

Do Self-Improving Coding Agents Beat Textbook Baselines? Evidence from a Darwin Gödel Machine Port and Two Published Systems

Tejasvi Singh Tomar

ORCID: <https://orcid.org/0000-0003-4668-3639>

Abstract

Self-improving agent harnesses such as the Darwin Gödel Machine (DGM) let a coding agent edit its own scaffolding, and they keep the edits that score better and report large gains on coding benchmarks. In three pre-registered rounds, we compare what such loops find with standard engineering changes (“textbook moves”), one-shot redesigns by a language model, and placebo loops that make no edits; we also re-run released agents of the DGM and of ADAS with newer models. On hard LiveCodeBench problems with Claude Haiku 5.5, the starting agent often stops on an empty turn because its output-token limit is too small for the model’s reasoning. A single textbook move, which raises this limit to 16,000 tokens and makes three other changes to the model loop, recovers about 90% of the gain of the best loop agent and costs less than half as much per task. Four of five one-shot redesigns by Claude Opus 5.5 also raise the limit, to 8,192 tokens, without evaluation feedback. The best loop agent still leads the textbook moves by 1–5 percentage points in pass rate. On held-out SWE-bench Verified tasks, run in the DGM’s harness, the DGM’s released best agent does not outperform its initial agent; three textbook moves on the initial agent outperform the best agent at a quarter of its cost, whether or not the patches of runs that ended early, most of them the best agent’s, are kept. Claims of self-improvement should be measured against textbook moves, one-shot redesigns, placebo loops, held-out tasks and newer models.

1 Introduction

A self-improving agent harness lets a coding agent edit the code around a frozen model, such as its prompts, tools and control flow, and keeps the edits that score better on a benchmark. The Darwin Gödel Machine (DGM) [37] reports SWE-bench rising from 20% to 50% this way, and related systems search over agent designs (ADAS, Automated Design of Agentic Systems [12]), guide the search with lineage statistics [28], or make the improvement procedure itself editable [38]. Recent audits show that such gains shrink on held-out tasks and against compute-matched baselines [11, 30, 7, 22], and that a setting change can match what a search finds [32]. What a self-improvement loop finds that direct engineering would not find remains an open question.

We address it in two ways. First, in a controlled port of the DGM to hard competitive-programming problems from LiveCodeBench, we compare the agents that loops select with four baselines: standard engineering changes applied without search (textbook moves), a single redesign by a language model without evaluation feedback (a one-shot redesign), loops whose diagnoses, a model’s analysis of a failed task, are replaced by requests drawn from a fixed list of generic improvement requests (random-diagnosis loops), and placebo loops whose children are exact copies of their parents. Second, we rebuild the DGM’s own released SWE-bench agents (the initial agent and the

reported best) and run them on SWE-bench Verified tasks the search never saw, next to the initial agent with textbook moves. We also run the agents released with ADAS under a current model. All three rounds of experiments were pre-registered: a first round on the loops themselves, a second round with the baselines, generality checks and published systems, and a third round that removes confounds the first two left. Changes made after each pre-registration, among them the choice of the second-round SWE-bench model after pilot runs whose outcomes were known, are summarized in Appendix A, and the full log is part of the release.

Each baseline removes one ingredient of a loop. Textbook moves are the changes an engineer would make without any search (ours were written while reading the logs of pilot runs; Appendix C); if they match a loop’s best agent, the search found nothing that an engineer could not match without it. A one-shot redesign keeps the language model’s knowledge and the failure logs but drops the evaluation feedback and the iteration. Random-diagnosis loops keep the search but replace the diagnosis. Placebo loops keep the selection on a small benchmark but make no edits, so they show how much of a reported gain selection alone produces.

Findings.

- (i) On competitive programming with Claude Haiku 5.5, most of what the loops found is standard engineering. The model’s reasoning often exhausts the starting agent’s 4,096-token output cap on hard problems, and the agent then stops on an empty turn, a response with neither text nor a tool call. One textbook move, L, which rewrites the agent’s model calls with a 16,000-token output cap, comes within about a tenth of the best loop agent’s gain at less than half its cost per task; in each of two sets of five one-shot redesigns by Claude Opus 5.5, one made with failure logs and one without, four also raise the cap, to 8,192 tokens. Even the best combinations of textbook moves remain one to five points below the best loop agent (§5; Table 8 gives the registered verdicts).
- (ii) Generic requests come close to the DGM’s diagnoses only when some of them target the empty-turn failure: without such requests, random-diagnosis loops gain 6.6 points less than the DGM loops (§5.3).
- (iii) On SWE-bench Verified tasks outside the DGM’s evaluation set, run in the DGM’s harness with Claude Sonnet 4.5 (a newer model than the DGM used), the released best agent does not improve significantly on its initial agent, and about half of the best agent’s runs end before a patch is recorded, for a cause we could not determine. The initial agent with three textbook moves does better than the best agent at a quarter of the best agent’s cost, also when the patches of those runs are kept and when both DGM agents are given the same 16,000-token cap (§6).
- (iv) Placebo loops raise their best score on the selection tasks by as much as order statistics predict. For the six DGM loops of the first two rounds, the gain on the selection tasks exceeds the gain on fresh problems by about four points on average, which is not significant (§3 and Figure 1).

Section 6 also re-runs the agents discovered by ADAS, which, under a current model, do no better than chain-of-thought self-consistency with a matched number of samples on three of four benchmarks, and reports a leak of hidden-test information in the DGM’s SWE-bench harness: in the three tasks we inspected, a leftover pytest cache named the failing hidden tests. Our runs remove the leak, and none of the DGM’s released chats reads the cache. Appendix D shows that the textbook moves together help under Claude Haiku 4.5 but do not transfer to gpt-oss-120b (exploratory).

2 Setup

Tasks. We use the LiveCodeBench code-generation problems (lite, release v6) that are labelled hard and whose private tests total at most 20 MB, 282 problems in all [13]. A fixed shuffle gives a 10-problem small stage, a 40-problem medium stage (together the selection set S) and 232 fresh problems F , split into halves F_1 and F_2 of 116. A problem counts as solved when the final `solution.py` passes all public and private tests (6 s per test). Programs that read standard input run exactly as written; class-method problems receive the usual imports.

Harness and loop. The seed is the DGM polyglot coding agent (a bash tool and a whole-file editor) with one change: its model client points to our proxy. The loop follows the DGM code: two self-improvement attempts per generation; parents drawn with probability proportional to a sigmoid of their score divided by one plus their number of children; a diagnosis of one of the parent’s failed tasks; the parent agent editing a copy of its own repository from the resulting problem statement; staged evaluation (medium stage if at least 4 of 10 small tasks pass); and an archive that keeps compiled children scoring at least the seed’s score minus 0.1. The diagnosis prompts are the DGM’s polyglot prompts with the language sentence changed. With probability 0.25 the loop uses the DGM’s hand-written “stochasticity” prompt, and for a task with no output its “empty patch” prompt; we call these two the hint prompts, and two exploratory loops run without them. The reported best agent is the archive agent with the highest score on the full selection set, as in the DGM. Loops run 40 generations (20 for the other task models). A second seed, the engineered seed, was written directly, without a search loop (Appendix C says when); it differs from the DGM seed in four files (§4).

Models. The task model and the self-editing model is Claude Haiku 5.5 with its default settings; diagnoses use Claude Sonnet 5.5. Other task models are Claude Haiku 4.5, Claude Sonnet 4.5 (SWE-bench), Claude Sonnet 4.6, Claude Sonnet 5.5, Gemini 2.5 Flash and gpt-oss-120b; one-shot redesigns use Claude Opus 5.5 and Claude Sonnet 5.5. Every model call passes through a proxy that records its cost, removes sampling parameters (which the API no longer accepts), requests one tool call per response (the behaviour the 2025 seed code assumes) and caps each request at 16,000 output tokens. Each task run has a dollar budget (\$0.25 under Haiku 5.5 and gpt-oss-120b, \$1 under the other models on competitive programming, \$10 on SWE-bench) and a wall-clock limit (600 s on competitive programming); a run that reaches the budget receives a final empty answer, and at either limit the solution present is graded. On competitive programming, harnesses run in network-isolated sandboxes. On SWE-bench, the task containers can reach the internet, because the DGM’s harness installs the agent’s requirements in them.

Measures. For an agent a , $S(a)$ is its pass rate on the selection set and $F(a)$ its pass rate on fresh problems. Fresh gains are differences of per-problem pass rates paired by problem, with 95% bootstrap intervals over problems. In the first round the seed’s fresh reference pools all its runs (12 per problem on F_1 , 3 on F_2), from different hours; morning runs scored about 2.5 points higher than evening runs. Every second-round batch carries its own runs of the seed, interleaved with the other agents problem by problem. Cost per task is the dollar total that the proxy records for that run. Table 1 lists the agents and loops compared.

Table 1: Agents and loops compared in the paper.

name	meaning
DGM seed (the seed)	the DGM’s polyglot coding agent as ported; it requests at most 4,096 output tokens
textbook moves E, B, L, P	a string-replace editor; a non-interactive shell; a model loop with a 16,000-token output cap; a workflow prompt (§4)
engineered seed	the DGM seed with all four textbook moves, written directly, without search
cap-only seed	the DGM seed with only its output cap raised to 16,000 tokens
one-shot redesign	one model call that rewrites the seed from its code and five failure logs (log-free: without the logs)
DGM loop	the DGM procedure on the seed; loops 1 to 3 are those of the first round
placebo loop	the same schedule, with children that are exact copies of their parents
random-diagnosis loop	a DGM loop whose diagnoses are replaced by requests drawn from a fixed list (second round; with a clean list in the third)
HGM-style loop	a DGM loop that picks each parent by Thompson sampling over the passes and failures of each archive agent and its descendants
textbook agent (SWE-bench)	the DGM’s initial SWE-bench agent with three textbook moves

3 Where a loop’s gain comes from

Selection on noise. The unchanged seed scores $62.5\% \pm 2.8$ on S over 254 evaluations (its own and those of placebo copies, which have identical code). The 6 placebo loops evaluate about 80 copies each on the full selection set and report best scores of 68 to 70%; the expected maximum of that many independent draws with the seed’s mean and spread is 69% (Figure 1a shows the three first-round placebo loops and the three first-round DGM loops). A loop’s best selection score therefore carries a noise term of about seven points, predictable from the rerun spread and the number of full evaluations. For the DGM loops the selection gain exceeds the fresh gain in every loop, by about four points on average, but with each loop’s sampling error this is not significant (H2; §5).

Compute. Evolved agents spend more per task than the seed. Rerunning the unchanged seed several times and submitting the first solution that passes the examples in the problem statement (Table 2) accounts for most of the fresh gains of two of the three first-round loops at similar cost: their best agents differ from the seed at matched cost by $+1.3 [-2.7, +5.8]$ and $+3.1 [+0.5, +6.3]$ points on F_1 . Resampling does not reproduce the third loop’s gain even at twelve attempts, which still cost less than that agent, probably because the seed fails the same problems on every attempt.

Repair. The seed requests at most 4,096 output tokens. The task model’s default reasoning often uses that budget up on hard problems, and the agent then receives neither text nor a tool call and stops; 28% of the seed’s fresh runs end with no solution. Loop 3’s best agent recovers from such empty turns by retrying with larger budgets, nudges and fallback calls. Raising the seed’s cap to 16,000 tokens and changing nothing else gains $+19.8$ points $[+15.1, +24.6]$ on all fresh problems in the final batch that re-evaluated every loop agent together (Table 5), where it trails loop 3’s agent by 2.6 points $[0.4, 5.0]$ at about a third of its cost per task; in the batch of the lineage ablation (§4) it trails by 4.5 $[2.2, 7.3]$. The cap-only seed is an exploratory arm.

Other first-round results (details in Appendix F). Single edits proposed from the agent’s code alone gain about two points; single DGM steps gain 1.9 times as much when children that failed to compile count as unchanged, and 2.7 times as much when those children are left out, mostly

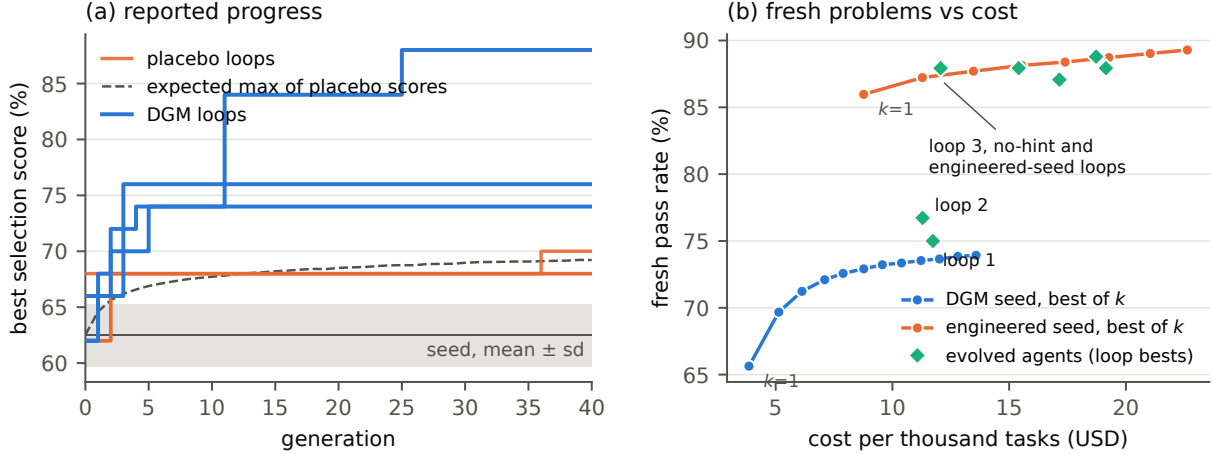


Figure 1: First-round loops. (a) Best selection score by generation. Placebo loops (children are exact copies) rise to the expected maximum of their own noisy scores (dashed); DGM loops rise above it. (b) Pass rate on F_1 against cost per task. Lines: the unchanged seeds run k times, submitting the first solution that passes the examples in the problem statement. Diamonds: best agents of the loops.

Table 2: The unchanged seed run k times, submitting the first solution that passes the sample tests in the problem statement (F_1 ; averaged over orderings of the available runs). Cost in dollars per thousand tasks.

	$k = 1$	$k = 2$	$k = 3$	$k = 6$	$k = 12$
DGM seed, pass rate	65.6	69.7	71.2	72.9	73.9
cost	3.9	5.1	6.1	8.8	13.6

through the DGM’s hand-written empty-patch prompt. Loops started from the engineered seed (a short harness that already avoids the empty-turn failure) gain about two points on fresh problems (+2.4 [90% CI +0.5, +4.3]). Under a bug in our first grader, most of the measured gain of loops started from that harness was recovery from the bug; re-scoring the saved archives with the corrected grader shows this without new agent runs. Under other task models, evolved agents gain most where the seed fails worst, and loop 3’s agent costs 2 to 13 times as much per task as the seed. These small first-round gains are measured against a seed reference that pools runs from different hours (Appendix F).

4 Baselines without self-improvement

The first round shows where the loops’ gains come from. It does not show whether a loop was needed to find them. A second pre-registration fixed four baselines (B1–B4 below: textbook moves, random diagnosis, one-shot redesign and a lineage ablation; the placebo loops were registered in the first round), three generality checks, and a test of published systems, before any of them ran.

Textbook moves (B1). The engineered seed differs from the DGM seed in four files, and each difference is a standard engineering move: E, a string-replace editor (view with line ranges, create,

replace a unique string, insert); B, a non-interactive shell with closed standard input, a per-command timeout and clipped output; L, a model loop that allows 16,000 output tokens, answers every tool call in a response, re-prompts once after an output cut off by the cap and stops after 40 turns; P, a workflow prompt that asks for sample tests, a brute-force cross-check and a timing check. We build all 16 combinations by copying the chosen files into the DGM seed and evaluate each twice on all 232 fresh problems, interleaving agents problem by problem so that every agent sees the same hours.

Random diagnosis (B2). Three loops run the DGM procedure unchanged except that each diagnosis is replaced by one request drawn uniformly from a fixed list of 20 generic improvement requests (for example, “Add a step that writes a brute-force reference solution and compares it with the main solution on random small inputs”). The list was written after the first round and before any second-round run; none of its requests mentions the output cap, but one asks the agent to recover from responses that contain neither text nor a tool call, and two are close to it: one reruns the agent when the solution file was not changed, and one logs the stop reason and usage of each response. This is search over textbook requests instead of diagnoses.

One-shot redesign (B3). A language model receives, once, the seed’s code and the logs of five failed selection-set runs of the seed (the material the DGM’s diagnosis step sees) and returns the files it would change. It gets no evaluation feedback. We draw five samples from Claude Opus 5.5 (pre-registered) and five from Claude Sonnet 5.5, the loops’ diagnosis model (added to separate the design from model strength). This is diagnosis without search.

Lineage ablation (B4). For the best agent of each first-round DGM loop we rebuild the agent from the seed by applying its lineage’s patches in order, and build variants with one patch left out and with one patch alone. We also revert the output cap in loop 3’s best agent while keeping all its other code.

Generality. Three more DGM loops and three more placebo loops on the same seed and tasks double the number of loops per arm. Three loops replace the DGM’s parent choice with HGM-style Thompson sampling: each archive member’s clade (the member and all its evaluated descendants) contributes its passed and failed selection runs to a Beta posterior, and the parent is the member with the largest draw [28]. Three DGM loops and one placebo loop use gpt-oss-120b as task and self-editing model (20 generations). As an exploratory addition, two DGM loops and one placebo loop use Claude Haiku 4.5, a model that rarely exhausts the seed’s output cap, so that the cap fix is not available to the search.

Analysis rules. Gains and intervals are computed as in §2. These intervals hold the realized loops fixed, so arm-level comparisons are also tested at the level of loops. Comparisons that pair agents from two batches (H6, H8 and the comparison of L with one retry in §5) are reported as raw differences in the text and also seed-adjusted, with the batch times and seed drifts, in Appendix B. When a best candidate is chosen among several (15 combinations, five redesigns), it is chosen on the first half of the fresh problems and compared on the second half, a rule fixed after the pre-registration but before any result was summarized (Appendix A, item 4); we also report the literal registered reading. The non-inferiority margin is 3 points. The final batch re-evaluates every loop’s reported best agent, the seed, the cap-only seed and L together on all fresh problems, twice.

Published systems. For the DGM’s SWE-bench run we rebuild the initial agent as it ran in 2025: the released repository’s history starts after the run, and two small differences from the run-time code (a later retry in the model call and an example line in the editor) are inferred from the context lines of the archive’s patches, after which all six patches of the reported best agent’s lineage apply without a rejected hunk. We compare the initial agent, the reported best agent, the initial agent with textbook moves (the string-replace editor, a 16,000-token output budget and five workflow steps in the prompt) and the same with up to three attempts selected by the repository’s regression tests, built from helper functions that the DGM’s initial agent already contains. All four run inside each task’s container through the DGM’s own harness, on SWE-bench Verified tasks outside the DGM’s 200-task evaluation set and on the DGM’s 60 selection tasks, keeping only tasks whose reference patch passes under today’s environments. For ADAS we take the archived agent with the best validation score in each domain and compare it with chain-of-thought self-consistency [29] given the same number of model calls, on ADAS’s own test sets, with Claude Haiku 5.5 replacing gpt-3.5. ADAS’s search and evaluation code is imported unchanged; we raise the output limit from 4,096 to 16,000 tokens for every agent, score a question 0 instead of aborting the run when an agent raises an error, read every agent’s JSON with the same tolerant parser, run 7 to 16 workers instead of 48, and send no sampling temperature, which the API no longer accepts.

Third round. A third pre-registration, frozen after the second-round results and before any of its runs, addressed three weaknesses of the first two rounds: the output limit that differs between the DGM’s agents and the textbook agent on SWE-bench, the random-diagnosis list, which contained a request aimed at empty responses, and the failure logs that both the engineered seed and the one-shot redesigns had seen. It reruns the SWE-bench comparison at equal output budget (§6.1), makes one-shot redesigns without failure logs (§5.2) and repeats the random-diagnosis loops with a clean request list (§5.3). Every primary contrast of this round is within one batch; Table 9 lists its hypotheses (H11–H15) and verdicts.

5 Results on competitive programming

5.1 Textbook moves

Every combination that contains the model-loop move L gains between +20.0 and +21.6 points over the seed, and every combination without it between −5.2 and −0.2 (Table 3). L bundles four changes (a 16,000-token budget, an answer to every tool call, one re-prompt after a truncated output, and a 40-turn limit); alone it gains +20.9 points. The editor and the shell change nothing measurable on their own, and the workflow prompt alone costs 4.7 points: its longer reasoning runs into the seed’s 4,096-token cap more often.

The registered test (H6). The second pre-registration asks whether the best textbook combination comes within three points of the best first-round loop agent, the one from loop 3. The best of the 15 combinations is chosen on F_1 and compared on F_2 (§4); we also named two single candidates to report on all of F , all four moves and L. Loop 3’s agent was re-evaluated in another batch, so these contrasts cross batches. We report raw contrasts here and seed-adjusted ones in Appendix B. All contrasts in this paragraph are the candidate minus loop 3’s agent, in points.

The combination chosen on F_1 (EBLP) is 4.7 points [0.4, 9.5] behind on F_2 , so H6 is not supported. Over the 8 combinations that contain L, the contrasts on F_2 range from −4.7 to −0.4, and the chosen one happens to be the lowest. On all of F , all four moves are 2.4 points [−0.2,

Table 3: Textbook moves added to the DGM seed (Haiku 5.5; all 232 fresh problems, two runs each, one interleaved batch). E: string-replace editor; B: non-interactive shell; L: model loop with a 16,000-token budget; P: workflow prompt. Gain over the seed in the same batch with 95% intervals; cost in dollars per thousand tasks; share of runs that end with no solution.

moves	F (%)	gain [95% CI]	cost	no solution (%)
(seed)	65.9	–	3.8	27.2
E	65.5	−0.4 [−3.2, +2.6]	3.8	27.6
B	65.3	−0.6 [−3.7, +2.4]	3.0	28.0
P	61.2	−4.7 [−8.4, −1.3]	4.4	30.4
E, B	65.7	−0.2 [−3.2, +2.8]	3.2	26.3
E, P	60.8	−5.2 [−8.8, −1.7]	4.3	31.5
B, P	60.8	−5.2 [−8.6, −1.9]	3.5	31.2
E, B, P	61.4	−4.5 [−7.8, −1.3]	3.5	30.2
L	86.9	+20.9 [+16.2, +25.9]	8.7	4.7
E, L	87.5	+21.6 [+16.8, +26.5]	8.8	3.9
B, L	86.6	+20.7 [+15.9, +25.6]	7.4	4.5
L, P	86.6	+20.7 [+15.7, +25.9]	11.4	5.4
E, B, L	87.3	+21.3 [+16.4, +26.5]	7.4	3.7
E, L, P	86.2	+20.3 [+15.3, +25.4]	10.9	5.0
B, L, P	86.0	+20.0 [+15.3, +25.2]	8.9	5.8
E, B, L, P	86.2	+20.3 [+15.5, +25.4]	9.0	5.4

5.0] behind and L is 1.7 [0.4, 3.2] behind. Of the 8 L combinations, 7 miss the margin and 4 are significantly behind. The literal registered reading chooses the best combination on all of F (EL) and scores it on the same problems, which favours it; its contrast, −1.1 [−2.6, +0.4], is within the margin on raw scores but not after seed adjustment. Every combination is cheaper per task than loop 3’s agent, at most 59% of its cost; L alone costs 45% as much, and all four moves cost 47%. Measured from the seed, the textbook moves thus close about nine tenths of the distance to the best loop agent and remain one to five points behind it: on all fresh problems the contrasts of the L combinations range from −1.1 to −2.6, and on F_2 the chosen combination is at −4.7.

All loops against one move (exploratory). The final batch measured every loop’s reported best agent, the seed, the cap-only seed and L together, two runs per problem (Table 5). There L gains +21.1 points [+16.4, +26.1] over the seed. No loop agent is significantly better than L and 10 of 14 are significantly worse; the best, loop 3’s agent, is +1.3 points [−0.6, +3.4] ahead, so non-inferiority of L within three points is not shown in this batch either. In the third-round batch, loop 3’s agent is 2.6 points [1.1, 4.3] ahead of L, a significant gap (§5.3). Section 3 reports how far the cap-only seed trails loop 3’s agent in the final and ablation batches. The proxy caps every request at 16,000 output tokens, so loop 3’s agent never receives more output than L; what it adds is more attempts after an empty turn. With one more attempt when its first solution fails the sample tests in the problem statement (the rule of the compute-matched baseline above), L reaches 88.4% at \$11.4 per thousand tasks, while loop 3’s agent reaches 88.6% at \$19.3, a contrast of −0.2 points [−1.3, +0.6]. This comparison pairs L’s runs from the grid batch with loop 3’s agent from the ablation batch (Appendix B); it was specified after the H6 result and before it was computed.

Table 4: One-shot redesigns of the DGM seed by a language model (Haiku 5.5 task model; fresh problems, two runs each). Each redesign saw the seed’s code and five failure logs once. Cap: the output cap the redesign set. Opus redesigns are registered; Sonnet redesigns are exploratory.

agent	cap	F (%)	gain [95% CI]	cost
Opus redesign 1	8192	83.8	+19.6 [+14.7, +24.6]	7.6
Opus redesign 2	4096	63.6	−0.6 [−3.2, +1.7]	3.0
Opus redesign 3	8192	86.4	+22.2 [+17.0, +27.4]	7.8
Opus redesign 4	8192	85.8	+21.6 [+16.4, +26.7]	7.6
Opus redesign 5	8192	86.2	+22.0 [+17.0, +27.2]	7.4
Sonnet redesign 1	4096	72.2	+8.0 [+5.2, +11.0]	9.4
Sonnet redesign 2	4096	69.0	+4.7 [+1.5, +8.2]	8.3
Sonnet redesign 3	8192	85.3	+21.1 [+16.4, +26.1]	20.1
Sonnet redesign 4	8192	82.5	+18.3 [+13.4, +23.3]	13.8
Sonnet redesign 5	8192	85.3	+21.1 [+15.7, +26.3]	15.3
DGM seed	4096	64.2	—	3.8
cap-only seed	16000	82.5	+18.3 [+13.4, +23.5]	7.1
engineered seed	16000	86.9	+22.6 [+17.5, +27.8]	9.4

5.2 One-shot redesigns

Four of the five Opus redesigns and three of the five Sonnet redesigns raise the output cap to 8,192 tokens and gain 18 to 22 points; the three that keep 4,096 change the pass rate by -1 to $+8$ points (Table 4). For the registered H8 we take contrasts as redesign minus loop, again across batches (Appendix B). The contrast of the average Opus redesign with the average loop best is $+2.6$ [$+0.6$, $+4.6$], so this part is supported. The best of five, chosen on F_1 , is 4.3 points [1.3 , 7.8] behind loop 3’s agent on F_2 , so that part is not. The best Opus redesign on all of F is 2.2 points [0.4 , 4.3] behind.

Without failure logs. In the third round, five further samples from Claude Opus 5.5 received the redesign prompt without the five failure logs: the agent summary, the seed’s code and the same task paragraph. Four of the five raise the output cap to 8,192 tokens, as four of five did with the logs; the fifth keeps 4,096 but continues after a truncated response and reruns the agent when no solution was written. In one batch with the seed, which scores 64.7%, the log-free redesigns gain 14.4 to 21.6 points, on average $+16.7$ [$+12.5$, $+20.9$], compared with $+16.9$ [$+12.8$, $+21.0$] for the redesigns made from failure logs; the difference is -0.2 points [-1.9 , $+1.5$], so the registered non-inferiority test passes. An exploratory comparison shows that the averages agree for different reasons. Among the redesigns that raise the cap, those made without logs gain 4.0 points [2.2 , 6.1] less than those made from logs. Of the two that keep 4,096, the log-free one gains $+16.6$ points and the log-based one gains $+1.3$. Claude Opus 5.5 thus finds the fix from the code alone when asked for a one-shot redesign, so the failure logs are not needed to find it. In the first round, none of the 24 diagnoses that Claude Sonnet 5.5 made from the agent’s code alone mentions the output cap, stop reasons or empty responses (Appendix F).

5.3 Search without diagnosis, another search rule, more loops

The random-diagnosis loops draw every request from the fixed list of §4; because one request targets empty responses, they search over textbook requests rather than without any diagnosis. In the final batch they gain $+10.6$ points [$+7.5$, $+14.0$] on average, whereas the first-round DGM loops gain

Table 5: Reported best agents of all Haiku 5.5 loops, re-evaluated together with the seed in one batch (all 232 fresh problems, two runs each). S : the selection score for which the loop reported the agent. Inflation: selection gain over the seed’s pooled selection score (62.5% over 254 evaluations) minus fresh gain. Cost in dollars per thousand tasks. The first-round DGM loops and the second-round arms (more DGM loops, random diagnosis, HGM-style) are registered; the loops without hint prompts are exploratory. The last rows are references measured in the same batch.

loop type	loop	S	fresh gain [95% CI]	inflation	cost
DGM	1	76	+7.5 [+4.5, +10.8]	+5.9	11.6
	2	74	+8.4 [+5.4, +11.6]	+3.1	10.7
	3	88	+22.4 [+17.5, +27.4]	+3.1	20.5
DGM, no hints	1	90	+20.9 [+16.2, +25.9]	+6.6	15.7
	2	84	+20.9 [+16.2, +25.6]	+0.6	19.6
DGM (more loops)	1	76	+10.1 [+7.1, +13.4]	+3.4	11.2
	2	76	+9.1 [+6.0, +12.3]	+4.4	6.2
	3	74	+8.0 [+4.7, +11.2]	+3.5	12.7
random diagnosis	1	74	+8.0 [+5.0, +11.2]	+3.5	9.8
	2	74	+11.6 [+7.8, +15.5]	−0.1	12.8
	3	74	+12.3 [+8.6, +16.2]	−0.8	6.3
HGM-style	1	78	+13.4 [+9.5, +17.2]	+2.1	18.6
	2	90	+21.8 [+16.8, +26.7]	+5.7	17.3
	3	80	+15.1 [+11.0, +19.4]	+2.4	13.4
seed	—	—	$F = 65.5\%$	—	3.6
cap-only seed	—	—	+19.8 [+15.1, +24.6]	—	7.6
L (textbook move)	—	—	+21.1 [+16.4, +26.1]	—	8.8

+12.8 [+9.7, +15.9]. The registered non-inferiority test (H7) fails: the difference is $-2.2 [-3.8, -0.4]$. Against all six DGM loops it is $-0.3 [-1.7, +1.1]$. HGM-style parent selection gains +16.7 [+12.8, +20.8], which is 5.8 [3.9, 7.9] points more than the six DGM loops (Table 5). These intervals resample problems (§4). Compared at the level of loops (three or six per arm), none of the arm differences is significant (Welch tests: HGM-style vs DGM $p = 0.15$, random vs first-round DGM $p = 0.70$, random vs all DGM $p = 0.92$). Four of the 14 loops reach the level of the single move L: loop 3, both loops without hint prompts and one HGM-style loop; one of the six DGM loops does.

For each DGM loop the selection gain exceeds the fresh gain, by 3.1 to 5.9 points. With each loop’s own sampling error (tasks resampled, as registered for H2) every interval includes zero, and over the six loops the hierarchical estimate is +3.9 points $[-1.3, +8.8]$. The placebo loops’ best selection scores are given in §3.

With a clean request list. Five loops ran the DGM procedure with every diagnosis replaced by a request drawn from a clean list, and three DGM loops ran at the same time. The clean list is the second-round list with the three failure-related requests described in §4 replaced by three generic ones. The one aimed at empty responses also asked the agent to answer every tool call, which is part of move L. In the third-round batch, which also re-evaluated the earlier loops’ agents and where the seed scores 65.7%, the five clean random-diagnosis loops gain +4.3 points [+2.0, +6.7] on average, with individual loops from -1.1 to +8.4. All nine DGM loops gain +10.8 [+7.8, +14.0], and the three that ran at the same time +11.0 [+7.6, +14.6]. The registered non-inferiority test fails: the clean loops trail the DGM loops by 6.6 points [4.9, 8.3], and at the level of loops the difference is

Table 6: The DGM’s released SWE-bench agents and the initial agent with textbook moves, run through the DGM’s own harness with Claude Sonnet 4.5 on SWE-bench Verified tasks whose reference patch passes today: fresh tasks outside the DGM’s evaluation set, and the DGM’s selection tasks. One run per task. Early: runs, of all 293, in which the agent process ended before the harness recorded a patch; they count as unresolved.

agent	fresh	DGM selection	\$/run	early
DGM initial agent	139/246 (56.5%)	26/47	1.19	22
DGM reported best agent	106/246 (43.1%)	20/47	4.48	142
initial + textbook moves	181/246 (73.6%)	34/47	1.20	11
+ regression-test retries	147/246 (59.8%)	30/47	3.48	58

significant (Welch test, $p = 0.03$). The registered secondary contrast, against the three same-time DGM loops, is -6.7 $[-8.9, -4.7]$ ($p = 0.02$).

The remaining contrasts in this batch are exploratory. The random-diagnosis loops of the second round gain $+11.1$ $[+8.0, +14.4]$. That is 6.9 points $[5.1, 8.9]$ more than the clean loops ($p = 0.03$) and level with all nine DGM loops ($+0.3$ $[-0.9, +1.5]$). The same-time DGM loops reproduce the earlier ones within 0.3 points. Search over generic requests therefore gains much less when no request points at the failure, and the DGM’s diagnosis, which reads the failure logs, does better. The single model-loop move L gains $+19.8$ $[+15.1, +24.8]$ in this batch, more than the average DGM loop but 2.6 points $[1.1, 4.3]$ less than the agent of loop 3.

5.4 What the lineages contain

The loops’ edits depend on each other. Removing one edit from a lineage and re-applying the rest leaves rejected hunks for all but 1 of loop 3’s 6 edits (2 of 4 for loop 2, 0 of 2 for loop 1), so the registered test of whether two edits carry most of the gain (H9) cannot be run. Removing loop 3’s last edit, the only leave-one-out variant that applies cleanly, costs 10.6 points $[6.9, 14.4]$. In an exploratory variant, setting the output cap of loop 3’s agent back to 4,096 tokens while keeping its other code costs only 1.9 points $[0.2, 3.9]$, probably because the agent also retries empty turns with nudges and fallback calls.

6 Published systems

6.1 The DGM’s SWE-bench agents

The DGM reports that its search took the initial agent from 20% to 50% on its 200 SWE-bench tasks with Claude 3.5 Sonnet, which is no longer served. We ran all four agents with Claude Sonnet 4.5 on Verified tasks outside the DGM’s 200-task evaluation set (Table 6); Sonnet 4.5 is newer than the model the DGM used. As run in the DGM’s harness, the reported best agent resolves 13.4 points $[6.1, 20.7]$ fewer fresh tasks than the initial agent it came from, and the initial agent with three textbook moves (defined in §4) resolves 30.5 points $[23.6, 37.0]$ more than the best agent, at about a quarter of its cost per task. The textbook moves alone add 17.1 points $[11.0, 23.2]$ to the initial agent at the same cost. On the DGM’s own selection tasks the textbook agent leads the best agent by 29.8 points $[12.8, 46.8]$.

Most of the best agent’s deficit relative to its initial agent comes from runs that end early. In 142 of its 293 runs the agent process ended before the harness recorded a patch, and the DGM’s harness then discards the run, as it does for every agent. We could not determine the cause for

this model; the exit codes, a scan of the saved logs and two pilots are described in Appendix G. The best agent’s design (up to three attempts, each checked with the repository’s regression tests, plus summarization of long conversations) produces long sessions: its median run took 75 minutes, compared with 9 for the textbook agent. Among the runs that finished, the best agent resolves the largest share of fresh tasks (85.5%; the textbook agent resolves 76.4%), but this comparison conditions on finishing and favours agents that end early on hard tasks.

In an exploratory sensitivity analysis, we graded the patch that each early-ended run left in the repository, as if the harness had kept it. This does not show what a completed run would have produced. Of the best agent’s 142 early-ended runs, 87 had a patch, and 57 of those resolve their task. With these patches kept, the best agent resolves 61.8% of fresh tasks. That is +3.7 points $[-3.3, +10.2]$ relative to the initial agent, and the textbook agent (75.6%) still leads it by 13.8 points $[8.1, 19.5]$. On the DGM’s selection tasks, with patches kept, the textbook agent’s lead shrinks to 8.5 points $[-4.3, 21.3]$, and the best agent is +8.5 $[-8.5, +25.5]$ relative to its initial agent. Under this newer model, then, neither reading of the fresh tasks shows a significant gain for the released best agent over the initial agent it came from, and the textbook agent remains ahead of it. The advantage the DGM reported (from 20% to 50%) does not carry over.

At equal output budget. The DGM’s initial and reported best agents allow at most 4,096 output tokens in each tool-calling request, and the textbook agent allows 16,000. We raised this limit of both DGM agents to 16,000 and changed nothing else (other model calls, such as the best agent’s summaries of its history, keep 4,096 in all three agents), and in the third round ran the three agents in one batch on 65 of the fresh tasks, drawn at random, with half the earlier parallelism. The textbook agent resolves 75.4% of the tasks at \$1.15 per task run, the initial agent resolves 67.7% at \$1.44, and the best agent resolves 41.5% at \$4.77. The textbook agent leads the best agent by 33.8 points $[20.0, 47.7]$ (McNemar $p = 2.7 \times 10^{-5}$), so the registered non-inferiority test passes and the lead is significant, and the best agent is 26.2 points $[12.3, 40.0]$ below its own initial agent ($p = 0.0015$). The initial agent is 7.7 points $[0.0, 16.9]$ below the textbook agent, which is not significant ($p = 0.18$).

The best agent still ends 30 of its 65 runs early, compared with 8 for the initial agent and 4 for the textbook agent, although the machine was far less loaded than in the earlier batch and memory pressure does not explain the endings (Appendix A, items 13 and 14). With the patches of early-ended runs kept, the textbook agent still leads the best agent by 16.9 points $[4.6, 29.2]$, and the best agent is 13.8 points $[0.0, 29.2]$ below its initial agent, which is not significant ($p = 0.11$).

Harness findings. The DGM’s harness runs the task’s evaluation script inside the agent’s container before the agent starts, to install the environment. That script applies the hidden tests, runs them and restores only the test files it modified. Afterwards a pytest cache directory remained in all 106 pytest-based tasks we ran (of 293; the django and sympy tasks use their own test runners); in the three we inspected, its `lastfailed` file named the failing hidden tests: in two it listed exactly the test that the reference patch must make pass, and in the third those 4 tests among 142. Test files that the hidden patch adds also stay in the repository (8 of 300 fresh and 7 of the DGM’s 200 tasks). To our knowledge this has not been reported; the nearest known case is agents reading future commits from the repository’s history [25]. None of the DGM’s released chats reads the cache, so its reported numbers do not appear to use it, but a self-modifying agent could learn to. All our runs remove this residue after setup. Separately, 59 of the 352 tasks we built no longer pass with their reference patch under today’s package versions and were excluded for every agent.

Table 7: ADAS’s best-by-validation discovered agents (found with gpt-3.5) against chain-of-thought self-consistency with a matched number of samples (one extra aggregation call on DROP), both run with Claude Haiku 5.5 on ADAS’s test sets. Calls per question in parentheses. Last column: the gain over CoT-SC that ADAS’s archive reports with gpt-3.5 (for DROP computed from archive medians that lie above their own reported intervals).

domain	ADAS agent	CoT-SC	CoT	agent – CoT-SC [95% CI]	agent – CoT	archive gain
MGSM	93.1 (5)	95.1 (5)	95.4	−2.0 [−3.2, −0.8]	−2.2	+25.2
DROP (F1)	78.0 (10)	84.2 (11)	84.7	−6.1 [−8.1, −4.2]	−6.7	+6.3
GPQA	88.3 (16)	85.1 (16)	83.3	+3.3 [+1.4, +5.4]	+5.1	+1.8
MMLU	90.1 (4)	90.2 (4)	90.6	−0.1 [−1.6, +1.4]	−0.5	+3.2

6.2 ADAS

Run with a current model, ADAS’s discovered agents lose to self-consistency with matched calls on MGSM and DROP, tie on MMLU and win on GPQA (Table 7); a single chain-of-thought call matches or beats them on three of four domains with a quarter to a tenth of the calls. On DROP the discovered agent gives longer answers, which ADAS’s own F1 metric penalizes: on the items where it scores below CoT-SC, its answer is longer in 134 of 142 (median 29 characters, compared with 9). The agents with the best test score in the archive behave the same way (on MGSM it is the same agent). This agrees with Jwalapuram et al. [15], who find automatically designed multi-agent systems behind CoT-SC with current models.

7 Related work

Self-improving agents and automated agent design. The DGM [37] and the Huxley-Gödel Machine [28] let coding agents rewrite their own code and select variants on benchmark scores; ADAS [12], AFlow [39] and agentic harness engineering [19] search over agent designs or workflows; the Gödel Agent [35], SICA [23], STOP [36] and Hyperagents [38] make the improver itself editable. Some recent systems report gains that transfer to held-out benchmarks and models [24, 19].

How such gains should be measured. Selection on a small benchmark inflates reported scores [11, 3]. Harness evolution often fails to beat test-time scaling at matched budgets [30], and LLM-driven program evolution loses most of its advantage over independent sampling at matched token budgets [22]; much of harness evolution’s gain splits into overfitting and extra inference [7]. A pre-registered study finds that prompt and parser search on small models mostly repairs a capped-thinking mismatch that a setting change also repairs [32], close to our cap result. In test generation, once the oracle is audited, equal-budget resampling beats evolution and a placebo feedback loop comes within about two points of real feedback [18]; placebo-controlled retries at matched budgets also match execution feedback on small code models [27], and an evolving harness repairs process failures rather than content failures [26]. Edits in the DGM’s logs repair their target failure no more often than other failures [9]; re-executing the DGM’s SWE-bench evaluations leaves its decisions unchanged [6]; and few published self-improvement studies compare improvers at matched budgets from identical starting points [4]. Meta-agents rarely match human-engineered baselines [20], while a one-shot rewrite did not explain the gains of one open-ended optimizer [33], in contrast to our one-shot redesigns. Automatically designed multi-agent systems lose to CoT-SC with current models [15]. For DGM-style self-modifying loops, our study adds exact-copy placebo loops, textbook-move

and one-shot-redesign baselines, search over a list of textbook requests, and a re-run of the DGM’s own released agents on fresh tasks.

Baselines and evaluation hygiene. Random search matched neural architecture search under equal budgets [17, 34], and seed variance changed conclusions in deep reinforcement learning [10]; agent evaluations should also control cost [16], and repeated sampling or self-consistency are strong references [2, 29]. SWE-bench [14, 5] has known leaks and weak tests [25, 21, 1, 31], agentic benchmarks often have setup and reward flaws [41], and agents exploit visible checks when they can [8, 40].

8 Discussion

Tables 8 and 9 list the registered hypotheses and their outcomes, and the findings in the introduction summarize them together with the exploratory comparisons. Read together, they show that the loops mostly found standard engineering, which a one-shot redesign also reaches at a lower cost per task whether or not it sees failure logs; they do not show that loops find nothing.

Two readings follow. The first concerns what search contributes. In most of our settings, a baseline that uses no search (textbook moves, or a single redesign from the same failure logs) reached the level of the loops’ best agents, or came within a few points of it, at lower cost. The exceptions run both ways: on gpt-oss-120b an agent found by a loop on another model gained more than our textbook set did, in a comparison across batches, and ADAS’s discovered agent wins on GPQA. Diagnosis also carries information: once the request list contains nothing aimed at the empty-turn failure, random-diagnosis loops gain much less than DGM loops (§5.3). The loops’ costs are many evaluations and inflated selection scores (§3). The second reading concerns transfer to a new task model. A harness is tuned to the model it was built for, whether by a loop or directly: the engineered seed loses to the DGM seed under gpt-oss-120b, and it is ahead of loop 3’s agent under Haiku 4.5 and Sonnet 4.6, about level under Sonnet 5.5, and behind it under gpt-oss-120b and Gemini 2.5 Flash (Appendix F). Published self-improvement results should therefore be re-measured when the task model changes.

Recommendations. Claims of self-improvement should report (i) gains on tasks that no selection step saw, (ii) a placebo loop or its order-statistics prediction, (iii) the seed resampled at the agent’s cost, (iv) the seed with textbook moves and a one-shot redesign from the same failure logs, measured in the same batch as the loop’s agent (our own registered tests did not fully meet this), and (v) the released agent re-run with a newer model. Harnesses should keep hidden tests out of the agent’s container, keep the agent’s work when its process ends early, and check that each task’s reference patch still passes.

Limitations. Loops were run on one task family with one to nine loops per arm, and the confidence intervals over problems hold the realized loops fixed. At the level of loops, differences between loop types are significant only for the clean random-diagnosis loops of the third round (§5.3). The SWE-bench and other-model evaluations use one run per task, and budget and time limits censor several agents under other models (Appendix E). Claude 3.5 Sonnet, the DGM’s original model, is no longer served, so we cannot separate the effect of the model change from a failure to replicate. We could not determine what ends the best agent’s runs early under Claude Sonnet 4.5; machine load and memory do not explain it (§6.1). The SWE-bench batch at equal output budget used 65 tasks. HGM’s released agent and the DGM’s Polyglot agents were not run. The proxy caps

output at 16,000 tokens for every arm. The engineered seed, the source of the textbook moves, was written while pilot runs that included 19 of the fresh problems took place, and their logs showed the DGM seed’s empty responses, so the textbook moves are not independent of what failure logs show; without those problems the main numbers barely change (Appendix C). The request list of the second-round random-diagnosis loops was drafted after the first round had exposed the empty-turn failure, and three of its requests aim at or near it (§4); the third round repeats the test without them (§5.3). Several comparisons were added after earlier results (Appendix A, items 3 and 15); those of item 3 were specified before they were computed, and those of item 15 after they were computed. Under Haiku 4.5 most runs of the DGM seed end with its agent process killed by a signal, which the engineered seed avoids (Appendix D). Heavy load on the shared machine affected part of one exploratory batch and the last generations of three loops, and the second-round SWE-bench run used an oversubscribed machine (Appendix A).

Table 8: Pre-registered hypotheses of the first two rounds (H1–H5, H6–H10), the estimate with its 95% interval and the verdict. (a) marks amendments made before the results; (p) marks readings or tests chosen after the results. Contrasts are candidate minus comparator. Contrasts across batches are raw; seed-adjusted values are in Table 10.

hypothesis (as registered)	estimate	verdict
H1 placebo loops show a positive selection gain of predictable size	first-round best scores inside leave-one-out predictive intervals; six loops 68–70% vs 69% predicted	supported
H2 DGM loops’ selection gain exceeds their fresh gain; fresh gain positive	per-loop excess (tasks resampled) 3.1 to 5.9 points, every interval includes 0; six loops (hierarchical) +3.9 [−1.3, +8.8]; fresh gains all positive	not supported
H3 best log-free edit reaches $\geq 0.8\times$ the loops’ fresh gain	(a) on F_1 : ratio 0.23	not supported
H4 from the engineered seed, fresh gains of all arms within ± 3 points, selection gains > 0	DGM loops: +2.4 [90% CI +0.5, +4.3] on F (p: 90% two one-sided tests) and +1.8 [+0.1, +4.2] on F_1 ; single-edit arms (F_1): +1.2 [−1.0, +3.7] and −4.1 [−9.5, +0.1]; selection gains of the loops +7.4 and +7.4	not supported (equivalence not shown)
H5 evolved-agent gains shrink or vanish under Haiku 4.5 and Sonnet 4.6	(a) on F_1 , loops 1 and 3 (loop 2’s agent not run); change of gain, loop 3’s agent: −7.7 [−18.0, +2.8], −11.1 [−20.8, −2.1]; loop 1’s: −7.7 [−16.1, +0.6], −3.4 [−12.4, +5.4]	supported only for loop 3 under Sonnet 4.6
H6 best textbook combination $\geq$ best first-round loop agent −3 (and cheaper)	(a) chosen on F_1 (EBLP), on F_2 : −4.7 [−9.5, −0.4]; literal, best on all of F (EL; chosen and scored on the same problems): −1.1 [−2.6, +0.4], seed-adjusted −0.6 [−3.9, +2.6]; (a) pre-specified single candidates on all of F : all four moves −2.4 [−5.0, +0.2], L −1.7 [−3.2, −0.4]; all cheaper	not supported; literal reading supported on raw scores, not after seed adjustment
H7 random-diagnosis loops $\geq$ first-round DGM loops −3 (mean fresh gain)	−2.2 [−3.8, −0.4]; (p) vs six DGM loops −0.3 [−1.7, +1.1]	not supported
H8 best of five one-shot redesigns $\geq$ best loop agent −3; mean redesign $\geq$ mean loop best −3	(a) best on F_2 : −4.3 [−7.8, −1.3]; literal, best on all of F : −2.2 [−4.3, −0.4]; mean: +2.6 [+0.6, +4.6]	best: not supported; mean: supported
H9 two edits carry $\geq 80\%$ of a loop’s fresh gain	loop 1 has exactly two edits; for loop 2 half and for loop 3 all but one of the leave-one-out variants do not apply	not testable (loops 2, 3)
H10 seed + textbook moves (+ retry) $\geq$ published best −3 on fresh tasks	(a; pilot results known) DGM SWE-bench with Sonnet 4.5, moves: +30.5 [+23.6, +37.0] as run, +13.8 [+8.1, +19.5] with early-ended patches kept; moves and retries: +16.7 [+9.3, +24.0] and +9.8 [+3.7, +15.9]; ADAS (CoT-SC vs agent): holds on MGSM, DROP, MMLU, fails on GPQA; HGM and DGM Polyglot not run	(a) supported for DGM SWE-bench; 3 of 4 for ADAS

Table 9: Pre-registered hypotheses of the third round (H11–H15), the estimate with its 95% interval and the verdict. Every contrast is within one batch.

hypothesis (as registered)	estimate	verdict
H11 at equal output budget, seed + textbook moves $\geq$ DGM best agent -3 (SWE-bench)	as run $+33.8$ [$+20.0$, $+47.7$]; early-ended patches kept $+16.9$ [$+4.6$, $+29.2$]	supported (also superior)
H12 at equal output budget, DGM best agent vs its initial agent (expected: no significant gain)	as run -26.2 [-40.0 , -12.3]; patches kept -13.8 [-29.2 , 0.0]	no gain; worse as run
H13 (descriptive) textbook agent vs initial agent with the raised limit	$+7.7$ [0.0 , $+16.9$]	–
H14 clean random-diagnosis loops $\geq$ all DGM loops -3 (mean fresh gain)	-6.6 [-8.3 , -4.9]	not supported
H15 log-free redesigns: ≥ 4 of 5 raise the cap; mean gain $\geq$ log-based -3	4 of 5; -0.2 [-1.9 , $+1.5$]	supported

Code and data availability

Code, the three pre-registrations, the change log and per-task results are available at <https://doi.org/10.5281/zenodo.23720890>; the same files are on GitHub at <https://github.com/tejasvi/self-improving-agents-textbook-baselines> (tag v4).

Acknowledgements

Generative AI tools were used to write code, run analyses and draft text; the author directed the work and takes full responsibility for its content.

References

- [1] Reem Aleithan, Haoran Xue, Mohammad Mahdi Mohajer, Elijah Nnorom, Gias Uddin, and Song Wang. SWE-Bench+: Enhanced coding benchmark for LLMs. *arXiv preprint arXiv:2410.06992*, 2024.
- [2] Bradley Brown, Jordan Juravsky, Ryan Ehrlich, Ronald Clark, Quoc V. Le, Christopher Ré, and Azalia Mirhoseini. Large language monkeys: Scaling inference compute with repeated sampling. *arXiv preprint arXiv:2407.21787*, 2024.
- [3] Gavin C. Cawley and Nicola L. C. Talbot. On over-fitting in model selection and subsequent selection bias in performance evaluation. *Journal of Machine Learning Research*, 11:2079–2107, 2010.
- [4] Zihan Chen, Di Zhu, and Lei Nico Zheng. Auditing the product, not the recursion: The evaluation evidence behind recursive self-improvement claims. *arXiv preprint arXiv:2610.27610*, 2026.
- [5] Neil Chowdhury, James Aung, Jun Shern Chan, Oliver Jaffe, Dane Sherburn, Giulio Starace, Evan Mays, Rachel Dias, Marwan Aljubei, Mia Glaese, Carlos E. Jimenez, John Yang, Leyton Ho, Tejal Patwardhan, Kevin Liu, and Aleksander Madry. Introducing SWE-bench Verified. OpenAI blog post, <https://openai.com/index/introducing-swe-bench-verified/>, August 2024. Published 13 August 2024, updated 24 February 2025.
- [6] Tianyu Ding, Yaocheng Tong, Muzhi Yang, Fei Jiang, Lei Yang, and Boya Yu. Re-executing coding-agent evaluations: Unchanged decisions and unresolved policy risk. *arXiv preprint arXiv:2610.14460*, 2026.
- [7] Wenxuan Ding, Qin Lu, Changlong Yu, Sha Li, Shuwei Jin, Xin Liu, Zhenyu Shi, and Greg Durrett. Harness-Delta Attribution: Dissecting gains from harness evolution. *arXiv preprint arXiv:2610.22466*, 2026.
- [8] Jonathan Gabor, Jayson Lynch, and Jonathan Rosenfeld. EvilGenie: A reward hacking benchmark. *arXiv preprint arXiv:2511.21654*, 2025.
- [9] Rong He. Does failure feedback direct agent self-modification? *arXiv preprint arXiv:2610.14662*, 2026.

- [10] Peter Henderson, Riashat Islam, Philip Bachman, Joelle Pineau, Doina Precup, and David Meger. Deep reinforcement learning that matters. *Proceedings of the AAAI Conference on Artificial Intelligence*, 32(1), 2018. doi: 10.1609/aaai.v32i1.11694.
- [11] Litao Hu and Yutong Tang. The winner’s curse in LLM self-improvement loops: Selection noise, lock-in, and acceptance rules. *arXiv preprint arXiv:2610.09239*, 2026.
- [12] Shengran Hu, Cong Lu, and Jeff Clune. Automated design of agentic systems. In *International Conference on Learning Representations*, 2025.
- [13] Naman Jain, King Han, Alex Gu, Wen-Ding Li, Fanjia Yan, Tianjun Zhang, Sida Wang, Armando Solar-Lezama, Koushik Sen, and Ion Stoica. LiveCodeBench: Holistic and contamination free evaluation of large language models for code. In *International Conference on Learning Representations*, pages 58791–58831, 2025.
- [14] Carlos E. Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik Narasimhan. SWE-bench: Can language models resolve real-world Github issues? In *International Conference on Learning Representations*, pages 54107–54157, 2024. URL <https://openreview.net/forum?id=VTF8yNQm66>.
- [15] Prathyusha Jwalapuram, Hehai Lin, Chuyuan Li, Fangkai Jiao, Sudong Wang, Yifei Ming, Zixuan Ke, Chengwei Qin, Giuseppe Carenini, and Shafiq Joty. The illusion of multi-agent advantage. *arXiv preprint arXiv:2606.13003*, 2026.
- [16] Sayash Kapoor, Benedikt Stroebl, Zachary S. Siegel, Nitya Nadgir, and Arvind Narayanan. AI agents that matter. *arXiv preprint arXiv:2407.01502*, 2024.
- [17] Liam Li and Ameet Talwalkar. Random search and reproducibility for neural architecture search. In Ryan P. Adams and Vibhav Gogate, editors, *Proceedings of The 35th Uncertainty in Artificial Intelligence Conference*, volume 115 of *Proceedings of Machine Learning Research*, pages 367–377. PMLR, 2020. URL <https://proceedings.mlr.press/v115/li20c.html>.
- [18] Yunhao Liang, Chengguang Gan, Ruixuan Ying, Hanjun Wei, Zhe Cui, and Shiwen Ni. Auditing and decomposing feedback-driven evolution in LLM test generation under the oracle problem. *arXiv preprint arXiv:2608.19626*, 2026.
- [19] Jiahang Lin, Shichun Liu, Chengjun Pan, Lizhi Lin, Shihan Dou, Zhiheng Xi, Xuanjing Huang, Hang Yan, Zhenhua Han, Tao Gui, and Yu-Gang Jiang. Agentic Harness Engineering: Observability-driven automatic evolution of coding-agent harnesses. *arXiv preprint arXiv:2604.25850*, 2026.
- [20] Xinyu Lu, Tianshu Wang, Pengbo Wang, Zujie Wen, Zhiqiang Zhang, Jun Zhou, Boxi Cao, Yaojie Lu, Hongyu Lin, Xianpei Han, and Le Sun. The Meta-Agent Challenge: Are current agents capable of autonomous agent development? *arXiv preprint arXiv:2606.04455*, 2026.
- [21] Nikolai Ludwig, Wasi Uddin Ahmad, Somshubra Majumdar, and Boris Ginsburg. Shortcutting the fix: Identifying and categorizing agentic exploits in software engineering benchmarks. *arXiv preprint arXiv:2609.06780*, 2026.
- [22] Tal Oved, Roi Pony, Oshri Naparstek, and Udi Barzelay. LLM evolutionary search strategies should be compared by tokens. *arXiv preprint arXiv:2610.23977*, 2026.

- [23] Maxime Robeyns, Martin Szummer, and Laurence Aitchison. A self-improving coding agent. *arXiv preprint arXiv:2504.15228*, 2025.
- [24] Dhruv Srikanth, Bingchen Zhao, Dixing Xu, Yuxiang Wu, and Zhengyao Jiang. Recursive self-improvement of AI research agents. *arXiv preprint arXiv:2609.26457*, 2026.
- [25] SWE-bench GitHub issue #465. Repo state loopholes during agentic evaluation. <https://github.com/SWE-bench/SWE-bench/issues/465>, 2025. Opened 3 September 2025.
- [26] Yuan Tian, Bing Hu, Hao Wang, Binghang Lu, and Fang Wu. Harness evolution hits a ceiling: When weight training should begin. *arXiv preprint arXiv:2610.11655*, 2026.
- [27] Yuvraj Verma. Try again, don’t look back: Blind resampling outperforms self-repair in small code models. *arXiv preprint arXiv:2607.26117*, 2026.
- [28] Wenyi Wang, Piotr Piękos, Li Nanbo, Firas Laakom, Yimeng Chen, Mateusz Ostaszewski, Mingchen Zhuge, and Jürgen Schmidhuber. Huxley-Gödel Machine: Human-level coding agent development by an approximation of the optimal self-improving machine. In *International Conference on Learning Representations*, pages 80356–80386, 2026.
- [29] Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc V. Le, Ed H. Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. Self-consistency improves chain of thought reasoning in language models. In *International Conference on Learning Representations*, 2023. URL <https://openreview.net/forum?id=1PL1NIMMrw>.
- [30] Yike Wang, Huaisheng Zhu, Zhengyu Hu, Yige Yuan, Zhengyu Chen, Shakti Senthil, Hannaneh Hajishirzi, Yulia Tsvetkov, Noah A. Smith, Pradeep Dasigi, and Teng Xiao. Rethinking the evaluation of harness evolution for agents. *arXiv preprint arXiv:2607.12227*, 2026.
- [31] You Wang, Michael Pradel, and Zhongxin Liu. Are “solved issues” in SWE-bench really solved correctly? an empirical study. *arXiv preprint arXiv:2503.15223*, 2025.
- [32] Bowen Xu and Boyu Chen. What does a harness repair? a preregistered study of visibility, baseline adequacy and evaluation defects. *arXiv preprint arXiv:2610.05533*, 2026.
- [33] Hui Xue and Fan Yang. Rethinking self-evolving agents: Do we still need prescribed optimization pipelines? *arXiv preprint arXiv:2608.09629*, 2026.
- [34] Antoine Yang, Pedro M. Esperança, and Fabio M. Carlucci. NAS evaluation is frustratingly hard. In *International Conference on Learning Representations*, 2020. URL <https://openreview.net/forum?id=HygrdpVKvr>.
- [35] Xunjian Yin, Xinyi Wang, Liangming Pan, Li Lin, Xiaojun Wan, and William Yang Wang. Gödel agent: A self-referential agent framework for recursively self-improvement. In Wanxiang Che, Joyce Nabende, Ekaterina Shutova, and Mohammad Taher Pilehvar, editors, *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 27890–27913, Vienna, Austria, 2025. Association for Computational Linguistics. doi: 10.18653/v1/2025.acl-long.1354. URL <https://aclanthology.org/2025.acl-long.1354/>.
- [36] Eric Zelikman, Eliana Lorch, Lester Mackey, and Adam Tauman Kalai. Self-Taught Optimizer (STOP): Recursively self-improving code generation. *arXiv preprint arXiv:2310.02304*, 2023.

- [37] 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*, pages 104223–104294, 2026.
- [38] Jenny Zhang, Bingchen Zhao, Wannan Yang, Jakob Foerster, Jeff Clune, Minqi Jiang, Sam Devlin, and Tatiana Shavrina. Hyperagents. *arXiv preprint arXiv:2603.19461*, 2026.
- [39] Jiayi Zhang, Jinyu Xiang, Zhaoyang Yu, Fengwei Teng, Xionghui Chen, Jiaqi Chen, Mingchen Zhuge, Xin Cheng, Sirui Hong, Jinlin Wang, Bingnan Zheng, Bang Liu, Yuyu Luo, and Chenglin Wu. AFlow: Automating agentic workflow generation. In *International Conference on Learning Representations*, pages 34040–34077, 2025.
- [40] Ziqian Zhong, Aditi Raghunathan, and Nicholas Carlini. ImpossibleBench: Measuring LLMs’ propensity of exploiting test cases. In *International Conference on Learning Representations*, pages 124959–124986, 2026.
- [41] Yuxuan Zhu, Tengjun Jin, Yada Pruksachatkun, Andy Zhang, Shu Liu, Sasha Cui, Sayash Kapoor, Shayne Longpre, Kevin Meng, Rebecca Weiss, Fazl Barez, Rahul Gupta, Jwala Dhamala, Jacob Merizian, Mario Giulianelli, Harry Coppock, Cozmin Ududec, Jasjeet Sekhon, Jacob Steinhardt, Antony Kellermann, Sarah Schwettmann, Matei Zaharia, Ion Stoica, Percy Liang, and Daniel Kang. Establishing best practices for building rigorous agentic benchmarks. *arXiv preprint arXiv:2507.02825*, 2025.

A Changes after pre-registration

The three pre-registrations and the full log of changes (DEVIATIONS.md) are in the release; this is a summary. The pre-registrations call the engineered seed the strong or hand-written seed (`seed_strong` in the released files).

- (1) The first launch was aborted because proxy restarts dropped model calls.
- (2) The first grader shadowed the built-in `pow`; the campaign was relaunched with a corrected grader, and five old-grader loops were kept for the grader analysis.
- (3) Exploratory comparisons added after earlier results: the loops without hint prompts, the cap-only seed, the compute-matched resampling baseline and its follow-up for L, the reversal of the output cap in loop 3’s agent, the Sonnet redesigns, the loops and textbook moves with Claude Haiku 4.5, the textbook moves, one-shot redesigns and resampled seed under gpt-oss-120b, the first-round transfer runs under Claude Sonnet 5.5, Gemini 2.5 Flash and gpt-oss-120b, the ADAS agents with the best test score, and the SWE-bench sensitivity analysis. Each was specified before it was computed.
- (4) The rule that chooses a best candidate on F_1 and compares it on F_2 (H6, H8) was fixed after the second pre-registration, while the grid batch was running (about three quarters done) and before any result was summarized (we had only glanced at the first 100 runs for crashes); the literal reading is reported as well.
- (5) H6, H8 and the L comparison pair agents from different batches. The pre-registration’s same-batch seeds “to control drift” and the pre-results rule that comparisons use same-batch references point to seed-adjusted contrasts, while the pre-results rule that names loop 3’s agent re-evaluated “in the same period” as the comparator fits raw ones; none of them says explicitly how such contrasts are formed. The analysis code, written before the comparator’s batch finished, computed raw contrasts, and seed-adjusted contrasts were added after the raw contrasts had been computed. Verdicts are given under both; they differ only for the literal reading of H6, which fails under the seed-adjusted contrast.
- (6) H2: the round-2 summary over six loops is given with the registered per-loop task bootstrap and a hierarchical bootstrap; an earlier analysis used a bootstrap over loop means only.
- (7) The random-diagnosis list of the second round was written after the first round had identified the empty-turn failure (§4); the third round replaced the requests aimed at it (§5.3).
- (8) Pilot overlap with the fresh problems: Appendix C.
- (9) Proxy spending caps were raised during the runs; evolved agents get one fresh run per problem in the first round; the improver swap used 12 diagnoses instead of 16.
- (10) H9 could not be run as registered for loops 2 and 3, because half (loop 2) or all but one (loop 3) of the leave-one-out variants do not apply. H3 and H5 are computed on F_1 instead of F , because the single edits and the transfer runs were evaluated on F_1 only; H5 covers loops 1 and 3, because loop 2’s agent was not run under the other models (to limit cost). H4 is reported for every arm started from the engineered seed.
- (11) In the registered SWE-bench test of the DGM, Claude Haiku 4.5 was replaced by Claude Sonnet 4.5 after a pilot in which most runs of both DGM agents ended early; the outcomes of a 15-task pilot with Claude Sonnet 4.5 (Appendix G), including resolve rates, were known when the model and protocol of the full run were fixed; the registered Claude Haiku 5.5 runs

were not made; the third textbook move is a workflow prompt rather than a loop change; the retry arm costs less than the best agent instead of the same; tasks whose reference patch fails today were excluded for all agents; hidden-test residue is removed from the agent’s container after setup.

- (12) The registered test of HGM’s released agent was not run, because the agent supports only OpenAI’s Responses API, and the registered test of the DGM on its Polyglot benchmark was not run.
- (13) Part of one exploratory batch and the selection runs of three loops near their end ran during heavy load on the shared machine (re-grading that batch’s saved solutions changed 5 of 1,624 outcomes); later batches ran on a separate machine. During the second-round SWE-bench run the machine’s CPU was oversubscribed (per-run mean load up to 167 on 64 cores). Load was sampled every minute over the whole run in 67% of runs and over part of it in 5%, and interpolated from checks every 30 minutes for the rest, including 170 of the textbook agent’s 293 runs. By load tertile (low, middle, high), the early-ending rates were: initial agent 7, 5 and 10%; best agent 49, 51 and 45%; textbook agent 3, 6 and 2%; retry variant 26, 16 and 16%. The resolve rate of the textbook agent fell from 81% to 66%, which works against it. The tertiles are formed per agent and are confounded with run order, so these figures do not rule out an effect of load. In the third round the one-minute load average stayed below 18 (in the second round it reached 200), and the best agent still ended 30 of 65 runs early, compared with 34 on the same tasks in the second round (§6.1).
- (14) Third round: the loops ran with eight task workers each instead of twelve (parallelism only) on a spot machine that was preempted three times during the runs, twice during the loops and once during the third-round batch. Each time the jobs resumed from their saved state, and an unfinished child or run was redone. The spending cap of that machine was raised by \$40 before the third-round batch, and its third-round spend was \$331.66, against the \$330 registered. The SWE-bench machine reached the internet only through address translation, and the requirement files that the swebench code reads from GitHub were served from a local copy of the same files. The third-round SWE-bench run used eight task containers per agent, half the earlier parallelism. Comparisons with the second round’s SWE-bench runs on the same tasks are exploratory, because they cross batches. In the third-round SWE-bench run, the early endings of the best agent were 27 on a signal and 3 with exit code 1, all those of the other two agents were on a signal, and 19 of the best agent’s tasks end early in both rounds. The kernel logged one out-of-memory kill; none of the 8 runs then in progress ended early within ten minutes of it, and 2 of them ended early at least 62 minutes later. With the patches of early-ended runs kept, the three agents resolve 76.9%, 73.8% and 60.0% (textbook, initial, best), and the initial agent is 3.1 points below the textbook agent ($p = 0.73$). In an exploratory comparison of the same tasks between the second-round batch and this one, the rate of the best agent changes by +1.5 points $[-12.3, +15.4]$ and that of the initial agent by +6.2 $[-6.2, +18.5]$, while the textbook agent reproduces its earlier rate within 1.5 points; the two batches differ in the output limit, the parallelism and the machine load.
- (15) Third round: exploratory analyses first computed during internal review and then added: the second-round random-diagnosis loops against the DGM loops in the third-round batch, loop 3’s agent against L in that batch, the redesigns that raise the cap with and without logs, the runs in progress at the out-of-memory kill, and the second-round early endings on the same 65 tasks.

B Contrasts across batches

H6, H8 and the comparison of L with one retry pair agents from different batches. Loop 3’s agent was re-evaluated in the ablation batch (23:19 to 04:05 UTC), which started together with the grid batch of the 16 combinations (23:15 to 00:32 UTC) and was about a quarter done when the grid ended; the redesign batch ran within the ablation batch’s hours (00:11 to 03:43 UTC). Relative to the seed in the grid batch, the seed in the ablation batch scores +0.4 points $[-2.4, +3.2]$ on all of F . Relative to the seed in the redesign batch, it scores +3.0 $[-0.4, +6.9]$ on F_2 and +2.2 $[-0.6, +5.0]$ on all of F , so raw contrasts are less favourable to the redesigns than seed-adjusted ones by these amounts. A seed-adjusted contrast is (candidate minus its batch’s seed) minus (comparator minus its batch’s seed), paired by problem; item 5 of Appendix A explains why both are given. Table 10 lists both. The verdicts agree except for the literal reading of H6, which passes on raw scores and fails after seed adjustment.

Table 10: Contrasts across batches, raw and seed-adjusted (candidate minus comparator, in points, with 95% intervals).

contrast	problems	raw	seed-adjusted
H6: combination chosen on F_1 (EBLP) vs loop 3’s agent	F_2	−4.7 $[-9.5, -0.4]$	−4.3 $[-10.8, +1.7]$
H6, literal: best combination on F (EL) vs loop 3’s agent	F	−1.1 $[-2.6, +0.4]$	−0.6 $[-3.9, +2.6]$
all four moves vs loop 3’s agent	F	−2.4 $[-5.0, +0.2]$	−1.9 $[-5.8, +1.9]$
L vs loop 3’s agent	F	−1.7 $[-3.2, -0.4]$	−1.3 $[-4.5, +1.9]$
H8: average Opus redesign vs average loop best	F	+2.6 $[+0.6, +4.6]$	+4.7 $[+1.3, +8.2]$
H8: best of five Opus redesigns (chosen on F_1) vs loop 3’s agent	F_2	−4.3 $[-7.8, -1.3]$	−1.3 $[-6.5, +3.9]$
H8, literal: best Opus redesign on F vs loop 3’s agent	F	−2.2 $[-4.3, -0.4]$	0.0 $[-3.4, +3.4]$
L with one retry vs loop 3’s agent	F	−0.2 $[-1.3, +0.6]$	+0.2 $[-2.8, +3.2]$

C Pilot overlap

The engineered seed, and with it the textbook moves E, B, L and P, was written while two short pilot runs of the DGM seed took place, and their logs were read as it was written. The second pilot covered 80 problems, 19 of which are in F (and 7 in the selection set); the first used a subset of them. Of the second pilot’s 160 runs, 16 ended without a solution, 13 of them on an empty response, the failure that L repairs. L is therefore a repair informed by failure logs, like a one-shot redesign, rather than a move chosen without them. Without those problems, L gains +20.7 points $[+15.7, +26.1]$ in the final batch, no loop agent is significantly better than L and 10 are significantly worse (best: +1.2 $[-0.9, +3.3]$), and the H6 contrast on F_2 is −3.8 $[-8.6, +0.5]$.

D Other task models

These comparisons are exploratory.

Claude Haiku 4.5. This model rarely reaches the seed’s cap (0.3% of the seed’s responses stop at it, among runs with logs), but another failure takes its place: in 70% of the seed’s runs the

agent process is killed by a signal before it finishes (72% in the first-round transfer runs; 0.1% under Haiku 5.5). The chat logs of these runs are empty, because the seed writes its log on exit; in the SWE-bench pilot with the same model, such endings came from cleanup commands that the agent itself ran (Appendix G). A batch of single textbook moves under this model, in which the seed scores 28.0%, separates the causes. The non-interactive shell B removes the signal endings entirely (0% of runs, compared with 70% for the seed) and gains +5.6 points [+0.9, +10.3]; the editor, the model loop and the workflow prompt alone show no gain (−2.2, −3.9 and −4.3 points), and their runs still end on a signal in about three quarters of cases. All four moves together gain +12.5 points [+7.8, +17.7], 6.9 more than the shell alone. Of the two loops, one found nothing (+0.4 [−4.7, +5.6] in the final batch; 73% of its runs end on a signal). The other gained +13.4 points [+8.2, +19.0] at \$671.4 per thousand tasks with four edits that rebuild parts of the engineered seed’s workflow (repeated attempts with the previous failure in context, a brute-force reference with random stress tests, timing on a maximum-size input, and a rule never to stop with a known time-out); 29% of its runs end on a signal, and others reach the budget or the time limit (Appendix E). The seed resampled eight times, at a lower cost than that agent, is 11.3 points [5.7, 17.4] below it on F_1 , so its gain is not a compute effect alone. The engineered seed, which combines the shell fix with the workflow, is +2.2 points [−2.6, +6.9] ahead of it at a lower cost in the final batch, where the seed scores 28.0%. The Haiku 4.5 placebo loop’s best selection score (34%) lies between those of the two loops (28% and 40%).

gpt-oss-120b. This model is slow: depending on the agent, 14 to 100% of its runs reach the 600s limit, and the solution present at that point is graded (Appendix E). In the final batch for this model the DGM seed solves 42.7% of the fresh problems; the engineered seed scores 10.3 points below it. One of three loops gained +8.2 points [+3.9, +12.5], with a workflow prompt and the rule to write the solution file at once and never end a turn without it; its best agent costs about three times as much per task as the seed, and the seed resampled 5 times at a similar cost reaches 57.7% on F_1 , and the contrast of the loop agent with it there is −4.3 points [−12.3, +3.4]. A second loop’s reported agent has a fresh gain of −2.6 [−8.6, +3.4] and reaches the 600s limit in 100% of its runs, and the third never got a child through the full selection set; neither did any of the placebo loop’s 40 copies. In a separate batch of single textbook moves, in which the seed scores 41.8%, the moves as written for Haiku 5.5 do not transfer: the editor alone scores 24.1 points below the seed, the move chosen on F_1 (L) gains +3.0 [−3.4, +9.5], and the loop’s agent, measured in the same batch, is ahead of that move by 7.8 [0.0, 15.5] on F_2 . Loop 3’s agent from the first round, evolved with Haiku 5.5, gained +22.4 points [+12.9, +31.9] on F_1 in the first-round transfer runs (Appendix F), more than any textbook move gains in the batch above; the two numbers come from different batches. This is consistent with the retries after empty turns that the loop found transferring where our textbook set does not, but we did not ablate them under this model. In a third batch we evaluated five one-shot redesigns made from five failure logs of this model. The best of five, chosen on F_1 , beats the loop’s agent on F_2 by +9.5 points [+1.7, +17.2]; the average of the five trails it by 8.9 [4.7, 13.1], because one redesign was broken and solved nothing. Choosing the best of five used 116 evaluated problems, more than the loops’ 50-problem selection set.

E Time and budget limits

This appendix gives the shares of runs that end on a signal, reach the wall-clock limit, or spend at least 99% of the per-run budget, for the batches in which any of these is common. In the final batch under Claude Haiku 4.5 (\$1 budget), the DGM seed ends on a signal in 70% of runs (72%

in the transfer runs) and the engineered seed in 0%; the engineered seed reaches the budget in 8%. Loop 1’s agent ends on a signal in 73% of runs and reaches the time limit in 26%. Loop 2’s agent ends on a signal in 29%, reaches the time limit in 30% and the budget in 36%. In the first-round transfer runs, loop 3’s agent reaches the \$1 budget in 75 of 116 runs under Claude Haiku 4.5 and in 50 under Claude Sonnet 4.6, and the DGM seed ends on a signal in 11% of its Sonnet 4.6 runs. Under gpt-oss-120b no run ends on a signal, and the share of runs that reach the 600 s limit ranges from 47 to 100% across agents in the final batch, from 14 to 60% in the textbook-move batch and from 28 to 94% in the redesign batch. In the Haiku 5.5 final batch at most 1% of runs end on a signal; time-outs reach 22% for one loop agent and are under 10% for every other agent. Pass rates under these limits are censored from above.

F First-round details

Table 11: Single edits of the DGM seed and log-free chains (the diagnosis sees only the agent’s code, no task log). Fresh gain on F_1 averaged over evaluated members, the same average counting children that failed to compile as unchanged, and the member with the best selection score.

diagnosis given to the editing step	tried	failed	evaluated	mean gain [95% CI]	with failures	best-by- S
agent code only (no task data)	24	2	22	+2.3 [+0.4, +4.2]	+2.1	+3.4
task log, no hint prompts	24	0	24	+1.6 [−0.7, +4.1]	+1.6	+0.8
DGM diagnosis (task log and hint prompts)	24	9	15	+6.3 [+3.4, +9.5]	+3.9	+10.3
log-free chain 1 (steps)	10	8	2	+2.9 [−1.1, +6.6]	+0.6	+4.2
log-free chain 2 (steps)	10	6	4	+5.7 [+1.9, +9.8]	+2.3	+7.7

Single edits. A diagnosis model that sees only the agent’s code proposes plausible generic changes; none of its 24 diagnoses mentions the output cap, stop reasons or empty responses. Implemented by the seed, the changes gain about two points on average (Table 11), about the same as edits from a task log without the DGM’s hint prompts. Single DGM steps gain two to three times as much (counting children that failed to compile as unchanged, or not); the children that came from the empty-patch prompt carry that gain, and all children from the stochasticity prompt failed to produce a working edit. Log-free chains mostly broke the agent. These single-edit gains, and the two-point gains of loops started from the engineered seed, are measured against a pooled seed reference whose morning runs scored about 2.5 points higher than its evening runs, so they depend on the reference to within about that much.

Grader quirk. Our first grader prepended `from math import *` to every solution, which replaces the built-in `pow` and breaks `pow(a, b, m)`. Loops from the engineered seed proposed tools that run a solution under several import preludes. Re-scoring the saved archives with the corrected grader shows that most of their measured gain (+22 and +20 points on the selection set) was recovery from the bug (+6 points each corrected); the two graders rank the archives with Spearman $\rho = 0.24$ and 0.94.

Improver swap. The same twelve DGM diagnoses of the seed were implemented on a fresh copy of the seed by the seed agent and by loop 3’s best agent. The seed agent returned no edit for 5 of the 12 diagnoses, and 7 working agents remained (mean selection score 66.0%; mean fresh gain +4.6 points over 7 with fresh results). Loop 3’s best agent returned an edit for every diagnosis; 5 of

its edits left an agent that wrote no solution, and 7 working agents remained (mean selection score 66.9%; mean fresh gain +6.0 points over 7 with fresh results). Among the evolved agent’s finished edits, broken edits changed 247–448 lines, working ones 70–253. A better coding agent does not yield more working agents from the same instructions.

Table 12: First-round agents under other task models (F_1 , one run per problem, \$1 budget, \$0.25 for gpt-oss-120b). Gains are paired against the DGM seed under the same model; budget and time limits censor some agents (Appendix E).

task model	DGM seed	engineered seed	loop-3 best: gain	loop-1 best: gain
Claude Haiku 4.5	25.0	47.4	+15.5 [+7.8, +22.4]	+1.7 [−5.2, +8.6]
Claude Sonnet 4.6	58.6	76.7	+12.1 [+5.2, +19.8]	+6.0 [−2.6, +13.8]
Claude Sonnet 5.5	87.9	90.5	+3.4 [+0.9, +6.9]	+2.6 [+0.0, +6.0]
Gemini 2.5 Flash	12.1	30.2	+38.8 [+30.2, +47.4]	+32.8 [+24.1, +41.4]
gpt-oss-120b	40.5	25.9	+22.4 [+12.9, +31.9]	−5.2 [−13.8, +3.4]

G SWE-bench details

Rebuilding the agents. The released DGM repository starts after the SWE-bench run. Its initial agent differs from the run-time agent in two places that the archive’s patch contexts reveal: a later retry around the model call and an example line in the editor. With both restored, the six patches of the reported best agent’s lineage apply in order without a rejected hunk. The textbook agents start from the restored initial agent; the retry variant uses helper functions the initial agent already contains (regression-test discovery, test runs and a report score).

Protocol. All four agents run through the DGM’s harness with one change (each task gets its own proxy token and the proxy address), on 246 fresh tasks (SWE-bench Verified minus the DGM’s 200, minus tasks whose environment does not build or whose reference patch fails today) and 47 of the DGM’s 60 selection tasks, one run per task, \$10 per run, 16 tasks in parallel per agent, all four agents at once in the same task order. A pilot with Claude Haiku 4.5 was stopped because most runs of both DGM agents ended early; Claude Sonnet 4.5 was used instead. A 15-task pilot with it showed 7 of the best agent’s 14 runs and 3 of the initial agent’s 15 runs ending early; the full run kept the same protocol. Grading follows the DGM’s own report script; evaluations that timed out were re-graded alone. One task (scikit-learn-14710) timed out in evaluation for both textbook agents even when re-graded alone and counts as unresolved; in the sensitivity analysis one patch of the best agent (psf__requests-1142) got no report and counts as unresolved.

Early endings. Of the best agent’s 142 early endings in the second-round run, 136 came on a signal (exit codes 137 and 143) and 6 with exit code 1. Of the saved logs of these runs, 24 are empty; 11 of the other 118 contain a command that ends processes (`pkill`, `killall` or `kill -9`), and 107 contain no such command. In the pilot with Claude Haiku 4.5 we traced such endings to cleanup commands issued through the agent’s own shell. Median run times were 20 minutes for the initial agent and 37 for the retry variant; among the runs that finished, the retry variant resolves 75.4% and the initial agent 60.7% of the fresh tasks.

H Costs

Model spend was about \$4,600 for the competitive-programming study (all three rounds) and the ADAS test together, and about \$3,700 for the SWE-bench tests; machine time was small by comparison. These totals cover the model calls of the experiments only.