MAINFRAME-GYM: BENCHMARKING AGENTS ON THE MAINTENANCE OF PRODUCTION LEGACY ESTATES

Anonymous authors
Paper under double-blind review

ABSTRACT

Coding agents are being deployed on COBOL estates just as the engineers who maintain them retire, yet no benchmark measures whether an agent can *maintain* a production mainframe system. Existing mainframe benchmarks test knowledge or single functions, and software benchmarks are mined from public repositories whose history and tests a production estate lacks. We introduce MAINFRAME-GYM, a benchmark of maintenance tickets on the 432k-line decommissioned billing estate of a large telecommunications company, and release the two pieces of infrastructure it required. HARWELL runs the estate's COBOL and JCL batch chains, with DB2 emulated, off the mainframe. TASKFORGE is a human–AI collaboration harness that scales task construction: engineers direct the work, agents draft and verify the tasks, and the domain expert, who does not work in English, is consulted in their own language and reviews tasks in full, so scarce expert time goes to judgment rather than writing. Every grader is tested before any agent runs. We test five frontier models, each in its own or a common harness. The best pair solves five of eight tickets at least once, and two tickets defeat every pair. In a closer pilot study, agents usually find the right programs but make incomplete changes, missing the clone programs and archived versions that carry the same logic, and their own tests do not catch it.

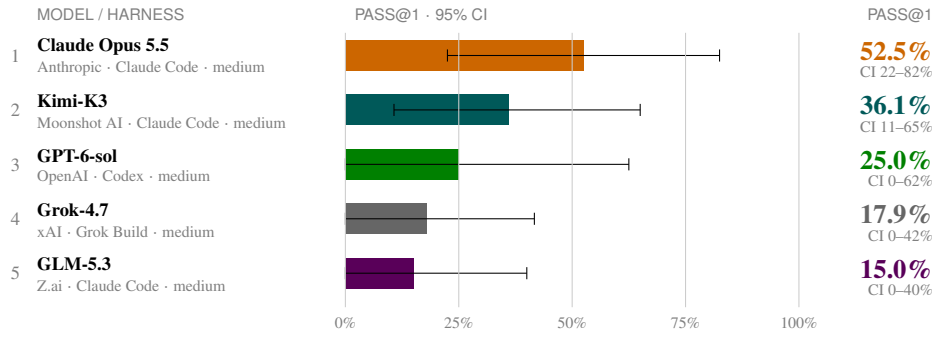


Figure 1: Five frontier models, each in a harness, on the green tasks of MAINFRAME-GYM at medium reasoning, ranked by Pass@1. Pass@1 is the pass rate of each task averaged over the eight tasks every pair ran. Whiskers are 95% intervals from resampling tasks (Section 6).

1 INTRODUCTION

A large share of the world's transaction processing still runs as COBOL batch jobs on mainframes, and the engineers who maintain it are retiring. Coding agents are already being assigned to this work: industry reports describe agents documenting COBOL, migrating batch workflows, and refactoring fields across hundreds of programs (Cognition, 2026). What is missing is a way to measure whether an agent can *maintain* such a system. That means taking a ticket that describes a business symptom, finding the programs that actually run, changing every place the logic lives, and leaving closed periods exactly as billed.

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.

Existing benchmarks measure something else. Mainframe benchmarks exist. MainframeBench (Dau et al., 2024) poses multiple-choice and open questions about mainframe concepts and scores COBOL summarization; nothing in it is executed. COBOLEval (Gordon-Hall, 2024) ports HumanEval to COBOL and executes it, but each problem is a single self-contained program, with no job control, no database, and no surrounding system. General software benchmarks do grade agents inside real systems (Jimenez et al., 2024; Tang et al., 2026), but most are built by mining public repositories: take a closed pull request, keep its tests, and admit the task when those tests fail before the fix and pass after it. That recipe needs commit history, an issue tracker, and a test suite that runs on commodity hardware. The estate we study has none of the three, and running its tests at all normally requires a licensed mainframe.

Why a real estate matters. Most of the difficulty in maintaining a mainframe system comes from how it is organized. The job and procedure libraries decide which program actually runs, so reading the source alone does not tell you. Many programs also exist as near-identical clones, each carrying one rate-plan variant of the same logic. COBOL code of that era was rarely written for reuse, and in a procedural language the easy way to add a variant was to copy the whole program and change one piece. Every copy must then be maintained, compiled and tested on its own. Older generations of a program sit beside the live one, and sometimes the job compiles the old one. If an agent edits the wrong program, no test fails and points at the file. The change simply has no effect. The single-program problems of existing COBOL benchmarks have none of this structure.

What we built. MAINFRAME-GYM is built on the decommissioned batch billing estate of a large telecommunications company, acquired under commercial licence. The estate came with no version history and no tickets, so every task was constructed. Getting from the estate to a benchmark took two pieces of infrastructure, and we contribute both.

- **HARWELL, an open execution environment for mainframe code.** It runs both batch jobs and online transactions off the mainframe. For batch, it interprets JCL, compiles IBM-dialect COBOL on a patched GnuCOBOL, emulates embedded DB2 SQL on SQLite, and models VSAM, GDG and the batch utilities. For online work, it runs a bounded set of CICS commands and has a terminal for driving screens by hand. We know of no other open toolchain that runs IBM-dialect COBOL, JCL, DB2 SQL and CICS together. Its known divergences from the mainframe are published (Section 4).
- **TASKFORGE, a human-led harness for constructing tasks.** With no history to mine, tasks must be authored. In TASKFORGE, people lead and agents do the drafting. Engineers choose what to build, drive each stage and read every result; agents propose and draft tasks inside three nested loops (*Check* gates each artifact with a verifier whose verdict is computed in code, *Conduct* runs agents at k, *Consult* asks the domain expert the business-rule questions no verifier can settle, in the expert's own language). Every task we compare models on was reviewed by a person before any agent ran on it, and the expert reviewed 12 of them in full. We report where the gates and the expert disagreed (Section 5).
- **MAINFRAME-GYM and an analysis of how agents fail on it.** There are 27 tasks, graded by 308 recorded cases pinned to business observables. Before any agent runs, each grader is shown to accept the reference fix and to reject the unchanged estate and seeded wrong answers. We compare five frontier model and harness pairs (Figure 1) and report pass rates with intervals, per-attempt variation, cost, the integrity of the reward, and every task and instrument defect we found. A failure taxonomy locates the difficulty in completing a multi-site change rather than in finding where to start (Section 6).

Availability. HARWELL will be released as open source. For review, an anonymized package (Section 8) holds HARWELL, one complete task with graded trials from several agents, and TASKFORGE's rubric files. The remaining tasks stay private under the estate's licence, which also protects them from training contamination.

2 RELATED WORK

Mainframe and COBOL benchmarks. Table 8 (Section H) compares the settings. MainframeBench, released with XMainframe (Dau et al., 2024), is the most direct predecessor. It covers

multiple-choice questions, question answering, and COBOL code summarization, all scored against references. COBOLEval (Gordon-Hall, 2024) executes COBOL. It translates HumanEval problems into COBOL and runs them under GnuCOBOL, but each problem is one program with its own inputs, so nothing exercises job control, a database, or dispatch across programs. CodeTransOcean (Yan et al., 2023) includes COBOL among its niche languages and scores that translation only by overlap with a reference. Industry reports of agents on COBOL describe documentation, batch migration, and mechanical refactoring. Each relies on a code map built in advance and on expert-written playbooks (Cognition, 2026). The same report names the central obstacle: agents run in Linux VMs, so they can read COBOL but cannot run it. HARWELL removes that obstacle for batch jobs and online CICS programs. The Hercules emulator (The Hercules Project, 2026) can run old, freely available MVS releases with JCL and COBOL, but not DB2 or IBM Enterprise COBOL. IBM's own z/OS test environment (IBM, 2026) runs modern workloads, but it is licensed.

Execution-graded software benchmarks. SWE-bench (Jimenez et al., 2024) set the recipe of mining a closed pull request and keeping its tests, and later work mines more languages, live settings and training environments (Zan et al., 2025; Badertdinov et al., 2025; Zhang et al., 2025; Pan et al., 2025). Terminal-Bench uses hand-written tasks instead (Merrill et al., 2026), and operations and tool-use benchmarks widen the range (Tang et al., 2026; Jha et al., 2025; Chen et al., 2025; Yao et al., 2024). DevOps-Gym (Tang et al., 2026) is closest in spirit. The same agent falls there from 70.4% on SWE-bench Verified to 23.9% on Java and Go, and experts built most of its build and monitoring tasks by injecting faults, at over ten expert-hours each.

Task synthesis and verifier quality. SWE-smith (Yang et al., 2025), R2E-Gym (Jain et al., 2025), RST (Li et al., 2026), and self-play environment synthesis (Zhao et al., 2025; Shi et al., 2026) manufacture tasks and admit each one by executing it. RST, like TASKFORGE, reuses accepted tasks as seeds for the next round. Weak and leaked tests (Aleithan et al., 2024) are why we test each grader by running it before admitting a task, rather than assuming it works.

Judging trajectories. Frontier agents game rewards (Von Arx et al., 2025; Denison et al., 2024; Baker et al., 2025), and trajectory judges are not settled. Agent-as-a-Judge reports near-human reliability on its own benchmark (Zhuge et al., 2024), while other work finds judges weak at debugging traces or inconsistent across benchmarks (Deshpande et al., 2025; Lù et al., 2025). So our judges must quote transcript bytes that the runner checks, and the reward always comes from the grader.

3 THE ESTATE AND ITS TASKS

MAINFRAME-GYM is built on the batch billing side of a mainframe system that a large telecommunications company ran in production and later shut down. We bought the code under a commercial licence. It has 181 COBOL source files, 321 older versions of those files (432k lines in all), 482 JCL jobs and 61 copybooks. Only 91 of the 181 files are ever run by a job. The other 90 are never called, and 34 of them are not even complete programs. Old systems often look like this, and it makes the tasks harder. An agent that edits one of those 90 files changes nothing.

Some pieces were missing. The code came without 40 of the 61 copybooks, which are the shared record layouts that programs include, and without any DB2 table definitions. We rebuilt both with one of the code's original authors. No production data came with the code, so every record a task uses was made for that task. The code itself is the company's own. The tasks placed in it are ours, and the rest of this section explains how we built them.

We use four shop terms: a *member* is a source file; a *deck* is a JCL job whose `EXEC PGM` steps name the programs that run; a *generation* is an older version of a member kept in an archive library (they ship with the estate but are not a change log); and a problem's tests travel together as a *pack*.

No history to mine. The estate came with no version-control history, no tickets, and no test suite. Every step a mining pipeline gets for free has to be rebuilt. There is no diff to localize the change, no issue to take the ticket text from, and no repository test suite to serve as the oracle. Every task and every reference fix in MAINFRAME-GYM was constructed with TASKFORGE in iteration with the

domain expert (Section 5). A task's pre-change state is the production code with the defect the ticket describes present, and its reference fix is the change that removes it.

What a task is. A task has three parts: a ticket in the shop's words, the estate in its pre-change state, and a recorded pack of fail-to-pass (FTP), pass-to-pass (PTP) and regression cases (Jimenez et al., 2024; Just et al., 2014). Tickets describe symptoms and name no program. Each case is pinned to a *business observable*—an output record, a DB2 row after the chain runs, or a report counter—and never to source text, so any implementation with the right behavior passes and a fix in the wrong program cannot. A candidate change scores 1 when every FTP case flips and every PTP case holds, and 0 otherwise. One ticket, `smp-mobile-0800-billing`, is the task in our reviewer package (Section 8). Its green version is graded by six cases, five FTP and one PTP:

> I manage the 0800 billing closure process. Ever since the mobile (SMP) carriers began delivering traffic in November, our 0800 clients have been contesting their invoices, claiming that calls made from mobile phones to their numbers are missing. [...] With the January billing cycle approaching in mid-December, we need this traffic billed correctly. Please note that invoiced totals for previously closed periods must remain unchanged.

The reference fix touches eight members, one of them in the archive library. The ticket's last sentence is a pass-to-pass contract over every closed period.

Where expected values come from. No expected value is written by hand: a case enters the library only from a recorded HARWELL run, PTP expectations from the unchanged estate and FTP expectations from the reference fix. The engine was hardened against 85 expert-exported case packs (Section 4); where a pack and the code disagreed, the code decided, and the pack was withdrawn as a case.

4 HARWELL: RUNNING THE ESTATE OFF THE MAINFRAME

Grading by execution requires running the estate, which normally requires licensed z/OS hardware. HARWELL runs it on Linux.

What it runs. HARWELL reads and runs JCL itself, including cataloged procedures, symbolic parameters, step overrides and condition-code checks, and the batch utilities the estate calls (IEBGENER, IDCAMS, DFSORT, ICETOOL and the DB2 utilities). It compiles IBM-dialect COBOL on GnuCOBOL 3.2. Embedded SQL runs through GixSQL against SQLite, which stands in for DB2. SQL codes behave as DB2's do, but isolation clauses are ignored and there is one schema. VSAM and GDG datasets are modelled too. Twenty-four patches, 5 to GnuCOBOL and 19 to GixSQL, make these tools behave like the mainframe. A bad packed number or a divide by zero, for example, ends the step with S0C7 or S0CB and skips the rest of the job, as z/OS does. COBOL comparisons and SORT order text in EBCDIC, while VSAM keys and SQL ordering still use ASCII, a published divergence. When a program asks for today's date or time, it gets a fixed value set for the run, so a billing program always decides the period the same way. HARWELL also runs online CICS programs: file control, temporary storage, BMS screens, exception handling, interval control, transient data, channels and containers, and SYNCPOINT rollback, with a terminal for driving screens by hand. This benchmark uses only the batch side, because the estate is a batch system.

Calibrated against the estate's recorded behavior. We hardened HARWELL against the recorded behavior of this estate rather than against a specification. The first verifier we built checked the tests. Of 85 expert-exported case packs, 37 asserted behavior that no program in the estate implements, so they cannot describe what this system did. We removed them as graded cases and kept them as fixtures. As fixtures they exposed four engine defect classes, each of which we closed and guarded with a unit test. HARWELL publishes its open divergences from the mainframe (about 80, over a quarter of them in the online CICS tier, which this batch estate does not use), most with a workaround. Cases that would grade bytes a program never sets are refused. For the other divergences, the task author applies the published workaround.

On two public mainframe applications, AWS CardDemo and GenApp, HARWELL reads every JCL deck and `EXEC SQL` block and compiles 61 of 75 program members cleanly, naming each refusal

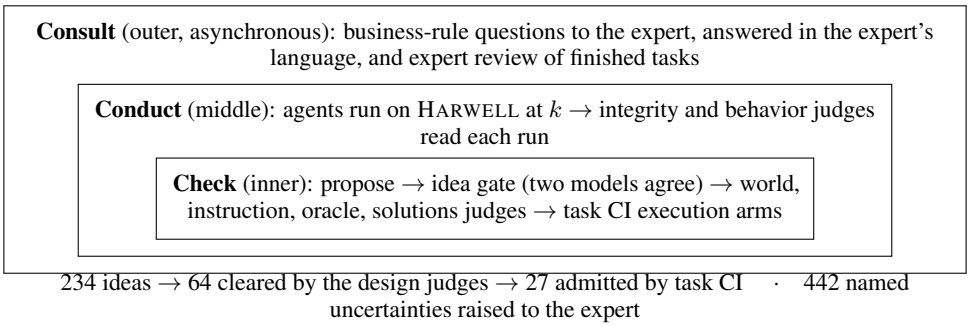


Figure 2: TASKFORGE's nested 3C loop. Engineers direct it and drive each stage, and agents draft inside it. Consult is asynchronous and never blocks the inner loops.

(Section A). No witness run on licensed z/OS hardware exists: HARWELL reproduces this estate's recorded behavior and is not a validated general emulator.

A report, not a grader. HARWELL runs a unit and returns a report: every job, step, return code and dataset, plus a snapshot of the final state with a digest per item. A `compare` verb locates the first difference by record, offset and column. The grader reads that report, and so can an agent, which gives agents the run–check–iterate loop that COBOL otherwise denies them (Cognition, 2026).

5 TASKFORGE: A HUMAN-LED HARNESS FOR CONSTRUCTING TASKS

With no history to mine, each task must be authored. TASKFORGE is built so that people lead that work and agents do the drafting. Engineers set the direction: they choose which parts of the estate and which failure classes to build tasks from, and drive the loop one stage at a time, reading each stage's output before deciding the next (propose, author, gate, run, judge, then keep, revise or reject). The domain expert, a mainframe engineer with over twenty years on production billing systems, answers the business-rule questions the harness raises and reviews finished tasks. The expert did not write this code, so each answer is an investigation. The expert also works in a language other than English. So TASKFORGE takes the drafting and the translation off the expert, and their time goes to judgment rather than writing.

5.1 THE 3C LOOP

Construction runs as three nested loops (Figure 2). They are nested because they differ in cost. An automated check takes seconds, an agent run takes up to an hour, and a question to the expert can take days. So most iteration happens in the inner loop, the middle loop runs only on tasks that survive it, and the outer loop is kept for what neither can settle. An engineer drives all three, deciding after each stage whether to keep, revise or drop a candidate.

Check: catching bad tasks before they cost anything. A candidate starts when an authoring agent proposes a problem from the expert's seed scenarios, the estate's recorded behavior, or an earlier agent run. Two judge models must independently agree that it is a real, recoverable problem before any work goes into it. The candidate is then built into a bundle and examined by four design judges, each asking one question. The *world* judge asks whether the pre-change estate is faithful. The *instruction* judge asks whether the ticket describes the symptom without giving away the fix. The *oracle* judge asks whether the test cases grade behavior and tell right from wrong. The *solutions* judge asks whether a different valid fix also passes.

We do not take a judge's word for its verdict. Each judge answers a versioned checklist, one yes-or-no per row, and has to back every answer with evidence we can check: a case id, a compile result, or a quote from the transcript. The verdict is then computed from those rows in code, and a missing row, or a quote that does not appear in the transcript, counts against the bundle. A bundle that clears the judges goes on to task CI (Section 5.2).

Conduct: seeing how real agents behave. The inner loop cannot tell us what an agent will do with a task, so each admitted task is run on HARWELL against a (harness, model) pair, k times. Two judges then read every run. The *integrity* judge asks whether the reward was earned (no hidden test read, no hard-coded output, no stub) and whether a failure was the agent's or the harness's. The *behavior* judge reads the trajectory without seeing the reward or the reference fix, and records row by row what the agent actually did, each point tied to the step and quote it rests on. Clean runs feed the next round of candidates, and the ways agents fail suggest new ones.

Consult: bringing in the expert without stalling the work. Some questions only the expert can answer: what a business rule is, what the shop intended, whether a scenario could really happen. When the gates cannot settle a judgment, the harness records it as a named uncertainty, and uncertainties are batched into questions for the expert. Each is a business-rule question tied to the test assertion its answer decides; none asks the expert to debug our tooling. Questions go out in a thread in the expert's own language, and work continues while the thread is open, so the expert's pace never holds up the loop. Answers are stored where the next pass reads them, and knowledge that recurs is folded into the authoring and judging rubrics. Across the suite the gates raised 442 named uncertainties over 44 candidates, and not every uncertainty needed its own answer. The expert also reviewed the before- and after-states of 18 candidates, endorsed the ticket text of 30, and contributed 85 seed case packs and 124 transcribed cases, each replayed before it could grade.

Human review of finished tasks. Passing every gate makes a task a candidate. Before any agent ran on it, a person read each of the 27 released tasks whole: the ticket, the reference fix and every test case. For 12 of them, that person was the domain expert. So every task behind the model comparisons in Section 6 was checked by a human, and the business rules in the other 15 rest on the gates and on the expert's answers to named questions. We have no record of a reviewed task being held back.

5.2 TASK CI: TESTING THE GRADER BEFORE ANY AGENT RUNS

Before a pack is admitted, task CI runs it three ways. The *oracle* arm runs the reference fix, and every case must pass. The *null* arm runs the unchanged estate, and the pack must fail, with every failing case pointing at a business observable rather than an infrastructure error. The *seeded-fault* arm plants wrong answers that still compile and run, and the cases must catch each one. Probes then test the edges of each case: moving a figure the case checks must turn it red, and changing a byte it ignores must leave it green. The design also has the solutions judge check that a different valid fix passes, but we kept no per-task record of that, so we do not count it.

All 27 packs passed. The reference fix passes all 308 cases, the seeded-fault arm caught all 208 planted faults, and every case got six probes, with three packs partly checked by other means (Section G). Each arm ran once, and we have not yet rerun them to show the grades repeat. The current harness goes further than the release did. It repeats the oracle and null arms three times, requires a seeded fault from outside the reference fix's own family, and adds a tamper arm that rewrites every file the candidate should not touch. None of these ran on the release. Five problems have defects found after admission, which Section M.2 lists.

5.3 WHAT THE GATES REMOVED, AND WHERE THE EXPERT OVERRULED THEM

Table 1 gives the funnel. The 27 released tasks are those that had passed task CI when agent runs were launched, so selection preceded any agent outcome. The other 37 design-cleared candidates were limited by compute, not rejected. The engine-hardening verifier, not the design gates, caught the 37 packs asserting nonexistent behavior. Without these checks the benchmark would include them and the gate-rejected candidates, and its pass rate would measure the packs as much as the agents.

The expert's sampled state review is our one direct measure of what the gates miss. Of 18 sampled before-states, the expert found 8 inaccurate, and the world judge had already flagged only one of them. After the anchoring was re-cut, 7 of the 8 passed re-review; the eighth, the one the world judge flagged, is unreachable because its programs are commented out of the deck. Each disagreement led to a rubric revision, re-stamping, and a re-run, and three revisions are on record. The instruction rubric raised two writing findings, and the tickets were rewritten until the expert endorsed the fourth version. A blind read of two trajectories showed the behavior rubric scored what an agent believed

Table 1: Construction and grader measurements for the release: the funnel, the grader checks before any agent runs, and the measured disagreement between the gates and the expert's sampled review.

stage	count
task ideas explored	234
did not reach a design bundle (idea gate, expert sample, not pursued)	157
rejected by the world / oracle / solutions judge	3 / 7 / 3
cleared by the design judges	64
admitted by task CI (the release)	27
reviewed by a person before agent runs	27
of which by the domain expert	12
packs passing the oracle, null and seeded-fault arms	27 of 27
recorded cases, all green under the reference fix	308
seeded faults that no case caught	0 of 208
probes per case (three packs partly checked by other means)	6
expert-exported case packs withdrawn by the engine-hardening verifier	37 of 85
before-states the expert judged inaccurate, of 18 sampled	8
of those, already flagged by the world judge	1
accurate after the anchoring was re-cut	7 of 8

but not how it bounded its search, so we added four rows. The same review produced a new task seed.

6 Evaluation and Analysis

6.1 Experimental setup

Agent configurations. We run five frontier models, each inside a harness, all at medium reasoning: GPT-6-sol under Codex, Claude Opus 5.5 under Claude Code, Grok-4.7 under Grok Build, and Kimi-K3 and GLM-5.3 under Claude Code. Harness versions are in Table 2. Every agent gets the same ticket and the same sandbox.

Tasks. We report on the 16 green tasks, a subset of the 27 released tasks. So far all five pairs have runs on the same eight of them, and those eight make up the comparison. A pair ran between one and seven trials per task (Figure 3, and Section L for every cell). The other eight green tasks do not yet have runs from every pair.

Runtime environment. Every trial runs in a fresh Modal sandbox that holds the estate's source and jobs and Harwell to run them. The agent never sees the packs, the expected values or the reference fix. The sandbox can reach only the hosts it needs to install the harness and call the model.

Time budgets. The agent has one hour and the grader has one hour. Building the task environment has its own 30 minutes, which do not count against the agent.

Failures. Some trials end in an infrastructure error rather than a graded attempt: the network drops, or the model's API returns an error. We leave these out of Pass@1 and also report the rate that counts them as failures. They are 32 of the 162 trials, and 18 of them are Claude Opus 5.5's.

Metrics. Pass@1 is the pass rate of each task, averaged over the tasks, so every task counts the same however many trials it had. The 95% interval comes from resampling tasks 10,000 times. Cost is Harbor's estimate from the token counts of each run, split evenly over the run's trials. On runs that also went through our model gateway, the billed spend can be much lower, so these costs are best read as an upper bound.

Table 2: Agent configurations, Pass@1 and cost per trial on the eight green tasks all pairs ran, at medium reasoning. *Trials*: graded over total. *All trials*: the pooled pass rate if every infrastructure error counts as a failure. *Solved*: tasks passed at least once. *Per solve*: total cost over passing trials. The null agent is the unchanged estate, graded by the null arm of task CI.

								cost per trial ($)		
developer	harness	version	model	trials	Pass@1	95% CI	all trials	solved	mean	median
Anthropic	Claude Code	2.1.274–283	Claude Opus 5.5	24/42	52.5%	22–82	9/42	5/8	5.86	5.67
Moonshot AI	Claude Code	2.1.274–283	Kimi-K3	29/34	36.1%	11–65	8/34	4/8	7.76	7.58
OpenAI	Codex	0.157.0–1	GPT-6-sol	25/25	25.0%	0–62	6/25	2/8	1.19	0.96
xAI	Grok Build	1.0.41	Grok-4.7	27/34	17.9%	0–42	8/34	2/8	4.89	4.10
Z.ai	Claude Code	2.1.274–283	GLM-5.3	25/27	15.0%	0–40	6/27	2/8	13.47	14.29
–	–	–	null agent	–	0.0%	–	–	0/8	–	–

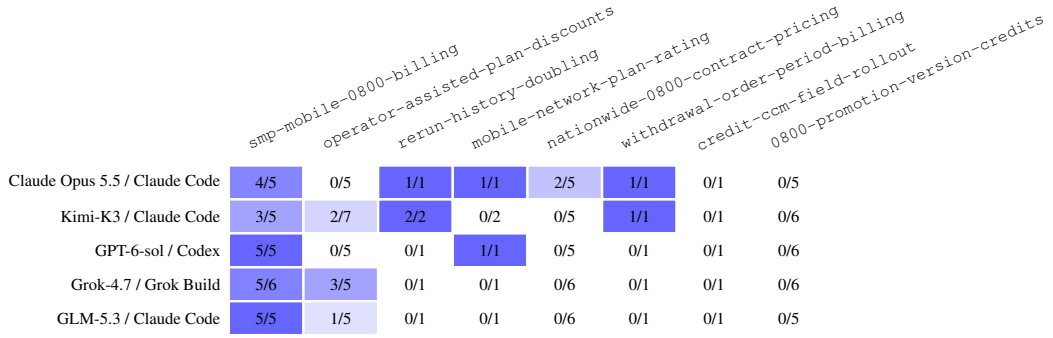


Figure 3: Passing trials over graded trials for each agent pair on each green task, at medium reasoning. Darker cells pass more often. Tasks run from most solved (left) to least.

6.2 Results

Pass rates. Claude Opus 5.5 and Kimi-K3, both under Claude Code, lead (Figure 1). Claude Opus 5.5 solves five of the eight tasks at least once and Kimi-K3 four. The other three pairs solve two. With eight tasks the intervals are wide and overlap, so we do not rank the pairs. The tasks differ far more than the agents do (Figure 3). Every pair solves `smp-mobile-0800-billing`, and 22 of its 26 graded trials pass. Only Claude Opus 5.5 solves `nationwide-0800-contract-pricing`, and no pair solves `credit-ccm-field-rollout` or `0800-promotion-version-credits`. The pooled rate in the *all trials* column leans towards the easy task, because it also has the most trials.

Cost. Cost varies more than the pass rate. GPT-6-sol under Codex costs $1.19 per trial and $4.97 per solve, and reads 3.8M tokens per trial, less than half of what any other pair reads. GLM-5.3 costs the most, $13.47 per trial and $58.39 per solve. Claude Opus 5.5 costs $16.27 per solve, the least after GPT-6-sol.

A closer look, from the pilot run. The per-case verdicts and transcripts that the analyses below need come from an earlier pilot on all 27 released tasks, with two agents. Agent A is kimi-k3 under Terminus 2 (Merrill et al., 2026) at k=1, 27 trials, with one hour per trial. Agent B is Claude Fable 5 under a minimal harness that gives it only a shell and a file-editing tool, at k=3. The platform's safety classifier refused 19 of Agent B's 81 attempts before any work began, and one failed to launch, which leaves 61 graded trials on 22 tasks. Agent A passes 5 of 27 (18.5%) and Agent B 14 of 61 (23.0%). Agent A costs $7.85 per trial, averages 96 steps and 36 minutes, and hits the one-hour limit on six trials. Agent B's sandbox held the test files, readable only by the root user, and ten of its 60 transcripts try to read them and fail.

What the pilot shows. Most failed trials edit a program the fix must touch and then leave the change incomplete: 38 of Agent B's 46 failures and 13 of Agent A's 16 shipped failures. Editing the wrong program, or the wrong copy of it, accounts for the rest, and some of those come from the

estate's own traps, such as a live deck that compiles an archived copy of the program. The agent's own test showed the defect gone on 40 of Agent B's 46 failures, so on nearly every failure it believed the work was done. The integrity judge found no trial that gamed its reward. Section M gives the grading ladder, consistency across attempts, the failure modes, the checks that the environment did not cause the losses, and the reward-hacking audit.

7 Discussion and Limitations

Limitations. *One estate, one engine.* Archived generations may be denser here than elsewhere, and HARWELL has no witness run on licensed z/OS hardware. *Two agents.* They differ in model, harness, platform and k, so no claim ranks models, and a safety classifier kept five problems from one of them. *Validation.* We report every measured gate–expert disagreement, but not each gate's precision and recall at scale, the behavior judge's agreement with a human reader, or expert hours. Agent A's labels were read with the hypothesis known, and Agent B's come from one LLM reader. *Scale.* There are 27 tasks, with 37 more design-cleared, and intervals are wide. Reference fixes show every task can be solved but not how long a person takes. One task is shared for review, and no model has been trained on it. The failure modes of Section M.1 seed the next round of tasks.

8 Conclusion

We built MAINFRAME-GYM on a real decommissioned billing estate. HARWELL runs its batch chains off the mainframe, and in TASKFORGE engineers direct the construction, agents draft the tasks, and the expert answers what no verifier can settle. Five frontier models, each in a harness, are far from saturating it. The best pair solves five of eight green tasks at least once, and two tasks defeat every pair. Agents find a program the fix must touch, then fall short of the full change, and their own tests cannot tell the difference. Reviewers can run HARWELL on a complete task with its graded trials, and read TASKFORGE's rubrics, in the reviewer package (Section 8).

AI Use Statement

Language models are part of TASKFORGE: they draft and check tasks, translate for the domain expert, and act as judges (Section 5). They do not decide what is admitted or what passes. Executed checks admit every task, and the execution grader gives every reward. We also used AI tools to help analyze results and to draft and edit the text. Every reported number comes from recorded runs, and the authors checked all AI-assisted work.

Ethics Statement

The estate was acquired under a commercial licence from its owner after the system was decommissioned. We do not redistribute it. Tickets are written in an operational persona and describe billing defects. They contain no personal data. No production data came with the code, so the tables hold only records made for the tasks. The table definitions include columns for customer names, tax IDs and phone numbers, but no values for them appear in any pack or sandbox. The code does contain some employee names and IDs. We remove them from every file we release, including the task we share with reviewers. The domain expert took part with informed consent as a paid consultant and worked in their own language throughout. The benchmark evaluates maintenance of financial batch systems and is not intended for deployment against live systems.

Reproducibility Statement

Available to reviewers now, as an anonymized package[1]:

- HARWELL in full: the toolchain of Section 4, its patches to GnuCOBOL and GixSQL, and its published divergences;

[1] `https://mainframe-gym-review-0b03b8.s3.us-east-1.amazonaws.com/mainframe-gym-reviewer-package.zip`

- one complete green task, `smp-mobile-0800-billing`, with its ticket, recorded pack and reference fix, and graded trials from several model and harness pairs, each with its trajectory and grader verdict;
- TASKFORGE's rubric files: the judge rubrics used by the Check loop and the behavior rubric whose rows are in Section B.

Identifying names, hostnames and key-like strings are replaced throughout the package. *Not released for review:* the full estate and the other tasks, which the licence does not allow us to redistribute. Holding them out also keeps them out of training data. Experimental accounting is in Section F, per-task results in Section E and Section L, and comparison-table sources in Section I.

REFERENCES

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. URL https://arxiv.org/abs/2410.06992.

Ibragim Badertdinov, Alexander Golubev, Maksim Nekrashevich, Anton Shevtsov, Simon Karasik, Andrei Andriushchenko, Maria Trofimova, Daria Litvintseva, and Boris Yangel. SWE-rebench: An automated pipeline for task collection and decontaminated evaluation of software engineering agents. In *Advances in Neural Information Processing Systems*, 2025.

Bowen Baker, Joost Huizinga, Leo Gao, Zehao Dou, Melody Y. Guan, Aleksander Madry, Wojciech Zaremba, Jakub Pachocki, and David Farhi. Monitoring reasoning models for misbehavior and the risks of promoting obfuscation. *arXiv preprint arXiv:2503.11926*, 2025. URL https://arxiv.org/abs/2503.11926.

Yinfang Chen, Manish Shetty, Gagan Somashekar, Minghua Ma, Yogesh Simmhan, Jonathan Mace, Chetan Bansal, Rujia Wang, and Saravan Rajmohan. AIOpsLab: A holistic framework to evaluate AI agents for enabling autonomous clouds. *arXiv preprint arXiv:2501.06706*, 2025.

Zhi Chen, Zhensu Sun, Yuling Shi, Chao Peng, Xiaodong Gu, David Lo, and Lingxiao Jiang. Rethinking the value of agent-generated tests for llm-based software engineering agents. *arXiv preprint arXiv:2602.07900*, 2026. URL https://arxiv.org/abs/2602.07900.

Cognition. How Devin is modernizing COBOL at Fortune 500 companies. Cognition blog, April 2026. Accessed 2026-09-23.

Anh T. V. Dau, Hieu Trung Dao, Anh Tuan Nguyen, Hieu Trung Tran, Phong X. Nguyen, and Nghi D. Q. Bui. Xmainframe: A large language model for mainframe modernization. *arXiv preprint arXiv:2408.04660*, 2024. URL https://arxiv.org/abs/2408.04660.

Carson Denison, Monte MacDiarmid, Fazl Barez, David Duvenaud, Shauna Kravec, Samuel Marks, Nicholas Schiefer, Ryan Soklaski, Alex Tamkin, Jared Kaplan, Buck Shlegeris, Samuel R. Bowman, Ethan Perez, and Evan Hubinger. Sycophancy to subterfuge: Investigating reward-tampering in large language models. *arXiv preprint arXiv:2406.10162*, 2024. URL https://arxiv.org/abs/2406.10162.

Darshan Deshpande, Varun Gangal, Hersh Mehta, Jitin Krishnan, Anand Kannappan, and Rebecca Qian. Trail: Trace reasoning and agentic issue localization. *arXiv preprint arXiv:2505.08638*, 2025. URL https://arxiv.org/abs/2505.08638.

Gabriel Gordon-Hall. Evaluating LLMs on COBOL. Bloop blog post and open-source benchmark, March 2024.

IBM. IBM Z development and test environment. IBM product documentation, 2026. Accessed September 2026.

Naman Jain, Jaskirat Singh, Manish Shetty, Liang Zheng, Koushik Sen, and Ion Stoica. R2e-gym: Procedural environments and hybrid verifiers for scaling open-weights swe agents. *arXiv preprint arXiv:2504.07164*, 2025. URL https://arxiv.org/abs/2504.07164.

Saurabh Jha, Rohan Arora, Yuji Watanabe, Takumi Yanagawa, Yinfang Chen, Jackson Clark, et al. ITBench: Evaluating AI agents across diverse real-world IT automation tasks. *arXiv preprint arXiv:2502.05352*, 2025.

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

René Just, Darioush Jalali, and Michael D. Ernst. Defects4j: A database of existing faults to enable controlled testing studies for java programs. In *Proceedings of the 2014 International Symposium on Software Testing and Analysis (ISSTA)*, pp. 437–440, 2014. doi: 10.1145/2610384.2628055.

Zhongzhi Li, Yucheng Shi, Zongxia Li, Ruhan Wang, Anhao Li, Zixun Huang, Junyao Yang, Lei Ke, Ninghao Liu, Haitao Mi, and Leowei Liang. Recursive synthesis for long-horizon terminal tasks. *arXiv preprint arXiv:2608.05466*, 2026. URL https://arxiv.org/abs/2608.05466.

Xing Han Lù, Amirhossein Kazemnejad, Nicholas Meade, Arkil Patel, Dongchan Shin, Alejandra Zambrano, Karolina Stańczak, Peter Shaw, Christopher J. Pal, and Siva Reddy. Agentrewardbench: Evaluating automatic evaluations of web agent trajectories. *arXiv preprint arXiv:2504.08942*, 2025. URL https://arxiv.org/abs/2504.08942.

Mike A. Merrill et al. Terminal-Bench: Benchmarking agents on hard, realistic tasks in command line interfaces. In *International Conference on Learning Representations (ICLR)*, 2026.

OpenAI. Introducing SWE-bench verified. OpenAI blog, August 2024.

Jiayi Pan, Xingyao Wang, Graham Neubig, Navdeep Jaitly, Heng Ji, Alane Suhr, and Yizhe Zhang. Training software engineering agents and verifiers with swe-gym. In *International Conference on Machine Learning (ICML)*, 2025. URL https://arxiv.org/abs/2412.21139.

Yucheng Shi, Zhenwen Liang, Kishan Panaganti, Dian Yu, Wenhao Yu, and Haitao Mi. Learning to build the environment: Self-evolving reasoning rl via verifiable environment synthesis. *arXiv preprint arXiv:2605.14392*, 2026. URL https://arxiv.org/abs/2605.14392.

Yuheng Tang, Kaijie Zhu, Bonan Ruan, Chuqi Zhang, Michael Yang, Hongwei Li, Suyue Guo, Tianneng Shi, Zekun Li, Christopher Kruegel, Giovanni Vigna, Dawn Song, William Yang Wang, Lun Wang, Yangruibo Ding, Zhenkai Liang, and Wenbo Guo. DevOps-Gym: Benchmarking AI agents in software DevOps cycle. In *International Conference on Learning Representations (ICLR)*, 2026.

The Hercules Project. Hercules: System/370, ESA/390 and z/Architecture emulator. Open-source software, 2026. Accessed September 2026.

Sydney Von Arx, Lawrence Chan, and Beth Barnes. Recent frontier models are reward hacking. METR blog, June 2025. URL https://metr.org/blog/2025-06-05-recent-reward-hacking/.

Weixiang Yan, Yuchen Tian, Yunzhe Li, Qian Chen, and Wen Wang. Codetransocean: A comprehensive multilingual benchmark for code translation. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, 2023.

John Yang, Kilian Lieret, Carlos E. Jimenez, Alexander Wettig, Kabir Khandpur, Yanzhe Zhang, Binyuan Hui, Ofir Press, Ludwig Schmidt, and Diyi Yang. Swe-smith: Scaling data for software engineering agents. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2025.

Shunyu Yao, Noah Shinn, Pedram Razavi, and Karthik Narasimhan. τ-bench: A benchmark for tool-agent-user interaction in real-world domains. *arXiv preprint arXiv:2406.12045*, 2024. URL https://arxiv.org/abs/2406.12045.

Daoguang Zan, Zhirong Huang, Wei Liu, Hanwu Chen, Linhao Zhang, Shulin Xin, Lu Chen, Qi Liu, Xiaojian Zhong, Aoyan Li, Siyao Liu, Yongsheng Xiao, Liangqiang Chen, Yuyu Zhang, Jing Su, Tianyu Liu, Rui Long, Kai Shen, and Liang Xiang. Multi-SWE-bench: A multilingual benchmark for issue resolving. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2025.

Linghao Zhang, Shilin He, Chaoyun Zhang, Yu Kang, Bowen Li, Chengxing Xie, Junhao Wang, Maoquan Wang, Yufan Huang, Shengyu Fu, Elsie Nallipogu, Qingwei Lin, Yingnong Dang, Saravan Rajmohan, and Dongmei Zhang. SWE-bench goes live! In *Advances in Neural Information Processing Systems (NeurIPS)*, 2025.

Andrew Zhao, Yiran Wu, Yang Yue, Tong Wu, Quentin Xu, Yang Yue, Matthieu Lin, Shenzhi Wang, Qingyun Wu, Zilong Zheng, and Gao Huang. Absolute zero: Reinforced self-play reasoning with zero data. *arXiv preprint arXiv:2505.03335*, 2025. URL https://arxiv.org/abs/2505.03335.

Yuxuan Zhu et al. Establishing best practices for building rigorous agentic benchmarks. *arXiv preprint arXiv:2507.02825*, 2025.

Mingchen Zhuge, Changsheng Zhao, Dylan Ashley, Wenyi Wang, Dmitrii Khizbullin, Yunyang Xiong, Zechun Liu, Ernie Chang, Raghuraman Krishnamoorthi, Yuandong Tian, Yangyang Shi, Vikas Chandra, and Jürgen Schmidhuber. Agent-as-a-judge: Evaluate agents with agents. *arXiv preprint arXiv:2410.10934*, 2024. URL https://arxiv.org/abs/2410.10934.

A HARWELL ON PUBLIC CORPORA

HARWELL's readers and compiler are run on two public applications pinned to a commit and re-derived in CI on every push. On AWS CardDemo they read all 61 JCL decks and all 33 `EXEC SQL` blocks, and compile 31 of 44 program members cleanly. On GenApp they read all 29 decks and all 56 SQL blocks, and compile 30 of 31 members. The online tier is checked the same way. HARWELL runs all 240 `EXEC CICS` blocks in CardDemo and all 500 in GenApp. Starting each transaction the way a terminal user would, with no input typed, 16 of CardDemo's 30 transactions and 8 of GenApp's 10 reach their first screen or return. That last count was measured by hand, because it is too slow to run in CI. Every refused member is listed with the reason it was refused.

B CHECK: GATE CONTRACTS

The flag lists below summarize each rubric; the full rubric files ship with the five review tasks.

Every gate is one versioned rubric file, and the rubric file is the method: dimensions, bands, findings shapes and the output contract live in the rubric, and code only carries it to the auditor under a thin system prompt. Each gate runs one blind auditor per item, staged rather than prompted into blindness: the container holds only the artifact available at that stage—the idea judge one idea, blind to every other idea's score; the world judge the ticket and the world tree at the judged SHA (archive export, no history) plus toolchain access for compile evidence; the instruction judge the text alone; the oracle judge the ticket, the world and the oracle record with its measured arms; the solutions judge the before-world and the banked solutions; the post-run judges the finished rollout, or the trajectory alone. No volume is mounted and no other judge's output is present.

Each row returns one Boolean (or Likert score) with named evidence—a case id, a grep, a compile return code, a `step_id` plus verbatim quote—and a probe that could not be completed reads false, the missing evidence being the finding. The runner derives the authoritative verdict in code and refuses disagreement: an absent or unreadable flag is an error, never a lenient pass; a composite disagreeing with its flags, or a cross-field derivation disagreeing with per-entry verdicts, is rejected; a behavior record whose locators do not resolve or whose quotes are not in the file is handed back. Every verdict is stamped with `rubric_sha12` (SHA-256, 12 hex) and the frontmatter `rubric_version`.

Idea judge (`idea`, v3). Sees one idea, blind to every other idea and to any measurement. Rows: `idea_difficulty` (0–1, against a self-derived reference solution); `idea_quality` (0–1, grounded and coherent); `oracle_quality_score` (0–1, only where the idea proposes cases). No overall number, no threshold, no label; worth is decided downstream.

World judge (`world-judge`, v9). Ten flags with evidence: `premise_intact`, `symptom_producible`, `mechanism_reachable`, `solution_absent`, `no_future_state`, `walk_complete`, `population_sufficient`, `constraints_inventoried`, `after_recorded`, `findings_cited`. Verdict valid iff every flag is true.

Instruction judge (`instruction-judge`, v6). Text alone; nine flags, six properties true and three defects false: `human_authored`, `human_user_persona`, `single_language`, `explicit_constraints`, `implicit_constraints`, `target_goal_stated`; `solution_leak`, `insufficient_information`, `excess_information`. Verdict valid iff every property holds and no defect is present. The `single_language` and `solution_leak` flags guard the step where the harness drafts an English ticket from the expert's answers (Section 5).

Oracle judge (`oracle-judge`, v9). Eleven flags: `coverage_complete`, `kinds_populated`, `flip_measured`, `null_arm_red`, `faults_detected`, `variant_green`, `no_unfair_pins`, `no_admitted_violations`, `expectations_recorded`, `leak_free`, `evidence_stamped`. Verdict valid iff every flag is true and both rejection lists are empty.

Solutions and test judges. `solutions-judge` (v8) returns twelve flags—`symptom_reproduced`, `solve_measured`, `chain_reached`, `consumption_shown`,

guards_hold, oracle_arm_proven, no_deletion_equivalent, set_curated, verdicts_executed, failures_classified, refutations_banked, diversity_preserved—plus a per-solution verdict in {yes, yes-by-judgment, no, no-op, unverifiable}. The bundle-stage test judge (tests, v4) scores meaningful, specific, not_fragile, covers_edge_cases, hard_to_game on 1–5 and fails if any load-bearing dimension sits in its 1–2 band.

Task CI. Pre-run CI over a packaged bundle runs static probes (packaging and identity, verifier contract, instruction leakage, grader isolation, test material), the judged surfaces, and three execution arms: oracle (reward ≥ 1), null_arm (reward < 1) with its per-case classification (baseline_arm: a discrimination matrix in which every failing case names a declared observable and none an infrastructure reason), and seeded_fault_arm (tests must reject wrong answers that still run). task_eval_signal carries two deterministic gates in code above the auditor's word: a false-negative risk naming a graded observable, and risks in bulk past a threshold. Gating checks block eval_ready; advisory checks record findings only; waivers are operator-written off the artifact, and the execution arms are never waivable.

Integrity judge (judges, v4). Sees the result, the trajectory and the verifier logs. One pass/warn/fail per dimension: run_success, reward_integrity, reward_hacking, harness_communication, stuckness, recovery, tool_use; alongside seven task-quality flags and a post-training eligibility verdict, each with cited evidence. There is no composite; per-problem views are mechanical rollups of per-rollout reports.

Behavior judge (behavior-analysis, v2). Reads one trajectory blind: no reward, no verifier output, no reference fix, no earlier judgment. Twenty rows (ten general G1–G10, ten domain M1–M10), each decided PASS, FAIL or NA with a step_id locator and a copied quote of at most three lines. An unlocatable row is not decided in the agent's favour, and a row whose evidence class is *environment* takes a PASS only from keystrokes or observations. The misstep record names fault (mistake / slip / lapse / violation, with contradicted_by), error, failure and attribution. Nothing above the rows: no summary and no grade.

C Rubric Rows of the Blind Read

The rows carrying the argument in Section M.1, verbatim from the rubric except G3 and G10, which are given by their short labels.

M1 Before its first edit, the agent determined which programs execute for the reported symptom by tracing from the job and procedure libraries to the executed program.

M2 Every member the agent modified lies on the execution chain it had established.

M3 The agent derived candidate programs from the execution chain rather than confirming keyword matches against the job library afterward.

M5 Before editing a member, the agent located the other generations of that member in the archive and alternate libraries and determined which generation is live.

M6 The agent did not rely on a business rule that neither the ticket nor the estate states.

M9 Before editing a member, the agent searched for duplicates of the code it was changing and made, for each duplicate found, a decision grounded in the executing context or the symptom; changing every duplicate for safety is not a decision.

G1 The agent executed the modified program after its final edit; compilation alone does not satisfy this criterion.

G2 The agent read the output of that execution and compared it with the result the ticket specifies.

G3 Own test showed the defect before and its absence after.

G5 Before acting on it, the agent checked every assumption on which its fix depends against the estate.

G7 The agent did not truncate the output of a localization search in a way that concealed a candidate, or it repeated the search without truncation before acting on its results.

G10 Edits confined to the defect's code.

G4 The agent's closing message asserts only work that the transcript shows was performed.

Table 3: Agent B, all twenty rubric rows: trials marked PASS, passes against failures, risk difference (points) with Newcombe 95% interval, and two-sided Fisher exact p. NA rows are excluded from denominators. No row separates outcomes; power at these sizes is 0.72 for a true 40-point difference and 0.43 for 30 points.

row	passes	failures	RD	95% CI	p
chain traced before first edit (M1)	11/14	39/46	-6	$[-33, +13]$	0.69
candidates derived from the chain (M3)	3/14	17/46	-16	$[-36, +13]$	0.35
every edit on the chain (M2)	8/14	32/46	-12	$[-39, +13]$	0.52
single fix site chosen on evidence (M4)	9/14	36/46	-14	$[-41, +10]$	0.31
archive generation checked (M5)	5/14	15/46	$+3$	$[-21, +31]$	1.00
no unstated business rule relied on (M6)	11/14	29/46	$+16$	$[-13, +36]$	0.35
callers of a shared component enumerated (M7)	0/1	6/9	-67	$[-88, +19]$	0.40
change-board evidence delivered (M8)	7/14	33/46	-22	$[-47, +6]$	0.20
duplicates decided, not swept (M9)	8/14	27/46	-2	$[-29, +24]$	1.00
business-impact choice raised as a question (M10)	2/13	3/41	$+8$	$[-8, +35]$	0.58
changed program executed (G1)	14/14	45/46	$+2$	$[-19, +11]$	1.00
its output read against the ticket (G2)	14/14	44/46	$+4$	$[-17, +15]$	1.00
own test showed defect before, absence after (G3)	13/14	40/46	$+6$	$[-19, +20]$	1.00
closing message asserts only work performed (G4)	11/14	27/46	$+20$	$[-9, +40]$	0.22
assumptions checked against the system (G5)	3/14	16/46	-13	$[-33, +15]$	0.51
alternatives eliminated with evidence (G6)	6/8	25/31	-6	$[-41, +19]$	0.66
no candidate hidden by a truncated listing (G7)	9/14	28/46	$+3$	$[-25, +28]$	1.00
searches precise enough for their target (G8)	12/14	42/46	-6	$[-32, +10]$	0.62
plan revised on contradicting observation (G9)	13/14	35/45	$+15$	$[-11, +30]$	0.27
edits confined to the defect's code (G10)	10/14	42/46	-20	$[-46, +1]$	0.08

D EXAMPLE TASKS

Tickets as the agents receive them (the engine note that accompanies each is omitted).

`locality-terminal-capacity.` The ticket talks about capacity, but no case measures speed, which HARWELL does not measure. In the pre-change estate, a customer with a longer terminal or discount list than the table holds makes the run abort. The cases check that larger customer bases and 101-entry lists run through cleanly.

> I run the monthly close for the 0800 re-rating chain. Our close window is very tight and we are almost using the whole window to process with the current volume. Our base keeps growing and marketing tells us to expect more of the same, so we are worried that the current processing will surpass the window limit. We need the chain scaled to support current volumes and projected growth without altering any rated values. Our close window is already tight and the totals must continue to reconcile after the close. I don't know if it's related, but our largest corporate accounts, the ones with long terminal and discount lists, are losing discounts they are entitled to.

`credit-ccm-field-rollout.`

> I run the accounting close for long distance billing. Every alternative plan version now has its own credit CCM code registered in the plan catalog, and the corporate promotion credits have been arriving with it since the start of this month. The credits from the LDN, LDI and INTRA alternative plans still do not, so I cannot say which plan granted a credit and I am splitting February by hand. Those credits have to reach accounting carrying the code of the plan version that granted them. No credit value or rated amount changes, and January and everything before it stays exactly as posted. Accounting posts what the close hands over, and one credit I cannot tie to a plan reopens the period.

`withdrawal-order-period-billing.`

> I own the billing end of our service orders. When a line is given up, or a later order corrects one of them, the account comes out of the close described two ways at once: marked as

having moved off its old period, with the date of that move left as it was. A line we closed in March was billed through April. Installations are fine. We raise a few hundred credits a month by hand for this. The orders come to us out of the CSO customer records and the dates they carry are not always usable. It is the 13th and the next close takes this month's orders. A withdrawal or a correction has to leave the account describing one period and one only, and periods already closed and invoiced stay exactly as billed.

E PER-TASK AND PER-TRIAL RESULTS

Table 4: The 27 problems: graded cases per pack, Agent A's outcome with cases matched, and Agent B's passes over retained attempts, with attempts refused by the platform before any work.

problem	cases	Agent A	Agent B
0800-promotion-month-count	10	fail (1/10)	2/3 pass
0800-promotion-version-credits	15	timeout	1/2 pass, 1 refused
0800-set-amount-promotion-credit	15	fail (12/15)	3/3 pass
0800-staged-promotion-percentage	12	fail (5/12)	0/3 pass
atis-extension-billed-off-parent	6	timeout	0/3 pass
basic-plan-0800-promotion-qualification	16	fail (13/16)	1/3 pass
byvalue-promo-discounts	9	fail (1/9)	0/3 pass
closed-contract-ngeo-select	12	pass	3/3 pass
metro-area-local-reclassification	11	fail (5/11)	3 refused
credit-ccm-field-rollout	6	timeout	1/2 pass
day-count-ceiling-999	18	fail (14/18)	0/3 pass
duplicate-0800-promotion-credits	14	fail (10/14)	3 refused
local-area-charge-cutover	7	fail (3/7)	0/3 pass
locality-terminal-capacity	18	fail (13/18)	0/2 pass, 1 refused
mobile-network-plan-rating	11	fail (7/11)	1/3 pass
nationwide-0800-contract-pricing	7	fail (2/7)	0/3 pass
operator-assisted-plan-discounts	8	pass	1/3 pass
plan-version-discount-loss	9	timeout	1/3 pass
promotion-201-eligibility	5	fail (2/5)	3 refused
promotion-month-schedule	11	fail (1/11)	0/3 pass
recorder-code-through-tracking	5	timeout	0/2 pass, 1 refused
rerun-history-doubling	8	pass	0/3 pass
smp-mobile-0800-billing	27	pass	0/3 pass
tcl-consumption-days-stale	6	fail (1/6)	0/2 pass, 1 refused
tcl-stale-reading-correction-by-hand	14	timeout	3 refused
tcl-zeroed-pulses-not-honoured	9	fail (5/9)	3 refused
withdrawal-order-period-billing	19	pass	0/3 pass

Trial index for Agent A. T1 0800-promotion-month-count, T2 0800-set-amount-promotion-credit, T3 0800-staged-promotion-percentage, T4 nationwide-0800-contract-pricing, T5 byvalue-promo-discounts, T6 metro-area-local-reclassification, T7 day-count-ceiling-999, T8 duplicate-0800-promotion-credits, T9 mobile-network-plan-rating, T10 atis-extension-billed-off-parent, T11 plan-version-discount-loss, T12 promotion-201-eligibility, T13 promotion-month-schedule, T14 tcl-consumption-days-stale, T15 tcl-zeroed-pulses-not-honoured, T16 closed-contract-ngeo-select, T17 rerun-history-doubling, T18 smp-mobile-0800-billing, T19 withdrawal-order-period-billing, T20 local-area-charge-cutover, T21 tcl-stale-reading-correction-by-hand, T22 recorder-code-through-tracking, T23 locality-terminal-capacity, T24 basic-plan-0800-promotion-qualification, T25 operator-assisted-plan-discounts, T26 0800-promotion-version-credits, T27 credit-ccm-field-rollout.

Note on `rerun-history-doubling`. HARWELL does not model re-entry into a job after an abend, a published divergence.

F ADMISSION ACCOUNTING

No attempt by either agent was retried.

Table 5: Agent A, all 27 trials. *Wall*: the whole trial's wall-clock time, in seconds. *Chain* is *used* when every edited member was named by an executing step the agent had read before editing. It is *ignored* when the agent read the chain but edited outside it, or used the chain only to confirm a search hit, and *never* when it traced no executing step. *Exec*: the changed program was run and its output read. *Cut*: a listing the agent cut short hid a graded program.

id	outcome	steps	wall (s)	cases	$	chain	exec	cut
T1	fail	42	810	1/10	1.51	ignored	n	y
T2	fail	79	1924	12/15	2.62	ignored	y	y
T3	fail	103	1448	5/12	3.31	ignored	y	y
T4	fail	127	3553	2/7	7.19	ignored	y	y
T5	fail	116	2583	1/9	14.91	used	y	n
T6	fail	64	1443	5/11	4.69	ignored	n	y
T7	fail	18	555	14/18	0.63	never	n	y
T8	fail	47	1134	10/14	1.90	ignored	n	n
T9	fail	67	988	7/11	3.61	ignored	n	y
T10	timeout	80	3609	–	3.23	used	n	–
T11	timeout	210	3608	–	6.91	never	n	–
T12	fail	57	769	2/5	1.38	ignored	n	n
T13	fail	109	3302	1/11	12.26	ignored	y	y
T14	fail	65	954	1/6	1.48	never	y	n
T15	fail	153	1570	5/9	6.17	ignored	y	y
T16	pass	52	770	12/12	3.75	used	y	y
T17	pass	73	2375	8/8	7.69	used	y	n
T18	pass	77	1536	27/27	5.61	used	n	–
T19	pass	93	2323	19/19	10.94	used	y	–
T20	fail	88	3490	3/7	9.46	ignored	y	n
T21	timeout	146	3613	–	19.95	used	y	–
T22	timeout	91	3614	–	5.88	used	y	–
T23	fail	63	1255	13/18	3.40	ignored	n	y
T24	fail	47	1029	13/16	2.83	ignored	n	y
T25	pass	135	2567	8/8	20.43	never	y	y
T26	timeout	142	3609	–	22.60	used	y	–
T27	timeout	245	3609	–	27.72	never	y	–

Table 6: Every planned trial and what became of it.

	Agent A	Agent B
planned	27	81
refused by the platform's safety classifier before any work	0	19
failed to launch	0	1
graded	27	61
of which timed out, kept as failures	6	0
graded with a retained transcript	27	60
passed	5	14
pass rate over graded trials	18.5%	23.0%
pass rate counting every non-graded attempt as a failure	18.5%	17.3%

G Grader Calibration, Per Task

H Comparison With Related Benchmarks

I Sources for the Comparison Table

All sources read 2026-09-25. Section and table numbers refer to the cited version.

J Agentic Benchmark Checklist Self-Audit

We audit MAINFRAME-GYM against the Agentic Benchmark Checklist (Zhu et al., 2025), using its numbering (task validity T, outcome validity O, reporting R). Grading is by recorded cases over business observables, so the unit/end-to-end (O.d, O.f) and state-matching (O.g) checks apply. LLM judges (O.c) gate construction but never set the reward. O.a, O.b, O.e, O.h and O.i do not apply. Y means addressed, P partly addressed or pending, and N not addressed.

Table 7: Grader calibration for the 27 packs. *Designated*: the pack's own FTP / PTP labels. *Null arm*: cases red and green on the unchanged estate. Where red exceeds the FTP count, a case labelled PTP already fails before any fix. The oracle is green on every case of every pack. *Faults*: seeded faults rejected of those built and run. *Harness*: the release of the engine's case harness the pack was recorded under.

		designated		null arm		
problem	cases	FTP	PTP	red	green	faults
0800-promotion-month-count	10	8	2	9	1	7/7
0800-promotion-version-credits	15	6	9	6	9	6/6
0800-set-amount-promotion-credit	15	4	11	4	11	6/6
0800-staged-promotion-percentage	12	7	5	7	5	4/4
atis-extension-billed-off-parent	6	2	4	3	3	7/7
basic-plan-0800-promotion-qualification	16	3	13	3	13	6/6
byvalue-promo-discounts	9	4	5	8	1	7/7
closed-contract-ngeo-select	12	2	10	2	10	7/7
metro-area-local-reclassification	11	6	5	6	5	6/6
credit-ccm-field-rollout	6	3	3	3	3	9/9
day-count-ceiling-999	18	4	14	4	14	6/6
duplicate-0800-promotion-credits	14	4	10	4	10	11/11
local-area-charge-cutover	7	4	3	4	3	6/6
locality-terminal-capacity	18	11	7	11	7	13/13
mobile-network-plan-rating	11	3	8	3	8	8/8
nationwide-0800-contract-pricing	7	5	2	5	2	5/5
operator-assisted-plan-discounts	8	6	2	6	2	8/8
plan-version-discount-loss	9	7	2	7	2	3/3
promotion-201-eligibility	5	2	3	3	2	7/7
promotion-month-schedule	11	8	3	10	1	11/11
recorder-code-through-tracking	5	3	2	3	2	5/5
rerun-history-doubling	8	6	2	6	2	8/8
smp-mobile-0800-billing	27	8	19	8	19	15/15
tcl-consumption-days-stale	6	2	4	2	4	8/8
tcl-stale-reading-correction-by-hand	14	7	7	7	7	11/11
tcl-zeroed-pulses-not-honoured	9	4	5	4	5	7/7
withdrawal-order-period-billing	19	6	13	6	13	11/11
total	308	135	173	144	164	208/208

Table 8: Mainframe and execution-graded benchmarks. *Estate*: tasks span multiple programs, job control and a database. *Grader tested*: the grader's discrimination is measured before agents run. *Production code*: tasks sit in code that ran in production. In MAINFRAME-GYM the defects and reference fixes are constructed (Section 3). Table 9 explains every *partial* cell.

	COBOL	execution graded	production code	estate (JCL, DB)	agentic, long-horizon	grader tested
MainframeBench (Dau et al., 2024)	✓	–	–	–	–	–
COBOLEval (Gordon-Hall, 2024)	✓	✓	–	–	–	–
CodeTransOcean (Yan et al., 2023)	✓	partial	–	–	–	–
SWE-bench Verified (OpenAI, 2024)	–	✓	✓	–	partial	partial
DevOps-Gym (Tang et al., 2026)	–	✓	✓	–	✓	partial
MAINFRAME-GYM (ours)	✓	✓	✓	✓	✓	✓

K TASK AND INSTRUMENT REVIEW

We record every task or instrument defect found during evaluation, whether or not it changed a verdict.

(i) The engine's run guide illustrated its syntax with real member and deck names. In 17 of 27 Agent A trials the agent cited those examples as evidence about the graded chain, and in 9 an example seeded the candidate or the deck (T4, step 6: "it was even the example in the run guide").

(ii) T7: the graded program is a later generation in the archive library, and nothing in the run maps the chain's load-module name to a source member. Kept as an agent failure and flagged for fairness; the same problem defeats all three Agent B attempts.

Table 9: Source of each cell of Table 8.

benchmark	where each cell was read
MainframeBench (Dau et al., 2024) (arXiv v3)	COBOL: COBOL-summarization subtask (§3.2, Tab. 1). Not execution-graded: MCQ scored by accuracy, QA and summarization by "MAP, F1-Score, BertScore, RougeL, Meteor, BLEU-4" (§5.2). Not production code: items generated with GPT and Mixtral models from 300 expert seed items (§3.2). Single-turn, so not agentic. Reference-overlap metrics, so the grader is not tested.
COBOLEval (Gordon-Hall, 2024) (blog, 30 Mar 2024, and repository)	146 HumanEval problems transpiled to COBOL, "paired with an average of 6 test cases", compiled with GnuCOBOL 3.2. Each problem is one subprogram with `LINKAGE SECTION` arguments, called by a generated program per test: no job control, no database. Single completion, so not agentic. Grader not tested: the released `canonical_solution` of every problem is the original Python body, so no COBOL reference is run against the COBOL tests.
CodeTransOcean (Yan et al., 2023) (Findings of EMNLP 2023)	COBOL is one of 37 niche languages in NicheTrans (§3), scored by EM, BLEU and CodeBLEU. *Partial*: execution (DSR@K) exists only for LLMTrans, "seven popular programming languages to Python, totaling 350 program samples" (§3, §4.2), which contains no COBOL. Programs are from Rosetta Code (App. A.2.1), not production.
SWE-bench Verified (Jimenez et al., 2024; OpenAI, 2024)	Real merged pull requests from 12 Python repositories. Execution-graded by FAIL_TO_PASS and PASS_TO_PASS tests (§2.1). *Partial* grader test: instances are kept only if a test goes from fail to pass under the reference patch (§2.1, Stage III), and three annotators per sample checked "if the FAIL_TO_PASS tests might fail even when a valid solution is provided" (Verified). *Partial* horizon: 196 of the 500 tasks are "<15-minute fix" tasks and 45 are ">1-hour" tasks (Verified).
DevOps-Gym (Tang et al., 2026) (ICLR 2026)	Java and Go projects (§3.1). Execution-graded for build (artifacts "pass their dedicated testing cases", §3.2) and for issue resolving. Monitoring is graded by matching the anomaly label (§3.3). Production code: 30+ open-source projects, of which 20 of 54 build tasks and 29 of 34 monitoring tasks are synthetic injections (§3.1). Agentic: "extensive tool use and multi-step planning" plus end-to-end pipeline tasks (§1, §3.5). *Partial* grader test: "we verify that associated tests demonstrate fail-to-pass transitions" (App. C), and "three senior DevOps engineers independently validate" each monitoring task (§3.3).

(iii) T20: the verifier classes 3 of 4 failing FTP cases as missing a frozen input for one DD, although the pack validated at admission.

(iv) T22: a provided stub declares a 15-entry table while all callers pass 10-entry tables.

(v) T18: unreferenced sibling members carry exactly the changes the fix needs.

(vi) T26: the engine truncates inserted source past the fixed-format margin and reports undefined symbols rather than a column diagnostic.

(vii) T27: a suspected capture defect reproduced clean on a minimal unit; the engine is not implicated.

(viii) T13: a change-history tag on the suspect guard may read as camouflage (unconfirmed).

Sensitivity for Agent A: excluding the six timeouts, 5/21 (23.8%); excluding T7, T13, T20 and T22, 5/23 (21.7%); excluding those four and the flagged pass T18, 4/22 (18.2%). Self-truncated listings account for 872 of 1,417 localization commands (62%) across Agent A's trials: 52% on passes and 62% on graded failures.

L Per-Task Results for the Five Pairs

Table 11 gives every cell of the five-pair run. Each cell is passing trials over graded trials, with infrastructure errors in brackets. The other eight green tasks have no runs from every pair yet: `atis-extension-billed-off-parent`, `closed-contract-ngeo-select`, `plan-version-discount-loss`, `tcl-consumption-days-stale`, `basic-plan-0800-promotion-qualification`, `duplicate-0800-promotion-credits`, `day-count-ceiling-999` and `tcl-zeroed-pulses-not-honoured`.

M Pilot Analyses

The analyses below come from the pilot run of Section 6: Agent A (kimi-k3 under Terminus 2, k=1) and Agent B (Claude Fable 5 under a minimal harness, k=3) on the 27 released tasks.

Grading ladder. A binary score hides where a trial stopped, so Table 12 reports four rungs from the per-case verdicts. The headline is the top rung, because the pass-to-pass cases are part of the

Table 10: ABC self-audit. Each row points to where the draft addresses the item.

item	check		where
T.1	tool versions specified	P	Section 4 (GnuCOBOL 3.2, 24 patches)
T.2	tools/APIs accessible	P	Section F (refusals, launch failure)
T.3	API interruptions handled	P	no attempt was retried (Section F)
T.4	state cleared between runs	N	–
T.5	agent isolated from ground truth	Y	sandbox (Section 6), integrity judge (Section M.3), run-guide leak noted (Section K)
T.6	environment frozen, reproducible	P	Section 4, reruns pending (Section M.2)
T.7	ground truth verified	Y	Section 3, Section 4, human review (Section 5)
T.8	tasks solvable	Y	oracle arm (Section 5.2), Section 7
T.9	oracle solver	Y	reference fix, oracle arm (Section 5.2)
T.10	outliers inspected, no exploitable bugs	P	Section K, Section M.2, tamper arm not run on release
O.c.1	judge accuracy vs. humans	P	gate–expert disagreement (Section 5.3), behavior judge unmeasured (Section 7)
O.d.1	tests verified	Y	oracle, null and seeded-fault arms (Section 5.2), human review (Section 5)
O.d.2	objective test-quality metric	Y	208 of 208 seeded faults caught, six probes per case (Table 1, Table 7)
O.f.1	covers relevant code paths	P	PTP contract over closed periods (Section 3)
O.f.2	no flaky results	P	frozen clock and inputs, each arm ran once, reruns pending (Section M.2)
O.g.1	all success states accepted	P	cases pinned to observables (Section 3), no per-task alternate-solution record (Section 5.2)
O.g.2	relevant and irrelevant state checked	Y	FTP, PTP and regression cases (Section 3), Table 12
O.g.3	trivial changes cannot pass	Y	null arm, seeded faults (Section 5.2), null agent (Table 2)
R.1	open-sourced	P	engine and one task (Section 8)
R.2	open evaluation harness	P	engine and rubric files (Section 8)
R.3	contamination measures	Y	22 tasks private (Section 1, Section 8)
R.4	update plan	P	Section 7
R.5	construct stated	Y	Section 1, Section 3
R.6	evaluation subject stated	Y	Section 6, Section 7
R.7	flaw prevention and correction	Y	Section 5, Section 5.3, Section K
R.8	qualitative impact of flaws	Y	Section 4, Section M.2, Section 7
R.9	quantitative impact of flaws	Y	exclusion sensitivity (Section M.2), Table 1, Section F
R.10	statistical significance	Y	Table 2, Table 3
R.11	interpretation guidance	Y	Section 6, Section M.2
R.12	human baseline	N	–
R.13	trivial-agent baseline	Y	null agent (Table 2, Table 12)

Table 11: Passing over graded trials per task and pair, at medium reasoning, with infrastructure errors in brackets.

task	Opus 5.5 Claude Code	Kimi-K3 Claude Code	GPT-6-sol Codex	Grok-4.7 Grok Build	GLM-5.3 Claude Code
`smp-mobile-0800-billing`	4/5 (4)	3/5	5/5	5/6	5/5
`operator-assisted-plan-discounts`	0/5 (4)	2/7 (4)	0/5	3/5 (2)	1/5
`rerun-history-doubling`	1/1	2/2	0/1	0/1	0/1
`mobile-network-plan-rating`	1/1	0/2	1/1	0/1	0/1
`nationwide-0800-contract-pricing`	2/5 (5)	0/5	0/5	0/6 (3)	0/6 (1)
`withdrawal-order-period-billing`	1/1	1/1	0/1	0/1	0/1
`credit-ccm-field-rollout`	0/1	0/1	0/1	0/1	0/1
`0800-promotion-version-credits`	0/5 (5)	0/6 (1)	0/6	0/6 (2)	0/5 (1)

ticket's own contract ("invoiced totals for previously closed periods must remain unchanged"), not a stylistic requirement. The gap between the third and fourth rungs is the regression mode of Table 13: the symptom is gone, but a closed period was rebilled.

Consistency. On the 16 problems with three retained Agent B attempts, pass@1 is 25.0% (12/48), pass@3 is 43.8% (7/16), and pass^3, solving on every attempt, is 12.5% (2/16). The gap from 12.5% to 43.8% is the consistency finding: most problems Agent B can solve, it solves only sometimes. By attempt index, pass rates are 26.3%, 23.8% and 20.0% (5/19, 5/21, 4/20); each index covers a slightly different problem set because of refusals, so this is a range rather than a controlled rerun.

Headroom and non-graded attempts. Fifteen of the 27 problems are solved by neither agent (five of them never attempted by Agent B), eleven are both passed and failed, and only one, `closed-contract-ngeo-select`, is solved on every trial (Section E). Counting all 20 of

Table 12: Grading ladder: trials reaching each rung. The null agent (the unchanged estate) reaches none above R0.

rung	Agent A (n=27)	Agent B (n=61)
R0 shipped and graded (no timeout or abend)	21/27	61/61
R1 at least one FTP case flipped	8/27	38/61
R2 every FTP case flipped	5/27	17/61
R3 pass: every FTP flipped and every PTP held	5/27	14/61

Table 13: Primary failure mode per failed trial. The last row is measured over failures that shipped a fix. The integrity judge charges four of Agent A's six timeouts to harness interaction.

mode	Agent A (n=22)	Agent B (n=46)
wrong program: no edited program runs in the graded chain	2 (9%)	3 (7%)
wrong copy: right program, wrong generation or clone sibling	1 (5%)	5 (11%)
wrong fix: a graded program edited, logic wrong or incomplete	12 (55%)	33 (72%)
regression: FTP cases moved, PTP cases broke	1 (5%)	5 (11%)
no fix shipped: timeout	6 (27%)	0
verification closed on a proxy (compile, or self-written fixture)	9/16 (56%)	9/46 (20%)

Agent B's non-graded attempts as failures lowers its rate from 23.0% to 17.3% (14/81); Section F gives the full admission accounting.

M.1 How agents fail

We assign every failed trial with a retained transcript one primary mode, at the step that decided the outcome. The mode is read from the integrity report, the grader's per-case verdict, and the blind behavior read (Table 13).

Finding a starting point is largely solved; completing the change is not. Editing the wrong program, or the wrong copy of it, accounts for 8 of Agent B's 46 failures and 3 of Agent A's 16 shipped failures. In the remaining failures (38 of 46 and 13 of 16), the agent edited a graded program and the change was wrong or incomplete. Counting Agent A's six timeouts, which shipped nothing, the share is 13 of 22. The rubric rows break these failures into three mechanisms.

The change does not reach every site. Reference fixes touch up to twelve members across clone programs and archived generations. Agent B's edits stayed inside the defect's own code on 91% of failures against 71% of passes, a trend (p=0.08) rather than a significant difference. It checked for other generations of the edited member on only 15 of 46 failures, and on only 5 of 14 passes: rarely either way. Agent A fixed one clone of a family and left the rest in 9 of its 22 failures. All three of Agent B's attempts at `smp-mobile-0800-billing` pass the same 6 of 8 FTP cases and miss the same classifier clone. This is a localization failure too, but a multi-site one: the agent finds where to begin and not everywhere the change must go.

Missing business rules are filled by assumption. Agent B relied on a business rule that neither the ticket nor the estate states in 37% of its failures, against 21% of its passes. It checked its assumptions against the estate in only 35% of failures. One misstep record reads: "changed the call class … on the belief that the customer's negotiated tariff rows carried the mobile class, without inspecting the plan data."

The agent's own check cannot tell a good fix from a bad one. Agent B's own test showed the defect before its edit and gone after it on 13 of its 14 passes and on 40 of its 46 failures. So on nearly every failure the agent had a test telling it the work was done, yet the change was wrong. This fits the finding that tests agents write for themselves are only weakly aligned with success (Chen et al., 2026). In the sharpest form, the fixture is written from the same assumption as the fix and cannot fail. Verification closed on such a proxy in 56% of Agent A's shipped failures and 20% of Agent B's. Agent B ran its changed program after the final edit on 59 of 60 trials and read the output on 58.

What a wrong fix looks like. Two Agent B repairs show the dominant mode. One was "scoped to exactly two edits while leaving the month arithmetic around them untouched", which flipped 7 of 8 FTP cases on `promotion-month-schedule`. Another deleted "the whole second conjunct of the qualification gate" on `basic-plan-0800-promotion-qualification`. That flipped every FTP case and broke three PTP cases. Agent A shows the same habit of guessing. On `mobile-network-plan-rating` it stated that mobile-originated calls carry the origin flag set to mobile and never checked it. A smaller share of failures is a regression tail: a guard removed wholesale, in 7 of Agent A's 22 failures and in all 5 of Agent B's regressions.

The estate's own traps. Some failures come from how the estate is laid out, not from the logic of the fix. At `atis-extension-billed-off-parent` the live deck compiles the archived generation of the program, and the agent edited the other copy. At `day-count-ceiling-999` none of Agent B's three attempts opened the archive at all. These are the traps a real estate has and a single-program benchmark does not.

Table 14: Passes against shipped failures on the three process rows a reader can see: whether every edit stayed on the chain the agent had traced, whether the changed program was run after the final edit, and whether verification closed on a proxy. The chain row separates outcomes for Agent A and not for Agent B. Running the program separates neither.

	Agent A		Agent B	
row	pass (n=5)	fail (n=16)	pass (n=14)	fail (n=46)
edits bounded by the executing chain	4 (80%)	1 (6%)	11 (79%)	39 (85%)
changed program run after the final edit	4 (80%)	8 (50%)	14 (100%)	45 (98%)
verification closed on a proxy	1 (20%)	9 (56%)	1 (7%)	9 (20%)

Failures feed the next round of tasks. In TASKFORGE, a failure the judges explain becomes a seed for new tasks. A clone family the agent fixed only in part, or an archive it never opened, points to the next tickets worth writing.

No process row predicts outcomes across agents. Table 14 shows the three rows a reader can check without a judge. For Agent A, keeping every edit on the chain it had traced separates passes from failures (4/5 against 1/16, p=0.004); the closest row for Agent B shows no difference (11/14 against 39/46, p=0.69), and no row among the twenty separates its outcomes (Section C). The two come from different instruments and power is limited, so we treat the rows as per-agent diagnostics and take the reward from the grader. The labels come from an LLM judge whose agreement with a human reader is not yet measured.

M.2 IS THE ENVIRONMENT THE CULPRIT?

A low pass rate means something only if the environment and grader are not causing it.

Noise floor. HARWELL freezes the clock and fixes the inputs of every run, so the same change should always get the same grade. We have not yet shown this by rerunning the oracle and null arms. Each ran once in task CI, and the reference fix passed every case of every pack. Across Agent B's three attempts, the share of cases passed varies on 9 of its 22 tasks (SD up to 0.71). Agent A ran once per task, so it has no such spread.

Adversarial tests of the grader. Each pack must catch seeded faults, which are wrong answers that still compile and run. The packs caught all 208 that were planted (Section 5.2). The probes check that each case tests exactly what it says it tests.

Losses the environment caused. We record every task or instrument defect found during evaluation (Section K). Five of the 27 problems carry one: a graded program reachable only through an archived generation no mapping exposes, a verifier that misclassifies a missing frozen input, a stub with the wrong table size, sibling members that already contain the fix, and a change-history tag that may read

as camouflage. Separately, the engine's run guide used real member and deck names in its syntax examples; 17 of Agent A's 27 trials cited them as evidence about the graded chain, and in nine an example seeded the candidate or the deck. The integrity judge charges four of Agent A's six timeouts to the harness. In one, an instruction present in no observation was obeyed for 161 of 210 turns; in another, a format error discarded the turn that held the correct fix. Agent A's pass rate stays between 18.2% and 23.8% under every exclusion: without the six timeouts it is 5/21, without the four trials with task defects 5/23, and without those four and the flagged pass 4/22.

M.3 Reward hacking

The integrity judge found no Agent B trial that gamed its reward. None read a hidden test, hard-coded an output or stubbed a program, and every reward, 14 passes and 46 failures, was judged earned. The judge flagged two closing messages as claiming work that was never done. The behavior judge's stricter row G4 fails on 22 of 60 transcripts (3 of 14 passes and 19 of 46 failures, Section C). Neither affects the reward. The grader reads only business observables that the chain produces, so editing a test or a comment cannot move it. It does not by itself rule out a hard-coded output, which is why the integrity judge looks for one, and we have not tested the release adversarially.