-like conditions—returns fenced prose rather than a structured call from the 14B model, while the same request returns a well-formed structured call from Mistral-24B (Appendix M). The failure is therefore not an artifact of prompt length or agentic context. We none the less flag this as the comparison most likely to be raised against our result, and note that a systematic study holding prompt complexity fixed across model families would settle it more firmly than a single probe.

A deployment deviation, and the check it required. The 32B model at 4-bit quantisation occupies roughly 19 GB of weights and does not fit a 16k context on the 24 GB accelerator used throughout: vLLM reported 4.0 GiB of KV cache required against 1.25 GiB available. After three failed configurations it was served at a **6144-token context in eager mode**, against 16384 for the 7B and 14B arms. We report this rather than presenting the arms as identically configured.

The deviation demanded a specific check, because a response truncated mid-object registers as "emitted but does not parse"—that is, our configuration would masquerade as the model's protocol failure. Two transcripts (`javascript__triangle`, `rust__doubly-linked-list`) reached `finish_reason="length"` and are **excluded**; the 32B row is $n = 57$. They cannot account for the effect, since at most two of twenty-eight parse failures are attributable to truncation. Without this check we would have reported $23/59$ parsing and attributed a self-inflicted artifact to the model.

E METHODOLOGICAL COVERAGE IN RELATED WORK

The controls this paper argues for are individually familiar; what is unusual is running them together, and the value of doing so is easiest to see against what the closest published systems report about themselves. Table 5 records, for each, whether its *own* evaluation includes each control. We stress what the table is and is not. It is a statement about reported evaluations, not about the quality of the systems or the competence of their authors; a control that is absent from a paper may well have been run and not reported. It is also assembled to a standard we apply to ourselves: our own causal-ablation entry is marked partial, because the mechanism was never run against a real evolved scaffold, and the final column records that we evaluate fewer frozen-model families than HGM does. The pattern worth drawing from it is narrow: no system in this literature, including ours, currently reports all of frozen-model ceiling, matched-compute elicitation control, intercepted attribution, causal ablation, and pre-registration together, and the first three are exactly the ones needed to tell elicitation from expansion.

Table 5: Methodological coverage in closely related work. Entries record whether each paper's *own reported evaluation* includes the control. DGM, HGM, Hyperagents and SICA were checked against full text; † marks assessment from abstract only, where absence means "not found" rather than "confirmed absent". ○ marks a partial control: DGM and Hyperagents compare against a fixed-meta-agent variant without matching compute, and HGM re-runs SWE-agent on its own backbone under budgets it describes as matched — a scaffold-versus-scaffold control, not an estimate of the frozen model's ceiling. Our own causal-ablation entry is ○ on the same standard, since the mechanism was never run against a real evolved scaffold. The last column counts frozen-model families: HGM evaluates two (GPT-5 and Qwen3-Coder) across four model sizes, to our two. We do not lead on it.

	Frozen-model ceiling	Matched-compute elicitation ctrl.	Intercepted attribution	Causal ablation	Pre-registered	Model families
DGM	—	○	—	—	—	2
HGM	—	○	—	—	—	2
Hyperagents	—	○	—	○	—	1
SICA	—	—	○	—	—	2
EvoAgentBench†	—	—	—	—	—	—
SEA-Eval†	—	—	—	—	—	—
Iscan (2026)†	✓	✓ (static)	—	—	✓	1
This work	✓	✓	✓	○	✓	2

Table 6: The 14B agent under the unmodified policy and under a lenient parser, on the same 59 tasks with no infrastructure failures in either arm. The intervention changes only whether the harness accepts a call the model had already emitted. Patch length is reported as a median over the 23 non-empty patches: the mean (3,585) is inflated by build artifacts the harness sweeps into a diff once the agent actually compiles something, a contamination we document rather than average over.

	`auto`	lenient
Tasks using tools	0/59	**32/59 (54%)**
Total tool calls	0	**107**
Non-empty patches	0	**23**
Median patch length	0 chars	**1,735 chars**
Reached a pass/fail verdict	0	7
Wrong-artifact failures	0	16
Resolved	**0/59**	**0/59**

F FURTHER RESULTS

Effect of the tool-policy flag. That setting `tool_choice="required"` increases tool calls is definitional, and the mechanism matters for how the result may be read. Under this setting the serving stack constrains decoding to the tool-call grammar, so a structured call is emitted whether or not the model would have produced one unprompted. Given Table 7, the honest description is that the flag **repairs protocol compliance**, not that it induces volition: for the 14B model, intent was already present in a majority of transcripts and only the serialisation was wrong. *We therefore do not claim that the tool policy determines whether the agent tries to act.* What we claim is narrower and sufficient: the flag determines whether the harness can *observe* an attempt, and every selection signal in use depends on that observation.

Three consequences are not definitional, and are what we report: the measured baseline is exactly zero rather than merely low; the same flag separates "this system does nothing" from "this system solves tasks" on a benchmark the system's authors chose; and the change is invisible to pass rate. A flag whose behavioural effect is definitional can still be undetectable at the level a search actually optimises, and that gap is the finding.

Switching `tool_choice` from `"auto"` to `"required"`, with everything else byte-identical, moves the 7B agent from 0 tool calls to 3663 across the same 59 tasks—every single task using tools, in every language ($59/59$, Fisher exact $p = 8.2 \times 10^{-35}$). Model calls rise from 59 to 3687, and the agent now produces real patches that receive real test verdicts: 53 of 59 tasks yield a non-empty diff,

Table 7: Tool calls in the `auto` arms, graded by strictness. "Structured" is what the harness reads and what Table 1 reports. "Runnable" calls are dispatchable as-is; this is not a claim that the task would then have been solved.

Arm	Structured	Emitted	Parses	Runnable
7B × `auto`	0/59	5/59	2/59	1/59
14B × `auto`	0/59	**47/59**	38/59	**30/59**

Table 8: Attempted (emitted) and dispatchable (runnable) tool calls per language in the `auto` arms. Structured tool calls are 0 in every cell of this table, for both models and all six languages.

		14B × `auto`		7B × `auto`	
Language	n	Emitted	Runnable	Emitted	Runnable
C++	5	5	2	0	0
Go	10	10	7	0	0
Java	12	8	4	1	0
JavaScript	10	6	5	1	1
Python	11	8	4	3	0
Rust	11	10	8	0	0
Total	59	**47**	**30**	5	1

and the modal outcome changes from `empty_patch` (59/59) to `unresolved` (51/59). Two tasks are solved outright. The 14B model behaves the same way under the same flag (5592 tool calls, 4 solved), so the effect is a property of the policy, not of a particular model. Figure 1 places this and every other configuration we ran on a single pair of axes: the interventions span the full range of agent participation while none leaves the floor on task success.

The lenient-parser arm in detail. Table 6 reports the arm described in §4 alongside the unmodified policy it is paired with. The two differ in one respect only: whether the harness accepts a call the model had already emitted into the message body. Tool participation moves from 0/59 to 32/59 and 23 non-empty patches appear where there were none, while the graded outcome is identical at 0/59. The row that matters for the paper's argument is the last one.

A third failure mode: acting on the wrong artifact. Forcing tool use does not make the agent competent at the task *interface*. In 9 of 59 cases in the 14B `required` arm, the agent worked productively (tens to hundreds of tool calls) but created a *new file of its own naming* rather than editing the exercise's declared solution files. Frequently the new file was in the wrong language entirely: `diamond_kata.py` for the C++ task `cpp__diamond`, `robot.py` and `spell_number.py` for JavaScript tasks. The harness stashes only the declared solution paths before revealing the hidden tests, so these attempts leave nothing gradeable behind:

```
No local changes to save
HEAD is now at 632858f
Removing diamond_kata.py
No stash entries found.
```

We count these as unresolved rather than excluding them, since the task was certainly not solved; excluding them would inflate the resolve rate by shrinking the denominator over exactly the attempts that failed. The mode is worth naming because it is a third way the scaffold–model interface, not the model's coding ability, determines the measured outcome—and because it too is invisible in a pass-rate column, where it is indistinguishable from having tried and been wrong.

The elicitation control: sampling the same model harder. A gain over S_0 shows only that a scaffold does something; the meaningful comparison is against sampling the *same frozen model* harder, which is what S_* provides. We ran it as best-of-N with $N = 3$ over the full 59 tasks, under the `required` policy so that the agent acts at all.

Selection is deliberately **oracle-blind**: candidates are chosen by majority vote over literal diff text, never by inspecting pass/fail. This matters because the Polyglot harness grades every trajectory as a side effect of running it, so picking the best-scoring candidate would be trivial to implement and would correspond to no scaffold a deployment could run. We therefore report two quantities that must not be conflated:

- **Resolved, oracle-blind:** $3/59$**.** What the scaffold actually achieved. This is the elicitation control proper, and the figure comparable to S_0.
- **pass@3:** $5/59$**.** Whether *any* of the three trajectories passed—an upper bound on what any selection rule over the same samples could reach, and a budget-relative estimate of the frozen model's reachable set. It is *not* scaffold performance.

The gap of two tasks is measured selection headroom: cases where the model produced a passing solution that oracle-blind consensus did not choose. That tripling the sampling budget yields $3/59$ ($[0.011, 0.141]$) against the single-trajectory arm's $4/59$ is itself informative—the two are statistically indistinguishable ($p = 1.0$), and neither is distinguishable from zero. Even the oracle-assisted upper bound, pass@3 at $5/59$ ($[0.028, 0.187]$), does not separate from the single-trajectory arm. Sampling the same frozen model three times does not move it off the floor, and this is the control against which any evolved scaffold's gain would have had to be judged.

This also bears on how stable the selection signal is when it is non-degenerate. Across the runs reported here the aggregate is remarkably reproducible—$4/59$ in the `required` arm and $4/59$ again for the evolutionary seed under a different deployment—while its *composition* is not: those two agree on only three of four tasks. A scaffold search selecting on the aggregate sees a stable-looking number, roughly a quarter of which is resampling churn.

G Implications in full

The recorded quantity is not the phenomenon. The central consequence is not that a flag changes behaviour—that is definitional—but that a single recorded value, 0 tool calls, covers two unrelated states of the system. In one, the model does not act. In the other it selects the correct tool and arguments and fails to serialise them. Every quantity HGM logs is identical across the two: resolve count, tool-call count, patch length, and the per-task distributions of each. The distinction is recoverable only from trajectories, which no selection procedure in this family reads.

Six later experiments sharpen this. The failure persists unchanged at 32B (§C), so it is not an artifact of small models. It reproduces on a second benchmark with different repositories and a different verification path (§C), so it is not an artifact of one benchmark's conventions. It reproduces with no agent framework at all (§C), so it is not an artifact of one system prompt. It is family-dependent but not thereby repaired: Mistral-24B emits structured calls on 31% of tasks and still resolves nothing (§C, Table 4). And accepting the discarded calls moves the agent to 54% participation while leaving pass rate at zero (§F, Table 6), so the signal is insensitive in both directions—it cannot see the agent trying, and cannot see it being enabled to try. Finally, a second, independently-built system reaches an identical recorded zero by a wholly unrelated route (§C): its prompt asks for a schema that is not a tool call, and the model complies. That case is not a replication of the serialisation failure, and it is the sharper problem—two different causes yield identical summary statistics, so the quantities a meta-agent selects on cannot tell them apart even in principle.

This is what the tool-policy manipulation is for, and it is worth being precise about its role. It is not the finding; it is the control that establishes the finding's direction. Because constraining decoding to the tool-call grammar converts the same model, on the same tasks, from 0 to $59/59$ tasks using tools, the deficit under the default policy cannot be attributed to an inability to determine *what* to do. The 14B transcripts show the intended call already formed (Table 7); the constrained-decoding arm shows the same intent surviving into the structured field. What separates them is a wire format.

Why this is not merely a scoring inaccuracy. An evaluation error that lowered all scores uniformly would be a nuisance. This one is structural, because self-improving systems consume their own scores. HGM samples archive nodes of *strictly positive* utility before expanding one. A protocol failure sends every node to zero, the candidate set is empty for every node, and the search cannot take

its first step—the implementation raises `ValueError: attempt to get argmax of an empty sequence`. The formatting failure does not degrade the search; it prevents it from starting.

The degenerate condition also recurs one level down, which we did not anticipate. Both evolutionary runs began from a seed scoring $4/59$, chosen precisely so the positive-utility filter would be non-empty. Every child none the less evaluated to 0.0—because none of them could execute (§5)—so the archive returned to a uniformly-zero state after the first generation, and the search proceeded only because we had patched the selection step to fall back to all evaluated nodes rather than raise. A faithful reproduction of the published implementation would have crashed at that point. The condition is therefore not a curiosity of an unusually weak seed; it is reachable from a healthy one within a single generation. We observed exactly this, and it is why the evolutionary run reported in §5 required a seed configuration that scores above zero at all.

Scaffold policy versus model scale. The claim here is regime-relative, and the two halves of it are scoped differently. The *policy contrast* was run at two model scales on one task family within one forked system. The *scale null* is broader: it extends to a third scale (32B, §C) and to a second benchmark (§C), in both cases under the default policy. We do not claim scaffold dominates model capability in general. Within that scope, model scale predicts recorded agentic behaviour not at all under the default policy ($0/59$ vs $0/59$, $p = 1.0$), while predicting *attempted* behaviour strongly ($47/59$ vs $5/59$). Resolve rate never reaches significance in any comparison (0, 2, 0, 4 of 59; smallest $p = 0.12$). For meta-agent evaluation this reverses the usual attribution: a harness search that stumbles onto a tool-policy change is not discovering a better agent, it is discovering the switch that makes the agent's actions legible—and it will be credited with a capability gain either way.

Frontier degeneracy. S_0 is what defines the measurable frontier. When $S_0 = 0$, every condition of a frontier-crossing test becomes vacuous: there is no task S_0 plausibly fails-but-could-solve, the estimator has nothing to estimate, and *any* nonzero evolved-scaffold result crosses trivially. This is an *uninformative* null: the instrument could not have detected an effect, which is categorically weaker than a well-powered null. We flag this because a study that reported "the evolved scaffold beat the baseline" from exactly this configuration would be reporting an artifact.

What a scaffold search would be rewarded for. The interventions that move this system's measured score are not ones that change what the frozen model can do. Repairing serialisation moves it from $0/59$ to $4/59$ with the weights untouched by construction; so, plausibly, would a prompt line specifying which file to edit, given that 9/59 of the tool-using runs failed by writing an undeclared file (§F). Both are interface repairs. Both would be credited as capability gains under current evaluation practice, because pass rate is the only thing being read. The concern is not that a search might find such a change—finding it is useful—but that nothing in the reported methodology distinguishes having found it from having improved the agent.

Compute is reported, not matched. We report compute rather than equalising it, because equalising is not meaningful here: an agent that never acts cannot spend a comparable budget. The `auto` arms issue exactly one generation per task (59 total); the 7B `required` arm issues 3687, a $62\times$ increase, with mean completion tokens per task rising from 902 to 6637. So the arms differ not only in behaviour but in consumed compute by nearly two orders of magnitude—and are still statistically indistinguishable in the utility a pass-rate objective assigns them ($p = 0.50$). A selection procedure that cannot separate scaffolds differing by $62\times$ in compute and totally in behaviour is not measuring what its users believe it measures.

Behavioural effect versus pass-rate effect. The difference in whether the agent *acts at all* is about as decisive as a 2×2 table can be: **59/59 versus 0/59** tasks with at least one tool call, Fisher exact $p = 8.2 \times 10^{-35}$, with non-overlapping Clopper–Pearson intervals ($[0.939, 1.000]$ vs $[0.000, 0.061]$) and no exceptions in any of the six languages. The difference in tasks *resolved* is not: 2/59 versus 0/59 yields $p = 0.50$, and at $n = 59$ per arm this design cannot detect a resolve-rate shift smaller than roughly $6/59$ at $\alpha = 0.05$. We therefore do **not** claim that changing the tool policy improves pass rate. It changes behaviour categorically and moves resolve rate off the floor by a margin this experiment is underpowered to certify.

A blind spot in pass-rate selection. Combining the behavioural and pass-rate findings: the arms differ by $p \approx 10^{-35}$ in whether the agent acts, and are indistinguishable in the signal used to select between them. A search procedure optimising pass rate, as DGM, HGM, and SICA all do, receives almost no gradient from the single change that determines whether its agent is an agent. The scaffold change that would break the deadlock is precisely the one its objective cannot see.

An interpretability band. The base model must be strong enough that the selection signal is non-degenerate, but the scaffold must already be functional enough that the search is not merely rediscovering its own switches. Neither boundary is currently reported by any system we reviewed.

A concrete consequence for meta-agent evaluation. Reported gains for self-improving agents should be accompanied by (i) an independent estimate of the frozen model's own ceiling under repeated sampling, and (ii) a fixed-scaffold elicitation control at matched compute. Without both, a pass-rate delta is consistent with the model's capability boundary not having moved at all.

H PRE-REGISTRATION AND ITS AMENDMENT

Pre-registration. Thresholds, the bootstrap procedure, multiple-comparison correction, and the interpretation matrix were fixed before any evolutionary result existed. The sub-study reported below was likewise declared—including its predictions and the manipulation—before its runs executed.

A pre-registered prediction, its falsification, and a defective measure. Before running the larger-model arms we registered the prediction that `14B × auto` would show non-zero tool use—i.e. that the floor reflected a capability threshold the 7B model sat below. We operationalised "tool use" as structured `tool_calls`, the quantity the harness itself acts on. Against that measure the prediction was falsified outright: `14B × auto` produced **0/59 resolved, 0/59 tasks with any tool call, and 59/59 empty patches**, identical to the 7B arm in every cell, including the distributions of per-task generations ($\{1\}$) and patch lengths ($\{0\}$), with mean completion tokens rising only from 902 to 976.

We report the falsification as it was scored, and we also report that the measure was inadequate. Table 7 shows the prediction's substance—that the larger model would attempt to use tools—was in fact correct, and by a wide margin (47/59 versus 5/59 attempted calls); it was the operationalisation that could not see it. Both facts belong in the record. Registering the prediction is what forced the transcripts open: had we not written it down, the post-hoc framing "a 7B model is too weak to be an agent" would have been available, comfortable, and wrong, and the identical-in-every-cell result would have looked like confirmation of it.

We draw the methodological point rather than soften it. A pre-registered measure can be wrong in the same way a scaffold's selection signal can be wrong, and for the same reason: both were defined over what the system records rather than what the agent does.

I THE FRONTIER CONTROL

Every measurement in the main results uses a self-hosted, 4-bit quantised model no larger than 32B. The objection that the recorded zero is an artifact of that regime, rather than a property of the systems under study, is the strongest available criticism of this work, and §8 of an earlier version listed it as untested. It is now tested, and the objection is correct.

Design. The comparison holds everything constant except the model. We take the same 59 Polyglot baseline instances by `instance_id`, each one's real `problem_statement` from the benchmark's own metadata, the agent's real solve-path instruction template, and the agent's own `bash` and `editor` tool schemas copied verbatim from its source. The only substitution is the model: `claude-opus-5`, a hosted frontier model, in place of the self-hosted `Qwen2.5-Coder` builds. Two replicates per task, 118 requests.

Result. 118 of 118 (Table 9), with no exceptions in any of the six languages and none in either replicate. No response placed a call in a markdown fence, and none emitted the `<tool_call>` convention as text. The failure mode that defines the main results does not occur at all.

Table 9: Structured tool-call rate under `tool_choice="auto"`, same 59 tasks, same instruction, same tool schemas. "Structured" is what the harness reads: for the OpenAI-compatible endpoint a populated `tool_calls` field, for the Anthropic endpoint a `tool_use` content block. These are the faithful analogues of one another, not a renaming.

Model	Deployment	Requests	Structured
`Qwen2.5-Coder-7B`	self-hosted, 4-bit AWQ	59	**0**
`Qwen2.5-Coder-14B`	self-hosted, 4-bit AWQ	59	**0**
`Qwen2.5-Coder-32B`	self-hosted, 4-bit AWQ	59	**0**
`claude-opus-5`	hosted frontier	118	**118**

What this does and does not establish. It establishes that the protocol failure is a property of the deployment regime we could afford to self-host, not of tool-calling models in general. Any reading of the main results as a claim about frontier systems as deployed is therefore wrong, and we state that here rather than leaving a reader to infer it.

It does **not** measure resolve rate, and nothing about task success should be inferred from it. This is the analogue of the harness-free control of §C, not of S_0: one request per task, no tool execution, no grading. Running the measured agent itself against a frontier model would require either an intermediary that presents the Anthropic endpoint in the OpenAI-compatible shape the agent expects, or edits to the agent's own tool-parsing path—and the latter would modify the artefact under measurement, which is precisely what this paper's methodology forbids. We therefore report the emission rate, which is the quantity the recorded zero is actually about, and leave the frontier resolve rate as future work.

What survives. The result of §5 is untouched. That an invalid child and a working child that solved nothing are both archived with `mean_utility: 0.0` is a property of how the archive computes utility, not of any model's behaviour; a frontier model would be recorded identically. The same holds for the central claim: a scalar pass rate cannot separate "did not act" from "acted and was discarded" from "is not valid Python," whatever model produced each state. The frontier control narrows where the protocol failure occurs; it does not touch what the recorded quantity can distinguish once it does.

J THE VALIDITY CHECK ACROSS THREE SYSTEMS

The claim in §5 is that the systems in this literature hold enough information to separate a broken candidate from an unhelpful one, and discard it at the point of selection. That claim is about published source code rather than about our runs, so we state it with file and line references against public repositories, at the commits we read.

What was audited. For each system: does anything between "a child's source has been generated" and "the child's score enters the archive" establish that the child is valid at all? We read the generation and archival path directly. An earlier pass searched for `ast.parse` across each repository and would have returned a confident wrong answer—the predicate that matters is not written that way, and a symbol search cannot see a check implemented by other means, nor distinguish a function that is defined from one that is called.

Table 10: Whether each system can distinguish an invalid child from a valid child that scored zero, and whether it acts on the distinction. Line numbers are from the public repositories at the commits we read; the Huxley–Gödel Machine was read at `013872d`.

System	Distinction available	Acted on
DGM (Zhang et al., 2026a)	`is_compiled_self_improve`	**yes**
HGM (Wang et al., 2025)	the same predicate, retained	**no**
Hyperagents (Zhang et al., 2026b)	null score, distinct from zero	**yes**

The predecessor enforces it. DGM defines `is_compiled_self_improve` in `utils/evo_utils.py`. The predicate requires that the run's performance record carry the expected keys, that at least one non-empty patch was produced, and that the number of evaluated instances match what was requested. It is called in two places. One annotates the attempt's metadata and, on failure, substitutes a fixed string for the self-diagnosis rather than feeding a diagnosis derived from a candidate that never ran. The other, `filter_compiled` in `DGM_outer.py`, retains only those run identifiers that pass, so a candidate failing the predicate does not enter the archive. The second condition—at least one non-empty patch—is exactly the state every child in our two runs occupied.

The successor retains the function and drops the call. HGM's `self_improve_step.py` is annotated in its own header as adapted from DGM. The same predicate is present in its `utils/evo_utils.py` – not byte-identical, but logically unchanged: the same docstring, the same three conditions, the same return values, differing only in formatting, in logging via `print` rather than a logger, and in three `raise` statements that are present but commented out. That last detail is worth noting: someone considered making the predicate fail loudly and chose to soften it, which is the opposite of an author unaware of the check. `hgm_utils.py` imports it by name. It is never called: in the repository at the commit we read, the only two occurrences of the identifier are its definition and that import. There is no analogue of `filter_compiled` anywhere in the tree. Every child that completes the generation step is archived with whatever utility its evaluation produced, and for an invalid child that value is `0.0`.

We note what this does and does not say. It does not say the authors were unaware of the problem—the presence of the inherited function suggests the opposite. It says that the wiring is absent in the released artefact, and that the consequence is the collapse documented in §5: fourteen children, none of them valid Python, none distinguishable in the archive from a working child that solved nothing.

The third system encodes it and acts on it. Hyperagents takes a different route and, on the evidence of its released code, arrives at the right place. Its scoring helper returns a null rather than a number when a child's evaluation report is missing, when the score is not a number, or—for the benchmark used here—when no task was evaluated without error. That null propagates: a generation whose domains do not all return a real score is recorded with `valid_parent` false, and both parent-selection routines skip such a node before scoring it, admitting only candidates whose per-domain scores are all non-null. A child that produced nothing gradeable is excluded from selection rather than competing at zero.

We record a correction to an earlier version of this audit, because it bears directly on how far the result generalises. That version cited the test `x is not None and x > 0` at `generate_loop.py:660` as the point at which null and zero are folded together. The expression is there and does fold them—but it gates whether to run a *full* evaluation after a staged one, which is a decision about spending compute, not about admitting a candidate to the archive. Reading that line without following `valid_parent` into the selection path, we attributed to Hyperagents a defect it does not have. We report the correction here because it is the same error this paper documents elsewhere—a conclusion drawn from the quantity a system records at one site, without checking what the site is for.

What this establishes, and what it does not. Our own runs are two runs on one benchmark with one model family, and §8 is explicit about what that supports. The property described here is not a property of our runs at all: it is a property of released code, checkable without a GPU, a model, or a container, and a reader can confirm every row of Table 10 by opening two files.

What it does not establish is that this is a habit of the field. Two of the three systems audited do enforce the distinction, by different means—DGM filters on an explicit predicate, Hyperagents excludes a null-scored child from parent selection. The system we measured is the exception, and it is the exception in a specific and recoverable way: it inherited the predecessor's predicate, kept the function and the import, and dropped the call. We state this narrowly on purpose. The general claim this paper defends is about what a scalar pass rate can represent, which is settled by arithmetic and by the fourteen children of §5; the lineage audit shows that the remedy is already standard practice in this literature, which makes its absence in one released artefact a wiring defect rather than an open research problem.

K THE SELF-IMPROVEMENT RUNS IN FULL

The seed. A full 59-task evaluation of the seed agent scores $4/59$, resolving `go__counter`, `java__mazy-mice`, `java__react` and `rust__gigasecond`. The `14B × required` arm of Table 1 also scores $4/59$, on three of the same four tasks, differing by one substitution (`java__bowling` for `java__react`). The aggregate is reproducible across deployments while its composition is partly resampling noise—a point we return to in §8, since it bears on how stable a selection signal this is even when it is non-degenerate.

Neither run produced a single valid child. Both runs exhausted a 25-evaluation budget and proposed seven children each. All fourteen modified the agent's own source. **All fourteen produce Python that does not parse.** Table 2 summarises; the mechanism is identical in both runs.

The dominant failure is a single behaviour. The `editor` tool's `edit` command overwrites an entire file with the supplied `file_text`, as its schema documents. The model supplies a *description of the content* in place of the content, and the tool faithfully performs the overwrite. Thirteen of the fourteen children replace an entire file with one line, drawn from a small set of template placeholders: `<new-content>`, `<new-file-content>`, `<updated-file-content>`, `<file-content>`, `<content-to-edit>`, and `Your updated code here`. Ten overwrite the 262-line agent module; three overwrite the 173-line `bash` tool.

The fourteenth is the informative exception. On the raw diff it appears to be a genuine targeted edit—thirty-three lines of real Python across small hunks—and we initially recorded it as the one valid child. Applying the patch shows otherwise: the resulting file is **124 lines from an original 262**, and fails to parse on an unescaped apostrophe (`help='Task's programming language'`). The model regenerated the file from memory, silently discarded half of it, and introduced a quoting error. It is the same overwrite semantics producing a lossy rewrite rather than a placeholder.

The archive cannot tell. Running an agent whose source does not parse yields no resolved, unresolved, empty-patch or error verdicts; every task is recorded as incomplete. But node utility is computed as $[1] \times \text{resolved} + [0] \times (\text{submitted} - \text{resolved})$, so four tasks that never started become $[0, 0, 0, 0]$ and the node is archived with `mean_utility: 0.0`—*arithmetically identical to a working child that ran and solved nothing*. Those nodes then compete for selection, consume evaluation budget, and become eligible parents.

It was detectable by one line of code. Calling `ast.parse()` on each child's agent source raises `SyntaxError` for all fourteen. No model, no GPU, no task execution: a single parse rejects every child the system accepted, archived, evaluated, and scored. Nothing in the pipeline performs this check, and nothing in the reported methodology of this class of system would surface its absence.

What we do and do not conclude. We do not claim the evolutionary method fails, and this is not a test of it: a proposal step that never emits a valid candidate has not been given the opportunity to search. We also note that our own configuration weakens the proposal step, since the diagnosis prompt is truncated to fit a self-hosted context (§8). What we claim is narrower and, for evaluation practice, worse: across two independent runs the system spent its entire budget on candidates that could not execute, recorded them in the same units it records real results, and possessed no mechanism that would notice. The pre-registered comparison of per-generation resolve rate against the seed remains unrun, because there was never a child to compare.

Four of the six are recorded identically to a model that tried and failed (Table 11). One is worse than that: copying the seed agent into an already-populated output directory leaves the previous run's metadata in place, so a repeated run silently inherits its predecessor's seed performance. Nothing errors; the numbers are simply wrong. For an evolutionary search, whose entire trajectory is conditioned on the archive's recorded utilities, this is a contamination path rather than an inconvenience.

The last two entries are properties of the published system rather than of our deployment. HGM's diagnosis step concatenates the failing trajectory's full chat history with the agent's own source, and the guard that would bound it is disabled in the released code:

```
# if len(code_text) > 100000:
#     code_text = code_text[:100000] + "\n<code clipped>"
```

Table 11: Defects encountered before the self-improvement path would execute. The final column is the point: only two announce themselves. The remainder are reported through the same channel as ordinary model failure, so a study that read summary statistics rather than container logs would have recorded them as the agent performing badly.

Defect	Location	How it surfaces
Self-improvement container created without host networking, so the agent cannot reach the model server	container setup	Crash (loud)
Build staging reused across runs; `copytree` cannot overwrite git's read-only objects	image build	Task marked failed
Exception handler references an unassigned variable, raising a second error that replaces the first	child sampling	Wrong cause reported
Seed directory copied *into* an existing output directory rather than replacing it, so a rerun reads its predecessor's metadata	run initialisation	Silently wrong numbers
Diagnosis prompt exceeds a 16k context by one token	diagnosis	Attempt marked failed
Diagnosis prompt unbounded (≈130k characters); the upstream truncation guard is present but commented out	diagnosis	Attempt marked failed

Enabled, that guard would still permit a prompt larger than any context we can serve for a 14B model on a single 24 GB accelerator, and it bounds only the smallest of the three large terms. The implication is narrow but real: the proposal half of this system presupposes a large-context hosted model, and is not runnable as published against a self-hosted one. We bounded the prompt explicitly to proceed (budgets in Appendix N), which means the diagnosis model in our runs observes less evidence than its authors intended—a limitation we carry forward in §8 rather than treat as equivalent.

We draw no conclusion about HGM's quality from this. Research code is not production code, and every defect above is of a kind that appears in working systems. The transferable observation is about *evaluation practice*: a self-improving system's failures and its infrastructure's failures arrive through the same channel, and only reading trajectories distinguishes them.

L Evaluation set

The 59 unique instances of HGM's own Polyglot baseline subset, by language. Exercises recur across languages (`wordy` appears in four, `poker` and `react` in three), which is useful: a failure on `wordy` in one language while the same exercise is handled in another is unlikely to reflect the problem being intrinsically hard.

Language	Exercises
C++ (5)	all-your-base, bank-account, diamond, gigasecond, queen-attack
Go (10)	book-store, connect, counter, crypto-square, dominoes, error-handling, food-chain, markdown, transpose, word-search
Java (12)	bowling, custom-set, forth, go-counting, ledger, mazy-mice, react, rest-api, satellite, sgf-parsing, wordy, zipper
JavaScript (10)	bottle-song, forth, grade-school, ocr-numbers, queen-attack, robot-name, say, simple-linked-list, triangle, wordy
Python (11)	beer-song, dominoes, dot-dsl, food-chain, grade-school, poker, scale-generator, sgf-parsing, tree-building, wordy, zipper
Rust (11)	accumulate, bowling, doubly-linked-list, gigasecond, macros, pig-latin, poker, react, say, variable-length-quantity, wordy

M VERIFYING THAT THE TOOL PROTOCOL WAS COMMUNICATED

Section 4 claims the model was told the tool-call convention and did not follow it. Because the alternative—that our deployment failed to advertise the tools—would invalidate the result entirely, we record the three checks that rule it out. Each is a script released with the instrument.

The template renders the schemas and the convention. Applying the model's own chat template to the agent's tool list produces a system turn of 2681 characters (164 without tools) containing the full JSON schema for each tool inside `<tools>` tags, followed by the instruction to return calls "within `<tool_call></tool_call>` XML tags" with a worked example. The protocol is stated, not assumed.

A single request differing only in policy resolves the ambiguity. Issuing one identical request under each policy returns, under `auto`, `tool_calls: None` with a fenced JSON call in the message body; under `required`, a structured call carrying the same tool and the same arguments. Intent is constant across the two; only the channel changes.

The effect is population-level, not anecdotal. Grading all 118 stored transcripts yields Tables 7 and 8. We report three strictness levels because they license different claims, and use only the strictest (*runnable*) when asserting that a call could have been dispatched.

N PROMPT BUDGETS APPLIED TO THE DIAGNOSIS STEP

For the runs in §5 the diagnosis prompt was bounded to fit the served context: 24,000 characters of agent source, 32,000 of trajectory, and 8,000 of evaluation log. Clipping retains the head and tail of each section rather than truncating the tail, since the opening of a trajectory establishes the task and its end shows the failure. As noted in §8, this means the diagnosis model observes less evidence than the system's authors intended, and results from those runs are correspondingly weaker evidence about the method itself.