AGENTSWE: CAN CODING AGENTS BUILD THE AGENT YOU ACTUALLY WANT?

Anonymous authors
Paper under double-blind review

ABSTRACT

As the production of software passes into the hands of software itself, coding agents are increasingly commissioned to build agents fitted to their users' own workflows. This delegation carries a quiet assumption, that the delivered agent will faithfully serve the requirement it was built for, rather than merely appear to. No existing benchmark tests that assumption. We introduce *AgentSWE*, a benchmark for *agent software engineering* spanning three stages of the agent development lifecycle: **Creation** of a complete agent from a natural-language requirement, **Editing** of production agent codebases, and **Optimization** toward a measurable target. Each of its 25 tasks is framed as a commission rather than an exam: the builder iterates against a handful of illustrative cases, while acceptance exercises the frozen submission on held-out cases designed to span the full requirement, grading behavior against requirement-derived rubrics under hard pass/fail gates or, for Optimization, against the source benchmark's own objective. Evaluating a range of coding agents across models and harnesses, we find that none comes close to delivering what a requirement asks for: the gap between what a builder demonstrates and what it delivers is the norm, not the corner case. An accompanying empirical analysis of the failure trajectories identifies recurring failure modes, among them overfitting to the visible cases, hollow runtimes behind code that looks complete, and unsupported claims of completion, turning verdicts into signals by which coding agents may learn to build the agents they are asked for.

1 INTRODUCTION

Coding agents such as Claude Code (Anthropic, 2026) and Codex (OpenAI, 2026a) now carry substantial engineering workloads (Chen et al., 2021; Yang et al., 2024b; Wang et al., 2025b), and many of their users are not engineers: they describe what they need and delegate the rest (Hitzig et al., 2026). Increasingly, what they delegate is the building of an agent, a program that wraps a model, tools, and state around one recurring job so that the job runs on a fixed runtime, at a fixed cost, without a coding agent attending each run. This delegation carries a quiet assumption: that a coding agent, handed a requirement in natural language, will deliver the agent its user intends rather than one that merely appears to comply. Whether that trust is warranted is what this paper measures.

Developing an agent is a lifecycle that spans three stages. It begins with *creation*, where a requirement stated in natural language must become a working system; continues through *editing*, where new capabilities join a living codebase without unsettling what already works; and matures into *optimization*, where the agent is pushed toward an explicit target, be it a public leaderboard or a production workload. Handing this lifecycle to coding agents invites failures subtler than broken code: an agent that passes every visible example yet misses the intent behind them (Liu et al., 2023; Zhao et al., 2026a; Lodkaew et al., 2026); a tool registered but never invoked (Soni, 2026; Fan et al., 2026); a goal that drifts over a long horizon (Arike et al., 2025; Menon et al., 2026); a report citing evidence no source contains (Onweller et al., 2026; Rao et al., 2026). Such failures are not hypothetical; they are the present condition of agent-built agents (Raj et al., 2026; Naik et al., 2026). The question is how far coding agents compromise the requirements they are handed, on which capabilities, and for what reasons.

These questions outrun the instruments we have. SWE-bench and its descendants grade the repair of conventional software against a reference patch or test suite (Jimenez et al., 2024; Chan et al.,

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

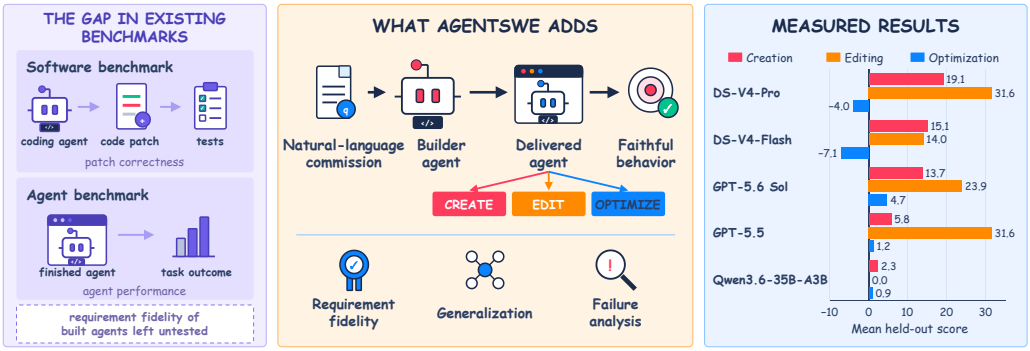

Figure 1: AgentSWE at a glance. (a) Existing benchmarks grade the patch a coding agent writes, or the performance of an agent someone else built; neither asks whether a coding agent delivers the agent a requirement describes. (b) AgentSWE commissions an agent from a natural-language requirement and grades what the delivered agent does on held-out cases, over creation, editing and optimization. (c) Mean held-out Result (S for Optimization) of five builders in the Codex harness (Table 1).

2024); NL2Repo-Bench commissions whole repositories from requirement documents (Ding et al., 2025); Paper2Code and PaperBench rebuild research implementations checked against the papers they reproduce (Seo et al., 2025; Starace et al., 2025). Each judges ordinary code against an answer known in advance. A commissioned agent wraps a stochastic policy around tools, answers to a requirement that has no reference solution, and keeps changing after delivery. Benchmarks that do evaluate agents, from AgentBench onward (Liu et al., 2024), examine a finished agent and ask how well it performs; none asks whether a coding agent could have built it. Work on agents that design or improve other agents (Hu et al., 2025; Mishra et al., 2026) comes closest, but it measures how far a search procedure or a meta-agent can raise a benchmark score, not whether a general coding agent delivers the agent a requirement describes (Section A). **Between judging built software and judging finished agents lies a step that current suites leave unmeasured: building the agent that a requirement asks for.**

We introduce *AgentSWE*, a benchmark that turns faithful delegation from anecdote into measurement (Figure 1) through three commitments. The first is to measure the whole lifecycle, through twenty-five tasks spanning all three stages. Ten *Creation* tasks commission a complete agent from a natural-language requirement, over workloads from GUI automation to formal theorem proving, each scoped against a mature open-source system; ten *Editing* tasks ask for new capabilities in the pinned codebase of a production agent such as OpenHands or Aider, capabilities that cut through crash recovery, permission fencing, and transactional state; and five *Optimization* tasks set a working agent against a measurable target. The second commitment is that evaluation should mirror the delegation it studies. Each task is framed as a commission: **the builder sees only what a customer could provide, while acceptance also tests the stated clauses its examples leave out.** The builder develops against the requirement and a few illustrative cases; held-out cases then test the frozen delivery in isolation. The third commitment follows from the absence of a prescribed reference implementation: every Creation and Editing execution is graded against a rubric derived from the requirement itself, and hard gates assign zero to invalid executions and inapplicable patches.

We put this instrument to work on five models as builders in the Codex harness, together with further pairings of harness and model. The verdict is sobering. The strongest builders average 19.1 on Creation and 31.6 on Editing, and in Optimization no builder closes more than 5 % of the remaining gap to the target on average, and only 13 of the 25 runs score above their starter. Creation runs that end development at 90 or above deliver agents averaging 9.9 on held-out cases. **On present evidence, the gap between what a builder demonstrates and what it delivers is not the corner case of agent software engineering; it is the norm.** The builders' trajectories reveal recurring failures. Builders overfit to the visible development cases, and the delivery fails on the clauses those cases leave out; they ship hollow runtimes, code that reads as complete but does no work when run; and both builders and the agents they deliver claim completions that their own records contradict. More

development cases, a prescribed design, an instruction to test, and a longer case budget shift the scores but leave held-out Result low under every condition. Each of these modes names a habit that the next generation of coding agents can be built and trained to avoid.

In sum, this paper makes three contributions. First, we formalize agent software engineering as a measurable problem, in which a builder must satisfy a requirement it can probe only through limited development feedback. Second, we deliver AgentSWE, twenty-five commissions whose acceptance executes the delivered agent on held-out cases and grades each run with a requirement-derived rubric or a native objective, and we evaluate a range of pairings of harness and model against it. Third, we contribute an empirical analysis of the failure trajectories across the three stages, which identifies recurring failure modes of coding agents and turns each into a direction for the next generation of coding agents.

2 Problem Formulation

A commission prescribes no reference implementation, so we define the requirement, the delivered agent, the builder, and the fidelity of a delivery through observable behavior alone.

2.1 Preliminaries

Requirements. A commission begins with a natural-language requirement $q = (g, \mathcal{I}, \mathcal{C}, \mathcal{R})$, which every AgentSWE task publishes in four parts: the goal g, which states what the agent is for and what counts as serving its user; the interface contract $\mathcal{I}$, which fixes how the agent is launched and which artifacts it must deliver; the constraints $\mathcal{C}$, which bound what may be edited, installed, or invoked; and the resources $\mathcal{R}$, such as data, repositories, and access to models and search. A *case* x is one concrete instance of the commission. The requirement comes with a small set D of public development cases, while the cases that matter, the ones the delivered agent will face in use, remain unwritten; we write $\mathcal{X}_q$ for the space of cases the requirement implies.

Agents. The deliverable is itself an agent $a = (h, m)$: a *harness* h, the orchestration code that plans, routes, and retries (Raj et al., 2026; Lee et al., 2026), around a *backbone model* m reached through a fixed gateway. Executing a on a case x in environment E yields observable outcomes $y \sim a(x; E)$. Because m samples, the same delivery may satisfy a requirement on one execution and betray it on the next, so acceptance executes the agent and grades what it does.

Builders. A coding agent, or *builder*, is an agent β whose actions are software engineering: reading a requirement, exploring and editing a workspace, running what it wrote, and deciding what to try next (Yang et al., 2024b; Wang et al., 2025b). Handed a commission, the builder produces an artifact $\hat{a} \sim \beta(q, a_0, D; B)$, where a_0 is the initial artifact, possibly empty, D the development cases it may probe freely, and B a budget on time and compute.

2.2 The Agent-for-Agent Problem

The customer cares about how faithfully $\hat{a}$ serves the requirement on cases it has not seen. Let $\mu_q(x, y) \in [0, 1]$ be the graded satisfaction of requirement q on case x with outcome y, and let $V(\hat{a}) \in \{0, 1\}$ be a validity gate on the non-negotiables of $\mathcal{I}$ and $\mathcal{C}$: the agent launches, its deliverables parse, its changes respect the authorized surface. The *fidelity* of a delivered agent is

$$F(\hat{a};\, q) \;=\; V(\hat{a}) \cdot \mathop{\mathbb{E}}_{x \sim \mathcal{X}_q} \; \mathop{\mathbb{E}}_{y \sim \hat{a}(x;E)} \big[\, \mu_q(x, y) \,\big]. \tag{1}$$

Definition 2.1 (Agent software engineering)**.** Given a requirement q, an initial artifact a_0, development cases D, and a budget B, a builder β must deliver an agent $\hat{a} \sim \beta(q, a_0, D; B)$ maximizing fidelity $F(\hat{a}; q)$.

We estimate fidelity on a held-out set $\bar{D} \subset \mathcal{X}_q$, disjoint from D and designed to exercise the promises in g, graded by a rubric that realizes μ_q. The gap between demonstrated and delivered fidelity, $F_D - F_{\bar{D}}$, measures how much better the delivered agent does on the cases the builder saw than on the cases it did not.

2.3 Task Families

The three lifecycle stages are instances of Definition 2.1 that differ in the initial artifact a_0 and in the shape of μ_q.

Creation ($a_0 = \emptyset$). The builder starts from an empty workspace and must deliver a complete agent under the interface contract $\mathcal{I}$; fidelity is estimated on held-out cases as in Equation (1).

Editing ($a_0 =$ a production system). The initial artifact is the pinned codebase of a working agent product, and the requirement commissions new capabilities. The constraints $\mathcal{C}$ confine the builder's changes to an authorized region, and satisfaction has two parts,

$$\mu_q \;=\; \mu_{\text{new}} \wedge \mu_{\text{reg}}, \tag{2}$$

the commissioned behavior μ_{new} and the preservation μ_{reg} of everything the product already did, checked by its public suite and published contracts (Jimenez et al., 2024).

Optimization ($a_0 =$ a working agent). The initial artifact already satisfies the interface, and the requirement names a measurable objective J on a 0–100 scale, whose ceiling $J^\star = 100$ is the target. Delivery is judged by the fraction of the remaining gap to the target that the builder closes (cf. Wijk et al., 2025),

$$\gamma(\hat{a}) \;=\; \min\left\{1,\; \frac{J(\hat{a}) - J(a_0)}{J^\star - J(a_0)}\right\}, \tag{3}$$

on held-out instances of the objective.

3 The AgentSWE Benchmark

AgentSWE turns Definition 2.1 into 25 commissions: ten Creation, ten Editing, and five Optimization tasks. Each task hands a coding agent, the builder, a requirement q, an initial artifact a_0, and development cases D, and grades the agent it delivers on held-out cases. Figures 2 and 3 show how tasks are built and evaluated; Section C walks through one task per family.

3.1 Task Construction

Each family fixes the requirement q and the initial artifact a_0 in its own way (Figure 2a). In every family, Codex with GPT-5.6 Sol (OpenAI, 2026c) at `xhigh` effort drafts the task materials from the analyzed source system, and the authors verify and refine each draft. Creation tasks come from open-source domain agents on GitHub, in domains from desktop automation and scientific PDF translation to formal theorem proving. Each repository is reverse-engineered into a self-contained goal and a set of functional requirements, extended with capabilities that broaden its purpose and raise the compositional difficulty, and published as input Markdown files that state the goal, interface contract, constraints, and resources of q; the initial artifact is empty.

Editing tasks extend a pinned repository snapshot a_0 of five general-purpose coding agents and five domain agents, among them Claude Code (Anthropic, 2026), Aider (Gauthier, 2026), OpenHands (Wang et al., 2025b), and OpenClaw (OpenClaw Contributors, 2026). The commissioned capabilities come from design brainstorming or from unresolved issues in the repository; we keep proposals with a consequential user need, a bounded implementation surface, and behavior that execution can verify, and write each as a feature specification with an acceptance boundary and the regression obligations that a_0 already carries.

Optimization tasks build on BrowseComp (Wei et al., 2025), τ^3 (Yao et al., 2024; Barrès et al., 2025), TerminalBench (Merrill et al., 2026), PinchBench (Kilo AI, 2026), and OSWorld (Xie et al., 2024) and keep their native task definitions. Each ships a runnable starter agent as a_0 (Table 5), and the requirement names the objective, the files the builder may edit, and the runtime resources it may not change.

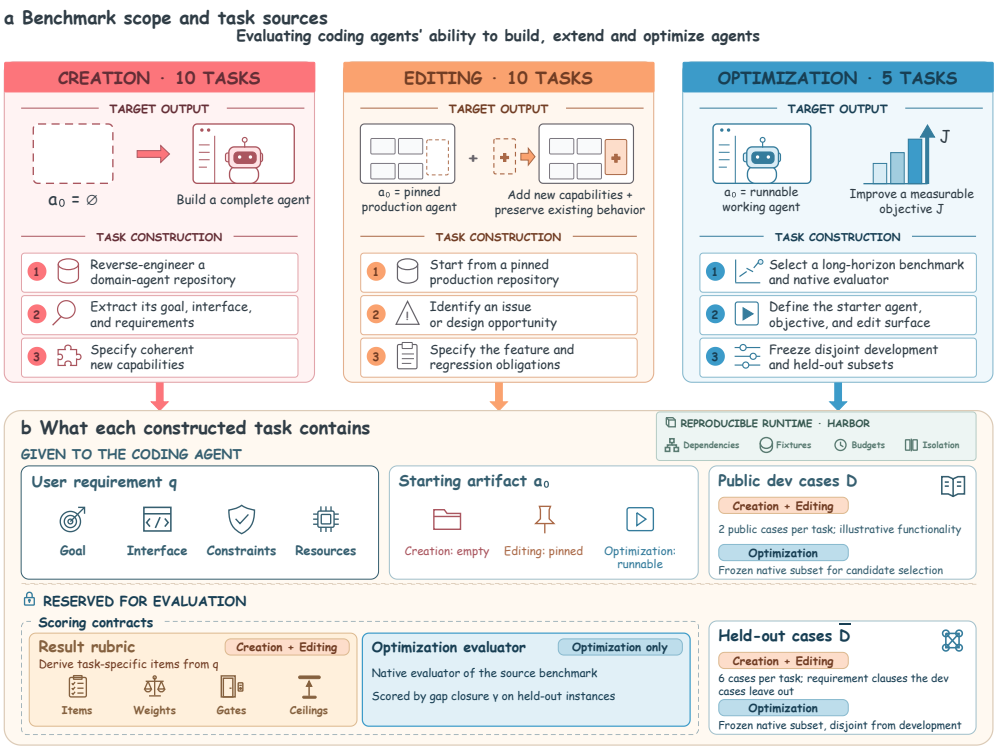

Figure 2: AgentSWE task families and benchmark construction. (a) Task objectives and construction processes. (b) Builder-visible inputs, evaluator-owned scoring contracts and held-out cases, supported by Harbor environments.

3.2 Task Package

Each task separates what the builder sees, namely q, a_0, and the development cases D, from what evaluation reserves (Figure 2b). Every Creation and Editing task publishes two development cases, which demonstrate the entry point, the delivery contract, and a small part of the required functionality; Optimization draws a development split from the instance pool of its source benchmark.

Evaluation reserves the held-out cases and the scoring contract. Each Creation and Editing task holds out six cases. Each case exercises one or more requirement clauses, among them clauses and capability interactions that the requirement states but the development cases leave out, and together the six cover diverse inputs, state, and data. Optimization holds out a disjoint split of the same pool. In total the suite has 150 development and 365 held-out cases. The scoring contract of a Creation or Editing task is a task-specific *Result* rubric whose weighted items derive from the functional requirements, with gates and ceilings for violations of the interface contract and the constraints (Section D); Optimization keeps the native evaluator of its source benchmark. Held-out inputs, evaluator state, scoring contracts, credentials, and model routing never enter the builder's workspace, and the builder receives feedback only on its development cases.

3.3 Evaluation Protocol

Development and freezing. Harbor (Harbor Framework Team, 2026), a container framework for agent evaluation, runs the builder in one persistent session (Figure 3, left). The builder works in a private writable workspace and may submit a snapshot of its delivery at any time. Each snapshot runs on the development cases in isolated jobs, and the scores and summarized feedback return to the same session, within fixed time and submission budgets (Section F). Development ends when the builder exits or exhausts its submissions or its time budget. Creation then freezes the workspace; Editing freezes the last accepted submission (one that passes the pre-run checks and is run on the development cases), so a builder whose submissions were never accepted delivers nothing; Optimization freezes

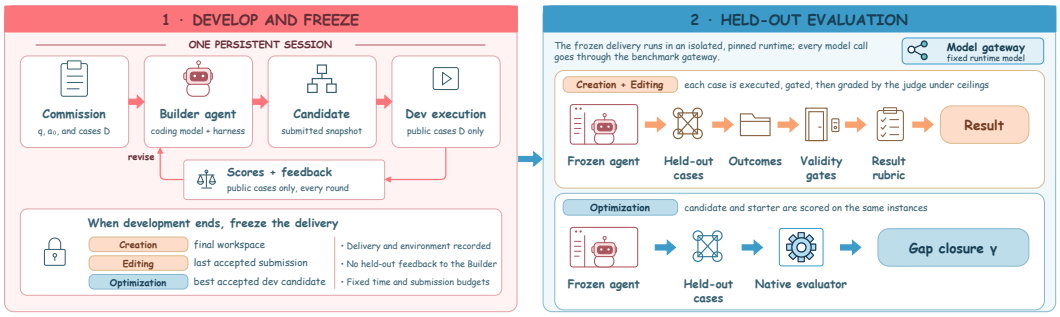

Figure 3: AgentSWE evaluation workflow: development with public feedback, delivery selection and freezing, and held-out evaluation using task-specific Result rubrics or native benchmark objectives.

the accepted snapshot with the highest development objective. Freezing records the delivery and its software environment, and the held-out evaluation verifies that neither has changed.

Held-out execution. Each held-out case runs the frozen delivery through its fixed entry point in an isolated, pinned runtime (Figure 3, right; Section E). Every model call the delivered agent makes goes through a benchmark-owned gateway to one runtime model, shared by all deliveries in a task family, so that the score reflects what the builder contributes: prompts, tool orchestration, context and state management, and output validation.

Scoring. Optimization scores the selected candidate and the starter with the native evaluator on the same held-out instances, a signal independent of the Result judge, and reports the gap closure of Equation (3). For Creation and Editing, deterministic gates first assign zero to an execution that does not launch, exceeds its time budget, or produces no parseable deliverable. A frontier reasoning model at its highest effort then applies the Result rubric (Zheng et al., 2023; Arora et al., 2025) (GPT-5.6 Sol for Creation, DeepSeek-V4.1-Flash (DeepSeek-AI, 2026b) for Editing; Table 8). The judge grades what the agent produced on one concrete case, its artifacts, execution trace, and resulting state, rather than the quality of its code, and ceilings bind the score to the violations the evaluator records. The task score is the mean partial-credit score over its held-out cases on a 0–100 scale, and family scores are macro-averages over tasks.

4 Experiments

Section 4.2 measures held-out performance, Section 4.3 analyzes the trajectories of these runs and the failure modes they reveal, and Section 4.4 examines how these failures respond to changes in the conditions of the build.

4.1 Setup

The main comparison evaluates five builders that pair the Codex harness with GPT-5.6 Sol, GPT-5.5 (OpenAI, 2026b), DeepSeek-V4-Pro, DeepSeek-V4-Flash (DeepSeek-AI, 2026a), and Qwen3.6-35B-A3B (Qwen Team, 2026); we refer to each builder by its model. A *run* is one builder developing one task, from the commission to the frozen delivery. Creation and Editing report the mean Result over the six held-out cases of each task. Optimization reports gap closure $S = 100\gamma = 100 \min\{1, \Delta/(100 - J(a_0))\}$, where $\Delta = J(\hat{a}) - J(a_0)$ is the change in the native objective on the held-out split; S is negative when the candidate scores below its starter. Section F gives the settings for each task family, and Section K its limitations.

4.2 Held-out Performance

Creation success is rare and task-specific. The 50 Creation runs in Table 1 average a Result of 11.2 (median 3.4), and 20 of them score zero. The highest-scoring builder, DeepSeek-V4-Pro, averages 19.1 yet scores below 20 on eight of ten tasks. Only three runs exceed 50, all on formal

Table 1: Held-out performance across the three lifecycle stages. Creation and Editing report mean Result (0–100); Optimization reports gap closure $S = 100\gamma$. Section H.3 gives per-run Optimization results. DS denotes DeepSeek.

Task	GPT-5.6 Sol	GPT-5.5	DS-V4-Pro	DS-V4-Flash	Qwen3.6
Creation					
Authorized vulnerability validation	0.0	0.0	14.0	14.0	0.0
Database analytics	3.0	9.8	14.3	0.0	0.0
Desktop GUI automation	18.0	3.2	17.5	12.0	0.0
Document-to-editable PPTX	0.0	9.8	18.7	16.8	21.7
Evidence-grounded document QA	3.3	12.2	0.0	3.5	0.0
Formal theorem proving	66.7	0.0	16.5	63.0	0.0
Repository bug repair	27.7	0.0	60.0	0.0	0.0
Schema-guided web extraction	0.0	0.0	6.7	0.0	0.0
Scientific PDF translation	18.3	21.8	25.3	26.0	0.0
Web research report	0.0	0.8	18.0	15.3	1.7
Mean	13.7	5.8	19.1	15.1	2.3
Editing					
Claude policy provenance	24.0	43.7	28.3	0.0	0.0
Aider worktree transaction	21.5	20.0	47.3	0.0	0.0
OpenHands effect recovery	4.0	20.5	24.3	0.0	0.0
OpenClaw channel handoff	38.5	12.3	19.5	0.0	0.0
Codex execution residual	11.7	70.0	35.0	0.0	0.0
AI-Scientist reproducibility gate	48.5	67.3	61.7	54.3	0.0
DeepCode claim traceability	26.5	41.3	32.7	47.0	0.0
DeepTutor adaptive remediation	54.2	14.3	31.0	25.7	0.0
Dyad acceptance-driven workflow	9.8	18.0	22.5	0.0	0.0
OpenWiki change-impact documentation	0.0	8.5	14.0	12.8	0.0
Mean	23.9	31.6	31.6	14.0	0.0
Optimization					
BrowseComp	32.3	18.4	−1.0	34.1	8.5
τ^3	4.3	0.0	15.8	−50.0	−23.8
TerminalBench	−10.0	7.7	0.0	2.8	7.7
PinchBench	−14.9	−11.1	−38.7	−34.4	12.1
OSWorld	12.0	−9.1	4.0	12.0	0.0
Mean	4.7	1.2	−4.0	−7.1	0.9

theorem proving or repository bug repair, and no builder exceeds 30 on more than one task. Success on one commission does not carry over to the next, and none of the evaluated builders reliably builds the agents that AgentSWE specifies.

Editing yields partial capabilities, or no delivery at all. The builders that deliver on nearly every task, DeepSeek-V4-Pro, GPT-5.5, and GPT-5.6 Sol, still realize only part of the commissioned capabilities: their means lie between 20 and 35, and only one run, GPT-5.5 on the Codex task, reaches 70. Qwen3.6 fails earlier: it submits no candidate on any task, typically ending its session right after announcing a plan. DeepSeek-V4-Flash delivers on four tasks; on most of the others it works until the budget runs out without ever submitting a candidate. Adding capabilities to a working product thus fails in two ways: the patch covers only part of the requirement, or the builder never finishes one.

Optimization gains are inconsistent across tasks. No builder closes more than 5 % of the remaining gap to the target on average. Of the 25 runs, 13 score above the starter, nine below it, and three match it; a per-run sign test supports three of the gains and two of the losses (Section H.3). Gains depend on the task: on BrowseComp four of five builders score above the starter, whereas on PinchBench four of five score below it.

4.3 Trajectory Analysis and Failure Modes

Analyzing the builders' trajectories, the frozen artifacts, and the delivered agents' execution traces in the main runs, we find three recurring failure modes (Section I gives the protocol).

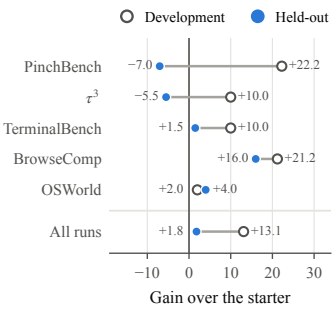

Figure 4: Optimization: gain over the starter on development (open) and held-out (filled) cases, averaged over five builders per benchmark.

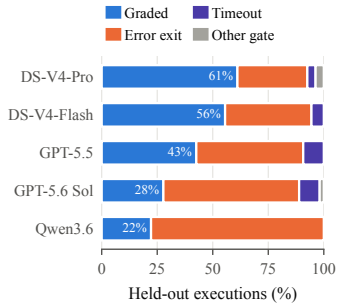

Figure 5: Outcomes of held-out Creation executions by builder. Only graded executions reach the Result rubric.

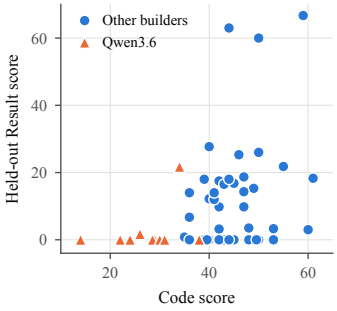

Figure 6: Code score against held-out Result score for the 50 Creation runs.

Overfitting to the visible cases. Development scores overstate what a delivery does on the rest of the requirement. Eleven Creation runs end development with a score of at least 90, yet their held-out Result averages 9.9. A DeepSeek-V4-Pro Editing delivery scores 100 on development cases and 28.3 on held-out cases. In Optimization, the selected candidate gains 13.1 points on development cases but only 1.8 on held-out cases (Section H.4), although both splits come from one instance pool; on four of the five benchmarks the gains shrink or reverse (Figure 4).

The trajectories and artifacts reveal case-specific implementations: builders fit their agents to the visible cases in 14 of 25 Optimization runs and 13 of the 33 Editing runs that deliver. On PinchBench, DeepSeek-V4-Flash and DeepSeek-V4-Pro route tasks by keyword to playbooks for the visible task categories, gaining 8.1 and 59.4 points on development cases and losing 15.2 and 15.1 on held-out cases. Even explicit feedback leaves such defects in place: in eight of ten DeepSeek-V4-Pro Editing runs, development feedback names a violated requirement that the delivery still violates on held-out cases (Section I.3). The requirement states what the held-out cases test; what fails is the step from a requirement with interacting capabilities to an agent that generalizes beyond the visible cases.

Hollow runtimes. Code that reads as complete often does not work when run. In Creation the failure is loud: 157 of the 270 traced non-GUI held-out executions (58.1 %) fail a validity gate, nearly all by exiting with an error or timing out (Figure 5). Yet a static review of every frozen Creation source, by the same judge as Result on an eight-dimension rubric (Section H.1), does not separate deliveries with zero Result from the rest (Figure 6): apart from Qwen3.6, lowest on both, deliveries with zero Result score 44.2 and those with nonzero Result 45.8.

In Editing, 28 of the 33 runs that deliver leave some added capabilities absent, uninvoked, or incompletely executed (Section I.3). These failures can be silent: an OpenWiki delivery passes every validity gate yet leaves four of its five required output categories empty on every held-out case.

Unsupported claims of completion. Reports of success are unreliable at both levels of the delegation. Checked against the builder's own trajectory, artifact, and feedback, 6 of the 42 final Creation reports and 3 of the 19 Optimization reports claim completions that these records contradict (Table 12). All nine assert completion; no report that admits partial failure is contradicted. Delivered Editing agents overclaim at run time too: in 13 of the 33 runs that deliver, they report outcomes that their own effect logs or state contradict, and the OpenWiki delivery above declares its run complete.

4.4 Discussion

More development cases do not help monotonically. We vary the conditions of the build on two Creation tasks, with three builder runs per condition (Section F.2). Going from zero to two development cases raises mean Result by 34.3 points on PDF translation and by 29.3 on PPTX generation (Table 2); without any case, all three PDF deliveries score zero. Two further cases add

Table 2: Development cases: mean (sd) Result over three builder runs per condition on the four held-out cases shared by all conditions. Builder: Codex with DeepSeek-V4.1-Flash.

Task	0 cases	1 case	2 cases	4 cases
PDF	0.0 (0.0)	21.7 (18.2)	34.3 (6.0)	33.2 (28.8)
PPTX	4.2 (7.2)	15.0 (2.8)	33.5 (15.0)	15.2 (1.8)

Table 3: Prompt priors: mean (sd) Result over three builder runs on six held-out cases, with the builder of Table 2.

Task	No prior	Strategy A	Strategy B
PDF	39.2 (14.2)	40.4 (9.3)	48.9 (12.6)
PPTX	29.6 (12.6)	7.4 (6.9)	31.2 (4.5)

Table 4: Pairings of harness and model, one run each. Creation columns are held-out Result; Optimization columns are S. Codex rows repeat Table 1; dashes mark pairings not run. Sources: DSH (DeepSeek Harness) (DeepSeek-AI, 2026), OpenCode (Anomaly, 2026), Pi (Earendil Works, 2026), and GLM-5.3 (GLM-5-Team et al., 2026).

		Creation		*Optimization*	
Harness	**Model**	**GUI**	**PDF**	τ^3	**PinchB.**
Codex	GPT-5.6 Sol	18.0	18.3	4.3	−14.9
Codex	GPT-5.5	3.2	21.8	0.0	−11.1
Codex	DS-V4-Pro	17.5	25.3	15.8	−38.7
Codex	DS-V4-Flash	12.0	26.0	−50.0	−34.4
DSH	DS-V4-Pro	19.7	20.3	−28.6	−39.7
DSH	DS-V4-Flash	18.7	30.5	4.8	−1.7
Claude Code	DS-V4-Pro	16.2	15.3	–	–
Claude Code	DS-V4-Flash	12.7	5.8	–	–
Claude Code	GLM-5.3	0.0	19.2	−19.0	13.0
OpenCode	GPT-5.6 Sol	0.0	26.7	–	–
OpenCode	GPT-5.5	0.0	42.8	–	–
Pi	GPT-5.6 Sol	0.0	18.3	−4.8	33.8
Pi	GPT-5.5	0.0	0.0	–	–

nothing on PDF and lower PPTX from 33.5 to 15.2, and all three four-case PPTX runs fall between 14.0 and 17.2 (Figure 7).

The tested prompt priors yield mixed results. We also append one of two prompt priors to the builder instruction (Section E.1): Strategy A specifies the components and how they connect, and Strategy B asks the builder to write additional cases and run its agent end to end. Strategy A leaves PDF unchanged and lowers PPTX from 29.6 to 7.4, with no PPTX run above 14; Strategy B raises PDF from 39.2 to 48.9, within run-to-run variation, and leaves the PPTX mean unchanged while narrowing its range from 15.2–38.8 to 26.0–34.0 (Table 3).

The failure modes persist in a second configuration. In the main runs, only 15 of 270 held-out Creation executions reach the 600 s limit (Figure 10). We also rebuild all ten Creation tasks with 1,500 s per case and a requirement that explicitly asks for a runtime agent emitting an execution trace, with DeepSeek-V4.1-Flash as builder (in Codex), runtime model, and judge. On eight of ten tasks the best-scoring case uses at most a third of the budget, yet both failure modes persist: the last development score exceeds held-out Result on every task, 70.3 against 31.5 on average (Table 9), and a third of the held-out executions produce no valid result. None reaches the 1,500 s limit.

Scores vary across harness–model pairings. With the model held fixed, scores differ across harnesses (Table 4): on scientific PDF translation, GPT-5.5 scores 21.8 under Codex and 42.8 under OpenCode, and DeepSeek-V4-Flash 5.8 under Claude Code and 30.5 under DSH. No harness leads for every model, so we evaluate pairings of harness and model, not models alone, and both are levers for closing the gap between what a builder demonstrates and what it delivers.

5 Conclusion

We introduced AgentSWE to measure whether coding agents deliver agents that serve their requirement on cases they have not seen. Current builders rarely do: in the main comparison, the strongest average 19.1 on Creation and 31.6 on Editing, and in Optimization none closes more than 5 % of the remaining gap to the target on average. The trajectories reveal recurring failures: builders fit the visible cases, ship runtimes that read as complete but do not work, and, like the agents they deliver, claim completions their own records contradict. These failure modes chart where coding agents need to improve; our future work will pursue harnesses that test against the requirement and verify completion claims, and training that rewards behavior on unseen cases.

AI Use Statement

We used generative AI tools for three tasks. First, to construct the benchmark, we used Codex with GPT-5.6 Sol to synthesize task materials from the analyzed source systems (Section 3.1): the requirement documents, the development and held-out cases, and the task-specific Result rubrics. The authors designed each commission and reviewed and revised the generated materials. Second, AI agents assisted the trajectory analysis (Sections 4.3 and I) by labeling failure modes from builder trajectories, execution traces, and evaluator records; the authors reviewed the labels and the resulting counts. Third, we used AI assistants to aid and polish the writing of this paper. Coding agents and language models are also the object of study, acting as builders, runtime models, and judges (Sections 4.1 and F); this use is part of the method rather than assistance to the authors. We have reviewed all AI-assisted content and take full responsibility for the paper.

Ethics Statement

AgentSWE asks coding agents to build agents that execute code, access the web, and, in one Creation task, validate software vulnerabilities. All builder sessions and candidate executions run in isolated containers with pinned fixtures, and every model call passes through a benchmark-owned gateway (Section E). The benchmark involves no human subjects. Its tasks derive from public repositories and public benchmarks, used under their licenses and terms. Our analysis documents how coding agents misreport their own work, which we regard as a safety-relevant property of delegating software to them.

Reproducibility Statement

Section 3 and Sections C and D describe the tasks, cases, and scoring. Sections E and F give the evaluation infrastructure, builder instructions, model versions, effort settings, and budgets. Section G reports repeated runs and the variation of repeated evaluation, Section H lists per-run results, and Sections I and J describe the analysis protocol and the traced cases. The task packages as builders receive them, together with the Creation and Editing rubrics and judge prompts, are available at https://anonymous.4open.science/r/agentswe. With the public version we will release the held-out cases and evaluators through a gated download that protects the held-out split, together with the frozen deliveries and per-run results; where an upstream license does not permit redistribution, we release our changes against the pinned upstream commit instead.

References

Lakshya A. Agrawal, Shangyin Tan, Dilara Soylu, Noah Ziems, Rishi Khare, Krista Opsahl-Ong, Arnav Singhvi, Herumb Shandilya, Michael J. Ryan, Meng Jiang, Christopher Potts, Koushik Sen, Alexandros G. Dimakis, Ion Stoica, Daniel Klein, Matei Zaharia, and Omar Khattab. GEPA: reflective prompt evolution can outperform reinforcement learning. *CoRR*, abs/2507.19457, 2025. doi: 10.48550/ARXIV.2507.19457. URL https://doi.org/10.48550/arXiv.2507.19457.

Anomaly. OpenCode: The open source coding agent. https://github.com/anomalyco/opencode, 2026. Version 1.18.25.

Anthropic. Claude Code. https://code.claude.com/docs/en/overview, 2026.

Rauno Arike, Elizabeth Donoway, Henning Bartsch, and Marius Hobbhahn. Technical report: Evaluating goal drift in language model agents. *CoRR*, abs/2505.02709, 2025. doi: 10.48550/ARXIV.2505.02709. URL https://doi.org/10.48550/arXiv.2505.02709.

Rahul K. Arora, Jason Wei, Rebecca Soskin Hicks, Preston Bowman, Joaquin Quiñonero Candela, Foivos Tsimpourlas, Michael Sharman, Meghan Shah, Andrea Vallone, Alex Beutel, Johannes Heidecke, and Karan Singhal. Healthbench: Evaluating large language models towards improved human health. *CoRR*, abs/2505.08775, 2025. doi: 10.48550/ARXIV.2505.08775. URL https://doi.org/10.48550/arXiv.2505.08775.

Victor Barrès, Honghua Dong, Soham Ray, Xujie Si, and Karthik Narasimhan. τ^2-bench: Evaluating conversational agents in a dual-control environment. *CoRR*, abs/2506.07982, 2025. doi: 10.48550/ARXIV.2506.07982. URL https://doi.org/10.48550/arXiv.2506.07982.

Jun Shern Chan, Neil Chowdhury, Oliver Jaffe, James Aung, Dane Sherburn, Evan Mays, Giulio Starace, Kevin Liu, Leon Maksin, Tejal Patwardhan, Aleksander Madry, and Lilian Weng. MLE-bench: Evaluating machine learning agents on machine learning engineering. *arXiv preprint arXiv:2410.07095*, 2024.

Guangyao Chen, Siwei Dong, Yu Shu, Ge Zhang, Jaward Sesay, Börje Karlsson, Jie Fu, and Yemin Shi. Autoagents: A framework for automatic agent generation. In *Proceedings of the Thirty-Third International Joint Conference on Artificial Intelligence, IJCAI 2024, Jeju, South Korea, August 3-9, 2024*, pp. 22–30. ijcai.org, 2024. URL https://www.ijcai.org/proceedings/2024/3.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, et al. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*, 2021.

Yizhe Chi, Wenyi Li, Deyao Hong, Xiaoqiu Wang, Mingju Gao, Kaisen Yang, Bingxiang He, Youjie Zheng, Calvin Xiao, and Qinhuai Na. Ai4ai-bench: Benchmarking llm agents in algorithmic design for recursive self-improvement, 2026. URL https://arxiv.org/abs/2608.20318.

DeepSeek-AI. Deepseek harness: Everything is a plugin. https://github.com/deepseek-ai/deepseek-harness, 2026.

DeepSeek-AI. Deepseek-v4: Towards highly efficient million-token context intelligence. *CoRR*, abs/2606.19348, 2026a. doi: 10.48550/ARXIV.2606.19348. URL https://doi.org/10.48550/arXiv.2606.19348.

DeepSeek-AI. DeepSeek-V4.1-Flash: Pushing the limits of KV cache compression, 2026b. URL https://arxiv.org/abs/2609.19969.

Jingzhe Ding, Shengda Long, Changxin Pu, Huan Zhou, Hongwan Gao, Xiang Gao, Chao He, et al. NL2Repo-Bench: Towards long-horizon repository generation evaluation of coding agents. *arXiv preprint arXiv:2512.12730*, 2025.

dyad-sh. Dyad: Local, open-source AI app builder. https://github.com/dyad-sh/dyad, 2026.

Earendil Works. Pi: An AI agent toolkit and coding-agent CLI. https://github.com/earendil-works/pi, 2026. Version 0.84.3.

Shengda Fan, Xuyan Ye, Yupeng Huo, Zhi-Yuan Chen, Yiju Guo, Shenzhi Yang, Wenkai Yang, Shuqi Ye, Jingwen Chen, Haotian Chen, Xin Cong, and Yankai Lin. AgentProcessBench: Diagnosing step-level process quality in tool-using agents. *arXiv preprint arXiv:2603.14465*, 2026.

Paul Gauthier. Aider: AI pair programming in your terminal. https://github.com/Aider-AI/aider, 2026.

GLM-5-Team, Aohan Zeng, Xin Lv, Zhenyu Hou, Zhengxiao Du, Qinkai Zheng, Bin Chen, Da Yin, Chendi Ge, Chenghua Huang, Chengxing Xie, Chenzheng Zhu, Congfeng Yin, Cunxiang Wang, Gengzheng Pan, Hao Zeng, Haoke Zhang, Haoran Wang, Huilong Chen, Jiajie Zhang, Jian Jiao, Jiaqi Guo, Jingsen Wang, Jingzhao Du, Jinzhu Wu, Kedong Wang, Lei Li, Lin Fan, Lucen Zhong, Mingdao Liu, Mingming Zhao, Pengfan Du, Qian Dong, Rui Lu, Shuang-Li, Shulin Cao, Song Liu, Ting Jiang, Xiaodong Chen, Xiaohan Zhang, Xuancheng Huang, Xuezhen Dong, Yabo Xu, Yao Wei, Yifan An, Yilin Niu, Yitong Zhu, Yuanhao Wen, Yukuo Cen, Yushi Bai, Zhongpei Qiao, Zihan Wang, Zikang Wang, Zilin Zhu, Ziqiang Liu, Zixuan Li, Bojie Wang, Bosi Wen, Can Huang, Changpeng Cai, Chao Yu, Chen Li, Chengwei Hu, Chenhui Zhang, Dan Zhang, Daoyan Lin, Dayong Yang, Di Wang, Ding Ai, Erle Zhu, Fangzhou Yi, Feiyu Chen, Guohong Wen, Hailong Sun, Haisha Zhao, Haiyi Hu, Hanchen Zhang, Hanrui Liu, Hanyu Zhang, Hao Peng, Hao Tai, Haobo Zhang, He Liu, Hongwei Wang, Hongxi Yan, Hongyu Ge, Huan Liu, Huanpeng Chu, Jia'ni Zhao, Jiachen Wang, Jiajing Zhao, Jiamin Ren, Jiapeng Wang, Jiaxin Zhang, Jiayi Gui,

Jiayue Zhao, Jijie Li, Jing An, Jing Li, Jingwei Yuan, Jinhua Du, Jinxin Liu, Junkai Zhi, Junwen Duan, Kaiyue Zhou, Kangjian Wei, Ke Wang, Keyun Luo, Laiqiang Zhang, Leigang Sha, Liang Xu, Lindong Wu, Lintao Ding, Lu Chen, Minghao Li, Nianyi Lin, Pan Ta, Qiang Zou, Rongjun Song, Ruiqi Yang, Shangqing Tu, Shangtong Yang, Shaoxiang Wu, Shengyan Zhang, Shijie Li, Shuang Li, Shuyi Fan, Wei Qin, Wei Tian, Weining Zhang, Wenbo Yu, Wenjie Liang, Xiang Kuang, Xiangmeng Cheng, Xiangyang Li, Xiaoquan Yan, Xiaowei Hu, Xiaoying Ling, Xing Fan, Xingye Xia, Xinyuan Zhang, Xinze Zhang, Xirui Pan, Xu Zou, Xunkai Zhang, Yadi Liu, Yandong Wu, Yanfu Li, Yidong Wang, Yifan Zhu, Yijun Tan, Yilin Zhou, Yiming Pan, Ying Zhang, Yinpei Su, Yipeng Geng, Yong Yan, Yonglin Tan, Yuean Bi, Yuhan Shen, Yuhao Yang, Yujiang Li, Yunan Liu, Yunqing Wang, Yuntao Li, Yurong Wu, Yutao Zhang, Yuxi Duan, Yuxuan Zhang, Zezhen Liu, Zhengtao Jiang, Zhenhe Yan, Zheyu Zhang, Zhixiang Wei, Zhuo Chen, Zhuoer Feng, Zijun Yao, Ziwei Chai, Ziyuan Wang, Zuzhou Zhang, Bin Xu, Minlie Huang, Hongning Wang, Juanzi Li, Yuxiao Dong, and Jie Tang. Glm-5: from vibe coding to agentic engineering, 2026. URL https://arxiv.org/abs/2602.15763.

Harbor Framework Team. Harbor: A framework for evaluating and optimizing agents and models in container environments. https://github.com/harbor-framework/harbor, 2026. Version 0.20.0.

Hugo He. PPT Master. https://github.com/hugohe3/ppt-master, 2026.

Prannay Hebbar, Yogendra Manawat, Samuel Verboomen, Alesia Ivanova, Selvam Palanimalai, Kunal Bhatia, and Vignesh Baskaran. SIA: self improving AI with harness & weight updates. *CoRR*, abs/2605.27276, 2026. doi: 10.48550/ARXIV.2605.27276. URL https://doi.org/10.48550/arXiv.2605.27276.

Zoe Hitzig, Maxim Massenkoff, Eva Lyubich, Shaoyi Zhang, Ryan Heller, and Peter McCrory. Agentic coding and persistent returns to expertise. Anthropic Economic Research, 2026. https://www.anthropic.com/research/claude-code-expertise.

Shengran Hu, Cong Lu, and Jeff Clune. Automated design of agentic systems. In *The Thirteenth International Conference on Learning Representations, ICLR 2025, Singapore, April 24-28, 2025*. OpenReview.net, 2025. URL https://openreview.net/forum?id=t9U3LW7JVX.

Qian Huang, Jian Vora, Percy Liang, and Jure Leskovec. Mlagentbench: Evaluating language agents on machine learning experimentation. In Ruslan Salakhutdinov, Zico Kolter, Katherine A. Heller, Adrian Weller, Nuria Oliver, Jonathan Scarlett, and Felix Berkenkamp (eds.), *Forty-first International Conference on Machine Learning, ICML 2024, Vienna, Austria, July 21-27, 2024*, volume 235 of *Proceedings of Machine Learning Research*, pp. 20271–20309. PMLR / OpenReview.net, 2024. URL https://proceedings.mlr.press/v235/huang24y.html.

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 (ICLR)*, 2024.

Jiajie Jin, Yuyang Hu, Kai Qiu, Qi Dai, Chong Luo, Guanting Dong, Xiaoxi Li, Tong Zhao, Xiaolong Ma, Gongrui Zhang, Zhirong Wu, Bei Liu, Zhengyuan Yang, Linjie Li, Lijuan Wang, Hongjin Qian, Yutao Zhu, and Zhicheng Dou. Toward generalist autonomous research via hypothesis-tree refinement. *CoRR*, abs/2606.11926, 2026. doi: 10.48550/ARXIV.2606.11926. URL https://doi.org/10.48550/arXiv.2606.11926.

Omar Khattab, Arnav Singhvi, Paridhi Maheshwari, Zhiyuan Zhang, Keshav Santhanam, Sri Vardhamanan, Saiful Haq, Ashutosh Sharma, Thomas T. Joshi, Hanna Moazam, Heather Miller, Matei Zaharia, and Christopher Potts. Dspy: Compiling declarative language model calls into self-improving pipelines. *CoRR*, abs/2310.03714, 2023. doi: 10.48550/ARXIV.2310.03714. URL https://doi.org/10.48550/arXiv.2310.03714.

Kilo AI. PinchBench: A benchmarking system for evaluating LLM models as OpenClaw coding agents. https://github.com/pinchbench/skill, 2026. Commit 819384a.

Jakub Lála, Odhran O'Donoghue, Aleksandar Shtedritski, Sam Cox, Samuel G. Rodriques, and Andrew D. White. Paperqa: Retrieval-augmented generative agent for scientific research. *CoRR*, abs/2312.07559, 2023. doi: 10.48550/ARXIV.2312.07559. URL https://doi.org/10.48550/arXiv.2312.07559.

Yoonho Lee, Roshen Nair, Qizheng Zhang, Kangwook Lee, Omar Khattab, and Chelsea Finn. Meta-harness: End-to-end optimization of model harnesses. *CoRR*, abs/2603.28052, 2026. doi: 10.48550/ARXIV.2603.28052. URL https://doi.org/10.48550/arXiv.2603.28052.

Xiaozhe Li, Jixuan Chen, Xinyu Fang, Shengyuan Ding, Haodong Duan, Qingwen Liu, and Kai Chen. OPT-BENCH: evaluating the iterative self-optimization of LLM agents in large-scale search spaces. In Maria Liakata, Viviane P. Moreira, Jiajun Zhang, and David Jurgens (eds.), *Findings of the Association for Computational Linguistics, ACL 2026, San Diego, California, United States, July 2-7, 2026*, pp. 28437–28452. Association for Computational Linguistics, 2026. doi: 10.18653/V1/2026.FINDINGS-ACL.1417. URL https://doi.org/10.18653/v1/2026.findings-acl.1417.

Zongwei Li, Zhonghang Li, Zirui Guo, Xubin Ren, and Chao Huang. Deepcode: Open agentic coding. *CoRR*, abs/2512.07921, 2025. doi: 10.48550/ARXIV.2512.07921. URL https://doi.org/10.48550/arXiv.2512.07921.

Jiawei Liu, Chunqiu Steven Xia, Yuyao Wang, and Lingming Zhang. Is your code generated by chatgpt really correct? rigorous evaluation of large language models for code generation. In Alice Oh, Tristan Naumann, Amir Globerson, Kate Saenko, Moritz Hardt, and Sergey Levine (eds.), *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*, 2023. URL http://papers.nips.cc/paper_files/paper/2023/hash/43e9d647ccd3e4b7b5baab53f0368686-Abstract-Conference.html.

Xiao Liu, Hao Yu, Hanchen Zhang, Yifan Xu, Xuanyu Lei, Hanyu Lai, Yu Gu, et al. AgentBench: Evaluating LLMs as agents. In *International Conference on Learning Representations (ICLR)*, 2024.

Thanawat Lodkaew, Johannes Ackermann, Soichiro Nishimori, Nontawat Charoenphakdee, Masashi Sugiyama, and Takashi Ishida. Do coding agents deceive us? detecting and preventing cheating via capped evaluation with randomized tests. *arXiv preprint arXiv:2606.07379*, 2026.

Achyutha Menon, Magnus Saebo, Tyler Crosse, Spencer Gibson, Eyon Jang, and Diogo Cruz. Inherited goal drift: Contextual pressure can undermine agentic goals. *arXiv preprint arXiv:2603.03258*, 2026.

Mike A. Merrill, Alexander Glenn Shaw, Nicholas Carlini, Boxuan Li, Harsh Raj, Ivan Bercovich, Lin Shi, Jeong Yeon Shin, Thomas Walshe, Estefany Kelly Buchanan, Junhong Shen, Guanghao Ye, Haowei Lin, Jason Poulos, Maoyu Wang, Marianna Nezhurina, Jenia Jitsev, Di Lu, Orfeas Menis-Mastromichalakis, Zhiwei Xu, Zizhao Chen, Yue Liu, Robert Zhang, Leon Liangyu Chen, Anurag Kashyap, Jan-Lucas Uslu, Jeffrey Li, Jianbo Wu, Minghao Yan, Song Bian, Vedang Sharma, Ke Sun, Steven Dillmann, Akshay Anand, Andrew Lanpouthakoun, Bardia Koopah, Changran Hu, Etash Kumar Guha, Gabriel H. S. Dreiman, Jiacheng Zhu, Karl Krauth, Li Zhong, Niklas Muennighoff, Robert Amanfu, Shangyin Tan, Shreyas Pimpalgaonkar, Tushar Aggarwal, Xiangning Lin, Xin Lan, Xuandong Zhao, Yiqing Liang, Yuanli Wang, Zilong Wang, Changzhi Zhou, David Heineman, Hange Liu, Harsh Trivedi, John Yang, Junhong Lin, Manish Shetty, Michael Yang, Nabil Omi, Negin Raoof, Shanda Li, Terry Yue Zhuo, Wuwei Lin, Yiwei Dai, Yuxin Wang, Wenhao Chai, Shang Zhou, Dariush Wahdany, Ziyu She, Jiaming Hu, Zhikang Dong, Yuxuan Zhu, Sasha Cui, Ahson Saiyed, Arinbjörn Kolbeinsson, Jesse Hu, Christopher Michael Rytting, Ryan Marten, Yixin Wang, Alex Dimakis, Andy Konwinski, and Ludwig Schmidt. Terminal-bench: Benchmarking agents on hard, realistic tasks in command line interfaces. *CoRR*, abs/2601.11868, 2026. doi: 10.48550/ARXIV.2601.11868. URL https://doi.org/10.48550/arXiv.2601.11868.

Abhishek Mishra, Selvam Palanimalai, Yogendra Manawat, Samuel Verboomen, Prannay Hebbar, Damir Vrabac, Deepak Nathani, Sumeet Ramesh Motwani, Kunal Bhatia, and Vignesh Baskaran. AIE-bench: Benchmarking agents that build agents. In *ICML 2026 Workshop on Combining*

Theory and Benchmarks: Towards A Virtuous Cycle to Understand and Guarantee Foundation Model Performance, 2026. URL https://openreview.net/forum?id=f9yc09BuxG.

Vishal Ishwar Naik, Chenyu Xu, Donna Dong, Hussein Hassan, Abhishek Pradhan, Ofer Mendelevitch, Tallat Shafat, and Humayun Irshad. GuardianAgentBench: Where agents fail and how to guard them. *arXiv preprint arXiv:2607.20982*, 2026.

Deepak Nathani, Lovish Madaan, Nicholas Roberts, Nikolay Bashlykov, Ajay Menon, Vincent Moens, Amar Budhiraja, Despoina Magka, Vladislav Vorotilov, Gaurav Chaurasia, Dieuwke Hupkes, Ricardo Silveira Cabral, Tatiana Shavrina, Jakob N. Foerster, Yoram Bachrach, William Yang Wang, and Roberta Raileanu. Mlgym: A new framework and benchmark for advancing AI research agents. *CoRR*, abs/2502.14499, 2025. doi: 10.48550/ARXIV.2502.14499. URL https://doi.org/10.48550/arXiv.2502.14499.

Hailey Onweller, Elias Lumer, Austin Huber, Pia Ramchandani, Vamse Kumar Subbiah, and Corey Feld. Cited but not verified: Parsing and evaluating source attribution in LLM deep research agents. *arXiv preprint arXiv:2605.06635*, 2026.

OpenAI. Codex CLI. https://github.com/openai/codex, 2026a.

OpenAI. GPT-5.5 system card. Technical report, OpenAI, April 2026b. URL https://deploymentsafety.openai.com/gpt-5-5.

OpenAI. GPT-5.6 system card. Technical report, OpenAI, July 2026c. URL https://deploymentsafety.openai.com/gpt-5-6.

OpenClaw Contributors. OpenClaw: Your own personal AI assistant. https://github.com/openclaw/openclaw, 2026.

Rongxin Ouyang, Chang Chu, Zhikuang Xin, and Xiangyao Ma. Pdfmathtranslate: Scientific document translation preserving layouts. In Ivan Habernal, Peter Schulam, and Jörg Tiedemann (eds.), *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing, EMNLP 2025 - System Demonstrations, Suzhou, China, November 4-9, 2025*, pp. 918–924. Association for Computational Linguistics, 2025. doi: 10.18653/V1/2025.EMNLP-DEMOS.71. URL https://doi.org/10.18653/v1/2025.emnlp-demos.71.

Marco Perini, Lorenzo Padoan, and Marco Vinciguerra. Scrapegraph-ai, January 2024. URL https://github.com/ScrapeGraphAI/Scrapegraph-ai.

Shuofei Qiao, Runnan Fang, Zhisong Qiu, Xiaobin Wang, Ningyu Zhang, Yong Jiang, Pengjun Xie, Fei Huang, and Huajun Chen. Benchmarking agentic workflow generation. In *The Thirteenth International Conference on Learning Representations, ICLR 2025, Singapore, April 24-28, 2025*. OpenReview.net, 2025. URL https://openreview.net/forum?id=vunPXOFmoi.

Yujia Qin, Yining Ye, Junjie Fang, Haoming Wang, Shihao Liang, Shizuo Tian, Junda Zhang, Jiahao Li, Yunxin Li, Shijue Huang, Wanjun Zhong, Kuanye Li, Jiale Yang, Yu Miao, Woyu Lin, Longxiang Liu, Xu Jiang, Qianli Ma, Jingyu Li, Xiaojun Xiao, Kai Cai, Chuang Li, Yaowei Zheng, Chaolin Jin, Chen Li, Xiao Zhou, Minchao Wang, Haoli Chen, Zhaojian Li, Haihua Yang, Haifeng Liu, Feng Lin, Tao Peng, Xin Liu, and Guang Shi. UI-TARS: pioneering automated GUI interaction with native agents. *CoRR*, abs/2501.12326, 2025. doi: 10.48550/ARXIV.2501.12326. URL https://doi.org/10.48550/arXiv.2501.12326.

Qwen Team. Qwen3.6-35B-A3B: Agentic coding power, now open to all, April 2026. URL https://qwen.ai/blog?id=qwen3.6-35b-a3b.

Harsh Raj, Vipul Gupta, Anas Mahmoud, Razvan-Gabriel Dumitru, Darvin Yi, Aakash Sabharwal, and Yunzhong He. Model or harness? an interaction-centric taxonomy for localizing agent failures. *arXiv preprint arXiv:2607.28802*, 2026.

Delip Rao, Eric Wong, and Chris Callison-Burch. Detecting and correcting reference hallucinations in commercial LLMs and deep research agents. *arXiv preprint arXiv:2604.03173*, 2026.

Maxime Robeyns, Martin Szummer, and Laurence Aitchison. A self-improving coding agent. *CoRR*, abs/2504.15228, 2025. doi: 10.48550/ARXIV.2504.15228. URL https://doi.org/10.48550/arXiv.2504.15228.

J. Rosser and Jakob N. Foerster. Agentbreeder: Mitigating the AI safety risks of multi-agent scaffolds via self-improvement. In Danielle Belgrave, Cheng Zhang, Laura N. Montoya, Hsuan-Tien Lin, Razvan Pascanu, Piotr Koniusz, Marzyeh Ghassemi, Nancy Chen, Iván Vladimir Meza Ruíz, and Arturo Loaiza-Bonilla (eds.), *Advances in Neural Information Processing Systems 38: Annual Conference on Neural Information Processing Systems 2025, NeurIPS 2025, San Diego, CA, USA, December 2-7, 2025 / Mexico City, Mexico, November 30 - December 5, 2025*, 2025. URL http://papers.nips.cc/paper_files/paper/2025/hash/dc2ccde7ee43e5719e08c68e848bd65a-Abstract-Conference.html.

Minju Seo, Jinheon Baek, Seongyun Lee, and Sung Ju Hwang. Paper2Code: Automating code generation from scientific papers in machine learning. *arXiv preprint arXiv:2504.17192*, 2025.

Yu Shang, Yu Li, Keyu Zhao, Likai Ma, Jiahe Liu, Fengli Xu, and Yong Li. Agentsquare: Automatic LLM agent search in modular design space. In *The Thirteenth International Conference on Learning Representations, ICLR 2025, Singapore, April 24-28, 2025*. OpenReview.net, 2025. URL https://openreview.net/forum?id=mPdmDYIQ7f.

Michael D. Skarlinski, Sam Cox, Jon M. Laurent, James D. Braza, Michaela M. Hinks, Michael J. Hammerling, Manvitha Ponnapati, Samuel G. Rodriques, and Andrew D. White. Language agents achieve superhuman synthesis of scientific knowledge. *CoRR*, abs/2409.13740, 2024. doi: 10.48550/ARXIV.2409.13740. URL https://doi.org/10.48550/arXiv.2409.13740.

Peiyang Song, Kaiyu Yang, and Anima Anandkumar. Lean copilot: Large language models as copilots for theorem proving in lean. In George J. Pappas, Pradeep Ravikumar, and Sanjit A. Seshia (eds.), *International Conference on Neuro-symbolic Systems, 28-30 May 2025, University of Pennsylvania, Philadelphia, Pennsylvania, USA*, volume 288 of *Proceedings of Machine Learning Research*, pp. 144–169. PMLR, 2025. URL https://proceedings.mlr.press/v288/song25a.html.

Harsh Soni. ToolFailBench: Diagnosing tool-use failures in LLM agents. *arXiv preprint arXiv:2607.04686*, 2026.

Brace Sproul. Introducing OpenWiki, an open source agent for repo documentation. https://www.langchain.com/blog/introducing-openwiki-an-open-source-agent-for-repo-documentation, July 2026. LangChain Blog.

Giulio Starace, Oliver Jaffe, Dane Sherburn, James Aung, Jun Shern Chan, Leon Maksin, Rachel Dias, Evan Mays, Benjamin Kinsella, Wyatt Thompson, Johannes Heidecke, Amelia Glaese, and Tejal Patwardhan. PaperBench: Evaluating AI's ability to replicate AI research. In *International Conference on Machine Learning (ICML)*, 2025.

Shiqian Su, Sen Xing, Xuan Dong, Muyan Zhong, Bin Wang, Xizhou Zhu, Yuntao Chen, Wenhai Wang, Yue Deng, Pengxiang Zhu, Ziyuan Liu, Tiantong Li, Jiaheng Yu, Zhe Chen, Lidong Bing, and Jifeng Dai. Miroflow: Towards high-performance and robust open-source agent framework for general deep research tasks. *CoRR*, abs/2602.22808, 2026. doi: 10.48550/ARXIV.2602.22808. URL https://doi.org/10.48550/arXiv.2602.22808.

Jiabin Tang, Tianyu Fan, and Chao Huang. Autoagent: A fully-automated and zero-code framework for LLM agents. In Maria Liakata, Viviane P. Moreira, Jiajun Zhang, and David Jurgens (eds.), *Findings of the Association for Computational Linguistics, ACL 2026, San Diego, California, United States, July 2-7, 2026*, pp. 42924–42974. Association for Computational Linguistics, 2026. doi: 10.18653/V1/2026.FINDINGS-ACL.2129. URL https://doi.org/10.18653/v1/2026.findings-acl.2129.

Weixi Tong and Tianyi Zhang. Codejudge: Evaluating code generation with large language models. In Yaser Al-Onaizan, Mohit Bansal, and Yun-Nung Chen (eds.), *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing, EMNLP 2024, Miami, FL, USA,*

November 12-16, 2024, pp. 20032–20051. Association for Computational Linguistics, 2024. doi: 10.18653/V1/2024.EMNLP-MAIN.1118. URL https://doi.org/10.18653/v1/2024.emnlp-main.1118.

usestrix. Strix: Open-source AI penetration testing tool. https://github.com/usestrix/strix, 2026.

vanna-ai. Vanna: Accurate text-to-SQL generation via LLMs using agentic retrieval. https://github.com/vanna-ai/vanna, 2026.

Wenyi Wang, Piotr Piekos, Nanbo Li, 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. *CoRR*, abs/2510.21614, 2025a. doi: 10.48550/ARXIV.2510.21614. URL https://doi.org/10.48550/arXiv.2510.21614.

Xingyao Wang, Boxuan Li, Yufan Song, Frank F. Xu, Xiangru Tang, Mingchen Zhuge, Jiayi Pan, Yueqi Song, Bowen Li, Jaskirat Singh, et al. OpenHands: An open platform for AI software developers as generalist agents. In *International Conference on Learning Representations (ICLR)*, 2025b.

Jason Wei, Zhiqing Sun, Spencer Papay, Scott McKinney, Jeffrey Han, Isa Fulford, Hyung Won Chung, Alex Tachard Passos, William Fedus, and Amelia Glaese. Browsecomp: A simple yet challenging benchmark for browsing agents. *CoRR*, abs/2504.12516, 2025. doi: 10.48550/ARXIV.2504.12516. URL https://doi.org/10.48550/arXiv.2504.12516.

Hjalmar Wijk, Tao Roa Lin, Joel Becker, Sami Jawhar, Neev Parikh, Thomas Broadley, Lawrence Chan, Michael Chen, Joshua Clymer, Jai Dhyani, Elena Ericheva, Katharyn Garcia, Brian Goodrich, Nikola Jurkovic, Megan Kinniment, Aron Lajko, Seraphina Nix, Lucas Jun Koba Sato, William Saunders, Maksym Taran, Ben West, and Elizabeth Barnes. Re-bench: Evaluating frontier AI r&d capabilities of language model agents against human experts. In Aarti Singh, Maryam Fazel, Daniel Hsu, Simon Lacoste-Julien, Felix Berkenkamp, Tegan Maharaj, Kiri Wagstaff, and Jerry Zhu (eds.), *Forty-second International Conference on Machine Learning, ICML 2025, Vancouver, BC, Canada, July 13-19, 2025*, volume 267 of *Proceedings of Machine Learning Research*. PMLR / OpenReview.net, 2025. URL https://proceedings.mlr.press/v267/wijk25a.html.

Tianbao Xie, Danyang Zhang, Jixuan Chen, Xiaochuan Li, Siheng Zhao, Ruisheng Cao, Toh Jing Hua, Zhoujun Cheng, Dongchan Shin, Fangyu Lei, Yitao Liu, Yiheng Xu, Shuyan Zhou, Silvio Savarese, Caiming Xiong, Victor Zhong, and Tao Yu. Osworld: Benchmarking multimodal agents for open-ended tasks in real computer environments. In Amir Globersons, Lester Mackey, Danielle Belgrave, Angela Fan, Ulrich Paquet, Jakub M. Tomczak, and Cheng Zhang (eds.), *Advances in Neural Information Processing Systems 37: Annual Conference on Neural Information Processing Systems 2024, NeurIPS 2024, Vancouver, BC, Canada, December 10 - 15, 2024*, 2024. URL http://papers.nips.cc/paper_files/paper/2024/hash/5d413e48f84dc61244b6be550f1cd8f5-Abstract-Datasets_and_Benchmarks_Track.html.

Yutaro Yamada, Robert Tjarko Lange, Cong Lu, Shengran Hu, Chris Lu, Jakob N. Foerster, Jeff Clune, and David Ha. The AI scientist-v2: Workshop-level automated scientific discovery via agentic tree search. *CoRR*, abs/2504.08066, 2025. doi: 10.48550/ARXIV.2504.08066. URL https://doi.org/10.48550/arXiv.2504.08066.

Chengrun Yang, Xuezhi Wang, Yifeng Lu, Hanxiao Liu, Quoc V. Le, Denny Zhou, and Xinyun Chen. Large language models as optimizers, 2024a. URL https://arxiv.org/abs/2309.03409.

John Yang, Carlos E. Jimenez, Alexander Wettig, Kilian Lieret, Shunyu Yao, Karthik Narasimhan, and Ofir Press. SWE-agent: Agent-computer interfaces enable automated software engineering. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2024b.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik R. Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models. In *The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*. OpenReview.net, 2023. URL https://openreview.net/forum?id=WE_vluYUL-X.

Shunyu Yao, Noah Shinn, Pedram Razavi, and Karthik Narasimhan. τ-bench: A benchmark for tool-agent-user interaction in real-world domains. *CoRR*, abs/2406.12045, 2024. doi: 10.48550/ARXIV.2406.12045. URL https://doi.org/10.48550/arXiv.2406.12045.

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 (eds.), *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2025, Vienna, Austria, July 27 - August 1, 2025*, pp. 27890–27913. Association for Computational Linguistics, 2025. doi: 10.18653/V1/2025.ACL-LONG.1354. URL https://doi.org/10.18653/v1/2025.acl-long.1354.

Jiarui Yuan, Tailin Jin, Weize Chen, Zeyuan Liu, Zhiyuan Liu, and Maosong Sun. Se-bench: Benchmarking self-evolution with knowledge internalization. *CoRR*, abs/2602.04811, 2026. doi: 10.48550/ARXIV.2602.04811. URL https://doi.org/10.48550/arXiv.2602.04811.

Siyu Yuan, Kaitao Song, Jiangjie Chen, Xu Tan, Dongsheng Li, and Deqing Yang. Evoagent: Towards automatic multi-agent generation via evolutionary algorithms. In Luis Chiruzzo, Alan Ritter, and Lu Wang (eds.), *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL 2025 - Volume 1: Long Papers, Albuquerque, New Mexico, USA, April 29 - May 4, 2025*, pp. 6192–6217. Association for Computational Linguistics, 2025. doi: 10.18653/V1/2025.NAACL-LONG.315. URL https://doi.org/10.18653/v1/2025.naacl-long.315.

Eric Zelikman, Eliana Lorch, Lester Mackey, and Adam Tauman Kalai. Self-taught optimizer (STOP): recursively self-improving code generation. *CoRR*, abs/2310.02304, 2023. doi: 10.48550/ARXIV.2310.02304. URL https://doi.org/10.48550/arXiv.2310.02304.

Guibin Zhang, Luyang Niu, Junfeng Fang, Kun Wang, Lei Bai, and Xiang Wang. Multi-agent architecture search via agentic supernet. In Aarti Singh, Maryam Fazel, Daniel Hsu, Simon Lacoste-Julien, Felix Berkenkamp, Tegan Maharaj, Kiri Wagstaff, and Jerry Zhu (eds.), *Forty-second International Conference on Machine Learning, ICML 2025, Vancouver, BC, Canada, July 13-19, 2025*, volume 267 of *Proceedings of Machine Learning Research*. PMLR / OpenReview.net, 2025a. URL https://proceedings.mlr.press/v267/zhang25bi.html.

Guibin Zhang, Leo Lu, Fangzhou Xie, Kang Zhu, Junhao Wang, Zhifei Xie, Zhaochen Yu, Zihang Liu, Zhongxiang Sun, Qiankun Li, Yue Liao, Heng Chang, Xiaobin Hu, Qibing Ren, Wangchunshu Zhou, and Shuicheng Yan. Jit-agent: Scaling harness intelligence via just-in-time harness evolution. *CoRR*, abs/2608.25593, 2026a. doi: 10.48550/ARXIV.2608.25593. URL https://doi.org/10.48550/arXiv.2608.25593.

Jenny Zhang, Shengran Hu, Cong Lu, Robert T. Lange, and Jeff Clune. Darwin godel machine: Open-ended evolution of self-improving agents. *CoRR*, abs/2505.22954, 2025b. doi: 10.48550/ARXIV.2505.22954. URL https://doi.org/10.48550/arXiv.2505.22954.

Jenny Zhang, Bingchen Zhao, Wannan Yang, Jakob N. Foerster, Jeff Clune, Minqi Jiang, Sam Devlin, and Tatiana Shavrina. Hyperagents. *CoRR*, abs/2603.19461, 2026b. doi: 10.48550/ARXIV.2603.19461. URL https://doi.org/10.48550/arXiv.2603.19461.

Jiayi Zhang, Jinyu Xiang, Zhaoyang Yu, Fengwei Teng, Xionghui Chen, Jiaqi Chen, Mingchen Zhuge, Xin Cheng, Sirui Hong, Jinlin Wang, Bingnan Zheng, Bang Liu, Yuyu Luo, and Chenglin Wu. Aflow: Automating agentic workflow generation. In *The Thirteenth International Conference on Learning Representations, ICLR 2025, Singapore, April 24-28, 2025*. OpenReview.net, 2025c. URL https://openreview.net/forum?id=z5uVAKwmjf.

Bingchen Zhao, Dhruv Srikanth, Yuxiang Wu, and Zhengyao Jiang. SpecBench: Measuring reward hacking in long-horizon coding agents. *arXiv preprint arXiv:2605.21384*, 2026a.

Bingxi Zhao, Jiahao Zhang, Xubin Ren, Zirui Guo, Tianzhe Chu, Yi Ma, and Chao Huang. Deeptutor: Towards agentic personalized tutoring. *CoRR*, abs/2604.26962, 2026b. doi: 10.48550/ARXIV.2604.26962. URL https://doi.org/10.48550/arXiv.2604.26962.

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric P. Xing, Hao Zhang, Joseph E. Gonzalez, and Ion Stoica. Judging llm-as-a-judge with mt-bench and chatbot arena. In Alice Oh, Tristan Naumann, Amir Globerson, Kate Saenko, Moritz Hardt, and Sergey Levine (eds.), *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*, 2023. URL http://papers.nips.cc/paper_files/paper/2023/hash/91f18a1287b398d378ef22505bf41832-Abstract-Datasets_and_Benchmarks.html.

Han Zhou, Xingchen Wan, Ruoxi Sun, Hamid Palangi, Shariq Iqbal, Ivan Vulic, Anna Korhonen, and Sercan Ö. Arik. Multi-agent design: Optimizing agents with better prompts and topologies. *CoRR*, abs/2502.02533, 2025. doi: 10.48550/ARXIV.2502.02533. URL https://doi.org/10.48550/arXiv.2502.02533.

Mingchen Zhuge, Wenyi Wang, Louis Kirsch, Francesco Faccio, Dmitrii Khizbullin, and Jürgen Schmidhuber. Gptswarm: Language agents as optimizable graphs. In Ruslan Salakhutdinov, Zico Kolter, Katherine A. Heller, Adrian Weller, Nuria Oliver, Jonathan Scarlett, and Felix Berkenkamp (eds.), *Forty-first International Conference on Machine Learning, ICML 2024, Vienna, Austria, July 21-27, 2024*, volume 235 of *Proceedings of Machine Learning Research*, pp. 62743–62767. PMLR / OpenReview.net, 2024. URL https://proceedings.mlr.press/v235/zhuge24a.html.

Terry Yue Zhuo. Ice-score: Instructing large language models to evaluate code. In Yvette Graham and Matthew Purver (eds.), *Findings of the Association for Computational Linguistics: EACL 2024, St. Julian's, Malta, March 17-22, 2024*, volume EACL 2024 of *Findings of ACL*, pp. 2232–2242. Association for Computational Linguistics, 2024. doi: 10.18653/V1/2024.FINDINGS-EACL.148. URL https://doi.org/10.18653/v1/2024.findings-eacl.148.

A Related Work

Agents that build and optimize agents. Research on automated agent development has expanded both the scope of what can be designed and the stage at which adaptation occurs. Early approaches generate task-specific roles, tools, and collaboration patterns from task descriptions or seed agents (Chen et al., 2024; Yuan et al., 2025; Tang et al., 2026). Subsequent work formulates agent development as optimization, first over textual components such as instructions and demonstrations (Yang et al., 2024a; Khattab et al., 2023; Agrawal et al., 2025), and then over broader design spaces containing agent modules, communication structures, and executable workflows (Zhuge et al., 2024; Shang et al., 2025; Zhang et al., 2025c;a; Zhou et al., 2025). This paradigm has further expanded to searching over complete code-defined agents and multi-agent scaffolds (Hu et al., 2025; Rosser & Foerster, 2025).

A related line relaxes the assumption that the improvement procedure itself remains fixed. Initial systems support self-referential modification of an improver or agent logic (Zelikman et al., 2023; Yin et al., 2025), while later approaches expose complete implementations, harnesses, and meta-level improvement mechanisms to revision (Robeyns et al., 2025; Zhang et al., 2025b; Wang et al., 2025a; Zhang et al., 2026b; Lee et al., 2026; Hebbar et al., 2026). Arbor (Jin et al., 2026) carries hypotheses and evidence across long-horizon artifact optimization, whereas JIT-Agent (Zhang et al., 2026a) synthesizes task-conditioned executable harnesses at inference time. Together, these developments establish agent construction and optimization as a capability in its own right. AgentSWE makes this capability the object of evaluation, measuring whether general-purpose coding agents can create, modify, and optimize agents from heterogeneous user requirements.

Benchmarks for agent construction. Most agent benchmarks hold the agent implementation fixed and evaluate its performance within a given environment. Recent benchmarks move beyond task completion by asking agents to construct or iteratively improve more complex artifacts. WorFBench (Qiao et al., 2025) evaluates generated workflow structures, while benchmarks for machine-learning experimentation and research engineering evaluate improvement of executable research artifacts (Huang et al., 2024; Nathani et al., 2025; Wijk et al., 2025). AI4AI-Bench (Chi et al., 2026) extends this setting to the redesign of model-training algorithms. Other evaluations isolate specific improvement mechanisms: SE-Bench (Yuan et al., 2026) studies knowledge internalization, whereas OPT-BENCH (Li et al., 2026) measures feedback-driven solution refinement. These benchmarks evaluate workflows, research artifacts, or optimized solutions rather than deployable agents.

Most closely, AIE-Bench (Mishra et al., 2026) evaluates whether a meta-agent can improve a pinned target-agent scaffold under controlled mutation boundaries and trial budgets. AgentSWE extends this setting to the full agent-development lifecycle: creating agents from natural-language commissions, modifying existing agent codebases without regression, and optimizing working agents toward measurable objectives. By freezing submissions and evaluating their behavior on held-out cases, AgentSWE measures whether coding agents can deliver and maintain the agents users actually request.

B Task Sources and Statistics

Table 5 lists the 25 tasks with the system each one derives from and its numbers of development and held-out cases. Creation tasks reverse-engineer the listed domain agent into a requirement, Editing tasks extend the pinned codebase of the listed product, and Optimization tasks keep the native evaluator of the listed benchmark and start from the listed agent.

C Representative Tasks and Case Coverage

Each card below shows, for one representative task per family, what the builder receives: the section structure of the four requirement documents with excerpts, one public development case, and a summary of the scoring. Excerpts are abridged, with omissions marked [. . .]; the builder never sees the held-out cases. Table 6 contrasts the requirements that development and held-out cases exercise.

Table 5: AgentSWE tasks, their source systems, and their development and held-out cases.

Family	Task	Source system	Dev / held-out
Creation	Authorized vulnerability validation	Strix (usestrix, 2026)	2 / 6
	Database analytics	Vanna (vanna-ai, 2026)	2 / 6
	Desktop GUI automation	UI-TARS-desktop (Qin et al., 2025)	2 / 6
	Document-to-editable PPTX	PPT Master (He, 2026)	2 / 6
	Evidence-grounded document QA	PaperQA (Lála et al., 2023; Skarlinski et al., 2024)	2 / 6
	Formal theorem proving	LeanCopilot (Song et al., 2025)	2 / 6
	Repository bug repair	SWE-agent (Yang et al., 2024b)	2 / 6
	Schema-guided web extraction	ScrapeGraphAI (Perini et al., 2024)	2 / 6
	Scientific PDF translation	PDFMathTranslate (Ouyang et al., 2025)	2 / 6
	Web research report	MiroFlow (Su et al., 2026)	2 / 6
Editing	Claude policy provenance	Claude Code† (Anthropic, 2026)	2 / 6
	Aider worktree transaction	Aider (Gauthier, 2026)	2 / 6
	OpenHands effect recovery	OpenHands (Wang et al., 2025b)	2 / 6
	OpenClaw channel handoff	OpenClaw (OpenClaw Contributors, 2026)	2 / 6
	Codex execution residual	Codex CLI (OpenAI, 2026a)	2 / 6
	AI-Scientist reproducibility gate	AI-Scientist-v2 (Yamada et al., 2025)	2 / 6
	DeepCode claim traceability	DeepCode (Li et al., 2025)	2 / 6
	DeepTutor adaptive remediation	DeepTutor (Zhao et al., 2026b)	2 / 6
	Dyad acceptance-driven workflow	Dyad (dyad-sh, 2026)	2 / 6
	OpenWiki change-impact documentation	OpenWiki (Sproul, 2026)	2 / 6
Optimization	BrowseComp (Wei et al., 2025)	Search ReAct agent (Yao et al., 2023)	50 / 100
	τ^3 (Yao et al., 2024; Barrès et al., 2025)	Policy-aware ReAct agent (Yao et al., 2023)	20 / 40
	TerminalBench (Merrill et al., 2026)	Terminus-2 agent (Merrill et al., 2026)	10 / 40
	PinchBench (Kilo AI, 2026)	OpenClaw productivity agent	20 / 40
	OSWorld (Xie et al., 2024)	Screenshot ReAct policy (Yao et al., 2023)	10 / 25
Total	25 tasks		150 / 365

†The public Claude Code repository holds its documentation and plugins, not the CLI core; the task adds a plugin at the documented hook interface.

C.1 Creation: Scientific PDF Translation

The builder starts from an empty workspace and must deliver a reusable translator with an offline viewer that reads in both directions.

Requirement | four Markdown documents

`01_task_goal.md` Background · Agent to Create · Target Users · Core Value · Final Objective · Non-Goals

"You must create an **Interactive Scientific PDF Translation Agent**. It receives one Markdown request that identifies one or more local scientific PDFs, source and target languages, a translated-PDF display mode, and optional terminology assets. It produces a faithful translated PDF plus a self-contained local reading viewer that displays the source and translation side by side with geometry-based paragraph correspondence."

`02_interface_and_delivery.md` Environment Preparation · Uniform Run Command · Input Contract · Output Directory and Repeated Runs · Required Successful Output · PDF Contract · Alignment Artifact · Viewer Interaction Contract · `run_report.json` · Exit Codes and Failure Reports

"Every case is invoked exactly as `python run_agent.py -input <input.md> -output <output_dir>` [...] A successful run must contain all of these paths:" `translated.pdf`, `alignment.json`, `run_report.json`, `viewer/index.html`, `styles.css`, `app.js`,

`documents/source.pdf`, `documents/translated.pdf`, and `vendor/`. "The browser must make no external network request after the output directory has been generated."

`03_requirements_and_constraints.md` nine functional requirements: interpret the complete request · inspect every page and recover reading order · translate faithfully · preserve scientific objects and associations · produce a readable translated PDF · build geometry-grounded alignment · build the local viewer · validate before success · degrade honestly; implementation constraints; observable evaluation boundary

`04_resources.md` Evaluator-owned model transport · Environment Ownership · Runtime Envelope · Credentials · Multimodal Responses API · API and Retrieval Budgets · Serper-Compatible Search · Search/Scrape Tool Layer · Network and Filesystem

"Wall-clock limit: 600 seconds per case, including translation and validation. Memory limit: 4 GiB per case. Generation is non-interactive."

Development case `dev_002` | Thermal Barrier Imaging Note

Translate `source_mixed.pdf` from English to Simplified Chinese for materials-science researchers. Apply `glossary.csv` as authoritative and create the complete interactive reading bundle.
Source condition. The four-page PDF is mixed: pages 1, 2, and 4 contain native selectable text, while page 3 is an embedded scan with mild skew, faint type, a plot, and the printed batch mark `TB-17`. Page 4 is landscape. Recover page 3 by OCR or visual analysis without inventing illegible characters.
Translated PDF. Produce a **bilingual English-Simplified Chinese PDF** with exactly four pages and the same portrait/portrait/portrait/landscape sequence. Within each prose region, place the English source first and its Chinese translation directly below. [...] Follow every glossary target exactly. Preserve `TB-17`, coating code `YSZ-8`, instrument `IRX-4`, citation markers, signs, decimal precision, units, $\alpha = k/(\rho c_p)$, $\mathrm{Fo} = \alpha t/L^2$, and equation numbers (1) and (2).
Alignment and viewer. Align source paragraphs to the Chinese part of each bilingual target region, not to the duplicated English. Include OCR-derived page-3 paragraphs with honest confidence. Visibly distinguish low-confidence links in the viewer. Bidirectional activation must work on native page 2, scanned page 3, and rotated landscape page 4. The supplied assets are the complete authority. Make zero search or retrieval calls and add no scan disclaimer page.

Scoring

Result rubric with six dimensions: semantic completeness and terminology 35, fidelity of scientific objects and data 35, page and reading structure 12, alignment auditability 8, offline viewer 8, delivery integrity and accessibility 2; gates and ceilings as described in Section D. The frozen source is also graded once on the shared static rubric (Section H.1).

The difficulty is compositional: a correct translation still fails part of the commission if its alignment points to the duplicated English, or if the viewer cannot find the rendered region.

C.2 Editing: OpenHands Effect Recovery

The builder receives a pinned OpenHands product and must add durable recovery through its production services, delivering a patch with edit and run reports.

Requirement | four Markdown documents and a checkpoint schema

`01_task_goal.md` Task goal: cross-instance transactional effect delivery

"Extend the supplied OpenHands Agent Canvas repository with a backend-scoped, versioned recovery ledger for conversation effects. The product must stay honest when two service instances or browser tabs share storage, a process stops after any delivery stage, a lease expires and moves to a new runtime, or pause/resume/cancel races an old callback. The ledger is a transactional outbox, not merely a history snapshot."

`02_interface_and_delivery.md` Submission · Frozen TypeScript adapter · Recovery-aware conversation event dispatcher · Durable cross-tab recovery sync coordinator · Transactional workspace recovery reconciler · Persistence, integrity, and migration · Delivery and crash semantics · Lease, ABA, event, and compaction semantics · Least-privilege inspection and production seams · Error contract for deterministic crash injection

"Submit exactly these top-level files: `solution.patch`: a non-empty UTF-8 unified Git patch relative to the supplied repository that applies with `git apply` exactly once. `edit_report.json` [...] `run_report.json` [...] Reports are evidence, not assertions of correctness."

`03_requirements_and_constraints.md` functional requirements · constraints · verifiable hard requirements

"Implement the durable outbox state machine: `queued -> claimed -> applied -> completed`, with `uncertain`, `blocked`, and `cancelled` recovery outcomes. Persist every stage before continuing, and never infer callback success from a callback return digest alone." [...] "Modify only `src/api/conversation-service/**`, `src/api/runtime-service/**`, `src/api/recovery/**`, `src/stores/**` [...] Do not modify lockfiles, package/build configuration, benchmark cases, generated output, authentication, network policy, or unrelated product code."

`04_resources.md` Pinned runtime · Prewarm and install · Runtime and memory envelope · Filesystem

Development case `dev_001` | Converge an interrupted workspace and a lossy history stream

Resume this public practice conversation. A chunk write response and a transport pull response were both lost after the far side had already acted on them, the history channel echoes its own notices back, and the workspace has a local-only and a remote-only path to reconcile. Five outcomes are required, and they are scored independently.

1. **Upload exactly the missing chunk, once**, after re-probing what the remote side genuinely lacks.
2. **Converge exactly once** on the union of the local-only and remote-only paths: one durable remote commit, one local apply.
3. **Retry the lost transport response from the same durable cursor**, not from the advisory remote cursor.
4. **Complete, bounded projection; no duplicate effect.** No side effect runs twice, and the public workspace projection carries every published stage field and no path, digest, chunk identifier, tab or incarnation identity, or token. An empty projection is not a safe projection.
5. **Keep runtime-event discipline.** A delivery that carries the current owner, lease epoch, fencing token, run generation and expected revision is genuine: accept it [...] A delivery whose fencing token or lease epoch does not match the live authority must be suppressed [...] Suppressing everything is not a safe answer.

Report what actually happened. Claiming an outcome the product did not reach scores worse than reporting it honestly as unmet.

Scoring

A patch that does not apply scores zero. Otherwise the Result rubric scores task completion 35, recovery quality 25, durable consistency 20, honest reporting 10, and privacy and scope 10, from numbered outcomes with observed facts, satisfaction flags, and safety flags; unmet outcomes impose ceilings (Section D reconstructs one recorded score).

The commission pairs a positive obligation to resume legitimate work with a negative obligation to reject stale work; an agent that refuses every action fails both.

C.3 Optimization: PinchBench Productivity Agent

The builder receives a runnable OpenClaw productivity agent and improves it against the native PinchBench objective. The requirement is short enough to show almost in full.

Requirement | four Markdown documents and a task contract

`01_task_goal.md` Goal · Non-goals

"You are optimizing the runnable OpenClaw Basic Productivity Agent already installed in the submission workspace. [...] Build a reusable tool-using productivity agent that completes file, calendar, analysis, browser, research, and codebase tasks in a controlled workspace, including recovery and cross-step planning." Non-goals: hard-coding the development cases or hidden answers, self-reported rewards, and copying a complete third-party agent or evaluator.

Table 6: Requirement coverage in development and held-out cases, with examples. The held-out split changes inputs and interactions within the commission.

Requirement	Public development example	Held-out example
PDF content and geometry	Mixed native and scanned pages, bilingual Chinese output, a landscape page (dev 002).	English-to-Japanese translation of a two-column paper with a paragraph split across pages 2 and 3 (test 002); Traditional Chinese with dense figure labels and preferred glossary terms (test 006).
OpenHands recovery	Lost responses, convergence applied exactly once, discipline for runtime events (dev 001); fencing of the replaced owner and scoped access (dev 002).	Pause, resume, and cancel followed by stale callbacks and a terminal burst (test 003); durable recovery across three workspace crash boundaries (test 002).
PinchBench transfer	Stock trend analysis (dev 006) and Apache top error analysis (dev 007).	Stock volatility analysis (test 004), Apache critical errors and timelines (tests 013 and 014), under the same fixed runtime and native grader.

`02_interface_and_delivery.md` one executable harness that emits an OpenClaw agent specification per input row

```
python run_harness.py --input <input.jsonl> \
    --output <predictions.jsonl> --run-dir <run_dir>

{"id": "dev_001",
 "agent": {"kind": "openclaw",
           "instructions": "reusable system-level working policy",
           "planning_guidance": "optional task planning policy",
           "recovery_guidance": "optional failure recovery policy",
           "verification_guidance":
             "optional artifact verification policy",
           "tool_profile": "coding"}}
```

"The evaluator locks the OpenClaw/model/provider versions, task prompt, workspace fixtures, timeout, model-call/token budget, transcript collection, official grader and hybrid judge. It ignores self-reported success or rewards."

`03_requirements_and_constraints.md` functional requirements · implementation constraints

"The supplied `run_harness.py` and `productivity_policy.py` are the only editable paths." [...] "Never access hidden cases, evaluator gold, reference trajectories, or other submissions."

`04_resources.md` pinned runtime and budgets

"The evaluator runs the submitted specification with pinned OpenClaw `2026.7.1-2` [...] Budget per row: at most 20 model calls, 600 seconds, and 200,000 model tokens. [...] Automated and hybrid scoring use the pinned PinchBench `grade_task()` implementation."

Development case `dev_007` | `task_log_apache_top_errors`

Optimization Task `task_log_apache_top_errors`. This task is executed by the pinned evaluator-owned OpenClaw runtime. Optimize a reusable productivity-agent policy; the official task prompt, fixtures, workspace, transcript and `grade_task()` remain evaluator-owned.

Scoring

The pinned native grader scores each PinchBench task, automatically or with the benchmark's hybrid judge. The starter and the selected candidate run on the same 40 held-out tasks, and the task reports gap closure S (Equation (3)); the 20 development tasks provide feedback during the five accepted rounds.

The builder edits a reusable policy; the objective measures the work the runtime completes under it.

Table 7: Worked OpenHands Result score. Ceilings for unmet outcomes and ceilings for safety failures apply together; their minimum per dimension sums to 10, and the recorded score is 4.

Dimension	Max.	Partial cap	Safety cap	Effective cap	Score
Task completion	35	4	3	3	0
Recovery quality	25	3	2	2	0
Durable consistency	20	2	2	2	1
Honest reporting	10	2	3	2	2
Privacy and scope	10	1	2	1	1

D Result Rubrics and a Worked Score

Each Result rubric scores the observable outcomes of its task. We describe two of them and reconstruct one recorded score.

PDF translation: content dominates presentation. The six Result dimensions allocate 35 points to semantic completeness and terminology, 35 to fidelity of scientific objects and data, 12 to page and reading structure, 8 to alignment auditability, 8 to the offline viewer, and 2 to delivery integrity and accessibility, so 70 points concern the translated content. The evaluator checks the final bundle, the rendered pages and their geometry, and the viewer in a browser; source architecture earns no Result credit. A run that does not launch, exceeds a resource limit, or produces no parseable translated PDF scores zero. A valid but incomplete translation is still scored, under ceilings: omitting or leaving untranslated at least 25 % of the substantive units caps the total at 35, a material reversal of meaning caps it at 45, and repeated corruption of formulas or values caps it at 40. The lowest applicable ceiling bounds the sum of the dimension scores. A valid PDF without alignment and viewer artifacts is capped at 75, which separates missing functionality from execution failure.

OpenHands: outcomes, ceilings, and earned credit. The dimensions are task completion (35), recovery quality (25), durable consistency (20), honest reporting (10), and privacy and scope (10). Each case lists numbered outcomes with observed facts, satisfaction flags, and safety flags. Unmet outcomes and safety failures impose ceilings on each dimension, and all applicable ceilings apply together. A ceiling only bounds the score.

Table 7 reconstructs the recorded score of GPT-5.6 Sol's OpenHands delivery on `test_002`. None of the five outcomes is satisfied: no missing chunk is uploaded, none of the three crash boundaries is reached, there are no remote commits and no local applies, no restart demonstrates durability, and the genuine event is not accepted. Four outcomes are safety requirements. The agent reports that it is blocked and acknowledges the unmet outcomes, which keeps some honesty credit. The score is $0 + 0 + 1 + 2 + 1 = 4$, although checkpoint and lease setup succeed.

E Evaluation Infrastructure and Builder Instructions

Builder development and candidate evaluation run as separate Harbor tasks. The builder's writable workspace holds the delivery and its permitted dependencies. The public package exposes the requirement and the development cases; the hidden cases and the evaluator stay outside it. Candidate snapshots execute in an environment built for the task, with pinned fixtures, services, and resource limits, and route model requests through the evaluator. Optimization also keeps the source benchmark's native executor and grader. OpenHands runs record both the product's own observations and a separate outcome ledger, so an audit can compare what the agent reports with the state it changed.

Base instruction and prompt priors. The PDF builder instruction specifies one continuous development loop:

> "Your entire build/dev/repair loop occurs in this one native session and this one writable workspace; no fresh repair conversation will be created."
>
> "The returned feedback is the only authoritative dev feedback. Diagnose it, edit the same `/workspace/submission`, and submit again in this same conversation."

The priors in Table 3 change the requested development strategy. Strategy A's PDF instruction includes "Separate extraction, translation, and reconstruction" and asks for an intermediate document model that carries geometry. Strategy B asks the builder to "create a private synthetic suite" and, before each run, to "write expected observable invariants"; it asks for at least six distinct, structurally different synthetic cases, full artifact checks, and a compact regression suite, and it lets the builder test its agent through the evaluator's model gateway. The full prior texts are in Section E.1. The feedback the builders received appears in Section J; runtime settings are in Section F.

E.1 Prompt Priors

In the prompt-prior conditions of Table 3, one of the texts below is appended to the base builder instruction. They are reproduced verbatim; the private environment path is abbreviated and omissions are marked `[...]`.

Strategy A | Scientific PDF translation

```
## Strategy A prior: architecture for scientific PDF translation

Treat the created system as a geometry-preserving document translation pipeline, not
as page-level text replacement. Preserve freedom in libraries and models, but make
these architectural responsibilities explicit:

1. Analyze each page before translation. Render and parse it, identify native versus
   scanned content, recover reading order and layout regions, and classify prose,
   formulas, code, tables, figures, captions, footnotes, references, headers, and
   protected identifiers. Use OCR or visual recovery only where native extraction is
   inadequate, with confidence attached to recovered units.
2. Maintain an intermediate document model whose units carry source page, reading
   order, text, geometry, type, protection policy, and confidence. Translation should
   operate on coherent prose units while scientific objects and their associations
   remain explicitly protected.
3. Separate extraction, translation, and reconstruction. Batch or cache translation
   where useful, enforce glossary/protected-token constraints, and validate numbers,
   signs, units, citations, equations, identifiers, negation, and modality before
   placing target text back into the document.
4. Reconstruct adaptively within the original page geometry. Choose suitable fonts,
   script direction, line breaking, local reflow, and bilingual pairing. Preserve
   page count, dimensions, rotations, columns, object associations, and page-local
   reading order while avoiding clipping, overlap, and text painted across graphics.
5. Derive alignment from the same intermediate geometry used for reconstruction.
   Map meaningful source/target units with real page boxes, multiple anchors when a
   unit splits, stable reading-order IDs, digests, status, and calibrated confidence.
   Do not invent decorative boxes or align against a detached transcript.
6. Build the viewer as an offline projection of the PDF and alignment artifacts.
   Bundle the renderer locally; use the alignment geometry for overlays; support
   bidirectional pointer and keyboard activation, active/ARIA synchronization,
   counterpart scrolling, page identity, and visible uncertainty.
7. Validate in layers before success: parse and render every PDF page; compare page
   count/dimensions/rotation; check glyph coverage and protected-token survival;
   validate alignment schema, bounds, order, digests, and visible overlap; then run a
   clean-browser offline interaction smoke at both required viewports.
8. Recover honestly. Retry bounded OCR/translation/layout operations, isolate
   page-local failures, preserve uncertain visible tokens, lower confidence when
   warranted, and never silently omit a page or fabricate an alignment.

The reusable pattern is page analysis -> typed geometry model -> constrained
translation -> adaptive reconstruction -> geometry-derived alignment -> offline
viewer -> parse/render/browser validation and targeted repair.
```

Strategy B | Scientific PDF translation

```
## Strategy B prior: self-created development and self-verification

Do not optimize only against `dev_001` and `dev_002`. During this same Builder
session, create a private synthetic suite under `/workspace/self_cases`, run the
current created agent on it, and judge the complete translation bundles yourself.
Keep synthetic cases, outputs, renders, and review notes outside
`/workspace/submission`.

The Builder container provides the same evaluator-owned lower-agent transport used by
official Candidate jobs. [...] Invoke the current agent with the documented
private environment, for example:
```

```
    <private-env>/bin/python \
      /workspace/submission/run_agent.py \
      --input /workspace/self_cases/<case>/input.md \
      --output /workspace/self_results/<case>

Create structurally different PDFs rather than paraphrases of the public examples.
Across the session cover native and scan-only pages, multi-column and landscape
layouts, mixed rotations, monolingual and bilingual output, glossary conflicts,
protected equations/identifiers/units, tables/figures/captions/footnotes/references,
long text expansion, low-confidence OCR, split units, and degraded but recoverable
input. Generate small synthetic PDFs and glossaries locally so expected content and
geometry are known.

Before each run, write expected observable invariants. Then parse and render every
output page; compare count/dimensions/rotation and protected tokens; inspect target
glyphs, clipping, overlap, reading order, scientific associations, and glossary
consistency; validate alignment schema/digests/bounds/visible box overlap/confidence;
serve the viewer offline and exercise bidirectional click and keyboard activation at
both viewports. Record results and mechanism-level causes in
`/workspace/self_test_log.md`, then repair general extraction, translation,
reconstruction, alignment, viewer, or validation mechanisms.

Use official submissions as periodic external calibration inside the ten-round
budget: study feedback, update the synthetic coverage matrix, run relevant
self-created cases with the lower agent, self-evaluate, repair, and retain a compact
regression suite. Aim for at least six distinct synthetic cases spanning every major
function. High scores on the two public documents do not prove robustness to new
page structures or languages.
```

Strategy A | Document-to-editable PPTX

```
## Strategy A prior: architecture for document-to-editable-PPTX generation

Treat the created system as a staged document-understanding and presentation
compiler, not as a case-specific slide template. Preserve freedom in implementation,
but make these architectural responsibilities explicit:

1. Normalize every authorized source into a provenance-bearing content model. Parse
   text, headings, lists, tables, figures, captions, links, page/section locations,
   and source authority before slide authoring. Keep claims and quantitative values
   tied to source records rather than passing around untraceable prose.
2. Separate content strategy from visual execution. First infer audience, purpose,
   required slide count, narrative arc, slide roles, key claims, and evidence. Then
   create a slide-by-slide plan with one communication objective per slide. Only then
   materialize editable PowerPoint objects.
3. Establish a deck-wide design system before individual slides: canvas, margins,
   grid, typography hierarchy, palette, recurring components, image treatment, and
   density rules. Allow deliberate variation by slide role while preserving visual
   continuity; do not generate disconnected pages independently.
4. Use semantic slide objects. Titles, body text, shapes, tables, charts, diagrams,
   images, captions, citations, and speaker notes should remain natively editable and
   meaningfully grouped. Use native charts/tables when data semantics matter. Do not
   use full-slide screenshots or rasterized text as a shortcut.
5. Build an explicit evidence path for factual content. Every sourced claim and
   externally obtained asset should appear in a source manifest with stable IDs,
   locations, usage, and claim links. Preserve exact numbers, units, labels, and
   required tokens through planning and rendering.
6. Validate at three levels: package validity (OOXML relationships, slide count,
   notes, editable object structure), semantic validity (coverage, provenance,
   quantitative consistency), and visual validity (render every slide, inspect
   clipping/overlap/legibility, and compare against the plan). Rendering is a build
   gate, not an optional final glance.
7. Use a repair loop driven by validator and render evidence. Fix environment and
   dependency failures first, then missing content/provenance, then layout and visual
   defects. Re-render all slides after shared-theme changes and the affected slides
   after local changes.
8. Keep the pipeline input-driven. Content extraction, narrative planning, design
   choices, object generation, and validation must work for new document structures,
   topics, audiences, aspect ratios, and slide counts rather than matching the two
   public examples.

The essential pattern is source model -> communication plan -> design system ->
editable slide scene graph -> OOXML export -> package/semantic/render validation ->
targeted repair.
```

Table 8: Main evaluation settings.

	Creation	**Editing**	**Optimization**
Tasks	10	10	5
Dev / held-out	2 / 6 per task	2 / 6 per task	10–50 / 25–100
Builder effort	GPT models: `xhigh`; DeepSeek and Qwen: `max`	GPT models: `xhigh`; DeepSeek and Qwen: `max`	GPT-5.5: `xhigh`; GPT-5.6 Sol: `xhigh` on BrowseComp, `high` otherwise; DeepSeek and Qwen: `high`
Development budget	Up to 10 submissions, 8 h	5 accepted submissions, 5 h	5 accepted rounds, 16 h
Case budget	600 s	600 s	Native call, token, and time caps
Runtime model	gpt-5.6-sol / `medium`	deepseek-v4.1-flash / `high`	gpt-5.6-sol, fixed by contract
Result evaluator	gpt-5.6-sol / `xhigh`	deepseek-v4.1-flash / `max`	Native benchmark objective
Held-out artifact	Frozen delivery	Final accepted submission	Highest dev mean; earliest round breaks ties

Strategy B | Document-to-editable PPTX

```
## Strategy B prior: self-created development and self-verification

Do not optimize only against `dev_001` and `dev_002`. During this same Builder
session, create a private synthetic suite under `/workspace/self_cases`, run the
current created agent on it, and judge the resulting PPTX artifacts yourself. Keep
synthetic inputs, outputs, renders, and notes outside `/workspace/submission`.

The Builder container provides the same evaluator-owned lower-agent transport used by
official Candidate jobs. [...] Invoke the current agent with the documented
private environment, for example:

    <private-env>/bin/python \
      /workspace/submission/run_agent.py \
      --input /workspace/self_cases/<case>/input.md \
      --output /workspace/self_results/<case>

Create structurally different cases rather than paraphrases of the public examples.
Across the session cover long and short sources, mixed headings/lists/tables/images,
quantitative claims, conflicting source authority, different audiences and slide
counts, sparse and dense slides, a chart/table-heavy deck, citations/notes, and an
edge case with a missing optional asset. Use only synthetic or openly usable private
fixtures that you create for this purpose.

Before each run, write expected observable invariants. Afterward parse the OOXML,
verify slide count/relationships/editable object types/notes/source manifest, render
every slide, inspect montage and individual slides for clipping/overlap/legibility,
and check claim/source/number consistency. Record results and mechanism-level causes
in `/workspace/self_test_log.md`; repair extraction, planning, design-system,
generation, provenance, or validation mechanisms rather than hard-coding case text.

Use official submissions as periodic external calibration inside the ten-round
budget: study feedback, update a synthetic coverage matrix, run the lower agent on
relevant self-created cases, self-evaluate, repair, and retain a regression subset.
Aim for at least six distinct synthetic cases spanning all major functions. A deck
that merely exists is not a pass; local parse and full-slide render validation are
mandatory self-test gates.
```

F Experimental Details

F.1 Main Evaluation Protocol

Each builder receives a requirement, a workspace, and access to development evaluation. Held-out evaluation executes the frozen delivery in the environment owned by the task. Creation starts from an empty workspace, Editing from a pinned product codebase, and Optimization from a runnable starter. The main builder harness is Codex 0.144.1. Table 8 lists the settings for each task family.

Optimization uses 50/100 development/held-out cases for BrowseComp, 20/40 for τ^3, 10/40 for TerminalBench, 20/40 for PinchBench, and 10/25 for OSWorld. Each run evaluates both its starter and its selected candidate on the held-out split, and S uses the starter score measured in that run.

F.2 Supplementary Evaluation Protocols

Development cases and prompt priors. These experiments use Codex 0.144.1 with DeepSeek-V4.1-Flash (DeepSeek-AI, 2026b) (`max`) as builder, and DeepSeek-V4.1-Flash through the DeepSeek API as runtime model (`high`) and judge (`max`). Each condition has three independent builder runs. The experiment on development cases gives the builder 0, 1, 2, or 4 visible cases and scores every condition on the same four held-out cases of each task: cases 3–6 for PDF translation and cases 1, 2, 4, and 6 for PPTX. The experiment on prompt priors gives two visible cases and scores all six held-out cases; its no-prior condition reuses the three deliveries of the two-case condition, scored on all six held-out cases. The two experiments contain 36 distinct builder runs. Strategy A supplies a component design and an integration plan; strategy B asks for development cases written by the builder and testing of the whole pipeline through the evaluator's model gateway.

Extended budget. All ten Creation tasks are rebuilt with Codex and DeepSeek-V4.1-Flash (`max`). The runtime and judge are DeepSeek-V4.1-Flash at `max` effort through the provider API, each case receives 1,500 s, and the requirement asks for a runtime agent that emits an execution trace; an invalid trace caps Result at 40, and all 60 held-out traces are valid. Table 9 reports the last development score observed in each run.

Harnesses. Both harness studies use DSH (DeepSeek Harness) (DeepSeek-AI, 2026) 0.1.1-rc.2, Claude Code (Anthropic, 2026) 2.1.251, OpenCode (Anomaly, 2026) 1.18.25, and Pi (Earendil Works, 2026) 0.84.3, with effort `xhigh` for the GPT models, `high` for DeepSeek-V4-Flash, and `max` for DeepSeek-V4-Pro and GLM-5.3 (GLM-5-Team et al., 2026). The Creation study adds nine pairings of harness and model on two tasks (18 runs); Table 4 shows them with the Codex rows of the main comparison. The Optimization study adds four pairings on τ^3 and PinchBench (eight runs): Pi with GPT-5.6 Sol, DSH with DeepSeek-V4-Flash or DeepSeek-V4-Pro, and Claude Code with GLM-5.3. Three of these Optimization entries evaluate the starter and the candidate in separate sessions: DSH with DeepSeek-V4-Pro on PinchBench and with DeepSeek-V4-Flash on τ^3, and Pi with GPT-5.6 Sol on PinchBench.

G Repeated Runs

Two kinds of repetition appear in the paper: independent builder runs under one condition, and repeated evaluation of one artifact. In the Creation studies of Tables 2 and 3, each condition has three independently built deliveries. Figure 7 shows the score of each run and the mean of each condition.

Optimization evaluates the starter afresh in every builder run, which measures how much a native objective varies when the same agent is evaluated again. Across the independent measurements of each benchmark (four on BrowseComp, five elsewhere), the starter score has a standard deviation of 10.4 on BrowseComp, 4.5 on τ^3, 14.5 on TerminalBench, 20.8 on PinchBench, and 7.8 on OSWorld.

H Additional Results

H.1 Creation: Static Source Assessment

The static assessment of Section 4.3 applies only to Creation, where the builder writes the entire agent. It grades each frozen Creation source once, without executing it, as in model-based code evaluation (Zhuo, 2024; Tong & Zhang, 2024; Seo et al., 2025), with the judge configuration used for Result (Table 8). Its shared rubric assigns 20 points to how well the mechanisms cover the requirements; 15 each to interface and lifecycle, to analysis and evidence integrity, and to safety, privacy, and side effects; 10 each to recovery and honest failure, to testability and observability, and to maintainability and generalization; and 5 to resource discipline. Figure 8 shows the total next to held-out Result. The mean static score across the 50 runs is 41.8.

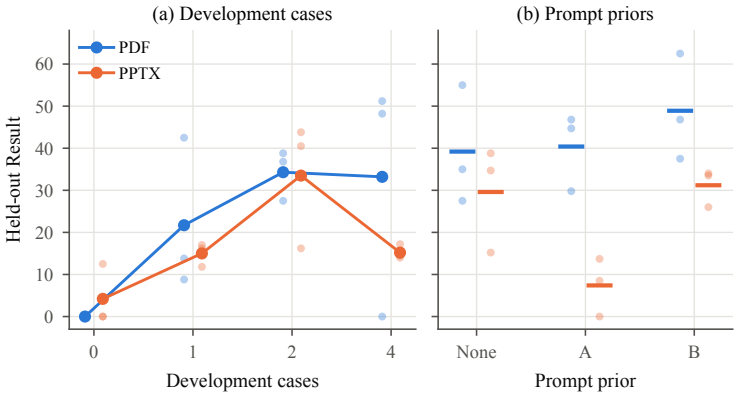

Figure 7: Independent Creation runs behind Tables 2 and 3, three per condition: held-out Result of each run (dots) and the condition mean (line or bar) for (a) the number of development cases and (b) the prompt priors. The two-case and no-prior conditions use the same deliveries, scored on four and on six held-out cases.

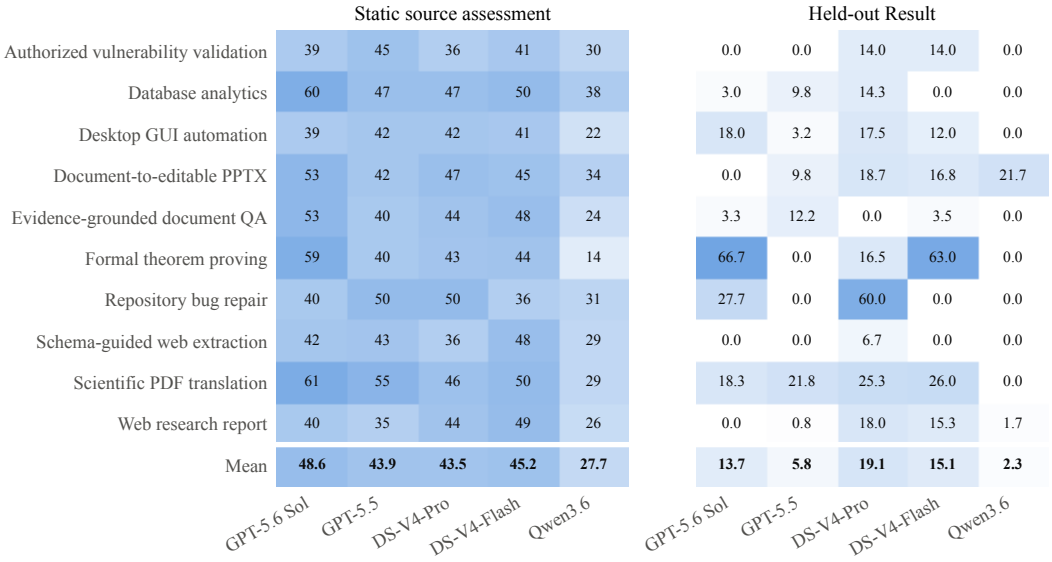

Figure 8: Static assessment of Creation deliveries (left): score of the frozen source on the shared rubric of eight dimensions (0–100), graded once per delivery without execution. Held-out Result from Table 1 (right), on the same color scale.

H.2 Creation: Extended Budget

Table 9 reports each task in the extended-budget setting (Section F.2). The last development score exceeds held-out Result on all ten tasks, by more than 30 points on seven. Formal theorem proving reaches 100 on development cases and 16.7 on held-out cases, where one of six cases succeeds; repository repair likewise succeeds on one held-out case.

Of the 60 extended-budget traces, 20 deliver no valid result: six reject the input before any model call, five repository cases end in repair loops, five formal proving cases end without a completed proof, three web research cases never write the requested report, and one schema extraction case exits normally with an invalid delivery. None reaches the 1,500 s limit. The web research failures run for 1,302 to 1,491 s; the repository and formal failures stop earlier, and the five failed formal traces contain a median of 23 retry steps. These agents stop without converging, well before the limit.

Table 9: Creation under the extended-budget setting, with DeepSeek-V4.1-Flash as builder (in Codex), runtime model, and judge. Held-out is the held-out Result; Last dev is the last development score observed in the run.

Task	Held-out	Last dev
Authorized vulnerability validation	0.0	40.0
Database analytics	70.0	94.0
Desktop GUI automation	28.5	63.0
Document-to-editable PPTX	10.2	46.5
Evidence-grounded document QA	81.2	98.8
Formal theorem proving	16.7	100.0
Repository bug repair	15.8	50.0
Schema-guided web extraction	11.7	79.5
Scientific PDF translation	61.0	96.0
Web research report	20.2	35.5
Mean	31.5	70.3

Table 10: Optimization held-out results. J_0 and J_1 are starter and candidate scores. Round is the selected round.

ID	Builder	Benchmark	J_0	J_1	Δ	S	W/L/T	Round
M01	GPT-5.6 Sol	BrowseC.	4.000	35.000	+31.00	32.3	32/1/67	3
M02	GPT-5.6 Sol	τ^3	42.500	45.000	+2.50	4.3	3/2/35	5
M03	GPT-5.6 Sol	TerminalB.	25.000	17.500	−7.50	−10.0	4/7/29	5
M04	GPT-5.6 Sol	PinchB.	27.625	16.825	−10.80	−14.9	4/11/25	1
M05	GPT-5.6 Sol	OSWorld	0.000	12.000	+12.00	12.0	3/0/22	1
M06	DS-V4-Flash	BrowseC.	18.000	46.000	+28.00	34.1	30/2/68	2
M07	DS-V4-Flash	τ^3	50.000	25.000	−25.00	−50.0	1/11/28	2
M08	DS-V4-Flash	TerminalB.	10.000	12.500	+2.50	2.8	2/1/37	5
M09	DS-V4-Flash	PinchB.	55.825	40.625	−15.20	−34.4	7/17/16	2
M10	DS-V4-Flash	OSWorld	0.000	12.000	+12.00	12.0	3/0/22	1
M11	DS-V4-Pro	BrowseC.	4.000	3.000	−1.00	−1.0	2/3/95	4
M12	DS-V4-Pro	τ^3	52.500	60.000	+7.50	15.8	5/2/33	1
M13	DS-V4-Pro	TerminalB.	2.500	2.500	+0.00	0.0	0/0/40	2
M14	DS-V4-Pro	PinchB.	60.975	45.875	−15.10	−38.7	6/20/14	5
M15	DS-V4-Pro	OSWorld	0.000	4.000	+4.00	4.0	1/0/24	1
M16	GPT-5.5	BrowseC.	13.000	29.000	+16.00	18.4	19/3/78	2
M17	GPT-5.5	τ^3	42.500	42.500	+0.00	0.0	5/5/30	4
M18	GPT-5.5	TerminalB.	2.500	10.000	+7.50	7.7	3/0/37	1
M19	GPT-5.5	PinchB.	67.675	64.075	−3.60	−11.1	12/13/15	1
M20	GPT-5.5	OSWorld	12.000	4.000	−8.00	−9.1	0/2/23	3
M21	Qwen3.6	BrowseC.	29.000	35.000	+6.00	8.5	13/7/80	5
M22	Qwen3.6	τ^3	47.500	35.000	−12.50	−23.8	3/8/29	2
M23	Qwen3.6	TerminalB.	35.000	40.000	+5.00	7.7	4/2/34	4
M24	Qwen3.6	PinchB.	21.275	30.825	+9.55	12.1	12/7/21	1
M25	Qwen3.6	OSWorld	16.000	16.000	+0.00	0.0	1/1/23	1

H.3 Optimization: Per-Run Results

Table 10 lists the native held-out scores behind the main Optimization block. W/L/T counts the cases on which the candidate scores higher than, lower than, or equal to the starter; over all 1,225 paired instances the counts are 175/125/925. A two-sided sign test on the W and L counts of each run rejects equal performance at the nominal 5 % level, without multiple-testing correction, in five runs: the BrowseComp gains of GPT-5.6 Sol, GPT-5.5, and DeepSeek-V4-Flash, and the losses of DeepSeek-V4-Flash on τ^3 and DeepSeek-V4-Pro on PinchBench.

H.4 Optimization: Development and Candidate Selection

Figure 9 separates the development gain of the selected candidate from its held-out gain Δ. The 25 runs average 13.1 and 1.8 points. Development gain exceeds held-out gain in 17 runs, and nine runs combine a development gain with a held-out loss. The behavioral analysis finds candidates specialized to the visible task categories in 14 runs; the four runs in which this is the primary cause account for 56 of the 125 held-out instances on which a candidate scores below its starter.

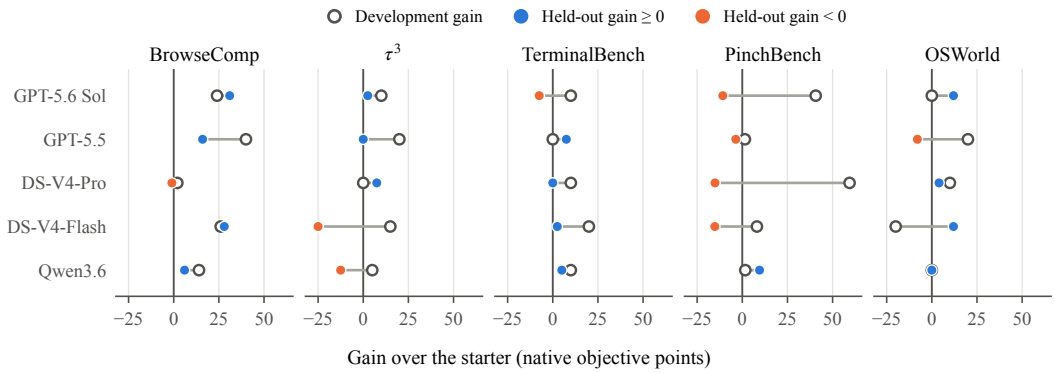

Figure 9: Optimization development and held-out gains per run, in native objective points. Development gain (open) is the selected candidate's development score minus the starter's; held-out gain (filled) is Δ from Table 10.

In nine runs no later round beats the first, and in 19 the selected round leads the runner-up by at most one solved development case, including eight ties broken toward the earliest round. Selecting the best of up to five rounds on development splits of 10 to 50 cases therefore inflates the development gain of the selected candidate.

I Behavioral Audit

I.1 Coverage and Counting Rules

The audit reads three kinds of evidence: the builder trajectory with the feedback it received, the frozen source or candidate diff, and the delivered agent's held-out execution trace. A run is one builder on one task, as in Section 4.1; a case is one held-out input. A mode is present in a run if it appears in at least one case or in the builder trajectory, and modes can co-occur. Labels proposed with AI assistance were reviewed by the authors against these records. Where a report names a primary cause, it is the failure that cost the run the most points.

Creation covers 270 held-out executions and the final reports of the 42 builder runs that write one; the ten extended-budget runs are analyzed separately. Optimization covers the 25 development logs, the frozen candidates, and the builder trajectories. Editing covers the 33 runs that deliver a candidate (Section I.3).

I.2 Creation: Validity Gates

Figure 10 gives the immediate outcome and execution time of each held-out execution. An error exit (nonzero exit code) can follow an implementation error, a failed self check, a rejected model response, or an external tool failure; a missing artifact follows from an earlier failure and is counted once.

Nearly half of the gate failures crash within one second, most of them from Qwen3.6 and the rest from the two GPT builders; neither DeepSeek builder crashes this way. Typical causes are input parsing and attempts to install dependencies into the submission directory, which the requirements forbid and the evaluator mounts read only. The other gate failures come after substantial execution, most often when the agent's own check rejects its output and the agent reports the failure.

Executions that pass the gate can still miss the requirement. Of the 113 graded executions, 33 trigger at least one rubric ceiling, and in 15 the ceiling sets the final score. Document QA agents, for example, violate the output contract with repeated DOM target identifiers or unsupported CSS references. An agent that launches can still fall short of what it was asked to do.

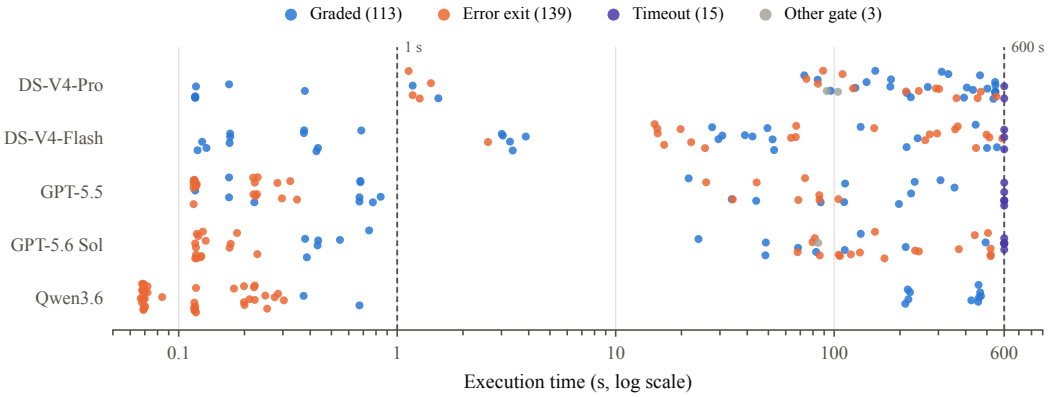

Figure 10: Execution time of each held-out Creation execution by builder, colored by its immediate outcome, with the categories of Figure 5: graded 113 (41.9 %), error exit 139 (51.5 %), 600 s timeout 15 (5.6 %), and other gates 3 (1.1 %; two missing required protected content, one other deterministic gate). Categories are mutually exclusive; gate failures total 157 (58.1 %).

Table 11: Editing modes in the runs that deliver a candidate. Rows can overlap; the third row is the union of two submodes.

Observed mode	**DS-V4-Pro (10)**	**GPT (19)**	**DS-V4-Flash (4)**	**Total (33)**
Fitting the visible cases	4	7	2	13
Gate failure or rejection of legal input	3	9	2	14
Capability absent, uninvoked, or incomplete execution	6	18	4	28
Runtime completion claim contradicted by records	6	5	2	13
Mutation that should have been refused	3	7	4	14
Budget spent without convergence	3	9	1	13

I.3 Editing: Runs with a Delivery

The Editing analysis covers the 33 runs that deliver a candidate: DeepSeek-V4-Pro and GPT-5.5 on all ten tasks, GPT-5.6 Sol on nine, and DeepSeek-V4-Flash on four, with 198 held-out executions. The other runs deliver nothing to execute (Section 4.2). Table 11 reports how many runs show each mode.

The third row covers six DeepSeek-V4-Pro runs, 18 of the 19 GPT runs, and all four DeepSeek-V4-Flash runs. Such capabilities can fail without any visible error: on one held-out case, an OpenClaw delivery issues 53 calls through its new handoff client and the product refuses every one, yet the execution passes the validity gates.

Development feedback names defects that survive freezing. Eight DeepSeek-V4-Pro runs receive feedback naming a violated cap identifier, and each of the eight still violates the same identifier on held-out cases. Nine of the ten DeepSeek-V4-Pro runs use all five submissions.

Preservation failures differ from missing behavior. In 14 runs the delivered product permits a mutation it should refuse, such as repeating an operation that must run once or overwriting state it must preserve. These violations concern the regression obligation in Equation (2).

I.4 Completion Claims

For Creation and Optimization the unit is the builder's final report, including any completion summary written into the frozen artifact. We compare it with the actions the trajectory records, the files and capabilities the delivery contains, and the development feedback the builder received, and count a

Table 12: Audit of builders' final reports.

	Creation	Optimization
All runs	50	25
Runs with a final report	42	19
Reports asserting completion	23	17
Reports contradicted by the record	6	3
Reports acknowledging partial completion or failure	19	2
Contradictions among the latter reports	0	0

report as unsupported only when those records contradict it. A held-out failure by itself does not count against a report about development testing.

All nine unsupported reports assert completion. None of the 21 reports that admit partial completion or failure is contradicted by the record. Four of the six unsupported Creation reports assert that code works or was verified, while the trajectory shows that the code, or the path the report vouches for, never ran in its final form. GPT-5.6 Sol, for example, reports that its final repository-repair entrypoint emits the required reports for both development cases, yet the frozen file differs from every version the builder ran. The other two contradict feedback or audits the builder had already seen: Qwen3.6 calls its web-research agent complete and validated, with all citations resolving, one round after the judge scored both development cases 5 and reported no matching citations. In Optimization, DeepSeek-V4-Flash reports that two PinchBench tasks stayed at zero in every round, while its own development record shows one of them at 36 and 22, and it omits the four regressions it had recorded for the delivered round.

In Editing the speaker is the delivered agent, and its runtime claim is checked against effect logs, state, and output artifacts on each held-out case; a run counts if any case contains a contradiction, and 13 of the 33 runs that deliver do. On OpenHands test 006, for example, the agent describes a workspace commit as applied exactly once, while the world records zero local applies and the local manifest still lacks the file that exists only remotely. A completed entry in the effect log does not show that the workspace converged.

J Execution Case Studies

Each case traces one failure from feedback or runtime actions to the frozen artifact or the observed state. Quoted text is verbatim with omissions marked; structured records are condensed to the relevant fields.

1. A development score built on two hard-coded cases. Qwen3.6's Database analytics delivery is a single script of 2,073 lines that describes itself as "pure Python stdlib, no external dependencies" and never calls the runtime model. It classifies each request by phrases taken from the two development cases and dispatches to queries written for them:

> **Case type:** `"commerce"` if the request contains "recognized net sales", "decline by market", [...]; `"adoption"` if it contains "adoption", "active_organization", [...]; otherwise none.
>
> **Queries:** `_build_commerce_q1_sql` [...] `_build_adoption_q3_sql`, with the quarter bounds and the refund cutoff written in as constants.

Its development means climb from 38 to 99 over ten rounds and end at 96. The final feedback already points at the hard-coding:

> **Round 10, dev 001 (92/100):** "`dashboard.html` switches between literal Harbor/Mesa panels instead of deriving the selected rows, total, and largest negative category from submitted result data" [...]

The held-out requests ask for other analyses over other databases. Five of them exit after 0.01 s with "Unknown case type: None", the sixth with "no such table: organizations", and all six fail the validity gate.

2. A higher development mean hides displaced failures. DeepSeek-V4-Flash's PinchBench run M09 adds playbooks for task categories and routes tasks to them by keyword. Its own assessment of round 2 records the tradeoff:

> "Round 2: **76.5** (up from 72.8)" [...] "But regressions: meeting tldr (97→0), spreadsheet (99→5), contract (100→62), syslog (36→22)."

Later rounds fall to 60.6, 63.7, and 51.65, so round 2 is frozen. Its held-out mean is 40.625 against the starter's 55.825, with 7 wins, 17 losses, and 16 ties. The higher development mean already contains large regressions on other visible tasks, and selection still favors it.

3. Cancellation succeeds, and late work still changes state. GPT-5.6 Sol's OpenHands delivery runs pause, resume, and cancel on `test_003`. The product observations and the burst probe record:

> **Lifecycle:** pause → generation 2; resume → generation 3; cancel → generation 4, cancellation revision 5.
> **Burst after cancellation:** 80 callbacks accepted, 0 rejected; audit length $1 \rightarrow 81$; revision $5 \rightarrow 85$; event cursor $0 \rightarrow 3320$; terminal state remains cancelled.

The commission requires those late callbacks to change nothing. The terminal state is correct, and the suppression fails: the audit log and the revision counter grow with every accepted callback. The agent's report acknowledges the burst.

4. A final report describes a discarded revision. The GPT-5.5 OSWorld run M20 selects round 3 (dev 40) after later rounds score 20 and 30. Its final report nevertheless describes harness changes:

> "`run_harness.py` provides BOM-tolerant input, deterministic JSONL output, atomic output/report writes, per-row agent mappings, and schema/length validation."

The round 4 harness contains the BOM handling and the atomic writes. The frozen round 3 `run_harness.py` is byte-identical to the starter's and contains neither; the round changes only the desktop policy. The report describes development work that the delivery does not contain.

A repair that reaches the delivered artifact. For contrast, DeepSeek-V4-Flash's BrowseComp run M06 answers a zero in the first round by finding a concrete protocol mismatch:

> "All 50 rows scored 0 with valid contracts" [...] "missing `Authorization` on model/search calls (the starter sent it on **all** calls)".

The next candidate restores the header, broadens response parsing, and revises the research policy. Its development score reaches 34 and it stays selected through three further rounds. The frozen source contains the restored header, and the trajectory records a local check of the whole pipeline against a mock broker. Held-out performance rises from 18 to 46 ($S = 34.1$). This repair fixes a fault the feedback exposed and reaches the delivered agent.

K LIMITATIONS

Repetition. The main comparison and the harness studies build each pairing of builder and task once. Native Optimization objectives vary between repeated evaluations of the same agent (Section G), so individual Optimization results carry that variation, and the reported means and counts summarize single builds rather than establish an ordering of builders. The Creation studies of development cases and prompt priors use three independent builder runs per condition, and we report their differences descriptively.

Model-based judging. Creation and Editing Result scores are assigned by a model judge (Zheng et al., 2023) that applies the task rubric to the delivered artifacts and execution traces. Validity gates and rubric ceilings bound the judge's discretion, but the scores still inherit its errors. The static source assessment of Section 4.3 uses the same judge. The task materials were drafted with GPT-5.6 Sol (Section 3.1), and in both judged families the judge shares a model family with at least one builder: GPT-5.6 Sol in Creation and DeepSeek-V4.1-Flash, beside DeepSeek-V4-Pro and DeepSeek-V4-Flash, in Editing.

Evaluation settings. Each task family fixes one runtime model and one judge (Table 8), so absolute levels are comparable within a family but not across families. The development-case, prompt-prior, and extended-budget studies use DeepSeek-V4.1-Flash as builder, runtime model, and judge (Section F.2), so their levels are not comparable to Table 1, and the analysis uses only comparisons within each study. The extended-budget setting changes the builder model, the case budget, the runtime model, the judge, and the requirement together, so we report it on its own rather than against Table 1. In the harness study, three Optimization entries evaluate the starter and the candidate in separate sessions (Section F.2).

Coverage of the failure analysis. Trace-level analysis of Creation covers the 270 of 300 held-out executions for which complete execution traces were recorded; the five desktop GUI runs have no per-case traces.