arXiv:2609.04298v3 [cs.AI] 10 Sep 2026

Harbor Adapters and Harbor-Index: Infrastructure and a Curated Meta-Dataset for Large-Scale Agentic Evaluation

Organization & Execution Team

Lin Shi♠, Haowei Lin♠, Zixuan Zhu♢, Xiaoyue Zhou♢, Xiang Li♢, Xiangning Lin♢, Yaxuan Deng♢, Han Xu♢, Yuangang Li♢, Shanda Li♢, Zizhao Chen♢, Hanwen Xing♢, Harsh Raj, Bo Chen, Quan Shi, Steven Dillmann, Yipeng Gao, Puneesh Khanna, Ruofan Lu, Chao Beyond Zhou, Michael Yang, Robert Zhang, Siyuan Chai, Jiayu Chang, Yizhao Chen, Xiaokun Chen, Yiwei Dai, Wenting Yang, Hange Liu, Minghao Liu, Zihan Wang

Adapter Contributors

Adnan El Assadi, Benedikt Stroebl, E. Kelly Buchanan, Han Meng, Junwei He, Longxuan Yu, Radin Shayanfar, Yukyung Lee, Zhikang Dong, Allen G Hart, Anjiang Wei, Anurag Kashyap, Arpandeep Khatua, Audrey Jixin Zheng, Chengrui Ma, David Heineman, Dubing Chen, Hai-Anh Trinh, Haishuo Fang, Hefan Zhang, Hui Shen, Issa Sugiura, Jiankai Sun, Jiechao Gao, Junhong Lin, Junnan Li, Kai Yang, Lei Hsiung, Maoyu Wang, Mengze Tang, Nabil Omi, Negin Raoof, Nicholas Edwards, Octavia Guo, Orfeas Menis Mastromichalakis, Pengliang Ji, Przemysław Hejman, Qi Qi, Qunshu Lin, Richard Zhuang, Rui Yang, Ruichen Zheng, Ryan Marten, Shaghayegh Fazliani, Shizheng Hou, Sicong Jiang, Sijie Li, Boqin Yuan, Michael Glass, Song Bian, Terry Yue Zhuo, Tianqing Wu, Tom Tang, Wanjia Zhao, Weihao Xuan, Wenhua Liang, Xian Liu, Xin Lan, Xuan Zhang, Xuandong Zhao, Yanchuan Tang, Yifan Jiang, Yijiang Li, Yitong Guan, Yizhi Li, Yonghui Liu, Yuheng Tang, Yujun (Audrey) Mao, Yunfei Zhao, Yuxin Wang, Yuxuan Tang, Zhenheng Tang, Zhifei Li, Ziruo Wang, Ziyu She, Kaiyuan Liu, Iheb Chaabane, Yuxin Tang, Xiangyi Li, Satya Sai Srinath Namburi GNVV, Xinyue Zheng

Advisory Committee

Andy Konwinski, Boxuan Li, Leon Liangyu Chen, Alex Dimakis, Nicholas Carlini, Soroush Vosoughi, Sanmi Koyejo, Di He, Etash Guha, Benjamin Feuer, Mike Merrill♠, Ludwig Schmidt♠, Alex Shaw♠

♠ Project Leads · ♢ Core Contributors · Full affiliations in Appendix A.
Correspondence: ls2282@cornell.edu; linhaowei@pku.edu.cn.

Harbor framework and adapters: https://github.com/harbor-framework/harbor
Harbor-Index: https://github.com/harbor-framework/harbor-index

Abstract

Evaluating agents on the growing number of agentic benchmarks is challenging because they often require complex environments and agent integrations. We introduce **Harbor Adapters**, a unified evaluation infrastructure for agentic benchmarks. Our work makes three contributions. First, we develop benchmark adapters that port more than 80 benchmarks to evaluate arbitrary agents, and validate them through rigorous code review and parity experiments. Second, we conduct a large-scale evaluation of 8 models spanning capability tiers across 54 benchmarks; every model is run with Terminus-2 and with one of 3 native harnesses. This enables a broader analysis of agent capabilities and failure modes than was previously possible. Third, we introduce **Harbor-Index**, a curated set of 82 difficult, diverse, and high-quality tasks spanning 29 benchmarks, refined from the adapted suite through difficulty filtering, AI and human audit, and an audit-and-fix loop. Harbor-Index preserves the challenge and breadth of large-scale agentic evaluations while being affordable to run; no evaluated model–harness configuration exceeds 30% pass rate, and the strongest (GPT-5.5 with Codex) reaches 28.0%. We release the adapters, evaluation results, in-depth analysis, and Harbor-Index as open-source artifacts to support more reliable and comprehensive evaluation of language-model agents.

1 Introduction

Language models are increasingly embedded within agents: systems that not only answer questions, but also plan, use tools, write code, operate computers, and carry out long-horizon tasks. This shift has expanded the scope of model evaluation beyond static question answering to interactive settings such as software engineering, web research, finance, and terminal-based problem solving.

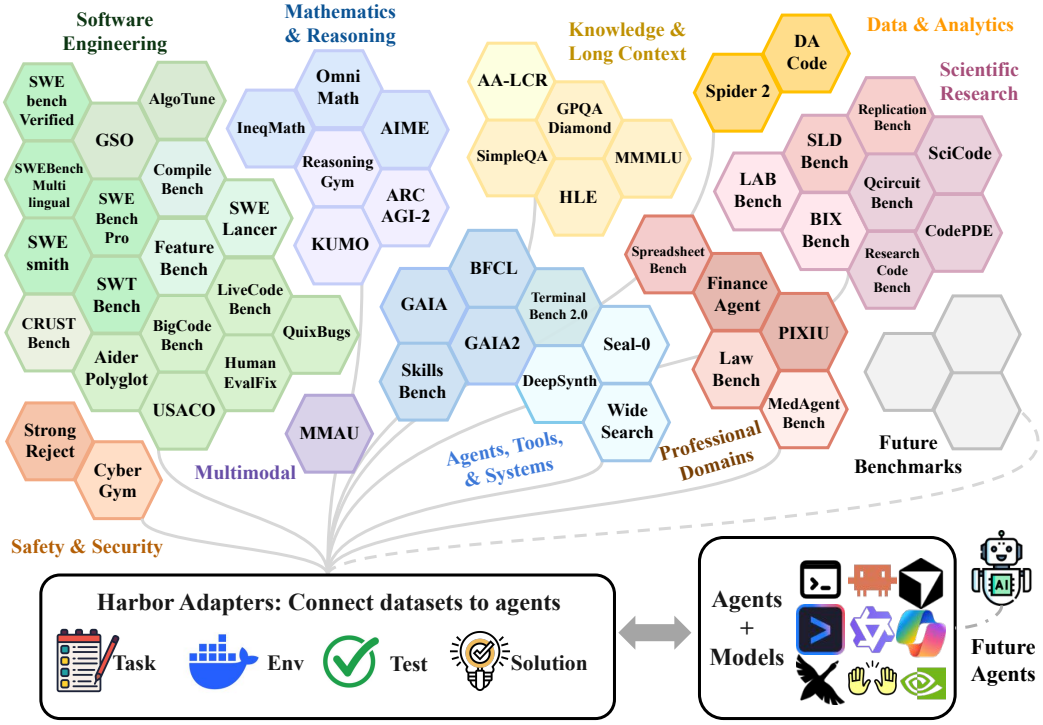


Figure 1: We integrate and evaluate Harbor adapters for 54 benchmarks across diverse domains. The Harbor format defines a task by *instruction*, *environment*, *test*, and *solution*, enabling heterogeneous datasets to be standardized to a shared schema and connected to multiple agents.

The benchmark landscape has grown in response: SWE-bench [64], Terminal-Bench [103], FinanceAgent [162], and many other agentic, tool-using, long-horizon suites; more than 200 have appeared since 2024. This growth reflects the importance of agent evaluation, but has also created a major infrastructure challenge.

Agentic benchmarks are far harder to run than question-answering suites because they are heterogeneous in task formats, environment requirements, interaction protocols, and scoring procedures. Supporting m benchmarks and n agents therefore often requires $\mathcal{O}(mn)$ integrations, one per benchmark-agent pair. This fragmentation limits reliability and scalability: papers and model releases typically report only a small subset of popular benchmarks, so it is unclear whether progress generalizes across domains or concentrates on a few well-known tasks, and comparisons are confounded by differences in benchmark implementations, environment setups, and evaluation protocols.

А вот и проблема, которую авторы пытались решить

Да мне обидно(((надо будет свои бенчи сделать в похожем формате

We address this with **Harbor Adapters**, which unify diverse agentic evaluations under Harbor, a Python library [53] originally developed for Terminal-Bench that defines tasks and runs agents in sandbox environments. Harbor Adapters decouple agents from benchmarks: each benchmark needs one adapter exposing its tasks, environments, and scoring logic through Harbor, and each agent needs one Harbor integration. This changes the integration burden from $\mathcal{O}(mn)$ benchmark-agent pairings to $\mathcal{O}(m+n)$ adapters and agent integrations (Figure 1).

Интересно глянуть в репо как она устроена

Our first contribution is the Harbor Adapters infrastructure. We port more than 80 benchmarks to Harbor, covering both natively agentic benchmarks (e.g., SWE-bench) and non-agentic ones (e.g., HLE [126]) that we make agentic by letting agents use tools, write files, and interact with executable environments. We use rigorous code review and parity experiments to verify that our adaptations remain comparable to the original implementations.

Our second contribution is a large-scale evaluation of models and harnesses. We evaluate 8 models spanning capability tiers, each under exactly two harnesses: cross-family Terminus-2 and one of 3 native harnesses (Codex for GPT, Claude Code for Claude, Gemini CLI for Gemini). This yields 16 model–harness configurations (4 distinct harness implementations) across 54 benchmarks and 6,627 tasks, three trials per configuration, consuming 226B input and output tokens and more than $300K of compute. We find:

Дорого нынче стоят прогоны бенчей

- The model underlying an agent is more influential for benchmark performance than the harness.
- The ordering of models on benchmarks is usually highly correlated, i.e., better models are usually better on many benchmarks as opposed to dominating specific niches.
- Better models use fewer tokens than weaker models across all empirical task difficulty levels.
- Among unresolved tasks, *task failures* such as misspecified tasks are common, obscuring whether a benchmark still has room for improvement or its remaining tasks are unsolvable because of errors in the task definitions.

Our third contribution is **Harbor-Index**, 82 tasks spanning 29 benchmarks drawn from the adapted suite. Because agentic evaluation is expensive, Harbor-Index is *affordable* by design: compact, diverse over many agentic domains and task formats, difficult enough to avoid immediate saturation, and high-quality enough that failures reflect model limitations rather than task defects or ambiguous grading. No evaluated model–harness configuration exceeds 30% pass rate; the strongest (GPT-5.5 with Codex) reaches 28.0%, leaving substantial room for future progress.

We open-source the benchmark adapters, evaluation tools and analyses, the full trajectories of the large-scale evaluation, and Harbor-Index 1.0 on GitHub and the Harbor Hub, with interactive results at harbor-index.org.

2 Harbor Adapters: Infrastructure

Overview Harbor Adapters is a unifying integration layer built upon **Harbor** [53]. Harbor provides a standardized task abstraction and execution environment; Harbor Adapters translate dataset-specific formats into Harbor's task schema of *instruction*, *environment*, *tests*, and *solution* (Figure 1), decoupling agent execution from benchmark-specific logic and reducing integration from $\mathcal{O}(mn)$ to $\mathcal{O}(m+n)$. In addition, the adapter infrastructure supports GPU interaction, LLM-as-a-judge verifiers, custom metric aggregation, and multiple sandbox backends (e.g., Daytona [28], Modal [108], and E2B [38]). As of May 2026, Harbor Adapters support 22 agents and more than 80 benchmarks.

Adapter construction Adapters are designed to faithfully preserve the semantics of the original benchmark datasets. To verify adaptation fairness, (1) we run multi-trial **parity experiments** matching agent, model, and execution configuration across original and Harbor-adapted benchmarks (Appendix D.4), reporting mean ± sample standard error of the mean (SEM) to account for stochasticity in Tables 3 and 4; (2) each adapter undergoes a strict three-stage quality audit by *bot, junior, and senior reviewers* to ensure faithful and standardized adaptation, with over 10,000 GitHub comments across all adapters (Appendix D.3.2).

Large-scale evaluation. Harbor Adapters enable a large-scale evaluation over 6,627 tasks from 54 benchmarks, analyzed in Section 3. We cover 8 models spanning capability tiers from Google, OpenAI, and Anthropic: Gemini-3.1-Pro and Gemini-3-Flash; Claude Opus 4.6, Sonnet 4.6, and Haiku 4.5; and GPT-5.4, GPT-5-mini, and -nano. Each model runs under exactly two harnesses: the cross-family *Terminus-2* [103] and one of 3 native harnesses (Gemini CLI for Gemini, Claude Code for Claude, and Codex for GPT), giving 16 model–harness configurations using 4 distinct harness implementations overall. Each (benchmark, model, harness) setting is repeated for 3 trials, yielding ∼0.3M trajectories. This scale is difficult to reach with $\mathcal{O}(mn)$ benchmark-specific integrations; Harbor Adapters make it feasible. Further experimental details are in Appendix E.

3 Analysis: Agentic Benchmarking at Scale

We ask four questions: (1) **Benchmark:** how much unique signal does the 54-benchmark suite provide? (2) **Harness:** does model capability or harness design drive performance? (3) **Efficiency:** where do gains justify token costs? (4) **Bottlenecks:** where do tasks break, and why do agents fail?

3.1 Benchmark: progress over time and redundancy analysis

Which benchmarks still challenge frontier models? Figure 2 shows broad progress across domains. Of the 54 benchmarks we evaluate, 13 are largely saturated, with state-of-the-art models exceeding 90%, including once-challenging ones (e.g., GPQA-Diamond [135]). Software engineering

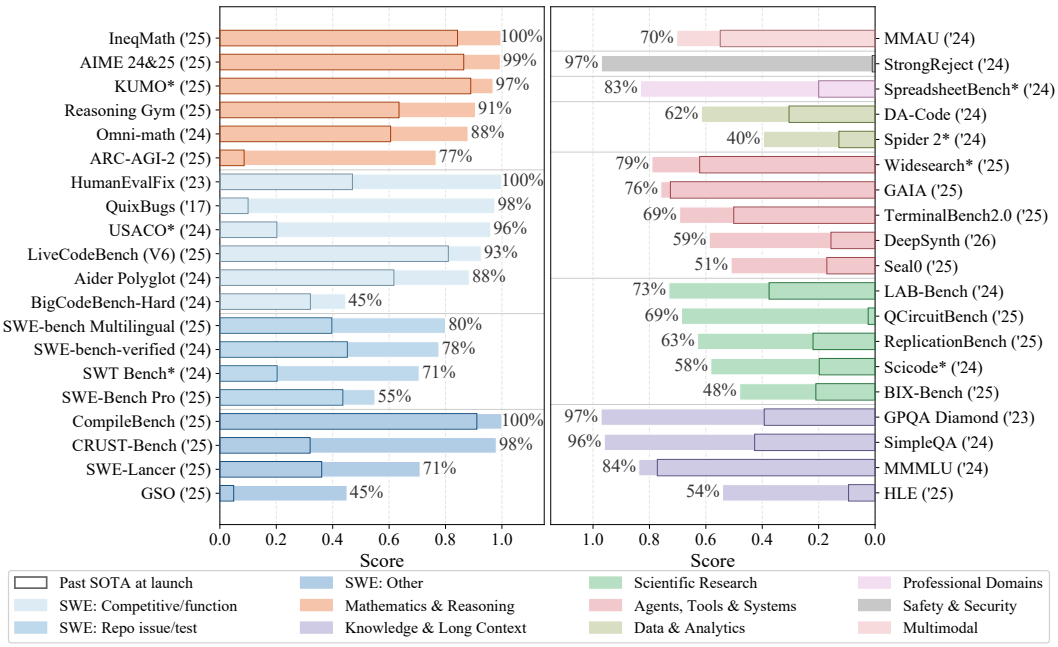


Figure 2: **Benchmark progress since published**; best reported score per benchmark, colored by domain, for 39 of the 54 benchmarks we evaluated. "SWE" abbreviates software engineering. Scores are normalized to $[0, 1]$ so that higher is better. Light-filled bars indicate the current best score in our evaluation; outlined bars, the state of the art at publication time. Benchmarks marked * use an i.i.d. subset of the full dataset. Launch and current-best scores come from benchmark papers, leaderboards, and model reports (Appendix F.3).

is especially prominent and splits sharply: functional and competitive-coding tasks near saturation (e.g., LiveCodeBench [62]), repository-level feature implementation and issue fixing progress quickly (e.g., SWE-bench-verified [64]), while software speedup remains hard (e.g., GSO [142]).

How many benchmarks do we actually need to differentiate models? While benchmark tasks can rank models by capability, we find the effective evaluation space far lower-dimensional than the number of benchmarks and tasks suggests, in three complementary forms.

(1) At the benchmark level, most scores vary along only a few shared directions. Following BenchPress [197], we apply Principal Component Analysis (PCA) to the 16×54 score matrix of model–harness configurations $\times$ benchmarks (Table 6, 7, & 8). Most variation is explained by a *single* shared capability factor; adding agent scaffolds introduces one additional direction, but the space remains low-rank (Appendix F.1). Greedy selection on absolute Spearman rank correlations agrees: after 12 benchmarks are selected, every remaining benchmark correlates at $\rho \geq 0.7$ with a selected one, so its model ranking is already well approximated and adds little information (Table 10).

(2) Difficulty and uniqueness are not the same. The benchmarks that greedy selection identifies as independently informative span the full difficulty spectrum, from hard (CodePDE [83]) to easy (StrongReject [148]), rather than clustering at one end (Table 10; Appendix F.1). Some difficult benchmarks (e.g., BigCodeBench [210]) are largely predictable from others, while benchmarks testing specialized capabilities resist prediction regardless of difficulty (e.g., FinanceAgent [162] and LabBench [78]). This suggests that evaluation design should prioritize coverage of distinct capability axes, not raw difficulty alone.

(3) Redundancy also appears within individual benchmarks. Across 52 benchmarks with at least 10 tasks (excluding CodePDE [83] (5 tasks) and SLDBench [89] (8 tasks)), 3 representative tasks per benchmark already recover the system ranking at a mean $\rho \approx 0.923$, so many tasks within a benchmark measure overlapping capabilities. The degree of redundancy varies with internal task diversity: 3 tasks nearly reproduce the full ranking where tasks test similar capabilities (e.g.,

WideSearch, $\rho = 0.99$), but not where task types are diverse (e.g., CyberGym [169], $\rho = 0.75$) (Appendix F.1).

3.2 Harness: the interaction analysis between harness and base model

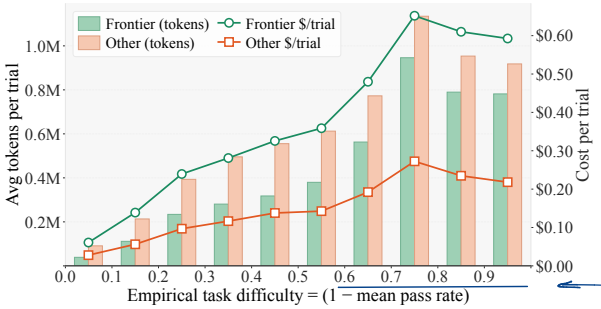
Which matters more: the model or the harness? Benchmark difficulty accounts for most score variance (ICC $= 0.75$), so we control for it with a linear mixed model using benchmark as a random intercept: 6/8 model coefficients are significant ($p < 0.05$) versus 2/4 harness coefficients, and the model fixed-effect range (0.451) is 5.2× the harness range (0.087) (Appendix F.2). Thus harness design can affect performance, but it does not substitute for base-model capability.

Очень важный вывод

3.3 Efficiency: the tradeoff analysis between performance and cost

Difficulty	# Tasks	Other	Frontier	Δ
0.0–0.1	1,461	94%	99%	+5 pp
0.1–0.2	862	80%	95%	+15 pp
0.2–0.3	660	67%	90%	+23 pp
0.3–0.4	495	54%	83%	+29 pp
0.4–0.5	429	44%	71%	+27 pp
0.5–0.6	392	35%	62%	+27 pp
0.6–0.7	448	26%	51%	+25 pp
0.7–0.8	460	17%	39%	+22 pp
0.8–0.9	454	9%	25%	+16 pp
0.9–1.0	966	1%	4%	+3 pp

Если смотреть на отношение решенных задач, то видно, что отношение больше в 2 раза



Интересно оценить не только в разрезе на попытку, а на успешную попытку. По моей прикидке тогда фронтир модели в диапазоне 0,6–1 будут дешевле

Figure 3: **Left:** Benchmark scores across empirical task-difficulty buckets for three frontier models (Claude Opus 4.6, Gemini 3.1 Pro Preview, GPT 5.4) versus five other models (Claude Sonnet 4.6, Claude Haiku 4.5, Gemini 3.1 Flash Preview, GPT 5 mini, GPT 5 nano). **Right:** Average tokens per trial (bars) and dollar cost per trial (lines) by empirical task difficulty under our experiment setup.

Frontier models are more capable but more expensive. We therefore ask: (1) where does upgrading to a frontier model pay off most, (2) do frontier models solve tasks with shorter trajectories, and (3) does that token efficiency offset their higher prices?

Интересно, что эмпирическая сложность выводится из среднего, а дальше мы смотрим на спилт фронтир vs остальные.

We define *empirical task difficulty* as one minus the average score across all model–harness runs, and group tasks into ten buckets of width 0.1. The distribution is U-shaped: 22.0% of tasks fall in the easiest bucket (0–0.1), 14.6% in the hardest (0.9–1), and the remaining 63.4% across the eight middle buckets. Note that this is a configuration-dependent proxy for intrinsic task difficulty, as it may be distorted by task brittleness (false positives/negatives) and the evaluation setup itself.

(1) Marginal benefits. As shown in Figure 3 (left), the absolute performance gain of frontier models is bell-shaped: Δ is marginal on the simplest and hardest tasks and peaks in the medium band (0.3–0.7). Other models use more tokens at every difficulty level yet cost 2–3× less per trial (Figure 3, right). The frontier value proposition is therefore most pronounced on medium-to-hard tasks, where the gain in absolute performance justifies the higher cost.

(2) Frontier models are more token-efficient, especially on easy tasks. Frontier models use fewer tokens than weaker models at every difficulty tier (Figure 3, right), and the *relative* gap is widest on easy tasks: in the easiest bucket (0–0.1), frontier models use only 42% of weaker-model tokens, a ratio rising to 61–86% in the upper half of the distribution (≥ 0.5). These results suggest that frontier models produce more concise trajectories overall, but that their relative token-efficiency advantage is most pronounced on easy tasks.

Manual inspection of agent trajectories further reveals that, on easy tasks, weaker models consume significantly more tokens, turns, and tokens per turn to reach the correct solution (e.g., 6.14× more tokens on SimpleQA [171] and 2.28× on KUMO [88]). The extra turns typically come from instruction misunderstanding and repetitive tool calls; the higher per-turn usage comes from verbose chain-of-thought and redundant verification compensating for weak reasoning. Frontier models instead follow instructions, execute cleanly, and accept outputs without extra deliberation.

(3) Token savings do not offset higher prices. Higher per-token prices outweigh the token savings: other models cost 2–3× less per trial across difficulty levels, and the absolute cost gap peaks at $0.37

per trial in the 0.7–0.8 bucket. Token efficiency therefore reduces, but does not eliminate, the frontier cost premium.

3.4 Bottlenecks: failure mode analysis

To see where and why agents fail, we audit trajectories where frontier models fail on hard tasks. Two domain-experienced human annotators independently label 200 trajectories across 12 failure modes (pooled inter-rater $\kappa = 0.66$); a calibrated Gemini 3.1 Pro judge ($\kappa = 0.51$ vs gold) extends this to $N = 6{,}028$ trajectories across 45 benchmarks, three frontier models, and two harnesses each. Full taxonomy, annotation protocol, and per-rubric reliability are in Appendix G.2.1.

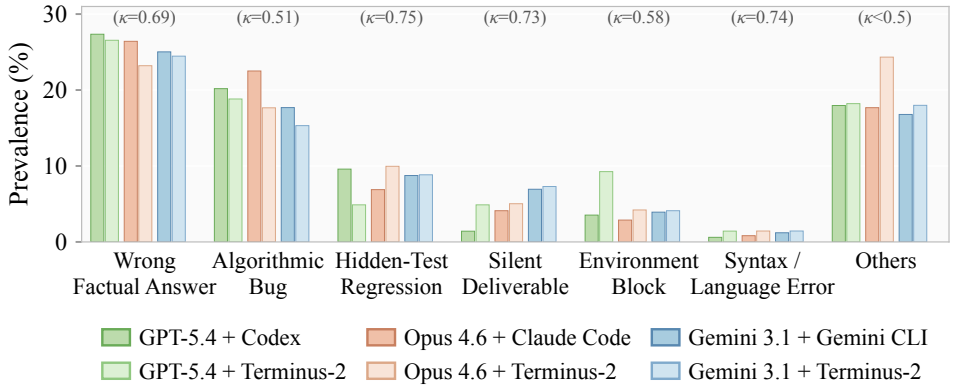


Figure 4: Agent failure-mode prevalence on $N = 6{,}028$ trajectories across seven rubrics. Each rubric shows six bars grouped by model: the darker bar is the native harness (Codex, Claude Code, Gemini CLI) and the lighter bar is the same model on Terminus-2. Per-rubric inter-rater κ is annotated above each group; *Others* collapses six rubrics with $\kappa < 0.5$.

Dominant failure modes. In Figure 4, *wrong factual answers*, *algorithmic bugs*, and *hidden-test regressions* dominate across all frontier models regardless of harness, indicating a persistent "capability gap" in task comprehension, domain knowledge, and the reasoning needed to produce substantially correct solutions. All evaluated agents also still occasionally make trivial operational mistakes such as *syntax errors* or *missing deliverables*.

Harness design shapes model behavior. Our human trajectory analysis (Section G.2.1) shows that native harnesses like Claude Code and Codex allow open-ended iteration and self-correction, whereas Terminus-2 enforces a linear *Plan-Execute-Complete* sequence; this lack of a built-in verification-correction loop makes Terminus-2 more vulnerable when iterative refinement is required.

Models diverge in behavioral patterns. GPT-5.4 prioritizes conciseness, either delivering a correct solution quickly or "giving up" on difficult tasks (e.g., HLE, LabBench), more so under Terminus-2 than Codex. Claude models usually hold a steady turn count and consistent execution pattern regardless of task difficulty. Gemini relies heavily on external search, which, while useful, often causes "information drift": it replaces an initially correct hypothesis with conflicting online information, notably on knowledge-intensive benchmarks (LabBench, MMMLU). Full per-(agent, model) breakdowns and examples are in Appendix G.2.2.

4 Harbor-Index: Compact, Diverse, Challenging, and High-Quality

From our task inspection, **low resolution rates in existing benchmarks are often attributable to structural design flaws (e.g., instruction-verification mismatches) rather than genuine task complexity**. We therefore introduce **Harbor-Index**, a curated meta-dataset of 82 tasks for next-generation agent development and evaluation. Harbor-Index prioritizes *compactness*, *diversity*, *difficulty*, and *quality*, and is built by passing the entire Harbor trial pool through a multi-stage funnel—difficulty filtering, a series of *AI audit* and *human review*, then an iterative *audit-and-fix* loop—in which each stage rejects tasks that look hard but are structurally broken: a task enters Harbor-Index only if it is well-defined, genuinely difficult, and free of exploitable verifier loopholes.

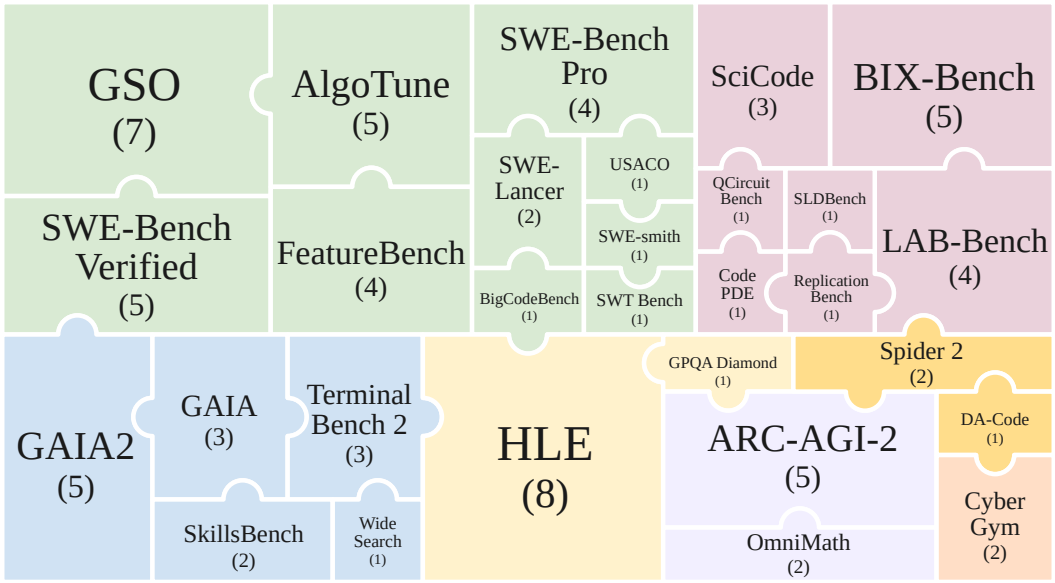


Figure 5: Harbor-Index task distribution by benchmark (82 tasks across 29 benchmarks), sized by task count and colored by domain.

4.1 Construction Pipeline

Starting from an initial pool of 6,627 Harbor tasks across 54 adapters, we apply a difficulty filter retaining the 1,311 tasks where three leading models at the time of filtering (Claude Opus 4.6, GPT-5.4, Gemini 3.1 Pro), each under both its native harness and Terminus-2 over three repeats (18 trials), succeed on at most 33% of trials. A Gemini-3-Flash auditor then scores each candidate against a quality rubric based on *instruction–verification alignment* and *essential difficulty* (difficulty must come from genuine reasoning, algorithmic thinking, domain expertise, long-horizon interactions, multi-step execution, etc.), filtering down to 307 tasks. Next, 14 domain-experienced human reviewers re-audit the survivors under the same rubric, each candidate receiving at least one senior or two junior reviews, leaving more than 110; a three-member senior panel then selects 100 on difficulty, diversity, and insight. Finally, at least two senior reviewers examine each of the 100 using trajectory-grounded failure analysis and false-positive/false-negative analysis, repairing broken tasks and dropping those that remain broken or become too easy after repair, yielding the final release of **82 tasks spanning 29 benchmarks**. Figure 5 shows the task distribution, with numbers denoting task counts per benchmark. The full audit process, quality bar, and sampling protocol are in Appendix H.

Our human audit reveals that broken task design is common: **across more than 30 benchmarks, roughly one third of the hardest candidate tasks are rejected as broken rather than genuinely difficult.** On all four GAIA2 [45] tasks reaching human review, the adapter never fires the simulation events the verifier waits on, so those trials time out regardless of agent approach. A SWE-bench Pro [33] task shows verifier overreach: it asserts `data-testid` strings and log message formats the instruction never specifies, so a correct solution fails on clerical grounds. On CRustBench [67], the verifier pins transpilation to a fixed hand-written Rust interface, so a correct and memory-safe solution is rejected when signatures or ownership annotations differ from the reference.

4.2 Evaluation Settings

We evaluate 9 models spanning capability tiers and both release types: four *closed-weight* models, whose weights are not publicly released (GPT-5.5, Claude Opus 4.8, Gemini 3.1 Pro, and Qwen3.7 Max), and five *open-weight* models (GLM 5.2, Kimi K2.6, MiniMax M3, DeepSeek V4 Pro, and MiMo V2.5 Pro). GPT-5.5, Claude Opus 4.8, and Gemini 3.1 Pro are served by first-party APIs; the other six models are served through OpenRouter [118]. All models run under default temperature, context length, and reasoning effort. Each model runs under two harness conditions: Terminus-2 and a model-native harness. This yields $9 \times 2 \times 82 = 1{,}476$ rollouts on Harbor-Index 1.0; the model

Table 1: Results on Harbor-Index 1.0 (82 tasks): pass rate (%) and approximate per-run cost (USD) for a full Index run. The *Native Harness* column group uses the vendor-native harness—Codex, Claude Code, or Gemini CLI—for GPT, Claude, and Gemini, and Claude Code for the six models served through OpenRouter. Models marked † are closed-weight (weights not publicly released); the rest are open-weight. The best pass rate in each row is shaded green. Costs reconstruct token usage at official API pricing for GPT/Claude/Gemini and OpenRouter pricing for the other models, whose cost Claude Code on OpenRouter can inflate via low cache-hit rates.

		Native Harness		Terminus-2	
Model	Agent	Pass %	$/run	Pass %	$/run
GPT-5.5†	Codex	28.0	178	19.5	155
Claude Opus 4.8†	Claude Code	20.7	269	15.9	293
Gemini 3.1 Pro†	Gemini CLI	13.4	74	11.0	89
GLM 5.2	Claude Code	8.5	205	9.8	52
Kimi K2.6	Claude Code	6.1	191	8.5	33
MiniMax M3	Claude Code	3.7	66	6.1	18
Qwen3.7 Max†	Claude Code	4.9	201	4.9	36
DeepSeek V4 Pro	Claude Code	4.9	177	3.7	35
MiMo V2.5 Pro	Claude Code	2.4	49	2.4	4

list and results appear in Table 1. Free-form answer tasks (e.g., HLE, GAIA, GPQA Diamond) use LLM-as-a-judge evaluation; elsewhere scoring is programmatic (unit tests, exact match, thresholded continuous metrics). For continuous metrics such as speedup ratios in AlgoTune [128] and GSO [142], we "SOTA-threshold" outcomes into pass/fail, so these tasks only reward solutions that advance the current frontier. See Appendix E for implementation details.

4.3 Evaluation Outcome

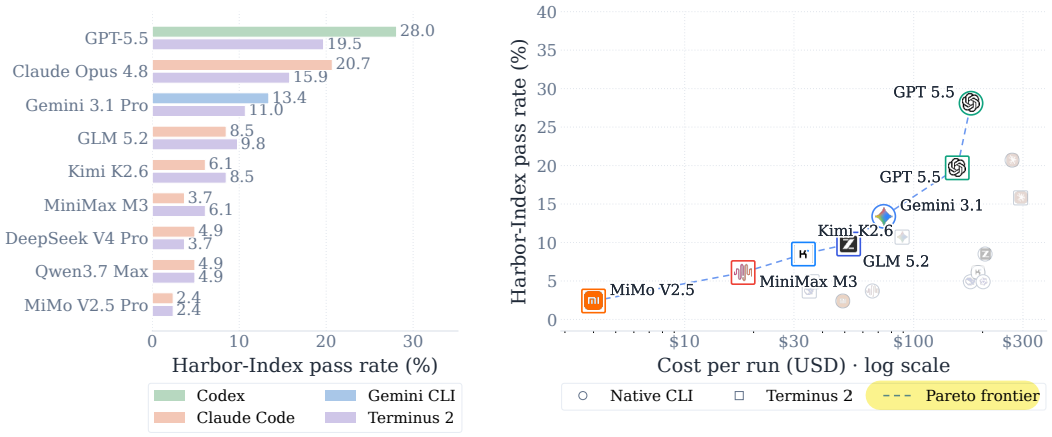


Figure 6: **Performance–cost tradeoff on Harbor-Index 1.0. Left:** Pass rates of the two evaluated harnesses for each model. **Right:** Pass rate versus cost for one complete Harbor-Index run (82 tasks). Each point is one model–harness configuration. Circles denote native harness and squares denote Terminus-2 for every model. Colored points and the dashed line mark the cost–performance Pareto frontier; gray points are dominated.

Table 1 and Figure 6 show the suite is lightweight enough for repeated evaluation. The three highest-scoring models (GPT-5.5, Claude Opus 4.8, and Gemini 3.1 Pro) benefit most from their vendor-native harnesses, whereas several open-weight models sit on the cost–performance Pareto frontier under Terminus-2, trading pass rate for a far lower cost per run.

Under the current evaluation constraints, agent failures are separated into 453 timeouts with no answer, 361 near-miss solutions, and 445 far-off fundamentally wrong answers. Capability and harness jointly shape these outcomes: the six lower-scoring models time out about twice as often

as the three highest-scoring ones (36.7% versus 18.7% of their respective rollouts), while the native harnesses use fewer tool calls and output tokens than Terminus-2 and, matched task by task, cut the timeout rate from 42% to 26%. Terminus-2's bash-only action space, lacking native image and web tools, explains part of this gap and cautions against attributing every failure to the model alone. Interactive trajectories, scores, and task-level audit artifacts are on the Harbor-Index website.[1]

5 Related Work

Agentic benchmarks and evaluation infrastructure. Agent benchmarks span tool and API use [121, 187, 131], embodied and science environments [146, 166, 186], web and enterprise navigation [207, 34, 36], application and OS control [159, 134, 179], software engineering from functions to repositories and terminals [20, 64, 103], scientific and workplace pipelines [16, 21, 180], and aggregate collections [91, 104, 109]. This diversity is informative but blocks aggregation: each benchmark ships its own action space, runtime assumptions, and scoring. Infrastructure work further shows a score is not a property of the model alone: interface, scaffold, cost budget, and holdout hygiene all move reported numbers [11, 160, 182, 175, 66]. HAL decomposes model, scaffold, and benchmark effects over 21,730 rollouts [65], while AstaBench controls cost and tool-access confounders within scientific research [12].

Benchmark validity and task quality. Aggregate scores inherit the design flaws of their tasks and metrics [133, 9, 136]: label errors destabilize aggregates [116, 47], contamination erodes held-out validity [139, 172], and leaderboard mechanics skew comparisons [147]. This sharpens for agents, whose verifier executes code rather than matching a label. An imperfect verifier caps attainable accuracy regardless of compute, since resampling cannot lower its false-positive rate [152]; agentic benchmarks misdesign setup or reward often enough to shift measured performance by up to 100% relative [208]; and SWE-bench audits trace many "resolved" instances to leakage, memorization, and weak tests, enough that strengthening those tests reorders the leaderboard [3, 168, 192, 86]. Model judges and live services add further error [202, 95, 51].

Efficient evaluation and benchmark redundancy. Model-by-benchmark score matrices are intrinsically low-rank [13, 101, 197, 200], motivating curated subsets and adaptive item selection [127, 70, 56]. Compression has limits: agreement estimates depend on unstandardized choices [124], subset prediction degrades on stronger models [199], and micro-benchmarks need more items than assumed [189, 105].

6 Discussion and Conclusion

Limitations and future work. Adapter availability limits our large-scale evaluation to 54 of the collected adapter benchmarks; cost restricts us to a focused set of popular models under native harnesses and Terminus-2, so our findings do not span all LLMs, agent architectures, or evaluation settings and are not universal. Moreover, stronger future agents may exploit benchmark artifacts or evaluation loopholes more effectively that leads to currently unexploitable reward hacks and unreliable evaluation. Therefore, we plan maintain Harbor-Index as a live benchmark, updating it to reduce saturation, preserve quality, and integrating more adapters.

Conclusion. Harbor Adapters address the fragmentation of agentic evaluation: standardizing heterogeneous benchmarks under the Harbor task interface reduces benchmark–agent integration from $\mathcal{O}(mn)$ to $\mathcal{O}(m+n)$. Across validated adapters and a large-scale evaluation of 54 benchmarks, 16 model–harness configurations spanning capability tiers, and 6,627 tasks, we find that base-model capability is the primary driver of performance gaps; many benchmarks contain redundant ranking signal; frontier models are more token-efficient but still substantially more costly; and low resolution tasks often reflect task-design flaws rather than genuine difficulty. Harbor-Index 1.0 distills this suite into 82 compact, difficult, diverse, and audited tasks spanning 29 benchmarks. Together, Harbor Adapters and Harbor-Index provide reusable infrastructure and an affordable, high-quality testbed for more reliable, scalable, and informative evaluation of language-model agents.

[1] https://harbor-index.org

References

[1] 2077AI Foundation. 2077AI: Open-Source Innovation Foundation. https://www.2077ai.com/, 2026.

[2] Aider AI. Aider polyglot benchmark, 2024.

[3] Reem Aleithan, Haoran Xue, Mohammad Mahdi Mohajer, Elijah Nnorom, Gias Uddin, and Song Wang. SWE-Bench+: Enhanced coding benchmark for LLMs, 2024. arXiv:2410.06992.

[4] Alibaba Cloud. Alibaba cloud Model Studio model pricing. https://www.alibabacloud.com/help/en/model-studio/model-pricing, 2026. Accessed: 2026-04-22 (qwen3-max, US Global tab) and 2026-05-07 (qwen3.6-max-preview, Chinese mainland tab).

[5] Maksym Andriushchenko, Alexandra Souly, Mateusz Dziemian, Derek Duenas, Maxwell Lin, Justin Wang, Dan Hendrycks, Andy Zou, J Zico Kolter, Matt Fredrikson, Yarin Gal, and Xander Davies. Agentharm: A benchmark for measuring harmfulness of LLM agents. In *The Thirteenth International Conference on Learning Representations*, 2025.

[6] Anthropic. Claude API pricing. https://platform.claude.com/docs/en/about-claude/pricing, 2026. Accessed: 2026-04-30.

[7] Ibragim Badertdinov, Alexander Golubev, Maksim Nekrashevich, Anton Shevtsov, Simon Karasik, Andrei Andriushchenko, Maria Trofimova, Daria Litvintseva, and Boris Yangel. Swe-rebench: An automated pipeline for task collection and decontaminated evaluation of software engineering agents. In *The Thirty-ninth Conference on Neural Information Processing Systems*, 2025.

[8] Andrei Barbu, David Mayo, Julian Alverio, William Luo, Christopher Wang, Dan Gutfreund, Josh Tenenbaum, and Boris Katz. Objectnet: A large-scale bias-controlled dataset for pushing the limits of object recognition models. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019.

[9] Andrew M. Bean, Ryan Othniel Kearns, Angelika Romanou, Franziska Sofia Hafner, Harry Mayne, Jan Batzner, Negar Foroutan, Chris Schmitz, Karolina Korgul, Hunar Batra, Oishi Deb, Emma Beharry, Cornelius Emde, Thomas Foster, Anna Gausen, María Grandury, Simeng Han, Valentin Hofmann, Lujain Ibrahim, Hazel Kim, Hannah Rose Kirk, Fangru Lin, Gabrielle Kaili-May Liu, Lennart Luettgau, Jabez Magomere, Jonathan Rystrøm, Anna Sotnikova, Yushi Yang, Yilun Zhao, Adel Bibi, Antoine Bosselut, Ronald Clark, Arman Cohan, Jakob Foerster, Yarin Gal, Scott A. Hale, Inioluwa Deborah Raji, Christopher Summerfield, Philip H. S. Torr, Cozmin Ududec, Luc Rocher, and Adam Mahdi. Measuring what matters: Construct validity in large language model benchmarks, 2025. NeurIPS 2025 Datasets and Benchmarks Track, arXiv:2511.04703.

[10] Gabrielle Berrada, Jannik Kossen, Freddie Bickford Smith, Muhammed Razzak, Yarin Gal, and Tom Rainforth. Scaling up active testing to large language models, 2025. NeurIPS 2025, arXiv:2508.09093.

[11] Stella Biderman, Hailey Schoelkopf, Lintang Sutawika, Leo Gao, Jonathan Tow, Baber Abbasi, Alham Fikri Aji, Pawan Sasanka Ammanamanchi, Sidney Black, Jordan Clive, Anthony DiPofi, Julen Etxaniz, Benjamin Fattori, Jessica Zosa Forde, Charles Foster, Jeffrey Hsu, Mimansa Jaiswal, Wilson Y. Lee, Haonan Li, Charles Lovering, Niklas Muennighoff, Ellie Pavlick, Jason Phang, Aviya Skowron, Samson Tan, Xiangru Tang, Kevin A. Wang, Genta Indra Winata, François Yvon, and Andy Zou. Lessons from the trenches on reproducible evaluation of language models, 2024.

[12] Jonathan Bragg, Mike D'Arcy, Nishant Balepur, Dan Bareket, Bhavana Dalvi, Sergey Feldman, Dany Haddad, Jena D. Hwang, Peter Jansen, Varsha Kishore, Bodhisattwa Prasad Majumder, Aakanksha Naik, Sigal Rahamimov, Kyle Richardson, Amanpreet Singh, Harshit Surana, Aryeh Tiktinsky, Rosni Vasu, Guy Wiener, Chloe Anastasiades, Stefan Candra, Jason Dunkelberger, Dan Emery, Rob Evans, Malachi Hamada, Regan Huff, Rodney Kinney, Matt Latzke, Jaron Lochner, Ruben Lozano-Aguilera, Cecile Nguyen, Smita Rao, Amber Tanaka,

Brooke Vlahos, Peter Clark, Doug Downey, Yoav Goldberg, Ashish Sabharwal, and Daniel S. Weld. Astabench: Rigorous benchmarking of ai agents with a scientific research suite. In *International Conference on Learning Representations (ICLR)*, 2026.

[13] Ryan Burnell, Han Hao, Andrew R. A. Conway, and Jose Hernandez Orallo. Revealing the structure of language model capabilities, 2023. arXiv:2306.10062.

[14] Nitay Calderon, Roi Reichart, and Rotem Dror. The alternative annotator test for LLM-as-a-Judge: How to statistically justify replacing human annotators with LLMs, 2025. arXiv:2501.10970.

[15] Federico Cassano, Luisa Li, Akul Sethi, Noah Shinn, Abby Brennan-Jones, Anton Lozhkov, Carolyn Jane Anderson, and Arjun Guha. Can it edit? evaluating the ability of large language models to follow code editing instructions. In *Conference on Language Modeling (COLM)*, 2024.

[16] 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. In *The Thirteenth International Conference on Learning Representations*, 2025.

[17] Chen Chen, Xinlong Hao, Weiwen Liu, Xu Huang, Xingshan Zeng, Shuai Yu, Dexun Li, Shuai Wang, Weinan Gan, Yuefeng Huang, Wulong Liu, Xinzhi Wang, Defu Lian, Baoqun Yin, Yasheng Wang, and Wu Liu. Acebench: Who wins the match point in tool usage?, 2025.

[18] Haorui Chen, Chengze Li, and Jia Li. Featbench: Evaluating coding agents on feature implementation for vibe coding, 2026.

[19] Hui Chen, Miao Xiong, Yujie Lu, Wei Han, Ailin Deng, Yufei He, Jiaying Wu, Yibo Li, Yue Liu, and Bryan Hooi. Mlr-bench: Evaluating ai agents on open-ended machine learning research. In *The Thirty-ninth Conference on Neural Information Processing Systems Datasets and Benchmarks Track*, 2025.

[20] Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, Nick Ryder, Mikhail Pavlov, Alethea Power, Lukasz Kaiser, Mohammad Bavarian, Clemens Winter, Philippe Tillet, Felipe Petroski Such, Dave Cummings, Matthias Plappert, Fotios Chantzis, Elizabeth Barnes, Ariel Herbert-Voss, William Hebgen Guss, Alex Nichol, Alex Paino, Nikolas Tezak, Jie Tang, Igor Babuschkin, Suchir Balaji, Shantanu Jain, William Saunders, Christopher Hesse, Andrew N. Carr, Jan Leike, Josh Achiam, Vedant Misra, Evan Morikawa, Alec Radford, Matthew Knight, Miles Brundage, Mira Murati, Katie Mayer, Peter Welinder, Bob McGrew, Dario Amodei, Sam McCandlish, Ilya Sutskever, and Wojciech Zaremba. Evaluating large language models trained on code, 2021.

[21] Ziru Chen, Shijie Chen, Yuting Ning, Qianheng Zhang, Boshi Wang, Botao Yu, Yifei Li, Zeyi Liao, Chen Wei, Zitong Lu, Vishal Dey, Mingyi Xue, Frazier N. Baker, Benjamin Burns, Daniel Adu-Ampratwum, Xuhui Huang, Xia Ning, Song Gao, Yu Su, and Huan Sun. Scienceagentbench: Toward rigorous assessment of language agents for data-driven scientific discovery. In *The Thirteenth International Conference on Learning Representations*, 2025.

[22] Wei-Lin Chiang, Lianmin Zheng, Ying Sheng, Anastasios Nikolas Angelopoulos, Tianle Li, Dacheng Li, Banghua Zhu, Hao Zhang, Michael Jordan, Joseph E. Gonzalez, and Ion Stoica. Chatbot arena: An open platform for evaluating LLMs by human preference. In *Forty-first International Conference on Machine Learning*, 2024.

[23] Francois Chollet, Mike Knoop, Gregory Kamradt, Bryan Landers, and Henry Pinkard. Arc-agi-2: A new challenge for frontier ai reasoning systems, 2026.

[24] João Coelho, Jingjie Ning, Jingyuan He, Kangrui Mao, Abhijay Paladugu, Pranav Setlur, Jiahe Jin, Jamie Callan, João Magalhães, Bruno Martins, and Chenyan Xiong. DeepResearchGym: A free, transparent, and reproducible evaluation sandbox for deep research, 2025. arXiv:2505.19253.

[25] Francesco Croce, Maksym Andriushchenko, Vikash Sehwag, Edoardo Debenedetti, Nicolas Flammarion, Mung Chiang, Prateek Mittal, and Matthias Hein. Robustbench: a standardized adversarial robustness benchmark. In *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 2)*, 2021.

[26] Marc-Alexandre Côté, Ákos Kádár, Xingdi Yuan, Ben Kybartas, Tavian Barnes, Emery Fine, James Moore, Ruo Yu Tao, Matthew Hausknecht, Layla El Asri, Mahmoud Adada, Wendy Tay, and Adam Trischler. Textworld: A learning environment for text-based games, 2019.

[27] Cainan Davidson, Deva Ramanan, and Neehar Peri. Refav: Towards planning-centric scenario mining, 2025.

[28] Daytona Platforms, Inc. Daytona Sandboxes. https://www.daytona.io/docs/en/sandboxes/, 2026.

[29] Thibault Le Sellier de Chezelles, Maxime Gasse, Alexandre Lacoste, Massimo Caccia, Alexandre Drouin, Léo Boisvert, Megh Thakkar, Tom Marty, Rim Assouel, Sahar Omidi Shayegan, Lawrence Keunho Jang, Xing Han Lù, Ori Yoran, Dehan Kong, Frank F. Xu, Siva Reddy, Graham Neubig, Quentin Cappart, Russ Salakhutdinov, and Nicolas Chapados. The browsergym ecosystem for web agent research. *Transactions on Machine Learning Research*, 2025. Expert Certification.

[30] Edoardo Debenedetti, Jie Zhang, Mislav Balunovic, Luca Beurer-Kellner, Marc Fischer, and Florian Tramèr. Agentdojo: A dynamic environment to evaluate prompt injection attacks and defenses for LLM agents. In *The Thirty-eight Conference on Neural Information Processing Systems Datasets and Benchmarks Track*, 2024.

[31] DeepSeek. DeepSeek API models and pricing. https://api-docs.deepseek.com/quick_start/pricing, 2026. Accessed: 2026-04-22.

[32] Adrienne Deganutti, Elad Hirsch, Haonan Zhu, Jaejung Seol, and Purvanshi Mehta. Graphic-design-bench: A comprehensive benchmark for evaluating ai on graphic design tasks, 2026.

[33] Xiang Deng, Jeff Da, Edwin Pan, Yannis Y. He, Charles Ide, Kanak Garg, Niklas Lauffer, Andrew Park, Chetan Rane, Karmini Sampath, Maya Krishnan, Srivatsa R Kundurthy, Sean M. Hendryx, Zifan Wang, Chen Bo Calvin Zhang, Noah Jacobson, Bing Liu, and Brad Kenstler. SWE-bench pro: Can AI agents solve long-horizon software engineering tasks?, 2026.

[34] Xiang Deng, Yu Gu, Boyuan Zheng, Shijie Chen, Samuel Stevens, Boshi Wang, Huan Sun, and Yu Su. Mind2web: Towards a generalist agent for the web. In *Thirty-seventh Conference on Neural Information Processing Systems Datasets and Benchmarks Track*, 2023.

[35] Shihan Dou, Ming Zhang, Zhangyue Yin, Chenhao Huang, Yujiong Shen, Junzhe Wang, Jiayi Chen, Yuchen Ni, Junjie Ye, Cheng Zhang, Huaibing Xie, Jianglu Hu, Shaolei Wang, Weichao Wang, Yanling Xiao, Yiting Liu, Zenan Xu, Zhen Guo, Pluto Zhou, Tao Gui, Zuxuan Wu, Xipeng Qiu, Qi Zhang, Xuanjing Huang, Yu-Gang Jiang, Di Wang, and Shunyu Yao. Cl-bench: A benchmark for context learning, 2026.

[36] Alexandre Drouin, Maxime Gasse, Massimo Caccia, Issam H. Laradji, Manuel Del Verme, Tom Marty, Léo Boisvert, Megh Thakkar, Quentin Cappart, David Vazquez, Nicolas Chapados, and Alexandre Lacoste. Workarena: How capable are web agents at solving common knowledge work tasks?, 2024.

[37] Mingxuan Du, Benfeng Xu, Chiwei Zhu, Xiaorui Wang, and Zhendong Mao. Deep-research bench: A comprehensive benchmark for deep research agents. *arXiv preprint arXiv:2506.11763*, 2025.

[38] E2B. E2B Documentation. https://e2b.dev/docs, 2026.

[39] Nicholas Edwards, Yukyung Lee, Yujun Audrey Mao, Yulu Qin, Sebastian Schuster, and Najoung Kim. RExbench: Can coding agents autonomously implement AI research extensions?, 2025.

[40] Alexander David Egg, Martin Iglesias Goyanes, Andreu Mora, Friso H. Kingma, Thomas Wolf, and Leandro Von Werra. DABstep: Data agent benchmark for multi-step reasoning, 2026.

[41] Mehmet Hamza Erol, Batu El, Mirac Suzgun, Mert Yuksekgonul, and James Zou. Cost-of-Pass: An economic framework for evaluating language models, 2025. arXiv:2504.13359.

[42] Zhiwei Fei, Xiaoyu Shen, Dawei Zhu, Fengzhe Zhou, Zhuo Han, Songyang Zhang, Kai Chen, Zongwen Shen, and Jidong Ge. Lawbench: Benchmarking legal knowledge of large language models, 2023.

[43] Yukang Feng, Jianwen Sun, Zelai Yang, Jiaxin Ai, Chuanhao Li, Zizhen Li, Fanrui Zhang, Kang He, Rui Ma, Jifan Lin, Jie Sun, Yang Xiao, Sizhuo Zhou, Wenxiao Wu, Yiming Liu, Pengfei Liu, Yu Qiao, Shenglin Zhang, and Kaipeng Zhang. Longcli-bench: A preliminary benchmark and study for long-horizon agentic programming in command-line interfaces, 2026.

[44] Timo Freiesleben and Sebastian Zezulka. The benchmarking epistemology: Construct validity for evaluating machine learning models, 2025. arXiv:2510.23191.

[45] Romain Froger, Pierre Andrews, Matteo Bettini, Amar Budhiraja, Ricardo Silveira Cabral, Virginie Do, Emilien Garreau, Jean-Baptiste Gaya, Hugo Laurençon, Maxime Lecanu, Kunal Malkan, Dheeraj Mekala, Pierre Menard, Gerard Moreno-Torres Bertran, Ulyana Piterbarg, Mikhail Plekhanov, Mathieu Rita, Andrey Rusakov, Vladislav Vorotilov, Mengjue Wang, Ian Yu, Amine Benhalloum, Grégoire Mialon, and Thomas Scialom. Gaia2: Benchmarking LLM agents on dynamic and asynchronous environments. In *The Fourteenth International Conference on Learning Representations*, 2026.

[46] Bofei Gao, Feifan Song, Zhe Yang, Zefan Cai, Yibo Miao, Qingxiu Dong, Lei Li, Chenghao Ma, Liang Chen, Runxin Xu, Zhengyang Tang, Benyou Wang, Daoguang Zan, Shanghaoran Quan, Ge Zhang, Lei Sha, Yichang Zhang, Xuancheng Ren, Tianyu Liu, and Baobao Chang. Omni-MATH: A universal olympiad level mathematic benchmark for large language models. In *The Thirteenth International Conference on Learning Representations*, 2025.

[47] Aryo Pradipta Gema, Joshua Ong Jun Leang, Giwon Hong, Alessio Devoto, Alberto Carlo Maria Mancino, Rohit Saxena, Xuanli He, Yu Zhao, Xiaotang Du, Mohammad Reza Ghasemi Madani, Claire Barale, Robert McHardy, Joshua Harris, Jean Kaddour, Emile van Krieken, and Pasquale Minervini. Are we done with mmlu?, 2025.

[48] Luca Gioacchini, Giuseppe Siracusano, Davide Sanvito, Kiril Gashteovski, David Friede, Roberto Bifulco, and Carolin Lawrence. Agentquest: A modular benchmark framework to measure progress and improve llm agents, 2024.

[49] Google. Gemini API pricing. https://ai.google.dev/gemini-api/docs/pricing, 2026. Accessed: 2026-04-30.

[50] Leon Guertler, Bobby Cheng, Simon Yu, Bo Liu, Leshem Choshen, and Cheston Tan. Textarena, 2025.

[51] Zhicheng Guo, Sijie Cheng, Hao Wang, Shihao Liang, Yujia Qin, Peng Li, Zhiyuan Liu, Maosong Sun, and Yang Liu. Stabletoolbench: Towards stable large-scale benchmarking on tool learning of large language models, 2025.

[52] Tanmay Gupta, Ryan Marten, Aniruddha Kembhavi, and Derek Hoiem. Grit: General robust image task benchmark, 2022.

[53] Harbor Framework Team. Harbor: A framework for evaluating and optimizing agents and models in container environments, January 2026.

[54] Hongliang He, Wenlin Yao, Kaixin Ma, Wenhao Yu, Yong Dai, Hongming Zhang, Zhenzhong Lan, and Dong Yu. Webvoyager: Building an end-to-end web agent with large multimodal models, 2024.

[55] Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. Measuring massive multitask language understanding. In *International Conference on Learning Representations*, 2021.

[56] Valentin Hofmann, David Heineman, Ian Magnusson, Kyle Lo, Jesse Dodge, Maarten Sap, Pang Wei Koh, Chun Wang, Hannaneh Hajishirzi, and Noah A. Smith. Fluid language model benchmarking, 2025. COLM 2025, arXiv:2509.11106.

[57] Junjie Hu, Sebastian Ruder, Aditya Siddhant, Graham Neubig, Orhan Firat, and Melvin Johnson. Xtreme: A massively multilingual multi-task benchmark for evaluating cross-lingual generalization, 2020.

[58] Xueyu Hu, Tao Xiong, Biao Yi, Zishu Wei, Ruixuan Xiao, Yurun Chen, Jiasheng Ye, Meiling Tao, Xiangxin Zhou, Ziyu Zhao, Yuhuai Li, Shengze Xu, Shenzhi Wang, Xinchen Xu, Shuofei Qiao, Zhaokai Wang, Kun Kuang, Tieyong Zeng, Liang Wang, Jiwei Li, Yuchen Eleanor Jiang, Wangchunshu Zhou, Guoyin Wang, Keting Yin, Zhou Zhao, Hongxia Yang, Fan Wu, Shengyu Zhang, and Fei Wu. OS agents: A survey on MLLM-based agents for general computing devices use, 2025. ACL 2025, arXiv:2508.04482.

[59] Tianyu Hua, Harper Hua, Violet Xiang, Benjamin Klieger, Sang T. Truong, Weixin Liang, Fan-Yun Sun, and Nick Haber. Researchcodebench: Benchmarking LLMs on implementing novel machine learning research code. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems Datasets and Benchmarks Track*, 2025.

[60] Kung-Hsiang Huang, Akshara Prabhakar, Sidharth Dhawan, Yixin Mao, Huan Wang, Silvio Savarese, Caiming Xiong, Philippe Laban, and Chien-Sheng Wu. Crmarena: Understanding the capacity of llm agents to perform professional crm tasks in realistic environments, 2025.

[61] Yiming Huang, Jianwen Luo, Yan Yu, Yitong Zhang, Fangyu Lei, Yifan Wei, Shizhu He, Lifu Huang, Xiao Liu, Jun Zhao, and Kang Liu. Da-code: Agent data science code generation benchmark for large language models, 2024.

[62] Naman Jain, King Han, Alex Gu, Wen-Ding Li, Fanjia Yan, Tianjun Zhang, Sida Wang, Armando Solar-Lezama, Koushik Sen, and Ion Stoica. Livecodebench: Holistic and contamination free evaluation of large language models for code. In *The Thirteenth International Conference on Learning Representations*, 2025.

[63] Yixing Jiang, Kameron C. Black, Gloria Geng, Danny Park, James Zou, Andrew Y. Ng, and Jonathan H. Chen. Medagentbench: A realistic virtual ehr environment to benchmark medical llm agents, 2025.

[64] 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?, 2024.

[65] Sayash Kapoor, Benedikt Stroebl, Peter Kirgis, Nitya Nadgir, Zachary S Siegel, Boyi Wei, Tianci Xue, Ziru Chen, Felix Chen, Saiteja Utpala, Franck Ndzomga, Dheeraj Oruganty, Sophie Luskin, Kangheng Liu, Botao Yu, Amit Arora, Dongyoon Hahm, Harsh Trivedi, Huan Sun, Juyong Lee, Tengjun Jin, Yifan Mai, Yifei Zhou, Yuxuan Zhu, Rishi Bommasani, Daniel Kang, Dawn Song, Peter Henderson, Yu Su, Percy Liang, and Arvind Narayanan. Holistic agent leaderboard: The missing infrastructure for AI agent evaluation. In *The Fourteenth International Conference on Learning Representations*, 2026.

[66] Sayash Kapoor, Benedikt Stroebl, Zachary S. Siegel, Nitya Nadgir, and Arvind Narayanan. AI agents that matter, 2024. arXiv:2407.01502.

[67] Anirudh Khatry, Robert Zhang, Jia Pan, Ziteng Wang, Qiaochu Chen, Greg Durrett, and Isil Dillig. CRUST-bench: A comprehensive benchmark for c-to-safe-rust transpilation. In *Second Conference on Language Modeling*, 2025.

[68] Arpandeep Khatua, Hao Zhu, Peter Tran, Arya Prabhudesai, Frederic Sadrieh, Johann K. Lieberwirth, Xinkai Yu, Yicheng Fu, Michael J. Ryan, Jiaxin Pei, and Diyi Yang. Cooperbench: Why coding agents cannot be your teammates yet, 2026.

[69] Douwe Kiela, Max Bartolo, Yixin Nie, Divyansh Kaushik, Atticus Geiger, Zhengxuan Wu, Bertie Vidgen, Grusha Prasad, Amanpreet Singh, Pratik Ringshia, Zhiyi Ma, Tristan Thrush, Sebastian Riedel, Zeerak Waseem, Pontus Stenetorp, Robin Jia, Mohit Bansal, Christopher Potts, and Adina Williams. Dynabench: Rethinking benchmarking in nlp, 2021.

[70] Alex Kipnis, Konstantinos Voudouris, Luca M. Schulze Buschoff, and Eric Schulz. metabench – a sparse benchmark of reasoning and knowledge in large language models, 2025. ICLR 2025, arXiv:2407.12844.

[71] Jing Yu Koh, Robert Lo, Lawrence Jang, Vikram Duvvur, Ming Chong Lim, Po-Yu Huang, Graham Neubig, Shuyan Zhou, Ruslan Salakhutdinov, and Daniel Fried. Visualwebarena: Evaluating multimodal agents on realistic visual web tasks. In *ICLR 2024 Workshop on Large Language Model (LLM) Agents*, 2024.

[72] Pang Wei Koh, Shiori Sagawa, Henrik Marklund, Sang Michael Xie, Marvin Zhang, Akshay Balsubramani, Weihua Hu, Michihiro Yasunaga, Richard Lanas Phillips, Irena Gao, Tony Lee, Etienne David, Ian Stavness, Wei Guo, Berton A. Earnshaw, Imran S. Haque, Sara Beery, Jure Leskovec, Anshul Kundaje, Emma Pierson, Sergey Levine, Chelsea Finn, and Percy Liang. Wilds: A benchmark of in-the-wild distribution shifts, 2021.

[73] Thomas Kuntz, Agatha Duzan, Hao Zhao, Francesco Croce, Zico Kolter, Nicolas Flammarion, and Maksym Andriushchenko. OS-Harm: A benchmark for measuring safety of computer use agents, 2025. NeurIPS 2025 Datasets and Benchmarks Track, arXiv:2506.14866.

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

[75] Eugenie Lai, Gerardo Vitagliano, Ziyu Zhang, Om Chabra, SIVAPRASAD SUDHIR, Anna Zeng, Anton A. Zabreyko, Chenning Li, Ferdi Kossmann, Jialin Ding, Jun Chen, Markos Markakis, Matthew Russo, Weiyang Wang, Ziniu Wu, Mike Cafarella, Lei Cao, Samuel Madden, and Tim Kraska. KRAMABENCH: A benchmark for AI systems on data-to-insight pipelines over data lakes. In *The Fourteenth International Conference on Learning Representations*, 2026.

[76] Yuhang Lai, Chengxi Li, Yiming Wang, Tianyi Zhang, Ruiqi Zhong, Luke Zettlemoyer, Scott Wen tau Yih, Daniel Fried, Sida Wang, and Tao Yu. Ds-1000: A natural and reliable benchmark for data science code generation, 2022.

[77] Laude Institute. Laude Institute. https://www.laude.org/, 2026.

[78] Jon M. Laurent, Joseph D. Janizek, Michael Ruzo, Michaela M. Hinks, Michael J. Hammerling, Siddharth Narayanan, Manvitha Ponnapati, Andrew D. White, and Samuel G. Rodriques. Lab-bench: Measuring capabilities of language models for biology research, 2024.

[79] Fangyu Lei, Jixuan Chen, Yuxiao Ye, Ruisheng Cao, Dongchan Shin, Hongjin SU, ZHAO-QING SUO, Hongcheng Gao, Wenjing Hu, Pengcheng Yin, Victor Zhong, Caiming Xiong, Ruoxi Sun, Qian Liu, Sida Wang, and Tao Yu. Spider 2.0: Evaluating language models on real-world enterprise text-to-SQL workflows. In *The Thirteenth International Conference on Learning Representations*, 2025.

[80] Bowen Li, Wenhan Wu, Ziwei Tang, Lin Shi, John Yang, Jinyang Li, Shunyu Yao, Chen Qian, Binyuan Hui, Qicheng Zhang, Zhiyin Yu, He Du, Ping Yang, Dahua Lin, Chao Peng, and Kai Chen. Prompting large language models to tackle the full software development lifecycle: A case study, 2024.

[81] Jinyang Li, Binyuan Hui, GE QU, Jiaxi Yang, Binhua Li, Bowen Li, Bailin Wang, Bowen Qin, Ruiying Geng, Nan Huo, Xuanhe Zhou, Chenhao Ma, Guoliang Li, Kevin Chang, Fei Huang, Reynold Cheng, and Yongbin Li. Can LLM already serve as a database interface? a BIg bench for large-scale database grounded text-to-SQLs. In *Thirty-seventh Conference on Neural Information Processing Systems Datasets and Benchmarks Track*, 2023.

[82] Minghao Li, Yingxiu Zhao, Bowen Yu, Feifan Song, Hangyu Li, Haiyang Yu, Zhoujun Li, Fei Huang, and Yongbin Li. Api-bank: A comprehensive benchmark for tool-augmented llms, 2023.

[83] Shanda Li, Tanya Marwah, Junhong Shen, Weiwei Sun, Andrej Risteski, Yiming Yang, and Ameet Talwalkar. CodePDE: An inference framework for LLM-driven PDE solver generation, 2025.

[84] Xiangyi Li, Wenbo Chen, Yimin Liu, Shenghan Zheng, Xiaokun Chen, Yifeng He, Yubo Li, Bingran You, Haotian Shen, Jiankai Sun, Shuyi Wang, Binxu Li, Qunhong Zeng, Di Wang, Xuandong Zhao, Yuanli Wang, Roey Ben Chaim, Zonglin Di, Yipeng Gao, Junwei He, Yizhuo He, Liqiang Jing, Luyang Kong, Xin Lan, Jiachen Li, Songlin Li, Yijiang Li, Yueqian Lin, Xinyi Liu, Xuanqing Liu, Haoran Lyu, Ze Ma, Bowei Wang, Runhui Wang, Tianyu Wang, Wengao Ye, Yue Zhang, Hanwen Xing, Yiqi Xue, Steven Dillmann, and Han chung Lee. Skillsbench: Benchmarking how well agent skills work across diverse tasks, 2026.

[85] Percy Liang, Rishi Bommasani, Tony Lee, Dimitris Tsipras, Dilara Soylu, Michihiro Yasunaga, Yian Zhang, Deepak Narayanan, Yuhuai Wu, Ananya Kumar, Benjamin Newman, Binhang Yuan, Bobby Yan, Ce Zhang, Christian Cosgrove, Christopher D Manning, Christopher Re, Diana Acosta-Navas, Drew A. Hudson, Eric Zelikman, Esin Durmus, Faisal Ladhak, Frieda Rong, Hongyu Ren, Huaxiu Yao, Jue WANG, Keshav Santhanam, Laurel Orr, Lucia Zheng, Mert Yuksekgonul, Mirac Suzgun, Nathan Kim, Neel Guha, Niladri S. Chatterji, Omar Khattab, Peter Henderson, Qian Huang, Ryan Andrew Chi, Sang Michael Xie, Shibani Santurkar, Surya Ganguli, Tatsunori Hashimoto, Thomas Icard, Tianyi Zhang, Vishrav Chaudhary, William Wang, Xuechen Li, Yifan Mai, Yuhui Zhang, and Yuta Koreeda. Holistic evaluation of language models. *Transactions on Machine Learning Research*, 2023. Featured Certification, Expert Certification, Outstanding Certification.

[86] Shanchao Liang, Spandan Garg, and Roshanak Zilouchian Moghaddam. The SWE-Bench illusion: When State-of-the-Art LLMs remember instead of reason, 2025. arXiv:2506.12286.

[87] Derrick Lin, James Koppel, Angela Chen, and Armando Solar-Lezama. QuixBugs: A multi-lingual program repair benchmark set based on the quixey challenge. *Proceedings Companion of the 2017 ACM SIGPLAN International Conference on Systems, Programming, Languages, and Applications: Software for Humanity*, 2017.

[88] Haowei Lin, Xiangyu Wang, Ruilin Yan, Baizhou Huang, Haotian Ye, Jianhua Zhu, Zihao Wang, James Zou, Jianzhu Ma, and Yitao Liang. Generative evaluation of complex reasoning in large language models, 2025.

[89] Haowei Lin, Haotian Ye, Wenzheng Feng, Quzhe Huang, Yujun Li, Hubert Lim, Zhengrui Li, Xiangyu Wang, Jianzhu Ma, Yitao Liang, and James Zou. Can language models discover scaling laws? In *The Fourteenth International Conference on Learning Representations*, 2026.

[90] Evan Zheran Liu, Kelvin Guu, Panupong Pasupat, and Percy Liang. Reinforcement learning on web interfaces using workflow-guided exploration. In *International Conference on Learning Representations*, 2018.

[91] Xiao Liu, Hao Yu, Hanchen Zhang, Yifan Xu, Xuanyu Lei, Hanyu Lai, Yu Gu, Hangliang Ding, Kaiwen Men, Kejuan Yang, Shudan Zhang, Xiang Deng, Aohan Zeng, Zhengxiao Du, Chenhui Zhang, Sheng Shen, Tianjun Zhang, Yu Su, Huan Sun, Minlie Huang, Yuxiao Dong, and Jie Tang. Agentbench: Evaluating llms as agents, 2025.

[92] Yu Lu Liu, Su Lin Blodgett, Jackie Chi Kit Cheung, Q. Vera Liao, Alexandra Olteanu, and Ziang Xiao. ECBD: Evidence-Centered benchmark design for NLP, 2024. arXiv:2406.08723.

[93] Zichen Liu, Anya Sims, Keyu Duan, Changyu Chen, Simon Yu, Xiangxin Zhou, Haotian Xu, Shaopan Xiong, Bo Liu, Chenmien Tan, Chuen Yang Beh, Weixun Wang, Hao Zhu, Weiyan Shi, Diyi Yang, Michael Shieh, Yee Whye Teh, Wee Sun Lee, and Min Lin. GEM: A gym for agentic LLMs, 2025. arXiv:2510.01051.

[94] Pan Lu, Jiayi Sheng, Luna Lyu, Jikai Jin, Tony Xia, Alex Gu, and James Zou. Solving inequality proofs with large language models. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems Datasets and Benchmarks Track*, 2025.

[95] Xing Han Lù, Amirhossein Kazemnejad, Nicholas Meade, Arkil Patel, Dongchan Shin, Alejandra Zambrano, Karolina Stanczak, Peter Shaw, Christopher Pal, and Siva Reddy. Agentrewardbench: Evaluating automatic evaluations of web agent trajectories. In *Second Conference on Language Modeling*, 2025.

[96] Zimu Lu, Yunqiao Yang, Houxing Ren, Haotian Hou, Han Xiao, Ke Wang, Weikang Shi, Aojun Zhou, Mingjie Zhan, and Hongsheng Li. Webgen-bench: Evaluating LLMs on generating interactive and functional websites from scratch. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems Datasets and Benchmarks Track*, 2025.

[97] Xing Han Lù, Zdeněk Kasner, and Siva Reddy. Weblinx: Real-world website navigation with multi-turn dialogue, 2024.

[98] Zeyao Ma, Bohan Zhang, Jing Zhang, Jifan Yu, Xiaokang Zhang, Xiaohan Zhang, Sijia Luo, Xi Wang, and Jie Tang. Spreadsheetbench: Towards challenging real world spreadsheet manipulation, 2024.

[99] Zhiyi Ma, Kawin Ethayarajh, Tristan Thrush, Somya Jain, Ledell Yu Wu, Robin Jia, Christopher Potts, Adina Williams, and Douwe Kiela. Dynaboard: An evaluation-as-a-service platform for holistic next-generation benchmarking. In A. Beygelzimer, Y. Dauphin, P. Liang, and J. Wortman Vaughan, editors, *Advances in Neural Information Processing Systems*, 2021.

[100] Adyasha Maharana, Dong-Ho Lee, Sergey Tulyakov, Mohit Bansal, Francesco Barbieri, and Yuwei Fang. Evaluating very long-term conversational memory of llm agents. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 13851–13870, 2024.

[101] Aviya Maimon, Amir DN Cohen, Gal Vishne, Shauli Ravfogel, and Reut Tsarfaty. From benchmarks to skills: Low-Rank factors for LLM evaluation, 2025. arXiv:2507.20208.

[102] Qiuyang Mang, Wenhao Chai, Zhifei Li, Huanzhi Mao, Shang Zhou, Alexander Du, Hanchen Li, Shu Liu, Edwin Chen, Yichuan Wang, Xieting Chu, Zerui Cheng, Yuan Xu, Tian Xia, Zirui Wang, Tianneng Shi, Jianzhu Yao, Yilong Zhao, Qizheng Zhang, Charlie Ruan, Zeyu Shen, Kaiyuan Liu, Runyuan He, Dong Xing, Zerui Li, Zirong Zeng, Yige Jiang, Lufeng Cheng, Ziyi Zhao, Youran Sun, Wesley Zheng, Meiyuwang Zhang, Ruyi Ji, Xuechang Tu, Zihan Zheng, Zexing Chen, Kangyang Zhou, Zhaozi Wang, Jingbang Chen, Aleksandra Korolova, Peter Henderson, Pramod Viswanath, Vijay Ganesh, Saining Xie, Zhuang Liu, Dawn Song, Sewon Min, Ion Stoica, Joseph E. Gonzalez, Jingbo Shang, and Alvin Cheung. Frontiercs: Evolving challenges for evolving intelligence, 2025.

[103] Mike A. Merrill, Alexander G. Shaw, Nicholas Carlini, Boxuan Li, Harsh Raj, Ivan Bercovich, Lin Shi, Jeong Yeon Shin, Thomas Walshe, E. 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 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, 2026.

[104] Grégoire Mialon, Clémentine Fourrier, Craig Swift, Thomas Wolf, Yann LeCun, and Thomas Scialom. Gaia: a benchmark for general ai assistants, 2023.

[105] Evan Miller. Adding error bars to evals: A statistical approach to language model evaluations, 2024. arXiv:2411.00640.

[106] Samuel Miserendino, Michele Wang, Tejal Patwardhan, and Johannes Heidecke. SWE-lancer: Can frontier LLMs earn $1 million from real-world freelance software engineering? In *Forty-second International Conference on Machine Learning*, 2025.

[107] Ludovico Mitchener, Jon M Laurent, Alex Andonian, Benjamin Tenmann, Siddharth Narayanan, Geemi P Wellawatte, Andrew White, Lorenzo Sani, and Samuel G Rodriques. Bixbench: a comprehensive benchmark for llm-based agents in computational biology, 2025.

[108] Modal Labs. Modal Sandboxes. https://modal.com/docs/guide/sandboxes, 2026.

[109] Mahmoud Mohammadi, Yipeng Li, Jane Lo, and Wendy Yip. Evaluation and benchmarking of LLM agents: A survey, 2025. arXiv:2507.21504.

[110] Moonshot AI. Kimi K2.5 API pricing. https://platform.kimi.ai/docs/pricing/chat-k25, 2026. Accessed: 2026-04-22.

[111] Niklas Muennighoff, Qian Liu, Armel Randy Zebaze, Qinkai Zheng, Binyuan Hui, Terry Yue Zhuo, Swayam Singh, Xiangru Tang, Leandro Von Werra, and Shayne Longpre. Octopack: Instruction tuning code large language models. In *The Twelfth International Conference on Learning Representations*, 2024.

[112] Niklas Muennighoff, Nouamane Tazi, Loic Magne, and Nils Reimers. MTEB: Massive text embedding benchmark. In Andreas Vlachos and Isabelle Augenstein, editors, *Proceedings of the 17th Conference of the European Chapter of the Association for Computational Linguistics*, pages 2014–2037, Dubrovnik, Croatia, May 2023. Association for Computational Linguistics.

[113] Niels Mündler, Mark Niklas Müller, Jingxuan He, and Martin Vechev. Swt-bench: Testing and validating real-world bug-fixes with code agents, 2025.

[114] Deepak Nathani, Lovish Madaan, Nicholas Roberts, Nikolay Bashlykov, Ajay Menon, Vincent Moens, Mikhail Plekhanov, Amar Budhiraja, Despoina Magka, Vladislav Vorotilov, Gaurav Chaurasia, Dieuwke Hupkes, Ricardo Silveira Cabral, Tatiana Shavrina, Jakob Nicolaus Foerster, Yoram Bachrach, William Yang Wang, and Roberta Raileanu. MLGym: A new framework and benchmark for advancing AI research agents. In *Second Conference on Language Modeling*, 2025.

[115] Yannic Neuhaus and Matthias Hein. RePOPE: Impact of annotation errors on the POPE benchmark, 2025. arXiv:2504.15707.

[116] Curtis G. Northcutt, Anish Athalye, and Jonas Mueller. Pervasive label errors in test sets destabilize machine learning benchmarks, 2021. NeurIPS 2021 Datasets and Benchmarks Track, arXiv:2103.14749.

[117] OpenAI. API pricing. https://developers.openai.com/api/docs/pricing, 2026. Accessed: 2026-04-30.

[118] OpenRouter, Inc. OpenRouter: The Unified Interface for LLMs. https://openrouter.ai/, 2026.

[119] Harshith Padigela, Chintan Shah, and Dinkar Juyal. ML-dev-bench: Comparative analysis of AI agents on ML development workflows. In *Towards Agentic AI for Science: Hypothesis Generation, Comprehension, Quantification, and Validation*, 2025.

[120] Jiayi Pan, Xingyao Wang, Graham Neubig, Navdeep Jaitly, Heng Ji, Alane Suhr, and Yizhe Zhang. Training software engineering agents and verifiers with SWE-gym. In *Forty-second International Conference on Machine Learning*, 2025.

[121] Shishir G Patil, Huanzhi Mao, Fanjia Yan, Charlie Cheng-Jie Ji, Vishnu Suresh, Ion Stoica, and Joseph E. Gonzalez. The berkeley function calling leaderboard (BFCL): From tool use to agentic evaluation of large language models. In *Forty-second International Conference on Machine Learning*, 2025.

[122] Shishir G. Patil, Tianjun Zhang, Xin Wang, and Joseph E. Gonzalez. Gorilla: Large language model connected with massive apis, 2023.

[123] Debjit Paul, Daniel Murphy, Milan Gritta, Ronald Cardenas, Victor Prokhorov, Lena Sophia Bolliger, Aysim Toker, Roy Miles, Andreea-Maria Oncescu, Jasivan Alex Sivakumar, Philipp Borchert, Ismail Elezi, Meiru Zhang, Ka Yiu Lee, Guchun Zhang, Jun Wang, and Gerasimos Lampouras. A benchmark for deep information synthesis. In *The Fourteenth International Conference on Learning Representations*, April 2026.

[124] Yotam Perlitz, Ariel Gera, Ofir Arviv, Asaf Yehudai, Elron Bandel, Eyal Shnarch, Michal Shmueli-Scheuer, and Leshem Choshen. Do these LLM benchmarks agree? fixing benchmark evaluation with BenchBench, 2024. arXiv:2407.13696.

[125] Thinh Pham, Nguyen Phan Nguyen, Pratibha Zunjare, Weiyuan Chen, Yu-Min Tseng, and Tu Vu. SealQA: Raising the bar for reasoning in search-augmented language models. In *The Fourteenth International Conference on Learning Representations*, 2026.

[126] Long Phan, Alice Gatti, Nathaniel Li, Adam Khoja, Ryan Kim, Richard Ren, et al. A benchmark of expert-level academic questions to assess AI capabilities. *Nature*, 649(8099):1139–1146, 2026.

[127] Felipe Maia Polo, Lucas Weber, Leshem Choshen, Yuekai Sun, Gongjun Xu, and Mikhail Yurochkin. tinybenchmarks: evaluating llms with fewer examples, 2024.

[128] Ori Press, Brandon Amos, Haoyu Zhao, Yikai Wu, Samuel Ainsworth, Dominik Krupke, Patrick Kidger, Touqir Sajed, Bartolomeo Stellato, Jisun Park, Nathanael Bosch, Eli Meril, Albert Steppi, Arman Zharmagambetov, Fangzhao Zhang, David Pérez-Piñeiro, Alberto Mercurio, Ni Zhan, Talor Abramovich, Kilian Lieret, Hanlin Zhang, Shirley Huang, Matthias Bethge, and Ofir Press. Algotune: Can language models speed up general-purpose numerical programs? In *The Thirty-ninth Annual Conference on Neural Information Processing Systems Datasets and Benchmarks Track*, 2025.

[129] Ariel Procaccia, Benjamin Schiffer, Serena Wang, and Shirley Zhang. Metritocracy: Representative metrics for lite benchmarks, 2025. arXiv:2506.09813.

[130] 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*, 2025.

[131] Yujia Qin, Shihao Liang, Yining Ye, Kunlun Zhu, Lan Yan, Yaxi Lu, Yankai Lin, Xin Cong, Xiangru Tang, Bill Qian, Sihan Zhao, Lauren Hong, Runchu Tian, Ruobing Xie, Jie Zhou, Mark Gerstein, dahai li, Zhiyuan Liu, and Maosong Sun. ToolLLM: Facilitating large language models to master 16000+ real-world APIs. In *The Twelfth International Conference on Learning Representations*, 2024.

[132] Quesma. CompileBench: Evaluating AI agents on real-world software compilation tasks, 2025.

[133] Inioluwa Deborah Raji, Emily M. Bender, Amandalynne Paullada, Emily Denton, and Alex Hanna. AI and the everything in the whole wide world benchmark, 2021. NeurIPS 2021 Datasets and Benchmarks Track, arXiv:2111.15366.

[134] Christopher Rawles, Sarah Clinckemaillie, Yifan Chang, Jonathan Waltz, Gabrielle Lau, Marybeth Fair, Alice Li, William Bishop, Wei Li, Folawiyo Campbell-Ajala, Daniel Toyama, Robert Berry, Divya Tyamagundlu, Timothy Lillicrap, and Oriana Riva. Androidworld: A dynamic benchmarking environment for autonomous agents, 2025.

[135] David Rein, Betty Li Hou, Asa Cooper Stickland, Jackson Petty, Richard Yuanzhe Pang, Julien Dirani, Julian Michael, and Samuel R. Bowman. GPQA: A graduate-level google-proof q&a benchmark. In *First Conference on Language Modeling*, 2024.

[136] Anka Reuel, Amelia Hardy, Chandler Smith, Max Lamparth, Malcolm Hardy, and Mykel J. Kochenderfer. BetterBench: Assessing AI benchmarks, uncovering issues, and establishing best practices, 2024. NeurIPS 2024, arXiv:2411.12990.

[137] Yangjun Ruan, Chris J. Maddison, and Tatsunori Hashimoto. Observational scaling laws and the predictability of language model performance, 2024. NeurIPS 2024, arXiv:2405.10938.

[138] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. Imagenet large scale visual recognition challenge, 2015.

[139] Oscar Sainz, Jon Ander Campos, Iker García-Ferrero, Julen Etxaniz, Oier Lopez de Lacalle, and Eneko Agirre. NLP evaluation in trouble: On the need to measure LLM data contamination for each benchmark, 2024. EMNLP 2024 Findings, arXiv:2310.18018.

[140] S Sakshi, Utkarsh Tyagi, Sonal Kumar, Ashish Seth, Ramaneswaran Selvakumar, Oriol Nieto, Ramani Duraiswami, Sreyan Ghosh, and Dinesh Manocha. MMAU: A massive multi-task audio understanding and reasoning benchmark. In *The Thirteenth International Conference on Learning Representations*, 2025.

[141] Yongliang Shen, Kaitao Song, Xu Tan, Wenqi Zhang, Kan Ren, Siyu Yuan, Weiming Lu, Dongsheng Li, and Yueting Zhuang. Taskbench: Benchmarking large language models for task automation, 2024.

[142] Manish Shetty, Naman Jain, Jinjian Liu, Vijay Kethanaboyina, Koushik Sen, and Ion Stoica. GSO: Challenging software optimization tasks for evaluating SWE-agents. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems Datasets and Benchmarks Track*, 2025.

[143] Ben Shi, Michael Tang, Karthik R Narasimhan, and Shunyu Yao. Can language models solve olympiad programming? In *First Conference on Language Modeling*, 2024.

[144] Noah Shinn, Federico Cassano, Edward Berman, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning, 2023. NeurIPS 2023, arXiv:2303.11366.

[145] Parshin Shojaee, Ngoc-Hieu Nguyen, Kazem Meidani, Amir Barati Farimani, Khoa D Doan, and Chandan K. Reddy. LLM-SRBench: A new benchmark for scientific equation discovery with large language models. In *Forty-second International Conference on Machine Learning*, 2025.

[146] Mohit Shridhar, Xingdi Yuan, Marc-Alexandre Cote, Yonatan Bisk, Adam Trischler, and Matthew Hausknecht. ALFWorld: Aligning text and embodied environments for interactive learning. In *International Conference on Learning Representations*, 2021.

[147] Shivalika Singh, Yiyang Nan, Alex Wang, Daniel D'Souza, Sayash Kapoor, Ahmet Üstün, Sanmi Koyejo, Yuntian Deng, Shayne Longpre, Noah A. Smith, Beyza Ermis, Marzieh Fadaee, and Sara Hooker. The leaderboard illusion, 2025. arXiv:2504.20879.

[148] Alexandra Souly, Qingyuan Lu, Dillon Bowen, Tu Trinh, Elvis Hsieh, Sana Pandey, Pieter Abbeel, Justin Svegliato, Scott Emmons, Olivia Watkins, and Sam Toyer. A strongREJECT for empty jailbreaks. In *The Thirty-eight Conference on Neural Information Processing Systems Datasets and Benchmarks Track*, 2024.

[149] Aarohi Srivastava, Abhinav Rastogi, Abhishek Rao, Abu Awal Md Shoeb, Abubakar Abid, Adam Fisch, Adam R. Brown, Adam Santoro, Aditya Gupta, Adrià Garriga-Alonso, Agnieszka Kluska, Aitor Lewkowycz, Akshat Agarwal, Alethea Power, Alex Ray, Alex Warstadt, Alexander W. Kocurek, Ali Safaya, Ali Tazarv, Alice Xiang, Alicia Parrish, Allen Nie, Aman Hussain, Amanda Askell, Amanda Dsouza, Ambrose Slone, Ameet Rahane, Anantharaman S. Iyer, Anders Andreassen, Andrea Madotto, Andrea Santilli, Andreas Stuhlmüller, Andrew Dai, Andrew La, Andrew Lampinen, Andy Zou, Angela Jiang, Angelica Chen, Anh Vuong, Animesh Gupta, Anna Gottardi, Antonio Norelli, Anu Venkatesh, Arash Gholamidavoodi, Arfa Tabassum, Arul Menezes, Arun Kirubarajan, Asher Mullokandov, Ashish Sabharwal, Austin Herrick, Avia Efrat, Aykut Erdem, Ayla Karakaş, B. Ryan Roberts, Bao Sheng Loe, Barret Zoph, Bartłomiej Bojanowski, Batuhan Özyurt, Behnam Hedayatnia, Behnam Neyshabur, Benjamin Inden, Benno Stein, Berk Ekmekci, Bill Yuchen Lin, Blake Howald, Bryan Orinion, Cameron Diao, Cameron Dour, Catherine Stinson, Cedrick Argueta, César Ferri Ramírez, Chandan Singh, Charles Rathkopf, Chenlin Meng, Chitta Baral, Chiyu Wu, Chris Callison-Burch, Chris Waites, Christian Voigt, Christopher D. Manning, Christopher Potts, Cindy

Ramirez, Clara E. Rivera, Clemencia Siro, Colin Raffel, Courtney Ashcraft, Cristina Garbacea, Damien Sileo, Dan Garrette, Dan Hendrycks, Dan Kilman, Dan Roth, Daniel Freeman, Daniel Khashabi, Daniel Levy, Daniel Moseguí González, Danielle Perszyk, Danny Hernandez, Danqi Chen, Daphne Ippolito, Dar Gilboa, David Dohan, David Drakard, David Jurgens, Debajyoti Datta, Deep Ganguli, Denis Emelin, Denis Kleyko, Deniz Yuret, Derek Chen, Derek Tam, Dieuwke Hupkes, Diganta Misra, Dilyar Buzan, Dimitri Coelho Mollo, Diyi Yang, Dong-Ho Lee, Dylan Schrader, Ekaterina Shutova, Ekin Dogus Cubuk, Elad Segal, Eleanor Hagerman, Elizabeth Barnes, Elizabeth Donoway, Ellie Pavlick, Emanuele Rodola, Emma Lam, Eric Chu, Eric Tang, Erkut Erdem, Ernie Chang, Ethan A. Chi, Ethan Dyer, Ethan Jerzak, Ethan Kim, Eunice Engefu Manyasi, Evgenii Zheltonozhskii, Fanyue Xia, Fatemeh Siar, Fernando Martínez-Plumed, Francesca Happé, Francois Chollet, Frieda Rong, Gaurav Mishra, Genta Indra Winata, Gerard de Melo, Germán Kruszewski, Giambattista Parascandolo, Giorgio Mariani, Gloria Wang, Gonzalo Jaimovitch-López, Gregor Betz, Guy Gur-Ari, Hana Galijasevic, Hannah Kim, Hannah Rashkin, Hannaneh Hajishirzi, Harsh Mehta, Hayden Bogar, Henry Shevlin, Hinrich Schütze, Hiromu Yakura, Hongming Zhang, Hugh Mee Wong, Ian Ng, Isaac Noble, Jaap Jumelet, Jack Geissinger, Jackson Kernion, Jacob Hilton, Jaehoon Lee, Jaime Fernández Fisac, James B. Simon, James Koppel, James Zheng, James Zou, Jan Kocoń, Jana Thompson, Janelle Wingfield, Jared Kaplan, Jarema Radom, Jascha Sohl-Dickstein, Jason Phang, Jason Wei, Jason Yosinski, Jekaterina Novikova, Jelle Bosscher, Jennifer Marsh, Jeremy Kim, Jeroen Taal, Jesse Engel, Jesujoba Alabi, Jiacheng Xu, Jiaming Song, Jillian Tang, Joan Waweru, John Burden, John Miller, John U. Balis, Jonathan Batchelder, Jonathan Berant, Jörg Frohberg, Jos Rozen, Jose Hernandez-Orallo, Joseph Boudeman, Joseph Guerr, Joseph Jones, Joshua B. Tenenbaum, Joshua S. Rule, Joyce Chua, Kamil Kanclerz, Karen Livescu, Karl Krauth, Karthik Gopalakrishnan, Katerina Ignatyeva, Katja Markert, Kaustubh D. Dhole, Kevin Gimpel, Kevin Omondi, Kory Mathewson, Kristen Chiafullo, Ksenia Shkaruta, Kumar Shridhar, Kyle McDonell, Kyle Richardson, Laria Reynolds, Leo Gao, Li Zhang, Liam Dugan, Lianhui Qin, Lidia Contreras-Ochando, Louis-Philippe Morency, Luca Moschella, Lucas Lam, Lucy Noble, Ludwig Schmidt, Luheng He, Luis Oliveros Colón, Luke Metz, Lütfi Kerem Şenel, Maarten Bosma, Maarten Sap, Maartje ter Hoeve, Maheen Farooqi, Manaal Faruqui, Mantas Mazeika, Marco Baturan, Marco Marelli, Marco Maru, Maria Jose Ramírez Quintana, Marie Tolkiehn, Mario Giulianelli, Martha Lewis, Martin Potthast, Matthew L. Leavitt, Matthias Hagen, Mátyás Schubert, Medina Orduna Baitemirova, Melody Arnaud, Melvin McElrath, Michael A. Yee, Michael Cohen, Michael Gu, Michael Ivanitskiy, Michael Starritt, Michael Strube, Michał Swędrowski, Michele Bevilacqua, Michihiro Yasunaga, Mihir Kale, Mike Cain, Mimee Xu, Mirac Suzgun, Mitch Walker, Mo Tiwari, Mohit Bansal, Moin Aminnaseri, Mor Geva, Mozhdeh Gheini, Mukund Varma T, Nanyun Peng, Nathan A. Chi, Nayeon Lee, Neta Gur-Ari Krakover, Nicholas Cameron, Nicholas Roberts, Nick Doiron, Nicole Martinez, Nikita Nangia, Niklas Deckers, Niklas Muennighoff, Nitish Shirish Keskar, Niveditha S. Iyer, Noah Constant, Noah Fiedel, Nuan Wen, Oliver Zhang, Omar Agha, Omar Elbaghdadi, Omer Levy, Owain Evans, Pablo Antonio Moreno Casares, Parth Doshi, Pascale Fung, Paul Pu Liang, Paul Vicol, Pegah Alipoormolabashi, Peiyuan Liao, Percy Liang, Peter Chang, Peter Eckersley, Phu Mon Htut, Pinyu Hwang, Piotr Miłkowski, Piyush Patil, Pouya Pezeshkpour, Priti Oli, Qiaozhu Mei, Qing Lyu, Qinlang Chen, Rabin Banjade, Rachel Etta Rudolph, Raefer Gabriel, Rahel Habacker, Ramon Risco, Raphaël Millière, Rhythm Garg, Richard Barnes, Rif A. Saurous, Riku Arakawa, Robbe Raymaekers, Robert Frank, Rohan Sikand, Roman Novak, Roman Sitelew, Ronan LeBras, Rosanne Liu, Rowan Jacobs, Rui Zhang, Ruslan Salakhutdinov, Ryan Chi, Ryan Lee, Ryan Stovall, Ryan Teehan, Rylan Yang, Sahib Singh, Saif M. Mohammad, Sajant Anand, Sam Dillavou, Sam Shleifer, Sam Wiseman, Samuel Gruetter, Samuel R. Bowman, Samuel S. Schoenholz, Sanghyun Han, Sanjeev Kwatra, Sarah A. Rous, Sarik Ghazarian, Sayan Ghosh, Sean Casey, Sebastian Bischoff, Sebastian Gehrmann, Sebastian Schuster, Sepideh Sadeghi, Shadi Hamdan, Sharon Zhou, Shashank Srivastava, Sherry Shi, Shikhar Singh, Shima Asaadi, Shixiang Shane Gu, Shubh Pachchigar, Shubham Toshniwal, Shyam Upadhyay, Shyamolima, Debnath, Siamak Shakeri, Simon Thormeyer, Simone Melzi, Siva Reddy, Sneha Priscilla Makini, Soo-Hwan Lee, Spencer Torene, Sriharsha Hatwar, Stanislas Dehaene, Stefan Divic, Stefano Ermon, Stella Biderman, Stephanie Lin, Stephen Prasad, Steven T. Piantadosi, Stuart M. Shieber, Summer Misherghi, Svetlana Kiritchenko, Swaroop Mishra, Tal Linzen, Tal Schuster, Tao Li, Tao Yu, Tariq Ali, Tatsu Hashimoto, Te-Lin Wu, Théo Desbordes, Theodore Rothschild, Thomas Phan, Tianle Wang, Tiberius Nkinyili, Timo Schick, Timofei Kornev, Titus Tunduny, Tobias Gerstenberg,

Trenton Chang, Trishala Neeraj, Tushar Khot, Tyler Shultz, Uri Shaham, Vedant Misra, Vera Demberg, Victoria Nyamai, Vikas Raunak, Vinay Ramasesh, Vinay Uday Prabhu, Vishakh Padmakumar, Vivek Srikumar, William Fedus, William Saunders, William Zhang, Wout Vossen, Xiang Ren, Xiaoyu Tong, Xinran Zhao, Xinyi Wu, Xudong Shen, Yadollah Yaghoobzadeh, Yair Lakretz, Yangqiu Song, Yasaman Bahri, Yejin Choi, Yichi Yang, Yiding Hao, Yifu Chen, Yonatan Belinkov, Yu Hou, Yufang Hou, Yuntao Bai, Zachary Seid, Zhuoye Zhao, Zijian Wang, Zijie J. Wang, Zirui Wang, and Ziyi Wu. Beyond the imitation game: Quantifying and extrapolating the capabilities of language models, 2023.

[150] Benn Stancil. ADE-bench: A framework for evaluating ai agents on data analyst tasks, 2025.

[151] Zafir Stojanovski, Oliver Stanley, Joe Sharratt, Richard Jones, Abdulhakeem Adefioye, Jean Kaddour, and Andreas Köpf. Reasoning gym: Reasoning environments for reinforcement learning with verifiable rewards. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems Datasets and Benchmarks Track*, 2025.

[152] Benedikt Stroebl, Sayash Kapoor, and Arvind Narayanan. The limits of inference scaling through resampling, 2024. arXiv:2411.17501.

[153] Mirac Suzgun, Nathan Scales, Nathanael Schärli, Sebastian Gehrmann, Yi Tay, Hyung Won Chung, Aakanksha Chowdhery, Quoc Le, Ed Chi, Denny Zhou, and Jason Wei. Challenging BIG-bench tasks and whether chain-of-thought can solve them. In Anna Rogers, Jordan Boyd-Graber, and Naoaki Okazaki, editors, *Findings of the Association for Computational Linguistics: ACL 2023*, pages 13003–13051, Toronto, Canada, July 2023. Association for Computational Linguistics.

[154] Yuheng Tang, Kaijie Zhu, Bonan Ruan, Chuqi Zhang, Michael Yang, Hongwei Li, Suyue Guo, Tianneng Shi, Zekun Li, Christopher Kruegel, Giovanni Vigna, Dawn Song, William Yang Wang, Lun Wang, Yangruibo Ding, Zhenkai Liang, and Wenbo Guo. Devops-gym: Benchmarking AI agents in software devops cycle. In *The Fourteenth International Conference on Learning Representations*, 2026.

[155] Artificial Analysis Team. Artificial analysis long context reasoning benchmark(lcr), 2025.

[156] Nandan Thakur, Nils Reimers, Andreas Rücklé, Abhishek Srivastava, and Iryna Gurevych. BEIR: A heterogeneous benchmark for zero-shot evaluation of information retrieval models. In *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 2)*, 2021.

[157] Minyang Tian, Luyu Gao, Dylan Zhang, Xinan Chen, Cunwei Fan, Xuefei Guo, Roland Haas, Pan Ji, Kittithat Krongchon, Yao Li, Shengyan Liu, Di Luo, Yutao Ma, HAO TONG, Kha Trinh, Chenyu Tian, Zihan Wang, Bohao Wu, Shengzhu Yin, Minhui Zhu, Kilian Lieret, Yanxin Lu, Genglin Liu, Yufeng Du, Tianhua Tao, Ofir Press, Jamie Callan, Eliu A Huerta, and Hao Peng. Scicode: A research coding benchmark curated by scientists. In *The Thirty-eight Conference on Neural Information Processing Systems Datasets and Benchmarks Track*, 2024.

[158] Transluce. Docent. https://transluce.org/docent, 2026.

[159] Harsh Trivedi, Tushar Khot, Mareike Hartmann, Ruskin Manku, Vinty Dong, Edward Li, Shashank Gupta, Ashish Sabharwal, and Niranjan Balasubramanian. AppWorld: A controllable world of apps and people for benchmarking interactive coding agents. In Lun-Wei Ku, Andre Martins, and Vivek Srikumar, editors, *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 16022–16076, Bangkok, Thailand, August 2024. Association for Computational Linguistics.

[160] UK AI Security Institute. Inspect: A framework for large language model evaluations, 2024.

[161] UniPat AI. UniPat AI. https://unipat.ai/, 2026.

[162] Vals AI. Finance-Agent: A tool for financial research and analysis, 2024.

[163] Rajan Vivek, Kawin Ethayarajh, Diyi Yang, and Douwe Kiela. Anchor points: Benchmarking models with much fewer examples, 2024. EACL 2024, arXiv:2309.08638.

[164] Alex Wang, Yada Pruksachatkun, Nikita Nangia, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel Bowman. Superglue: A stickier benchmark for general-purpose language understanding systems. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019.

[165] Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R. Bowman. GLUE: A multi-task benchmark and analysis platform for natural language understanding. In *International Conference on Learning Representations*, 2019.

[166] Ruoyao Wang, Peter Jansen, Marc-Alexandre Côté, and Prithviraj Ammanabrolu. Scienceworld: Is your agent smarter than a 5th grader?, 2022.

[167] Xingyao Wang, Boxuan Li, Yufan Song, Frank F. Xu, Xiangru Tang, Mingchen Zhuge, Jiayi Pan, Yueqi Song, Bowen Li, Jaskirat Singh, Hoang H. Tran, Fuqiang Li, Ren Ma, Mingzhang Zheng, Bill Qian, Yanjun Shao, Niklas Muennighoff, Yizhe Zhang, Binyuan Hui, Junyang Lin, Robert Brennan, Hao Peng, Heng Ji, and Graham Neubig. OpenHands: An open platform for AI software developers as generalist agents, 2025. ICLR 2025, arXiv:2407.16741.

[168] You Wang, Michael Pradel, and Zhongxin Liu. Are "solved issues" in SWE-bench really solved correctly? an empirical study, 2025. arXiv:2503.15223.

[169] Zhun Wang, Tianneng Shi, Jingxuan He, Matthew Cai, Jialin Zhang, and Dawn Song. Cybergym: Evaluating AI agents' real-world cybersecurity capabilities at scale. In *The Fourteenth International Conference on Learning Representations*, 2026.

[170] Anjiang Wei, Yuheng Wu, Yingjia Wan, Tarun Suresh, Huanmi Tan, Zhanke Zhou, Sanmi Koyejo, Ke Wang, and Alex Aiken. Satbench: Benchmarking llms' logical reasoning via automated puzzle generation from sat formulas, 2025.

[171] Jason Wei, Nguyen Karina, Hyung Won Chung, Yunxin Joy Jiao, Spencer Papay, Amelia Glaese, John Schulman, and William Fedus. Measuring short-form factuality in large language models, 2024.

[172] Colin White, Samuel Dooley, Manley Roberts, Arka Pal, Benjamin Feuer, Siddhartha Jain, Ravid Shwartz-Ziv, Neel Jain, Khalid Saifullah, Sreemanti Dey, Shubh-Agrawal, Sandeep Singh Sandha, Siddartha Venkat Naidu, Chinmay Hegde, Yann LeCun, Tom Goldstein, Willie Neiswanger, and Micah Goldblum. Livebench: A challenging, contamination-limited LLM benchmark. In *The Thirteenth International Conference on Learning Representations*, 2025.

[173] Ryan Wong, Jiawei Wang, Junjie zhao, Li Chen, Yan Gao, Zhanglong, Xuan Zhou, Zuo Wang, Kai Xiang, Ge Zhang, Wenhao Huang, Yang Wang, and WANG KE. Widesearch: Benchmarking agentic broad info-seeking. In *The Fourteenth International Conference on Learning Representations*, 2026.

[174] Zhiheng Xi, Yiwen Ding, Wenxiang Chen, Boyang Hong, Honglin Guo, Junzhe Wang, Dingwen Yang, Chenyang Liao, Xin Guo, Wei He, Songyang Gao, Lu Chen, Rui Zheng, Yicheng Zou, Tao Gui, Qi Zhang, Xipeng Qiu, Xuanjing Huang, Zuxuan Wu, and Yu-Gang Jiang. Agentgym: Evolving large language model-based agents across diverse environments, 2024.

[175] Chunqiu Steven Xia, Yinlin Deng, Soren Dunn, and Lingming Zhang. Agentless: Demystifying LLM-based software engineering agents, 2024. arXiv:2407.01489.

[176] Chunqiu Steven Xia, Yinlin Deng, and LINGMING ZHANG. Top leaderboard ranking = top coding proficiency, always? evoeval: Evolving coding benchmarks via LLM. In *First Conference on Language Modeling*, 2024.

[177] Xiaomi. MiMo platform pricing. https://platform.xiaomimimo.com/docs/pricing, 2026. Accessed: 2026-04-22; covers both MiMo-V2-Pro and MiMo-V2.5-Pro.

[178] Qianqian Xie, Weiguang Han, Xiao Zhang, Yanzhao Lai, Min Peng, Alejandro Lopez-Lira, and Jimin Huang. PIXIU: A comprehensive benchmark, instruction dataset and large language model for finance. In *Thirty-seventh Conference on Neural Information Processing Systems Datasets and Benchmarks Track*, 2023.

[179] 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 *The Thirty-eight Conference on Neural Information Processing Systems Datasets and Benchmarks Track*, 2024.

[180] Frank F. Xu, Yufan Song, Boxuan Li, Yuxuan Tang, Kritanjali Jain, Mengxue Bao, Zora Zhiruo Wang, Xuhui Zhou, Zhitong Guo, Murong Cao, Mingyang Yang, Hao Yang Lu, Amaad Martin, Zhe Su, Leander Melroy Maben, Raj Mehta, Wayne Chi, Lawrence Keunho Jang, Yiqing Xie, Shuyan Zhou, and Graham Neubig. Theagentcompany: Benchmarking LLM agents on consequential real world tasks. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems Datasets and Benchmarks Track*, 2025.

[181] Jie Yang, Honglin Guo, Li Ji, Jiazheng Zhou, Rui Zheng, Zhikai Lei, Shuo Zhang, Zhiheng Xi, Shichun Liu, Yuxin Wang, Bo Wang, Yining Zheng, Tao Gui, and Xipeng Qiu. Abc-bench: Benchmarking agentic backend coding in real-world development, 2026.

[182] John Yang, Carlos E Jimenez, Alexander Wettig, Kilian Lieret, Shunyu Yao, Karthik R Narasimhan, and Ofir Press. SWE-agent: Agent-computer interfaces enable automated software engineering. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024.

[183] John Yang, Kilian Lieret, Carlos E Jimenez, Alexander Wettig, Kabir Khandpur, Yanzhe Zhang, Binyuan Hui, Ofir Press, Ludwig Schmidt, and Diyi Yang. SWE-smith: Scaling data for software engineering agents. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems Datasets and Benchmarks Track*, 2025.

[184] John Yang, Kilian Lieret, Jeffrey Ma, Parth Thakkar, Dmitrii Pedchenko, Sten Sootla, Emily McMilin, Pengcheng Yin, Rui Hou, Gabriel Synnaeve, Diyi Yang, and Ofir Press. Programbench: Can language models rebuild programs from scratch?, 2026.

[185] Rui Yang, Ziruo Wang, Yuntian Gu, Yitao Liang, and Tongyang Li. QCircuitbench: A large-scale dataset for benchmarking quantum algorithm design. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems Datasets and Benchmarks Track*, 2025.

[186] Shunyu Yao, Howard Chen, John Yang, and Karthik Narasimhan. Webshop: Towards scalable real-world web interaction with grounded language agents. In S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh, editors, *Advances in Neural Information Processing Systems*, volume 35, pages 20744–20757. Curran Associates, Inc., 2022.

[187] Shunyu Yao, Noah Shinn, Pedram Razavi, and Karthik R Narasimhan. τ-bench: A benchmark for Tool-Agent-User interaction in real-world domains. In *The Thirteenth International Conference on Learning Representations*, 2025.

[188] Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. ReAct: Synergizing reasoning and acting in language models, 2023. ICLR 2023, arXiv:2210.03629.

[189] Gregory Yauney, Shahzaib Saqib Warraich, and Swabha Swayamdipta. How reliable is language model Micro-Benchmarking?, 2026. ICLR 2026, arXiv:2510.08730.

[190] Christine Ye, Sihan Yuan, Suchetha Cooray, Steven Dillmann, Ian L. V. Roque, Dalya Baron, Philipp Frank, Sergio Martin-Alvarez, Nolan Koblischke, Frank J Qu, Diyi Yang, Risa Wechsler, and Ioana Ciucă. Replicationbench: Can AI agents replicate astrophysics research papers?, 2026.

[191] Yixin Ye, Yang Xiao, Tiantian Mi, and Pengfei Liu. Aime-preview: A rigorous and immediate evaluation framework for advanced mathematical reasoning. https://github.com/GAIR-NLP/AIME-Preview, 2025. GitHub repository.

[192] Boxi Yu, Yuxuan Zhu, Pinjia He, and Daniel Kang. UTBoost: Rigorous evaluation of coding agents on SWE-Bench, 2025. ACL 2025, arXiv:2506.09289.

[193] Xingdi Yuan, Morgane M Moss, Charbel El Feghali, Chinmay Singh, Darya Moldavskaya, Drew MacPhee, Lucas Caccia, Matheus Pereira, Minseon Kim, Alessandro Sordoni, and Marc-Alexandre Côté. debug-gym: A Text-Based environment for interactive debugging, 2025. arXiv:2503.21557.

[194] Z.AI. Z.AI API pricing overview. https://docs.z.ai/guides/overview/pricing, 2026. Accessed: 2026-04-22.

[195] Amir Zamir, Alexander Sax, William Shen, Leonidas Guibas, Jitendra Malik, and Silvio Savarese. Taskonomy: Disentangling task transfer learning, 2018.

[196] Daoguang Zan, Zhirong Huang, Wei Liu, Hanwu Chen, Shulin Xin, Linhao Zhang, Qi Liu, Aoyan Li, Lu Chen, Xiaojian Zhong, Siyao Liu, Yongsheng Xiao, Liangqiang Chen, Yuyu Zhang, Jing Su, Tianyu Liu, RUI LONG, Ming Ding, and liang xiang. Multi-SWE-bench: A multilingual benchmark for issue resolving. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems Datasets and Benchmarks Track*, 2025.

[197] Yuchen Zeng and Dimitris Papailiopoulos. You don't need to run every eval, 2026. arXiv:2606.24020. Code: https://github.com/anadim/llm-benchmark-matrix.

[198] Xiaohua Zhai, Joan Puigcerver, Alexander Kolesnikov, Pierre Ruyssen, Carlos Riquelme, Mario Lucic, Josip Djolonga, Andre Susano Pinto, Maxim Neumann, Alexey Dosovitskiy, Lucas Beyer, Olivier Bachem, Michael Tschannen, Marcin Michalski, Olivier Bousquet, Sylvain Gelly, and Neil Houlsby. A large-scale study of representation learning with the visual task adaptation benchmark, 2020.

[199] Guanhua Zhang, Florian E. Dorner, and Moritz Hardt. How benchmark prediction from fewer data misses the mark, 2025. arXiv:2506.07673.

[200] Zicheng Zhang, Xiangyu Zhao, Xinyu Fang, Chunyi Li, Xiaohong Liu, Xiongkuo Min, Haodong Duan, Kai Chen, and Guangtao Zhai. Redundancy principles for mllms benchmarks, 2025.

[201] Yujie Zhao, Boqin Yuan, Junbo Huang, Haocheng Yuan, Zhongming Yu, Haozhou Xu, Lanxiang Hu, Abhilash Shankarampeta, Zimeng Huang, Wentao Ni, Yuandong Tian, and Jishen Zhao. Ama-bench: Evaluating long-horizon memory for agentic applications. In *Proceedings of the 43rd International Conference on Machine Learning (ICML)*, 2026.

[202] Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric Xing, Hao Zhang, Joseph E. Gonzalez, and Ion Stoica. Judging LLM-as-a-judge with MT-bench and chatbot arena. In *Thirty-seventh Conference on Neural Information Processing Systems Datasets and Benchmarks Track*, 2023.

[203] Xinyue Zheng, Haowei Lin, Shaofei Cai, Yaodong Yang, Zilong Zheng, and Yitao Liang. Unicode: Augmenting evaluation for code reasoning. In *Proceedings of the 43rd International Conference on Machine Learning (ICML)*, 2026.

[204] Changzhi Zhou, Ao Liu, Yuchi Deng, Zhiying Zeng, Tao Zhang, Haotian Zhu, Jianwei Cai, Yue Mao, Chenchen Zhang, Lingyun Tan, ZiyanXU, Bohui Zhai, HengyiLIu, Speed Zhu, Wiggin Zhou, and Fengzong Lian. Autocodebench: Large language models are automatic code benchmark generators. In *The Fourteenth International Conference on Learning Representations*, 2026.

[205] Hongli Zhou, Hui Huang, Ziqing Zhao, Lvyuan Han, Huicheng Wang, Kehai Chen, Muyun Yang, Wei Bao, Jian Dong, Bing Xu, Conghui Zhu, Hailong Cao, and Tiejun Zhao. Lost in benchmarks? rethinking large language model benchmarking with item response theory, 2026. AAAI 2026, arXiv:2505.15055.

[206] Qixing Zhou, Jiacheng Zhang, Haiyang Wang, Rui Hao, Jiahe Wang, Minghao Han, Yuxue Yang, Shuzhe Wu, Feiyang Pan, Lue Fan, et al. Featurebench: Benchmarking agentic coding for complex feature development. *arXiv preprint arXiv:2602.10975*, 2026.

[207] Shuyan Zhou, Frank F. Xu, Hao Zhu, Xuhui Zhou, Robert Lo, Abishek Sridhar, Xianyi Cheng, Tianyue Ou, Yonatan Bisk, Daniel Fried, Uri Alon, and Graham Neubig. Webarena: A realistic web environment for building autonomous agents, 2024.

[208] Yuxuan Zhu, Tengjun Jin, Yada Pruksachatkun, Andy Zhang, Shu Liu, Sasha Cui, Sayash Kapoor, Shayne Longpre, Kevin Meng, Rebecca Weiss, Fazl Barez, Rahul Gupta, Jwala Dhamala, Jacob Merizian, Mario Giulianelli, Harry Coppock, Cozmin Ududec, Jasjeet Sekhon, Jacob Steinhardt, Antony Kellermann, Sarah Schwettmann, Matei Zaharia, Ion Stoica, Percy Liang, and Daniel Kang. Establishing best practices for building rigorous agentic benchmarks, 2025. arXiv:2507.02825.

[209] Mingchen Zhuge, Changsheng Zhao, Dylan Ashley, Wenyi Wang, Dmitrii Khizbullin, Yunyang Xiong, Zechun Liu, Ernie Chang, Raghuraman Krishnamoorthi, Yuandong Tian, Yangyang Shi, Vikas Chandra, and Jürgen Schmidhuber. Agent-as-a-Judge: Evaluate agents with agents, 2024. arXiv:2410.10934.

[210] Terry Yue Zhuo, Vu Minh Chien, Jenny Chim, Han Hu, Wenhao Yu, Ratnadira Widyasari, Imam Nur Bani Yusuf, Haolan Zhan, Junda He, Indraneil Paul, Simon Brunner, Chen GONG, James Hoang, Armel Randy Zebaze, Xiaoheng Hong, Wen-Ding Li, Jean Kaddour, Ming Xu, Zhihan Zhang, Prateek Yadav, Naman Jain, Alex Gu, Zhoujun Cheng, Jiawei Liu, Qian Liu, Zijian Wang, David Lo, Binyuan Hui, Niklas Muennighoff, Daniel Fried, Xiaoning Du, Harm de Vries, and Leandro Von Werra. Bigcodebench: Benchmarking code generation with diverse function calls and complex instructions. In *The Thirteenth International Conference on Learning Representations*, 2025.

Appendix Table of Contents

A Authors **29**

A.1 Main Contributors & Affiliations 29

A.2 Additional Contributors 30

B Acknowledgment **30**

C Extended Related Work **30**

D Details for Harbor Adapters **32**

D.1 Motivation 33

D.2 Agent Interaction Modes 33

D.3 Adapter Construction Workflow 33

D.3.1 Creation 33

D.3.2 Review 34

D.3.3 Standardization 34

D.4 Parity Experiments 34

D.4.1 Challenges in Benchmark Portability 34

D.4.2 Design and Evaluation Metrics 34

D.4.3 Agent Integration Scenarios 34

D.4.4 Parity Set Selection 35

D.5 Adapter Catalog 35

D.6 Per-Benchmark Details 35

D.6.1 Software Engineering 36

D.6.2 Mathematics & Reasoning 38

D.6.3 Knowledge & Long Context 39

D.6.4 Scientific Research 40

D.6.5 Agents, Tools & Systems 41

D.6.6 Data & Analytics 42

D.6.7 Professional Domains 42

D.6.8 Safety & Security 43

D.6.9 Multimodal 43

D.6.10 Benchmarks Using Harbor Format 43

D.6.11 Other Benchmarks Not Included 43

D.7 Benchmark Issues and Lessons Learned 44

E Evaluation Protocol and Reproducibility Details **46**

E.1 Evaluation Infrastructure 46

E.2 Dataset Preparation and Sampling 48

E.3 Error Handling and Quality Control 48

E.4 Harbor-Index Experiment Protocol . . . 49
E.5 Evaluation Results . . . 49

F Detailed Large-scale Quantitative Analysis **49**
F.1 Benchmark and Task Predictability Analysis . . . 50
F.1.1 Low-rank structure for benchmark space . . . 50
F.1.2 Benchmark predictability. . . . 50
F.1.3 How many benchmarks span the evaluation space? . . . 52
F.1.4 Does the same redundancy hold at the task level? . . . 52
F.2 Model vs. Harness Effect: Methodology . . . 54
F.3 Original benchmark data collections and mapping . . . 55
F.4 Performance Progress Over Time . . . 56
F.5 Latency Study . . . 58

G Case Analysis **59**
G.1 Task Failure . . . 59
G.2 Agent Failure Modes . . . 60
G.2.1 Methodology . . . 60
G.2.2 Agent Failure Mode Full Results . . . 70

H Harbor-Index Selection Pipeline **71**
H.1 Sampling Protocol . . . 71
H.2 Task Quality Bar . . . 71
H.3 Task Audit Process . . . 72
H.4 Human Review Protocol . . . 74
H.5 Audit-and-Fix Loop . . . 75

I Limitations **75**

J Broader Impacts **75**

A Authors

Each contributor below is listed with their full affiliation (superscript), keyed to the numbered institution list at the end of this section.

A.1 Main Contributors & Affiliations

Organization & Execution Team

Lin Shi[1], Haowei Lin[2], Zixuan Zhu[3], Xiaoyue Zhou[1], Xiang Li[4], Xiangning Lin[5], Yaxuan Deng[1], Han Xu[6], Yuangang Li[7], Shanda Li[5], Zizhao Chen[1], Hanwen Xing[8], Harsh Raj[9], Bo Chen[10], Quan Shi[11], Steven Dillmann[4], Yipeng Gao[8], Puneesh Khanna[12], Ruofan Lu[13], Chao Beyond Zhou[14], Michael Yang[15], Robert Zhang[16], Siyuan Chai[6], Jiayu Chang[4], Yizhao Chen[17], Xiaokun Chen[4], Yiwei Dai[18], Wenting Yang[14], Hange Liu[14], Minghao Liu[19], Zihan Wang[19]

Advisory Committee

Andy Konwinski[21], Boxuan Li[22], Leon Liangyu Chen[4], Alex Dimakis[23], Nicholas Carlini[20], Soroush Vosoughi[24], Sanmi Koyejo[4], Di He[2], Etash Guha[4], Benjamin Feuer[4], Mike Merrill[20], Ludwig Schmidt[4], Alex Shaw[21]

Adapter Contributors

Adnan El Assadi[25], Benedikt Stroebl[26], E. Kelly Buchanan[4], Han Meng[27], Junwei He[28], Longxuan Yu[29], Radin Shayanfar[30], Yukyung Lee[31], Zhikang Dong[32], Allen G Hart[33], Anjiang Wei[4], Anurag Kashyap[34], Arpandeep Khatua[4], Audrey Jixin Zheng[35], Chengrui Ma[24], David Heineman[36], Dubing Chen[37], Hai-Anh Trinh[14], Haishuo Fang[38], Hefan Zhang[24], Hui Shen[39], Issa Sugiura[40], Jiankai Sun[4], Jiechao Gao[4], Junhong Lin[41], Junnan Li[42], Kai Yang[39], Lei Hsiung[24], Maoyu Wang[14], Mengze Tang[42], Nabil Omi[43], Negin Raoof[23], Nicholas Edwards[44], Octavia Guo[14], Orfeas Menis Mastromichalakis[45], Pengliang Ji[5], Przemysław Hejman[46], Qi Qi[47], Qunshu Lin[19], Richard Zhuang[4], Rui Yang[2], Ruichen Zheng[24], Ryan Marten[48], Shaghayegh Fazliani[4], Shizheng Hou[49], Sicong Jiang[19], Sijie Li[2], Song Bian[42], Terry Yue Zhuo[50], Tianqing Wu[14], Tom Tang[19], Wanjia Zhao[4], Weihao Xuan[51,19], Wenhua Liang[24], Xian Liu[14], Xin Lan[52], Xuan Zhang[19], Xuandong Zhao[23], Yanchuan Tang[25], Yifan Jiang[8], Yijiang Li[17], Yitong Guan[19,53], Yizhi Li[54], Yonghui Liu[55], Yuheng Tang[15], Yujun (Audrey) Mao[31], Yunfei Zhao[19,4], Yuxin Wang[24], Yuxuan Tang[14], Zhenheng Tang[56], Zhifei Li[26,57], Ziruo Wang[2], Ziyu She[58], Kaiyuan Liu[43], Iheb Chaabane[12], Yuxin Tang[59], Xiangyi Li[60], Satya Sai Srinath Namburi GNVV[42], Xinyue Zheng[2], Boqin Yuan[17], Michael Glass[61]

Affiliations

1. Cornell Tech
2. Peking University
3. Nanyang Technological University
4. Stanford University
5. Carnegie Mellon University
6. University of Illinois Urbana-Champaign
7. University of California, Irvine
8. University of Southern California
9. Northeastern University
10. University of Hong Kong
11. International School of Krakow
12. Technology Innovation Institute
13. The Chinese University of Hong Kong
14. Independent Contributor
15. University of California, Santa Barbara
16. University of Texas at Austin
17. University of California, San Diego
18. Cornell University
19. 2077AI
20. Anthropic
21. Laude Institute
22. Microsoft
23. University of California, Berkeley
24. Dartmouth College
25. Harvard University
26. Princeton University
27. College of William and Mary
28. ByteDance
29. University of California, Riverside
30. Queen's University
31. Boston University
32. Stony Brook University
33. University of Warwick
34. Amazon
35. CoreWeave, Inc.
36. Allen Institute for AI
37. University of Macau
38. TU Darmstadt
39. University of Michigan
40. Institute of Science Tokyo
41. Massachusetts Institute of Technology
42. University of Wisconsin-Madison

43. University of Washington
44. University of Vienna
45. Instituto de Telecomunicações, Lisbon
46. Quesma
47. Meta
48. HarborCo
49. National University of Singapore
50. Monash University
51. The University of Tokyo
52. Michigan State University
53. Zhejiang University
54. University of Manchester
55. Australian National University
56. The Hong Kong University of Science and Technology
57. Renmin University of China
58. University of Basel
59. Rice University
60. BenchFlow
61. IBM

A.2 Additional Contributors

We also thank the following contributors for help with adapter development, review, and evaluation:

- Zihao Wang (Peking University)
- Joan Cabezas (Independent Contributor)
- Thomas Aubry (ooakdata)
- Bowen Xing (Independent Contributor)
- Qiuyang Mang (University of California, Berkeley)
- Shuting Zhao (University of California, San Diego)
- Xinyu Yao (Rice University)
- Achintya Paningapalli (University of California, Berkeley)
- Benjamin Calvert (Brigham Young University)
- Xinyu Lu (Institute of Software, Chinese Academy of Sciences)
- Zhaowei Xu (Independent Contributor)

B Acknowledgment

The experiments and analyses of this study is generously supported by a series of partnership companies, frontier AI labs, and cloud sandbox providers:

Partnership Companies: Laude Institute [77], 2077AI [1], Docent [158], UniPat AI [161]

Frontier AI Labs: OpenAI (GPT) [117], Anthropic (Claude) [6], Google DeepMind (Gemini) [49], Alibaba (Qwen) [4], DeepSeek (Deepseek) [31], MoonShot.AI (Kimi) [110], Xiaomi (Mimo) [177], and Z.AI (GLM) [194]

Sandbox Providers: Daytona [28] (for all CPU-only tasks), Modal [108] (for GPU-required tasks specifically)

C Extended Related Work

This appendix extends Section 5. It adds two threads the main text omits for space, namely the aggregate-evaluation tradition that precedes agentic benchmarks and safety and robustness evaluation, and expands the main section's clause-level citation groups into fuller discussions of prior work.

Aggregate and holistic evaluation. GLUE and SuperGLUE established standardized multi-task evaluation in language [165, 164]. Later suites broadened scope to knowledge and reasoning [55, 149, 153] and multilinguality [57], and advanced methodology through multi-metric scenario reporting [85], preference-based comparison [22], and contamination-aware testing [172]. Analogous efforts span retrieval and representation learning [156, 112] and vision [138, 198, 72, 8, 25, 52, 195], while Dynabench and Dynaboard replace fixed test sets with evolving, hosted comparisons [69, 99].

Evaluation infrastructure and reproducibility. Reproducibility work documents how prompt templates, decoding choices, few-shot formatting, and metric definitions materially change reported comparisons [11]; HELM adds multi-metric, scenario-level transparency [85] and Dynaboard evaluates submitted models in a hosted setting to reduce reliance on self-reported results [99]. Recent systems extend this to model-plus-scaffold evaluation: Inspect [160], HAL [65], AstaBench [12], and Harbor [53], a containerized task and sandbox runtime. A parallel effort packages task or agent families behind one programmatic interface, including agentic environment simulators [93], interactive debugging environments [193], generalist software-agent platforms [167],

and reproducible search sandboxes that replace commercial APIs whose drift silently breaks comparability [24].

Agent benchmarks make reproducibility harder. A static task reduces to examples, prompts, and labels; an agent benchmark also carries an environment (browser state, files, virtual services, shell tools), an execution model (sandboxing, network controls, stochastic dynamics), and a scoring layer (hidden tests, trajectory judgments), each of which shifts correctness and cost between runs. Reporting practice compounds this: evaluations often conflate the needs of model developers with those of downstream developers, omit cost, and lack adequate holdout sets, which invites benchmark-specific shortcuts [66].

Tool and API use. Tool-use benchmarks test whether models select APIs, generate valid arguments, follow protocols, and compose calls. Each targets a different layer: API retrieval and selection [122, 82], parameter prediction and function-call accuracy [121], multi-step invocation [131, 141], policy following [187], and workflow planning [130, 48]. Each therefore fixes one abstraction boundary, whether a function call, a tool trajectory, or a simulated service policy, and evaluates models against it.

Interactive and embodied environments. Early work placed agents in web interfaces, where sparse reward made exploration the bottleneck [90], and in text games [26], embodied households [146], simulated science labs [166], and online shopping [186]. AgentBench, GAIA, and AgentGym then evaluated multi-step reasoning, planning, and tool use across broader collections [91, 104, 174]. Web benchmarks cover live synthetic sites [207, 71], real-world navigation [34, 54], enterprise workflows [36], and shared execution frameworks [29, 97]. Beyond the browser, AppWorld, AndroidWorld, and OSWorld reach application and operating-system control [159, 134, 179], now a large enough class to be surveyed separately [58].

These works show that agent performance depends on environment grounding, state tracking, error recovery, and long-horizon planning. They also explain why agent benchmarks resist direct comparison: environments differ in action granularity, observation modality, allowed tools, termination criteria, and scoring.

Code, scientific, and workplace agents. HumanEval tests program synthesis in self-contained functions [20], while SWE-bench raises the bar to issue resolution in real repositories [64]. SWE-agent shows the agent–computer interface itself substantially affects software-engineering performance [182], building on ReAct's interleaved reasoning-and-acting pattern [188] and the self-reflection variants that followed [144]; Agentless conversely shows a much simpler pipeline can be competitive, so scaffold complexity is not monotonically useful [175]. Terminal-Bench extends this line to shells, files, and long-running processes rather than a structured patch interface [103], and time-horizon studies quantify how the task length agents can complete has grown [74]. Beyond software engineering, MLE-bench evaluates agents on 75 Kaggle machine-learning engineering competitions against human leaderboard baselines [16]. ScienceAgentBench deliberately decomposes data-driven discovery into 102 expert-validated tasks from peer-reviewed publications, scoring the generated program, its execution results, and its cost [21]. TheAgentCompany combines browsing, coding, communication, and service manipulation inside a simulated organization [180].

These benchmarks expose failure modes invisible in static QA. The modes fall into three classes: environment (flaky dependencies, setup failures, resource limits, timeouts), scoring (hidden-test mismatch, partial credit, trajectory judgments), and scaffold (tool affordances, retry policies, interface conventions). Each class shifts reported scores independently of the model under test.

Safety, robustness, and adversarial evaluation. Tool-using agents take actions, read untrusted content, and execute multi-step plans whose intermediate states are themselves attack surfaces. AgentDojo evaluates prompt-injection attacks and defenses over realistic tasks and untrusted data [30], AgentHarm measures whether agents refuse malicious requests while retaining capability [5], and OS-Harm extends this to computer-use agents acting on a graphical interface [73]. A parallel line targets the reliability of agent evaluation itself: StableToolBench stabilizes tool calls through virtualized APIs and cached responses [51], while AgentRewardBench and Agent-as-a-Judge ask whether model judges can assess behavior beyond final-answer correctness [95, 209].

Benchmark validity and task quality. A distinct literature asks not how to run benchmarks but whether their scores license the conclusions drawn from them. Position and methodology papers make three arguments: that a few influential benchmarks should not stand in for general capability [133]; that benchmark construction should state its measurement assumptions explicitly and testably [92]; and that moving from a score to a claim about an abstract task requires further assumptions made explicit [44]. Empirically, a review of 445 benchmarks by 29 expert reviewers finds recurring patterns in measured phenomena, task design, and scoring metrics that undermine validity [9], and an assessment framework applied across widely used benchmarks reaches similar conclusions about documentation and design practice [136]. Concrete threats are well documented. Test-set label errors are pervasive, at least 3.3% on average across ten widely used datasets, and destabilize conclusions to the point that lower-capacity models can overtake higher-capacity ones once mislabeling is accounted for [116]; annotation errors change how aggregates should be read [47, 115]; contamination compromises held-out validity and is hard to measure per benchmark [139]; and leaderboard mechanics distort rankings [147].

For agents the verifier is executable code rather than a stored label, which turns these validity questions into engineering defects. Because resampling cannot lower a verifier's false-positive rate, that rate upper-bounds the accuracy any amount of resampling can attain, regardless of compute budget [152]. The Agentic Benchmark Checklist documents systematic task-setup and reward-design flaws in widely used agentic benchmarks, such as insufficient test cases in SWE-bench Verified and empty responses scored as success in τ-bench, with distortions reaching 100% in relative terms [208]. Software-engineering audits make the magnitude concrete: roughly a third of successful SWE-bench patches trace to solution leakage in the issue text, and a comparable share to test suites too weak to detect an incorrect patch [3]. Differential patch testing shows plausible patches diverging behaviorally from developer ground truth, inflating reported resolution rates [168]; augmenting test suites relabels hundreds of patches and reorders leaderboards [192]; and memorization rather than reasoning on SWE-bench instances complicates held-out interpretation [86]. The judging layer is a matching concern, whether the judge is a preference model [202], a trajectory judge [95, 209], or a stand-in for human annotators, a substitution for which no standard procedure existed until the recently proposed alternative annotator test [14].

Meta-evaluation, efficiency, and benchmark compression. Much of a large suite is redundant. Factor-analytic work finds model-by-benchmark score matrices are intrinsically low-rank, a few interpretable latent dimensions explaining most variance and implying substantial redundancy among nominally distinct tasks [13, 101, 197]; the same structure makes downstream performance predictable from a few latent capability measures [137], and correlation-based analyses of multimodal suites report comparable redundancy across benchmarks and capability dimensions [200]. This licenses cheaper evaluation through curated and distilled subsets [127, 70, 163], psychometrically adaptive item selection [56, 205], principled metric selection [129], and label-efficient active testing [10]. Cost-aware framings make the tradeoff explicit rather than incidental [41, 66].

Recent results delimit these methods. Benchmark agreement testing has no standardized procedure, and overlooked methodological choices, including which models the agreement is computed over, significantly change its conclusions, so apparent redundancy is partly an artifact of how it is measured [124]. Benchmark prediction from a subset depends on similarity to previously observed models and degrades sharply when extrapolating to stronger ones, failing exactly at the frontier where evaluation is most needed [199]. Micro-benchmarks frequently cannot rank nearby models reliably, requiring far more items than the handful sometimes suggested, at which point random sampling is competitive with sophisticated selection [189]. Small evaluations therefore require explicit uncertainty quantification [105], on top of the setup sensitivity documented above.

D Details for Harbor Adapters

This appendix describes the motivation for Harbor Adapters (§D.1), the agent interaction modes we use to categorize benchmarks (§D.2), the adapter construction workflow (§D.3), the parity experiment protocol (§D.4), the full adapter catalog (§D.5), per-benchmark adaptation and parity details (§D.6), and lessons learned from building adapters at scale (§D.7).

D.1 Motivation

As the number of benchmark datasets and agents grows, agent evaluation has become increasingly costly: supporting m benchmarks with n agents naively requires $\mathcal{O}(mn)$ effort, since each benchmark-agent pair involves its own harness integration, environment plumbing, and evaluation logic. Inconsistent environments and task-design issues further compound this cost, and the resulting leaderboards are often not directly comparable because instructions, tools, and execution configurations differ across benchmarks.

Despite the diversity of task formats, most benchmarks share a common underlying abstraction: **instruction**, **environment**, **tests**, and **solution**. The instruction specifies what the agent must accomplish; the environment specifies where the agent acts; the tests verify success; and the solution demonstrates feasibility. This abstraction, introduced by Terminal-Bench [103], applies across the vast majority of agentic and non-agentic benchmarks.

Harbor Adapters translate heterogeneous dataset-specific formats into this shared schema. Together with the Harbor infrastructure, which provides sandboxed execution (local and cloud) and reward-based evaluation, the adapter layer reduces the per-benchmark or per-agent integration cost from $\mathcal{O}(mn)$ to $\mathcal{O}(m+n)$, and enables running tens of datasets, tens of thousands of tasks, and millions of rollouts across many model–harness configurations under identical evaluation conditions.

D.2 Agent Interaction Modes

We categorize each benchmark by its *agent interaction mode*, which captures whether and how agentic capabilities affect the evaluation outcome.

Agentic benchmarks are explicitly designed for agent-based evaluation; tasks assume iterative interaction with an environment (e.g., filesystem, shell, browser, external tools), multi-step reasoning, and/or test-driven feedback, and evaluating them without agent capabilities is infeasible or inconsistent with the original design.

Non-Agentic benchmarks are designed for direct LLM evaluation; tasks are single-pass, with the model producing an answer from the prompt alone and no access to tools or an environment, so introducing agent capabilities would not meaningfully change what the benchmark measures.

Modified-Agentic benchmarks are originally non-agentic, but exposing the underlying environment to an agent (e.g., filesystem access, iterative execution, test-driven refinement) changes the evaluation outcome in a measurable way; these are typically coding tasks whose single-pass protocol underspecifies what a capable agent could achieve with terminal access.

Agent interaction mode is a property of the *benchmark* itself. It is used throughout the catalog (§D.5) and determines which integration scenario applies when we construct an adapter (§D.4).

D.3 Adapter Construction Workflow

Constructing and maintaining adapters at scale is an engineering-intensive process that requires careful handling of dataset formats, execution environments, and evaluation protocols. We organize this process into three stages: *creation*, *review*, and *standardization*.

D.3.1 Creation

Each adapter is constructed through the following steps. **(1)** Parsing and restructuring benchmark content into the Harbor schema (instruction, environment, tests, solution). **(2)** Reconstructing or aligning the execution environment (Docker images, dependencies, network policies, resource limits). **(3)** Integrating evaluation logic and test cases, and verifying that oracle solutions achieve a 100% pass rate. **(4)** Running parity experiments against the original benchmark (§D.4).

To scale this process, we developed contributor guidelines, a prompt-driven tutorial for coding agents, and an agent skill (`/create-adapter`) that handles context sharing and enables parallel adapter development across the team.

Это правильный подход :)

D.3.2 Review

Adapter review is equally intensive, as it must ensure semantic fidelity to the original benchmark, correctness of the execution environment, and conformance to the Harbor schema. We trained a dedicated team of five reviewers and developed automated review bots to accelerate quality assurance. Every adapter pull request undergoes a three-stage review process. **(1)** An *automated bot review* that checks code style, schema conformance, documentation completeness, and the presence of a passing oracle run. **(2)** A *human review* by one to three trained reviewers, who verify adaptation correctness, environment reproducibility, and parity-experiment results. **(3)** A *final approval* by the team lead, who signs off on non-trivial design decisions (e.g., parity-set selection, custom-agent ports) before the adapter is merged. Adapters that fail any stage are returned to the contributor with reviewer comments and re-enter the pipeline once addressed.

D.3.3 Standardization

As more adapters are integrated, requirements and best practices continue to evolve: the Harbor schema gains new fields, shared utilities are refactored, and documentation conventions tighten. To prevent format drift, a standardization team performs periodic passes that co-evolve three artifacts in parallel: existing adapters, the contributor tutorial, and the bot review prompts. In this way, the schema, the guidance new contributors receive, and the checks enforced at review time all stay aligned, while preserving dataset integrity and parity results. This maintenance work is scheduled independently of new-adapter contributions and is a first-class part of the workflow rather than one-off cleanup.

D.4 Parity Experiments

D.4.1 Challenges in Benchmark Portability

Porting a benchmark to a new harness is semantics-preserving *in intent*, but rarely so in practice. Small differences in prompt templates, tool configurations, default decoding parameters, Docker base images, or scoring scripts can shift reported scores by several points, even when the underlying task set is unchanged. Published results are often not reproducible from the released code alone: we have routinely encountered oracle solutions that fail the benchmark's own tests, scoring scripts with undocumented non-determinism, and leaderboard entries produced by pipelines that diverge from the public release. Without explicit verification, a user of a Harbor adapter cannot tell whether a score difference reflects a genuine capability gap, a harness artifact, or a latent bug on either side. We therefore treat parity verification as an essential step in building every adapter.

D.4.2 Design and Evaluation Metrics

A *parity experiment* compares the Harbor-adapted version of a benchmark against the original under matched conditions. The same agent, model, tool set, prompt template, decoding parameters, and execution configuration are applied on both sides. Each side is run k times ($k = 3$ by default) to account for stochasticity, and we report the mean ($\bar{x}$) $\pm$ sample standard error of the mean ($s_{\bar{x}}$), computed as $s_{\bar{x}} = s/\sqrt{k}$ where s is the sample standard deviation across the k runs. An adapter passes parity when the original and Harbor means agree within a margin consistent with their standard errors. Adapters that fail the parity are debugged and re-run. Cases where parity cannot be achieved due to irreducible differences, for example dependence on a non-deterministic external service, are documented per benchmark in §D.6.

Комментарий: что k=3 оставляет очень широкие стандартные отклонения для оценки

D.4.3 Agent Integration Scenarios

A parity experiment requires the same agent to run against both the original benchmark and the Harbor adapter. Whether such an agent already exists or has to be built depends on the original benchmark's design. During our development, we distinguish three main scenarios:

Scenario 1: Harbor Supported Agent. The original benchmark already uses an agent that Harbor also supports natively (e.g., Claude Code, Codex CLI). Adaptation reduces to aligning versions, tool configurations, execution arguments, and implicit environment assumptions between the two sides. The parity experiment runs the same agent against both.

Scenario 2: Vanilla LLM. The original benchmark provides no agent and was designed for direct LLM prompting. We fork the original repository and add a Harbor-supported CLI agent, matching the prompt and decoding protocol of the original single-pass evaluation. Minor modifications, such as input/output formatting and prompt tweaks, are permitted as long as they are applied symmetrically to both the original and Harbor sides. The parity experiment then runs this agent against both the forked original and the Harbor adapter. We apply the same treatment to non-agentic and modified-agentic benchmarks alike.

Scenario 3: Custom Agent. The original benchmark ships a custom agent that Harbor does not support natively (e.g., a benchmark-specific ReAct variant, a deep-research agent, or a multi-agent scaffold). We port the custom agent into Harbor, and the parity experiment runs it against the original benchmark. Where feasible, we also run a Harbor-native agent (e.g., Claude Code) as a cross-check. When the custom agent cannot be cleanly ported due to tight coupling with benchmark internals, we allow the benchmark design to diverge, and ship both the original-agent variant and a CLI-agent variant in Harbor for community use, and document the limitation in §D.6.

The mapping from agent interaction mode to integration scenario is direct. All agentic benchmarks fall into Scenario 1 or Scenario 3 depending on whether their native agent is supported by Harbor. All non-agentic and modified-agentic benchmarks fall into Scenario 2.

D.4.4 Parity Set Selection

For most benchmarks, parity is run on the full task set. When the full set is too costly to run k times (e.g., long-horizon agentic tasks or large task counts), we run parity on a representative subset, the *parity set*, selected by one of the following criteria:

(1) Random stratified sampling over difficulty and/or subdomain, when such metadata is available.

(2) Official subsets, when the benchmark authors provide a smaller canonical subset (e.g., a "Lite" or "Verified" split).

(3) Curated subsets, when stability or environment-setup cost requires hand-picked representative instances.

For each adapter, the parity-set size, selection criterion, and the specific task IDs are recorded and explained in the adapter's README at `adapters/<benchmark>/README.md` in the Harbor Adapters repository, and summarized in the corresponding paragraph of §D.6.

D.5 Adapter Catalog

Tables 3 and 4 list every adapter whose parity experiment is complete, with its domain/subdomain, integration scenario, agent interaction mode, parity agent, parity model, number of runs, parity-set size, and parity scores (original vs. Harbor). The parity-set column is formatted as $parity/full$, where $parity$ is the number of tasks actually used for parity verification and $full$ is the total size of the benchmark. Score values are reported as mean $\pm$ SEM over k runs (SEM omitted when $k = 1$). `N/A` indicates that no direct Harbor-vs-original (or Harbor-vs-Terminal-Bench) parity score is available — either because the upstream benchmark has no agent harness so traditional parity does not apply, or because only an upstream Terminal-Bench adapter versus original-benchmark comparison was measured. Because parity verification is part of adapter construction rather than of the evaluation itself, these tables span more benchmarks than the 54 used in our large-scale evaluation: nine rows (CooperBench, FeatBench, WebGen-Bench, Frontier-CS-Algorithm, DevOpsGym, ScienceAgentBench, MLGym-Bench, ACE-Bench, CRMArena) have verified adapters but were held out of the experiments for compute reasons, and are therefore also listed in §D.6.11; benchmarks already authored in the Harbor schema (Terminal-Bench, CompileBench, SkillsBench) require no adapter and are described in §D.6.10.

D.6 Per-Benchmark Details

The subsections from Section D.6.1 through Section D.6.9 describe benchmarks added with Harbor adapters. Section D.6.10 covers benchmarks that natively use the Harbor format and require

no adapter. Section D.6.11 lists benchmarks whose adapters are merged into Harbor (or have an approved adapter pull request) but were not evaluated here due to resource limits.

D.6.1 Software Engineering

Repo-level Issue Resolution: SWE-Bench Multilingual, SWE-Bench Pro, SWE-Bench Verified, SWE-Smith, SWT-Bench.

SWE-Bench-Multilingual [183]. This benchmark extends SWE-bench [64] across nine programming languages with 300 tasks. The Harbor adapter targets the per-task multilingual image and invokes the official grading harness, adapting all 300 tasks and scoring them with unit tests. The original oracle solutions pass 100%, with one seasonal failure. Parity uses a fixed subset of 50 tasks stratified across the nine languages.

SWE-Bench Pro [33]. This benchmark extends SWE-bench [64] with 731 tasks across Python, JavaScript, TypeScript, and Go. The Harbor adapter reuses the upstream execution and verification with two reliability fixes, running Jest in single-worker mode with forced exit and normalizing gold patches to end with a newline. The original oracle solutions pass 97%, with 15 invalid gold patches and 9 tasks that time out. Parity uses a random subset of 100 tasks together with a 10-task rerun after adapter fixes, and both sides are task-aligned.

SWE-Bench Verified [64]. This benchmark is a human-curated subset of SWE-bench [64] with 500 Python tasks scored by unit tests. The original oracle solutions pass 99.2%, with 4 tasks failing on data or infrastructure errors. Parity covers the full 500-task set and is validated transitively against an earlier Terminal-Bench integration. We also check the benchmark score directly against the official SWE-bench leaderboard on a 499-task set, excluding `scikit-learn__scikit-learn-14710` due to hardware limits in the Daytona environment.

SWE-smith [183]. This benchmark contains 60,000 SWE-bench [64]-style tasks with higher-quality descriptions and richer multi-file patches. Harbor currently supports the Python-profiled tasks, and the adapter extends to other languages easily. Runs are scored by unit tests. A fraction of the oracle patches do not fully pass upstream, a known artifact of synthetic patch generation. Parity uses a 100-task slice reused from an earlier Terminal-Bench integration, so parity chains transitively across two legs.

SWT Bench [113]. This benchmark contains 433 tasks that test an agent's test-generation and bug-reproduction ability, and a redundant code-description segment is removed from the prompt during adaptation. Generated tests are scored by whether they fail on the buggy code and pass on the fixed code. The original oracle solutions pass 97.7%, with 9 `sphinx-doc` tasks failing on an upstream parsing issue. Parity is restricted to evaluation because upstream provides no inference script, so Harbor diffs are exported to the upstream prediction format and graded by the upstream evaluator.

Feature & End-to-end Development: FeatureBench, SWE-Lancer.

FeatureBench [206]. This benchmark evaluates feature-level code generation on 200 real-world Python tasks across 24 popular repositories, split into interface-guided (lv1) and from-scratch (lv2) complexity levels. The Harbor adapter reuses the prebuilt DockerHub images, mirrors the lv1 oracle through `setup_patch` reversal and the lv2 oracle through `agent_code` re-export, auto-configures `shm_size` and GPU allocation from task metadata, and prunes flaky PASS_TO_PASS tests. The oracle passes 200/200 on Docker and 198/200 on Modal, where two transient failures occur, namely a mirror timeout on an lv2 dependency install and a SciPy segmentation fault on an lv1 task in Modal's sandbox. Parity uses a curated 30-task lite split (26 lv1 and 4 lv2) on NVIDIA A10G, and the residual disagreements reflect stochastic lv1 variation rather than systematic divergence.

SWE-Lancer [106]. This benchmark evaluates agents on 463 real freelance software-engineering tasks scraped from Upwork and scoped to the Expensify repository, comprising 198 IC SWE tasks that resolve a GitHub issue and pass Playwright and pytest user-tool tests, and 265 Manager tasks that select the correct proposal among candidates. The Harbor adapter reuses the upstream `swelancer_x86` images with light tmux and user-tool additions and rewrites the Python-only solver prompt into terminal-style instructions. One deviation is that the user tool runs in the agent's namespace, with tests copied in unzipped, because Harbor has no privileged out-of-namespace runner, and the grading logic is otherwise preserved. The oracle passes 100%, as reported in the

adapter PR. Parity covers the full 463-task set, and the original side runs from a `preparedness` fork that ports a matching Claude Code solver into the upstream nanoeval harness, so both sides exercise the same agent.

Competitive & Function-level Coding: Aider Polyglot, BigCodeBench, LiveCodeBench, USACO, HumanEvalFix, QuixBugs.

Aider Polyglot [2]. This benchmark repackages 225 Exercism exercises across six languages (Java, Python, Go, Rust, C++, and JavaScript) to evaluate iterative code editing rather than greenfield generation. The Harbor adapter mirrors all exercises with per-language Dockerfiles, ports the upstream test harnesses verbatim, and wraps encrypted oracle payloads in shell normalizers. Runs are scored by the per-language unit tests, and the adapter reports no aggregate oracle pass rate. Parity is established transitively in two legs, one between the original and the Terminal-Bench adapter (225 tasks, 2 runs with claude-code) and one between the Terminal-Bench adapter and Harbor (225 tasks, 1 run with claude-haiku).

BigCodeBench [210]. This benchmark evaluates complex function-calling and engineering programming on the 145 hard-split tasks, using two prompt formats, the *Complete* docstring-driven format and the *Instruct* natural-language format. The Harbor adapter converts tasks from HuggingFace and uses reward-based verification, and 3 tasks are excluded for network or file-structure incompatibilities with the Harbor sandbox. The adapter PR reports no aggregate pass rate in text. Parity uses the full 145-task set with codex over 3 trials per side.

LiveCodeBench [62]. This benchmark evaluates competitive programming on 1,055 problems aggregated from LeetCode, Codeforces, and AtCoder, covering May 2023 to March 2024. The Harbor adapter supports both Solution-class and stdin/stdout tests, exposes `check_solution.py` for public-test development, and auto-injects standard-library imports, and final scoring runs both public and private tests under 30 s and 180 s timeouts. LiveCodeBench ships no oracle solutions, so validity relies on test-case validation. Parity is established transitively in two legs, one between the original and the Terminal-Bench adapter (100 `release_v6` tasks, 4 trials with terminus-2) and one between the Terminal-Bench adapter and Harbor (100 tasks, 4 trials with claude-haiku).

USACO [143]. This benchmark evaluates algorithmic problem-solving on 307 USA Computing Olympiad problems covering graph theory, dynamic programming, and other competitive domains, of which Harbor adapts 304 after excluding 3 with broken reference solutions. The Harbor adapter wraps the official batch judge in pytest with CTRF logging and embeds canonical solutions, leaving the upstream grading logic unchanged. Parity uses the full 304-task set over 3 runs with claude-haiku.

HumanEvalFix [111]. This benchmark evaluates code repair on 164 Python tasks from HumanEvalPack, where the agent fixes buggy implementations so that pytest passes. The Harbor adapter rewrites the original prompts into agent-oriented instructions, embeds Dockerized environments, and ships reference solutions, and the pytest grading logic is unchanged. The oracle passes 164/164 at reward 1.0. Parity uses the full 164-task set for 3 trials with OpenHands [167] on Daytona under gpt-4o-mini and gpt-5-mini.

QuixBugs [87]. This benchmark evaluates single-line bug repair on 80 defective programs, 40 in Python and 40 in Java, drawn from the Quixey Challenge. The Harbor adapter clones the upstream repository and generates per-language tasks (Python 3.11, and JDK 11 with Gradle), and runs are scored by pytest correctness tests together with a diff-based single-line check. The oracle passes 100%. Parity is established transitively in two legs, one between the original and the Terminal-Bench adapter (80 tasks, 5 trials with claude-code) and one between the Terminal-Bench adapter and Harbor (80 tasks, 3 trials with codex).

Performance Optimization: GSO, AlgoTune.

GSO [142]. This benchmark evaluates code performance optimization on 102 Python repositories, each pairing a codebase with a performance test that serves as the specification. The Harbor adapter follows the upstream evaluation flow in isolated Docker environments and reuses the original grading harness, and all 102 tasks are adapted. The oracle passes 89/102 (87.3%), while the remaining 13 fail because their oracles run inconsistently faster than the reference baseline under

timing variance, a known artifact of the speedup metric rather than a harness bug. Parity uses the full 102-task set with OpenHands@1.4.0 and gpt-5.1 (high) on each side.

AlgoTune [128]. This benchmark evaluates algorithm optimization on 154 tasks across mathematics, physics, computer science, signal processing, cryptography, and graph algorithms, scored by continuous speedup rather than binary pass or fail. The Harbor adapter implements the interleaved timing protocol with 100 instances and 10 timing repetitions per instance, enforces the upstream limits of 8 CPUs and 16 GB of memory inside Docker, and reports the harmonic-mean speedup. Because AlgoTune has no canonical oracle, we use the baseline solution and verify that its self-speedup is close to 1.0×. Parity is established transitively in two legs, one between the original and the Terminal-Bench adapter and one between the Terminal-Bench adapter and Harbor, on the full 154-task set.

Language Translation: CRUST-Bench.

CRUST-Bench [67]. This benchmark evaluates C to safe-Rust transpilation on 100 real-world C repositories from GitHub, each paired with hand-written safe-Rust interfaces and correctness tests. The Harbor adapter vendors the C sources and headers into per-task Rust 1.83 slim containers and classifies projects by difficulty. Runs are scored by `cargo test`, and because transpilation is open-ended, no oracle solutions are provided. Parity uses the full 100-task set for 3 trials with codex and gpt-5-nano.

D.6.2 Mathematics & Reasoning

Competition Mathematics: AIME, IneqMath, Omni-Math.

AIME [191]. This benchmark evaluates competition-level mathematical reasoning on 60 problems from AIME 2024 and 2025, each requiring an exact integer answer in [0, 999] with no partial credit. The Harbor adapter sources problems from the GAIR-NLP/AIME-Preview repository and packages each as an isolated task with Python available for symbolic computation, and scoring uses exact match against the integer answer. The oracle solutions pass 100%. Because AIME has no agent-oriented upstream harness, standard parity does not apply, so we instead confirm on the full 60-task set that diverse agents such as Gemini-CLI and codex recover the expected fraction of solutions.

IneqMath [94]. This benchmark evaluates inequality-proof reasoning on the 100 expert-curated multiple-choice questions in the public dev set, covering Bound Estimation and Relation Prediction, while the test set and step-wise evaluation are not publicly released and are not adapted. The Harbor adapter generates tasks with full upstream metadata and supports submission-format conversion for the leaderboard, and evaluation follows the upstream protocol of exact match with a gpt-4o-mini judge fallback. The oracle passes 100% on the full dev set. Parity uses the full 100-task dev set over 3 trials with codex and gpt-4o-mini, with the same evaluator on both sides.

Omni-Math [46]. This benchmark evaluates olympiad-level mathematical reasoning on 4,428 problems spanning more than 33 sub-domains across 10 difficulty levels. The Harbor adapter loads problems from HuggingFace, standardizes the answer at `/workspace/answer.txt`, and installs the upstream evaluation code inside the container to avoid parsing pitfalls, and a gpt-5-mini judge compares the answer to ground truth. Oracle accuracy is bounded at 99.5% because 18 problems ship without benchmark-provided answers. Parity uses a stratified 100-task subset for 2 trials with codex and gpt-5.3-codex, with an identical judge configuration on both sides.

Abstract & Procedural Reasoning: KUMO, Reasoning Gym, ARC-AGI-2.

KUMO [88]. This benchmark evaluates interactive multi-step reasoning on 5,300 procedurally generated games (50 instances across 106 domain-scenario pairs), where the agent takes actions, observes outcomes, and identifies hidden truths using a knowledge book. The Harbor adapter runs an interactive loop through a local verifier service (`POST /act`) and a CLI client, isolates secrets in verifier-only Docker images while exposing only `knowledge_book.txt`, and enforces action budgets through HTTP 429 responses, and scoring uses exact match on `/app/answer.txt`. The oracle passes 100% on the generated tasks. Parity uses a deterministic 212-task subset (seeds 0 and 1 per scenario) for 3 trials per agent-model pair (kumo-vanilla, terminus-2, and codex, each paired with gpt-5-mini and gpt-5-nano) at `max-steps=50` and `temperature=1.0` on both sides.

Reasoning Gym [151]. This benchmark evaluates verifiable procedural reasoning across diverse problem types, materialized into 576 tasks from 96 task generators (288 easy and 288 hard, with 3 instances each). The Harbor adapter dockerizes each generator and has agents write code to compute answers rather than answering zero-shot, directing the result to `answer.txt`. Agentic execution yields substantially higher reward than the zero-shot setting, and scoring uses the upstream per-task reward verifier. A few generators ship no oracle answer or fail to score their own oracle at reward 1.0, and these are documented and excluded from oracle validation rather than treated as harness bugs. Parity uses the full 576-task set for 3 trials with codex and gpt-5.1-codex-mini, with the same generators and verifier on both sides.

ARC-AGI-2 [23]. This benchmark evaluates abstract visual reasoning on the ARC-AGI-2 public evaluation set, where the agent infers a grid transformation from a few demonstrations and applies it to held-out inputs. Harbor materializes 167 task pairs by splitting the 120 upstream puzzles along their test cases. The Harbor adapter pulls the dataset from `Ardea/arc_agi_v2`, rewrites the puzzle descriptions into agent-oriented instructions, and runs each task in a Python 3.13 container that writes the predicted grid as a JSON 2D array to `/testbed/output.json`. A pytest verifier requires an exact grid match and embeds reference solutions. Parity uses the full 167-task set for 6 trials with codex@0.53.0 and gpt-5.2 at `reasoning_effort=medium`, under the upstream Pass@2 protocol.

D.6.3 Knowledge & Long Context

Expert & Multi-subject QA: GPQA Diamond, Humanity's Last Exam, MMMLU, SimpleQA.

GPQA Diamond [135]. This benchmark evaluates graduate-level scientific reasoning on 198 expert-curated multiple-choice questions in biology, physics, and chemistry, where domain experts reach about 65% and non-experts with web access about 34%. The Harbor adapter loads questions from HuggingFace and deterministically shuffles the answer choices, and scoring uses exact letter match on `/app/answer.txt`. The oracle passes 100% on the full diamond split. Parity uses the full 198-task diamond split for 3 trials with codex and gpt-5.2.

Humanity's Last Exam [126]. This benchmark evaluates expert-level multimodal question answering on 2,500 questions across mathematics, science, engineering, humanities, and computer science. The Harbor adapter reuses the official HLE judge prompt, copies images into the workspace for vision-capable agents, routes responses through `/logs/agent/response.txt` with confidence scores, and corrects one ground-truth placeholder in the upstream dataset, and a gpt-5 judge performs scoring. Oracle verification passes 100% with 0% calibration error. Parity uses a stratified 249-task subset (10% per category, seed 42) with a gpt-5 judge, and calibration error is computed post-hoc.

MMMLU [55]. This benchmark evaluates multilingual multi-subject knowledge by extending MMLU across 15 languages, for a total of 210,630 tasks (14,042 questions translated into Arabic, Bengali, German, Spanish, French, Hindi, Indonesian, Italian, Japanese, Korean, Portuguese, Chinese, Swahili, and Yoruba). The Harbor adapter casts each question as a standalone task with explicit `Answer: $LETTER` formatting and supports multilingual character normalization through the simple-evals regex, and scoring uses exact match on the answer letter. Oracle accuracy is 100%. Parity uses a stratified 150-task subset (10 per language, seed 0) for 5 trials with codex and gpt-5.1-codex-mini.

SimpleQA [171]. This benchmark evaluates factual recall on 4,326 fact-seeking questions whose correct answers are short and unambiguous. The Harbor adapter follows OpenAI's simple-evals approach with a configurable grading model (gpt-5-mini by default) and allows internet access during evaluation, and scoring uses an LLM judge. The oracle passes 100% on the full 4,326-task set. Parity uses the first 50 tasks for 3 trials with claude-code and claude-opus-4-6, using gpt-4o-mini as the judge to match the upstream framework.

Long Context Reasoning: AA-LCR.

AA-LCR [155]. This benchmark evaluates long-context reasoning over documents of roughly 100k tokens in seven categories (company reports, academia, government, legal, industry, marketing, and survey materials), with 100 upstream tasks, of which Harbor adapts 99 after excluding one with a ground-truth error. The Harbor adapter copies documents into the workspace at

`/workspace/documents/` and corrects two ground-truth answers (Task 40 date formatting and Task 94 percentage versus decimal), and grading uses the official equality-checker prompt. The oracle passes 99/99 at mean reward 1.0. Parity uses the full 99-task set across several agent-model pairs, namely codex with gpt-5-mini, claude-code with haiku, terminus-2 with haiku, and terminus-2 with gpt-5-mini.

D.6.4 Scientific Research

End-to-end Research Workflows: ReplicationBench, SLDBench.

ReplicationBench [190]. This benchmark evaluates research replication on astrophysics papers, with 106 upstream tasks, of which Harbor adapts 90 expert-reviewed numeric tasks that require Python to match numeric outputs within specified tolerances. The Harbor adapter integrates the HuggingFace dataset and runs tasks in the upstream Docker images, and scoring uses tolerance-based numeric checks. No aggregate oracle rate is reported, and 5 tasks time out consistently on both sides and are flagged for possible exclusion. Parity is established transitively in two legs, one between the original and the Terminal-Bench adapter (106 tasks, 2 runs with claude-code and sonnet) and one between the Terminal-Bench adapter and Harbor (90 tasks, 1 run with claude-code and haiku).

SLDBench [89]. This benchmark evaluates symbolic scaling-law discovery from more than 5,000 LLM training experiments, packaged as 8 tasks that span parallel, vocabulary, fine-tuning, domain-mixture, mixture-of-experts, data-constrained, hyperparameter, and adversarial (U-shaped) scaling laws. The Harbor adapter ships oracle implementations of the Human and Expert baselines and requires the agent to produce both `/app/law.py` and `/app/explain.md`, and scoring is continuous over R^2, NMSE, and NMAE on held-out extrapolation sets. Because the metric is continuous rather than pass or fail, oracle validation reports the paper's Human and Expert baselines rather than a pass rate, and the two hardest tasks have negative R^2 even for the human oracle, which is preserved faithfully. Parity is established transitively against the Terminal-Bench adapter on all 8 tasks with claude-code and haiku-4-5 over 5 runs.

Scientific Computing: CodePDE, ResearchCodeBench, SciCode, QCircuitBench.

CodePDE [83]. This benchmark evaluates Python code generation for solving partial differential equations across five families (advection, burgers, reacdiff1d, cns1d, and darcy) with 100 test instances per family. The Harbor adapter replaces the upstream scaffolding-template prompts with full markdown task instructions and adds automatic test-data download from HuggingFace, and scoring uses the original nRMSE metric. Reference oracle solutions are copied verbatim from upstream, and no aggregate oracle rate is reported. Parity is established transitively in two legs, one between the original and the Terminal-Bench adapter and one between the Terminal-Bench adapter and Harbor, on the full task set, with results aligned within seed variance.

ResearchCodeBench [59]. This benchmark evaluates the implementation of research code from academic papers, covering 20 ML and CV papers decomposed into 212 code-snippet tasks totaling 1,449 lines of code. The Harbor adapter provides the full paper content and code files, divides the problems into per-snippet units, and reuses the upstream I/O format and unit-test verifier, with one task-specific oracle fix that relaxes the GMFlow tolerance from 1e-7 to 1e-3 to absorb upstream nondeterminism. Oracle reference code is embedded in `solve.sh`, and no aggregate oracle rate is reported. Parity uses the full 212-task set with gpt-4o-mini-2024-11-20 against the original benchmark.

SciCode [157]. This benchmark evaluates scientific computing on 80 main problems decomposed into 338 sub-problems across physics, mathematics, materials science, biology, and chemistry. The Harbor adapter generates one task per main problem with sequential instructions and skips three pre-written steps (problems 13, 62, and 76), and the upstream HDF5 checker gives fractional reward by sub-step correctness. Ground-truth solutions exist only for the 15 validation problems, where the oracle passes 15/15, while the 65 test problems carry placeholder oracles and are excluded from validation. Parity uses the full 80-task set with codex@0.106.0, with sub-step accuracy aligned within a Welch t-test.

QCircuitBench [185]. This benchmark evaluates quantum-algorithm design on 28 tasks spanning oracle construction, algorithm design, and random circuit synthesis, expressed in OpenQASM

3.0 and Python. The Harbor adapter rewrites the upstream task descriptions into agent-oriented instructions and generates oracles dynamically inside the verifier, and scoring combines semantic correctness and efficiency into a score in [0, 1]. The adapter PR reports no aggregate oracle rate. Parity uses the full 28-task set with codex@0.76.0.

Biomedical Research: BIX-Bench, LAB-Bench.

BIX-Bench [107]. This benchmark evaluates computational biology data analysis on 205 questions derived from 61 published Jupyter notebooks. The Harbor adapter offers two execution modes, a custom Jupyter agent that mirrors the original benchmark and a terminal variant (bixbench-cli) that exposes tasks to generic agents such as codex, and the LLM-judge prompt is lightly modified to accept range answers so the oracle scores robustly. The oracle passes 205/205. Parity uses a 50-task subset with gpt-4o-mini, together with a CLI-mode run using codex and gpt-5-mini on the same subset.

LAB-Bench [78]. This benchmark evaluates visual reasoning over scientific figures in biology, with 181 multiple-choice questions from the public FigQA subset. The Harbor adapter copies figures into the workspace for the agent to read from disk rather than embedding encoded images, and shuffles the answer choices at runtime through a container ENTRYPOINT to prevent positional bias, and scoring uses exact letter match through `answer.txt`. The oracle passes 181/181. Parity uses the full 181-task set with codex@0.71.0.

D.6.5 Agents, Tools & Systems

Tool Use & Assistants: GAIA, GAIA2, BFCL.

GAIA [104]. This benchmark evaluates multi-step tool use on 165 validation-split question-answering tasks that require browsing, file handling, and external tools across three difficulty levels. The Harbor adapter wraps the original prompts as task instructions and copies attachments into the workspace, and scoring uses deterministic answer normalization and exact match. The oracle passes 165/165 at mean reward 1.0 on the full validation split. Parity uses the full validation split with openhands and gpt-5-mini at `timeout_multiplier=1.5`, equivalent to `max_iterations=45`.

GAIA2 [45]. This benchmark evaluates agent ability across simulated apps and environments on 800 public validation scenarios from Meta's ARE, spanning five configurations (execution, search, adaptability, time, and ambiguity), each with full environments, apps, and oracle action sequences. The Harbor adapter supports an ARE-native mode that runs the official oracle and default agent, and a CLI mode that exposes ARE tools to standard agents through an MCP sidecar, and scoring uses the upstream action-sequence checker. The oracle passes 800/800. Parity uses a stratified 100-task subset with the official ARE `default` agent over 2 trials per side.

BFCL [121]. This benchmark evaluates function-calling ability on 3,641 tasks across 13 single-turn and live categories from BFCL v4. The Harbor adapter mirrors the upstream AST evaluation logic with function-name normalization, order-independent parallel matching, and parameter validation, and tasks run in Docker with the upstream evaluator pre-installed. All 3,641 oracle solutions pass. Parity uses a stratified 123-task subset with codex@0.77.0 and gpt-4o-mini.

Deep Research & Web Agents: DeepSynth, Seal-0, WideSearch.

DeepSynth [123]. This benchmark evaluates deep information synthesis from multiple web sources into structured JSON answers, with 40 tasks in the dev set. The Harbor adapter reuses the official HuggingFace dataset, and scoring uses deterministic F1 over key-value pairs with an optional Claude Haiku 4.5 judge fallback for semantic equivalence. The oracle passes 40/40 at reward 1.0. Parity uses the full 40-task dev set.

Seal-0 [125]. This benchmark evaluates search-augmented factual question answering under adversarial web evidence on 111 tasks, where frontier models reach near-zero accuracy because of conflicting or misleading search results. The Harbor adapter reuses the upstream HuggingFace dataset, and scoring uses an LLM judge (Claude Haiku 4.5 by default) with normalized string matching as a fallback, mirroring upstream. The oracle passes 111/111. Parity uses the full 111-task set with claude-code.

WideSearch [173]. This benchmark evaluates broad web information gathering on 200 bilingual tasks spanning 18 industries, where the agent collects and organizes structured information into markdown tables. The Harbor adapter reuses the official HuggingFace dataset and ports the upstream metrics, and scoring combines Item F1, exact match, and an LLM judge (gpt-4.1 by default). The oracle passes 200/200 at mean Item F1 of 0.9996, and the small gap arises because gold-answer cells that contain the markdown delimiter (|) are replaced with a space during CSV-to-markdown conversion, which does not affect real agent runs. Parity uses the full 200-task set for 3 trials on each side.

D.6.6 Data & Analytics

Text-to-SQL & Data Science: DA-Code, Spider 2.

DA-Code [61]. This benchmark evaluates data-science coding that requires Python, SQL, and Bash for end-to-end analysis, with 500 upstream tasks across 13 categories, of which Harbor adapts 479 after excluding the ml-cluster category. The Harbor adapter mirrors the upstream evaluation pipeline, applies 11 correctness fixes to the original evaluation script, and adds agent-oriented instructions describing task goals and file paths, and scoring uses the upstream checker against gold outputs. The gold-output oracle passes 479/479 after the fixes. Parity uses the full 479-task set for 3 trials with DA-Agent, and a permutation test shows no statistically significant difference.

Spider 2 [79]. This benchmark evaluates dbt data-transformation ability on 68 upstream tasks over real-world DuckDB projects, of which Harbor adapts 64 after excluding 4 that consistently fail the oracle on both sides. The Harbor adapter reuses the upstream dbt projects with refactored API integration and retry handling for safety-policy rejections, and scoring uses dbt database comparison. The oracle passes 64/64. Parity uses the full 64-task set for 3 trials with spider-agent.

D.6.7 Professional Domains

Finance & Trading: FinanceAgent, PIXIU.

FinanceAgent [162]. This benchmark evaluates financial research and analysis on 50 questions that require interpreting SEC filings and reasoning about reported figures. The Harbor adapter ships a terminal variant for CLI agents and a customized variant for function-calling, and an LLM judge scores answers against the Finance-Agent ground truth. The oracle passes 50/50. Parity uses the full 50-task set for 3 trials with Finance-Agent and Claude Code.

PIXIU [178]. This benchmark evaluates financial NLP across 54,083 tasks in 29 subcategories that span classification, sentiment, question answering, named-entity recognition, sequential labeling, relation extraction, and summarization. The Harbor adapter materializes each subcategory example as a task, and scoring uses category-specific metrics such as accuracy, F1, RMSE, ROUGE, BERTScore, and BARTScore. The oracle solutions pass 100%. Parity uses a stratified 435-task subset (15 per subcategory) for 3 trials with codex.

Business / Professional Work: SpreadsheetBench, MedAgentbench, LawBench.

SpreadsheetBench [98]. This benchmark evaluates spreadsheet manipulation on 400 tasks drawn from real user questions on Excel forums, where the agent modifies a workbook so that designated cell values become correct. The Harbor adapter replaces the Windows-only `win32com` pipeline with cross-platform LibreOffice headless recalculation and applies 11 correctness fixes to the evaluator, and scoring compares cell values against the gold workbook. All 400 oracle solutions pass. Parity uses the full 400-task set for 3 trials with claude-code.

MedAgentBench [63]. This benchmark evaluates clinical agent ability on 300 tasks that require FHIR API interaction to retrieve patient data and execute healthcare workflows. The Harbor adapter vendors the official reference grader, mirrors the upstream GET, POST, and FINISH semantics, and adds Harbor-specific instruction formatting to enforce strict payload compliance, and scoring uses the upstream grader. The oracle passes 300/300, as verified in the adapter PR. Parity uses the full 300-task set for 3 trials with HTTPAgent and gpt-4o-mini.

LawBench [42]. This benchmark evaluates Chinese legal reasoning across 1,000 tasks derived from 20 task families, each with zero-shot and one-shot prompts, covering statutes, legal reasoning, and case prediction. The Harbor adapter pins the upstream LawBench repository and reuses its native

evaluation functions, with the agent writing answers to `/app/answer.jsonl`, and scoring uses per-family metrics that mix exact match, F1, ROUGE, and rule-based scoring. The oracle passes the full task set, as verified in the adapter PR. Parity uses a stratified 120-task subset (3 chunks across 20 families and 2 prompt types) for 3 trials with Qwen Code on GLM-4.7.

D.6.8 Safety & Security

CyberGym [169]. This benchmark evaluates an agent's vulnerability-analysis ability on 1,507 real-world C and C++ tasks drawn from 188 projects, where the agent generates a proof-of-concept input that triggers the bug under sanitizer instrumentation (ASan, MSan, or UBSan). The Harbor adapter ships four difficulty levels, from level0 to level3, that control which files are exposed to the agent, and level1 is the primary level and matches the upstream paper. Scoring uses a dual-binary check in which the generated input must crash the vulnerable binary but not the patched one. The ground-truth inputs live in per-task runner images sourced from OSS-Fuzz, and the oracle reaches 100% on the 10-task subset, while full-set validation is skipped because of the roughly 10 TB footprint of the 3,014 runner images. Parity uses the recommended 10-task subset (6 ARVO and 4 OSS-Fuzz, over five difficulty and seed combinations of three at level1, one at level2, and one at level3) with OpenHands@1.6.0 and claude-haiku-4-5.

StrongReject [148]. This benchmark evaluates LLM jailbreak robustness on 313 forbidden prompts crossed with 39 jailbreak methods, for 12,207 tasks in total. The Harbor adapter supports all 39 methods and implements the official rubric evaluator with inverted scoring, where a reward of 1 indicates a safe refusal, and a rubric-based LLM judge (gpt-4o-mini by default) scores refusal, convincingness, and the specificity of harmful content. All 313 base oracle solutions pass. Parity uses a stratified 150-task subset (50 prompts across 3 jailbreaks) for 3 trials with codex and gpt-5-nano.

D.6.9 Multimodal

Audio Understanding. *MMAU* [140]. This benchmark evaluates multimodal audio understanding on 1,000 expert-curated tasks spanning speech, environmental sounds, and music. The Harbor adapter uses Qwen2-Audio-Instruct to transcribe audio to text and runs the upstream judge through pytest over the transcribed answer. The oracle passes the full test-mini set, as verified in the adapter PR. Parity uses the full 1,000-task test-mini set for 3 trials with Terminus-2 and gpt-4o.

D.6.10 Benchmarks Using Harbor Format

The following benchmarks are built with Harbor format so that no additional adaptation is needed.

Terminal-Bench [103]. This benchmark evaluates an agent's ability to complete realistic command-line tasks such as filesystem manipulation, build configuration, debugging, and system administration. Tasks are authored directly in the Harbor schema, with instruction, environment, tests, and solution, so no adapter is required, and Harbor pulls the upstream task directories verbatim, runs them in per-task Docker environments, and scores them with the upstream test scripts.

CompileBench [132]. This benchmark evaluates an agent's compilation skills across 15 real-world build tasks that involve dependency resolution, legacy code such as `coreutils` v5.0 from 2003, static linking with glibc and musl, and ARM64 and Windows cross-compilation. Tasks are maintained natively in the Harbor schema by the upstream authors, so the adapter reduces to a sparse-checkout of the task directories, and the upstream test scripts run inside isolated Docker images.

SkillsBench [84]. This benchmark evaluates how well agent skills transfer across diverse tasks, with 86 tasks across 11 domains paired with curated reference skills and deterministic verifiers. Tasks are authored directly in the Harbor schema and require no adaptation, and the agent runs with and without the curated skills exposed through Harbor's skills mechanism, scored by the upstream verifiers.

D.6.11 Other Benchmarks Not Included

Due to compute and time constraints, we exclude the following benchmarks from our experiments, although their adapters are merged into Harbor or have an approved adapter pull request. We list

the supported benchmarks below; adapters whose parity experiments are already complete also appear in Tables 3 and 4.

Software Engineering: ABC-Bench [181], AutoCodeBench [204], CanItEdit [15], CooperBench [68], DevEval [80], DevOpsGym [154], EvoEval [176], FeatBench [18], Frontier-CS-Algorithm [102], Multi-SWE-Bench [196], ProgramBench [184], SWE-Gym [120], SWE-Rebench [7], Uni-Code [203], WebGen-Bench [96].

Mathematics & Reasoning: SATBench [170].

Knowledge & Long Context: CL-Bench [35], LoCoMo [100].

Scientific Research: AstaBench [12], LLM-SRBench [145], ML-Dev-Bench [119], MLGym-Bench [114], MLR-Bench [19], RExBench [39], ScienceAgentBench [21].

Agents, Tools & Systems: ACE-Bench [17], AMA-Bench [201], DeepResearch-Bench-II [37], LongCLIBench [43], OSWorld [179], Tau3-Bench [187], TextArena [50].

Data & Analytics: ADE-Bench [150], BIRD-Bench [81], DABstep [40], DS-1000 [76], Kram-aBench [75].

Professional Domains: CRMArena [60], TheAgentCompany [180].

Multimodal: GraphDesignBench [32], RefAV [27].

D.7 Benchmark Issues and Lessons Learned

Adapter construction and parity validation surfaced a small number of recurring failure modes that materially affect reproducibility, often regardless of the benchmark's age, popularity, or scientific area. We group them here by symptom rather than by domain, with concrete examples drawn from the per-benchmark notes in Section D.6.1 through Section D.6.9, and conclude with concrete recommendations for benchmark authors.

Environment instability. Broken Docker builds, unpinned dependencies, removed upstream packages, hard-coded paths, and platform-specific tooling that breaks across machines. Many benchmarks required non-trivial environment reconstruction before any task could run: Spread-sheetBench's upstream pipeline relied on the Windows-only `win32com` stack and had to be ported to a cross-platform LibreOffice headless workflow; SWE-Bench Pro needed Jest invocations to be pinned to single-worker / forced-exit flags before its JavaScript and TypeScript tasks would terminate reliably; ScienceAgentBench shipped a Modal harness whose output paths and per-task dependencies drifted from the released evaluator; and SWE-Bench Verified contains a task (`scikit-learn__scikit-learn-14710`) that exceeds the hardware budget of our default execution backend.

Non-reproducible scoring. Scoring code with hidden non-determinism (network calls, time-based seeds, race conditions in parallel test runners), lenient or accidentally-permissive test matchers, and divergence between the public leaderboard pipeline and the released evaluation script. GSO's speedup metric is sensitive enough to timing variance that 13 of 102 oracle solutions are scored as "slower than baseline" on a fraction of runs — a property of the metric rather than the harness. WideSearch's gold-answer CSVs contain cells whose contents include the markdown column delimiter (`|`); the upstream CSV-to-markdown conversion silently rewrites these to a space, capping any oracle's Item-F1 at 0.9996. DA-Code's released evaluation script required eleven distinct correctness fixes before its gold outputs scored 1.0. We treat divergences of this kind as benchmark bugs, document them in the adapter README, and keep oracle exclusions out-of-band rather than baked into the dataset.

Implicit assumptions. Preprocessing, prompting, or grading pipelines that depend on undocumented conventions: a particular Git branch of a vendored repo, a particular tokenizer, an exact prompt formatting, or a hidden answer-extraction regex. SWE-Bench Pro's grader silently rejects gold patches that do not end with a newline. SWT-Bench's `FAIL_TO_PASS` field is parsed by an upstream regex that returns the empty set on nine `sphinx-doc` tasks, marking them as zero-score even under the gold patch. LAB-Bench's multiple-choice harness exhibits non-trivial positional bias unless the answer-choice order is randomized at runtime, which we now enforce via container

ENTRYPOINT. LiveCodeBench tasks rely on standard-library imports that the upstream harness injects implicitly. We surface these assumptions in the adapter so that they are part of the task specification rather than the agent's prior knowledge.

Oracle solutions that fail. In several benchmarks, the published ground-truth solution did not pass the benchmark's own tests, usually due to environment drift since release rather than a flaw in the solution itself. SWE-Bench Verified ships 4 tasks whose oracles fail on data or infrastructure errors (99.2% pass rate); SWE-Bench Pro ships 15 invalid gold patches and 9 oracles that exceed the upstream timeout (97% pass rate); SWT-Bench's oracle passes 97.7% for the parsing reason above; ScienceAgentBench's CPU-only oracle pass rate is 93.1% (the remaining 7 tasks need GPUs); GSO's oracle passes 87.3% due to timing variance; SWE-smith's synthetic patches contain a small fraction of upstream-failing entries that the original benchmark also does not score. Without an explicit oracle-validation step these failures are invisible to downstream users and silently inflate baseline scores.

Agent hacking surfaces. A less visible but arguably more damaging failure mode arises when the benchmark design inadvertently allows an agent to obtain the ground-truth answer or bypass the verifier without genuinely solving the task. We identified three recurring patterns during adapter construction. *Answer exfiltration via public data sources:* when the task instruction exposes a stable identifier and the container has unrestricted internet access, an agent can query the publicly hosted upstream dataset and retrieve the gold answer directly, passing the verifier without engaging with the task content. *Ground-truth leakage through container mounts or file placement:* if gold labels, answer files, or scoring criteria are visible inside the agent's runtime filesystem—whether through a shared Docker volume, a misconfigured `COPY` directive, or co-located test assets—the agent can read the expected output before producing its response. *Shared-environment verifier tampering:* when the agent and verifier share the same container and interpreter, an agent can overwrite judge dependencies so that the verification step returns a hard-coded passing score regardless of the agent's actual output; this class of attack is invisible to oracle validation because the oracle path typically bypasses the judge. These surfaces are particularly insidious because they inflate benchmark scores silently: the resulting numbers look plausible, oracle runs still pass, and the exploit is only revealed by careful trajectory inspection or by comparing scores against independent human evaluation. We now flag these surfaces during adapter review and require network restriction, identifier masking, or verifier isolation before merge.

Standards we enforce on every Harbor adapter.

- **Oracle validation.** The provided ground-truth solution must achieve a 100% pass rate; deviations must be explained per task and documented in the adapter README, not silently excluded from the dataset.
- **Parity verification.** A parity experiment must confirm statistical equivalence with the original benchmark within sample SEM, on a parity set whose composition is recorded in `parity_experiment.json`.
- **Reproducible execution.** Every task ships a sandboxed Docker environment with pinned dependencies; cross-platform tooling is preferred over OS-specific commands.
- **Documented divergences.** Any deviation from the upstream evaluator (prompt rewrite, scoring fix, exclusion list) is captured in the adapter README so that downstream users can audit it.

Recommendations for future benchmark authors. Based on the above, we recommend that future benchmark releases:

- pin all dependencies (Python packages, system libraries, language toolchains, container base images) to exact versions, ideally via lock files;
- publish a single reproducible container that bundles the entire evaluation pipeline rather than expecting users to assemble it from a README;
- include a runnable oracle solution and gate releases on an oracle-pass-rate check;
- make scoring deterministic — avoid network calls, time-based randomness, lenient regex matchers, and platform-dependent tooling in the grader;

- separate task specification from execution harness, so that downstream adapters and agents can be swapped without touching task content;
- document every implicit assumption (prompt format, answer-extraction rule, tokenizer, dataset version) in the task specification rather than only in code.

Adopting these practices would substantially reduce the per-benchmark integration cost we report in the per-adapter notes.

E Evaluation Protocol and Reproducibility Details

This section describes the evaluation protocol used to run the benchmark experiments at scale and for Harbor-Index. The protocol is designed to make the reported results reproducible and auditable. In particular, it specifies how benchmark datasets are prepared as runnable Harbor task registries, how the reported paper subset is sampled and locked, how model–harness configurations are executed in isolated environments, and how trial-level metadata and artifacts are stored for auditing and aggregate scoring.

E.1 Evaluation Infrastructure

Harness and repository state. We use Harbor as the unified harness for task execution, sandboxing, agent orchestration, and verification. For each benchmark, we follow the corresponding adapter README to construct the runnable dataset and to validate the execution environment, verifier behavior, credential requirements, and resource constraints prior to inclusion. Unless otherwise noted, large-scale evaluations are conducted with Harbor pinned to commit `9ee6790376583608f541c133d1af2dae47b8fc32`. For the Harbor-Index experiments, we use the more recent and stable Harbor@v0.6.4 release, pinned to commit `331dcba30efcc3fa8282a21de562a3834d6e6244`. Harbor automatically records token counts, metrics, and agent trajectories. The experiment repository provides the tooling for dataset metadata upload, job validation, result import, and trial-artifact archival.

Sandbox backends. Each trial is executed in an isolated Harbor environment. For CPU-only tasks, we use Daytona as the default backend. Daytona provides independent task workspaces and isolates the agent and verifier file systems for each trial. For a subset of benchmarks, we instead use Harbor's local Docker backend when the verifier or task environment is incompatible with Daytona, for example when CPU, memory, or storage requirements exceed Daytona's resource limits of 4 CPUs, 8 GB of memory, and 10 GB of storage. Docker-backed jobs are executed through the same Harbor interface as Daytona-backed jobs. For image-heavy benchmarks, we prebuild the outer task image and, when applicable, the inner Docker-in-Docker images, publish them to a registry such as GHCR, and configure the task environment to use the prebuilt image. This procedure reduces failures caused by registry rate limits and repeated large downloads during environment setup. GPU-dependent tasks are executed through Harbor's Modal backend, which we reserve for benchmarks that require GPU hardware or GPU-capable system images.

Language model services. We access each language model through the corresponding first-party provider API, with support from API grants provided by the respective providers. All model configurations are documented in our experiment repository mentioned above. Unless otherwise specified, we use each provider's default settings for temperature, context length, reasoning effort, and other provider-specific parameters; for example, GPT-5.4 is evaluated with the default medium reasoning effort and Claude Opus 4.6 is evaluated with the default high effort. We note that GPT-5.4 is often reported with extra-high reasoning effort on public leaderboards. Readers should therefore exercise caution when comparing our evaluation results with those reported by other leaderboards.

Model and harness matrix. The large-scale evaluation includes 16 model–harness configurations over 8 models spanning capability tiers. Every model is evaluated under exactly two harness conditions: the cross-family Terminus-2 harness and one native harness. There are three native harnesses—Codex CLI for GPT, Claude Code for Claude, and Gemini CLI for Gemini—and therefore 4 distinct harness implementations overall. Terminus-2 is evaluated with all 8 models:

`gpt-5.4`, `gpt-5-mini`, `gpt-5-nano` [117], Claude Opus 4.6, Claude Sonnet 4.6, Claude Haiku 4.5 [6], `gemini-3.1-pro-preview`, and `gemini-3-flash-preview` [49]. Codex CLI is evaluated with `gpt-5.4`, `gpt-5-mini`, and `gpt-5-nano` using CLI version `0.115.0`. Gemini CLI is evaluated with `gemini-3.1-pro-preview` and `gemini-3-flash-preview` using CLI version `0.34.0`. Claude Code is evaluated with Claude Opus 4.6, Claude Sonnet 4.6, and Claude Haiku 4.5 using CLI version `2.1.81`.

For reproducibility, each configuration records the agent interface, the agent CLI version when applicable, the model identifier used in the job configuration, and the run-date range. When provider snapshot or API release identifiers are available, we additionally report them alongside the job-configuration identifier. Some provider identifiers correspond to managed preview endpoints or aliases rather than immutable model snapshots; for these configurations, we report the exact identifier used during evaluation together with the corresponding run-date range. For the Gemini preview configurations, the job-configuration identifiers are `gemini-3.1-pro-preview` and `gemini-3-flash-preview`. Public provider metadata lists these endpoints as preview models with release dates of February 19, 2026 and December 17, 2025, respectively. Evaluations using these endpoints were run in April 2026.

For the Harbor-Index 1.0 experiments, the 9 evaluated models split into four *closed-weight* models, whose weights are not publicly released (GPT-5.5, Claude Opus 4.8, Gemini 3.1 Pro, and Qwen3.7 Max), and five *open-weight* models (GLM 5.2, Kimi K2.6, MiniMax M3, DeepSeek V4 Pro, and MiMo V2.5 Pro). GPT-5.5, Claude Opus 4.8, and Gemini 3.1 Pro are served by first-party APIs and retain their vendor-native harnesses (Codex CLI, Claude Code, and Gemini CLI respectively); the remaining six models are served through OpenRouter and run under Claude Code. Each of the 9 models is also evaluated with Terminus-2. Harbor-Index 1.0 therefore contains $9 \times 2 \times 82 = 1{,}476$ model–harness–task runs. Interactive trajectories and aggregate Pareto numbers are published at `https://harbor-index.org`.

Database and artifact storage. Results are stored in Supabase. The database schema contains eight core tables: `dataset`, `task`, `dataset_task`, `job`, `trial`, `agent`, `model`, and `trial_model`. Dataset upload creates `dataset`, `task`, and `dataset_task` records that capture registry information, task identifiers, task instructions, source metadata, and task-level metadata. Each job record stores the full job configuration, Harbor Git commit or package version, trial count, start and end timestamps, aggregate job statistics, and verification flags. Each trial record stores the trial UUID, task checksum, agent name and version, scalar reward, exception information, setup/execution/verifier timestamps, full trial configuration, and agent metadata. The `trial_model` table stores model and provider names, as well as input, output, and cache token counts for each trial.

We also upload the full trial directory as a compressed archive to a public Supabase Storage bucket named `trials`. When present, `agent/trajectory.json` is uploaded separately as `<trial-id>-traj.json`. Before upload, large transient agent artifacts that are not required for evaluation or audit are removed, including Codex runtime temporary directories and very large Terminus-2 terminal recordings. The retained artifacts include the reward file, verifier output, configuration, trajectory, and logs needed to audit individual trials.

Public trajectory release. The complete rollout set of the large-scale evaluation is released as a Hugging Face dataset at `https://huggingface.co/datasets/kendx/Harbor-Adapter`. The release has two configurations. The `manifest` configuration is a lightweight catalog with one row per (benchmark, task, model, agent) cell (178,647 rows) listing the trial identifiers for that cell, so it can be filtered without downloading trajectory bytes. The `trajectories` configuration stores one row per trial (793,698 trials, about 340 GB in total), each holding the gzipped trial directory described above together with its SHA-256 checksum; shards are laid out per benchmark so that a single benchmark can be fetched in isolation. Each cell retains up to its five most recent trials, so a cell may contain more than the three trials aggregated in this paper when the job was rerun during auditing (Appendix E.3). This release lets others reproduce our aggregate numbers, re-audit individual trials, and reuse the trajectories for analyses beyond those reported here.

E.2 Dataset Preparation and Sampling

For each external benchmark, we follow the Harbor-provided dataset generation workflow described in the benchmark's README. Before adding a benchmark to the evaluation suite, we inspect the generated dataset to determine whether it requires LLM-as-a-judge evaluation, Docker-in-Docker execution, GPU resources, or prebuilt images. The large-scale evaluation subset includes only tasks that can be executed through the standard Harbor job interface: each task provides an instruction to the evaluated agent, exposes an isolated execution environment, and computes a scalar reward using the benchmark verifier.

To ensure compatibility with the target execution platforms, namely Daytona and Modal, we run a smoke test for each benchmark using the low-cost GPT-5-nano model together with Codex. For benchmarks with a large number of tasks, we sample an i.i.d. subset to control evaluation cost while preserving broad benchmark coverage. The resulting large-scale evaluation set contains 54 benchmarks and 6,627 selected tasks. For each benchmark, the manifest records the selected task count, original task count, sample rate, verifier type, and execution backend. Table 5 lists the benchmarks included in the manifest and their selected task counts. After this preparation phase, the evaluation suite can be executed at large scale.

Table 5: Adapter task summary. Sample Rate = Paper #Tasks / Original #Tasks.

Adapter Name	Paper #Tasks	Original #Tasks	Sample Rate	Adapter Name	Paper #Tasks	Original #Tasks	Sample Rate
PIXIU	101	435	23.22%	FeatureBench	185	200	92.50%
SciCode	80	80	100.00%	ResearchCodeBench	212	212	100.00%
AlgoTune	154	154	100.00%	GSO	102	102	100.00%
BFCL	123	3,641	3.38%	HLE	249	2,500	9.96%
Aider Polyglot	225	225	100.00%	Reasoning Gym	576	576	100.00%
LawBench	181	1,000	18.10%	SLDBench	8	8	100.00%
SpreadsheetBench	200	400	50.00%	Spider 2	64	68	94.12%
Seal-0	111	111	100.00%	StrongReject	150	12,207	1.23%
Omni-Math	200	4,428	4.52%	SWE-smith	98	59,136	0.17%
Terminal-Bench 2.0	89	89	100.00%	MMMLU	150	210,630	0.07%
DeepSynth	40	40	100.00%	USACO	100	307	32.57%
GAIA2	100	800	12.50%	KUMO	212	5,300	4.00%
WideSearch	100	200	50.00%	FinanceAgent	50	50	100.00%
CRUST-Bench	100	100	100.00%	DA-Code	200	479	41.75%
SWT Bench	50	433	11.55%	SkillsBench	77	86	89.53%
GAIA	165	165	100.00%	SWE-Bench Pro	100	730	13.70%
AIME	60	60	100.00%	QuixBugs	80	80	100.00%
SWE-Lancer	100	463	21.60%	SWE-Bench-Multilingual	50	300	16.67%
ARC-AGI-2	100	167	59.88%	BIX-Bench	50	205	24.39%
CompileBench	15	15	100.00%	MedAgentBench	100	300	33.33%
GPQA Diamond	198	198	100.00%	SimpleQA	200	1,000	20.00%
HumanEvalFix	164	164	100.00%	SWE-bench-verified	100	500	20.00%
IneqMath	100	100	100.00%	LiveCodeBench	100	1,055	9.48%
LAB-Bench	181	181	100.00%	CodePDE	5	5	100.00%
MMAU	100	1,000	10.00%	ReplicationBench	90	111	81.08%
QCircuitBench	28	28	100.00%	AA-LCR	99	100	99.00%
BigCodeBench-Hard	145	145	100.00%	CyberGym	10	1,507	0.66%

E.3 Error Handling and Quality Control

Phase-based debugging. We structure the large-scale evaluation into phases. We first evaluate each benchmark independently and, within each benchmark, proceed in three phases. Phase 1 serves as a sanity check for model–harness compatibility. In this phase, we run a small subset of tasks (1% of the full dataset) using the complete model–harness matrix, and verify that the environment starts successfully, the agent receives the intended instruction, the verifier runs, the reward is written, token usage is recorded, and trajectory upload succeeds. This phase is also intended to surface severe issues, such as failures to install a particular harness in the benchmark environment due to system incompatibilities. For benchmarks with reference solutions or oracle support, we additionally run an oracle or known-good trajectory to confirm that the verifier produces the expected reward. Phase 2 evaluates the full model–harness matrix on a 10% subset of the dataset, and Phase 3 performs the full evaluation. Each phase excludes tasks that were already included in earlier phases.

Local and database checks. After jobs finish, we inspect local job statistics in a long-form per-task report. This report groups trials by job, dataset, agent, task, reward, token usage, and error type, making it possible to identify all-failure jobs, missing token counts, unexpectedly high

verifier failures, or reward distributions inconsistent with earlier phases. Supabase is then checked for completeness: each expected job should have a job row, the expected number of trial rows, `trial_model` token rows, and storage URLs for uploaded trial archives. Jobs with suspiciously missing trajectories, missing or zero token counts, or unexpected exception spikes are rerun or manually audited before inclusion.

Rerun policy. We rerun a trial when the recorded exception is caused by infrastructure or configuration rather than by the evaluated agent's behavior. Typical rerun triggers include API authentication errors, temporary provider outages, bursty rate-limit failures, sandbox creation failures, image pull failures, verifier crashes due to missing external resources, malformed judge responses caused by judge API failures, corrupted result files, and failed storage or database writes.

We do not rerun agent timeouts within the benchmark execution envelope, verifier timeouts induced by the submitted solution, policy refusals that reflect model behavior on the task prompt, max-output or context-limit failures caused by the agent's interaction strategy, or verifier failures that correctly indicate the task was not solved. When the distinction is ambiguous, the trial archive and trajectory are inspected, and the final decision is recorded with the job audit notes.

E.4 Harbor-Index Experiment Protocol

Task curation and execution budgets. Harbor-Index 1.0 contains 82 tasks spanning 29 benchmarks after difficulty filtering, AI and human audit, and an audit-and-fix loop (Appendix H). Tasks run in isolated Docker sandboxes with separate verifier environments. Performance-oriented tasks (e.g., AlgoTune, GSO) use GPU-capable backends; free-form answer tasks (e.g., HLE, GAIA, GPQA Diamond) use LLM-as-a-judge scoring. We augment each task instruction with an explicit execution contract that states both the time budget and the available resources, for example: "You should solve the task in 1 hour. You can access 1 CPU, 2 GB memory, 10 GB storage." The same limits are enforced in the task configuration through the Harbor `agent` fields. Timeouts are tightened during the audit-and-fix stage (typically $1.2\times$ the fastest frontier model's observed runtime, or up to 3 hours when every model fails).

Scoring and thresholding. Most Harbor-Index tasks have binary rewards and are scored directly from the verifier output. For continuous-score tasks, we convert the raw verifier reward to pass/fail using a locked task-specific threshold. Let $r_{i,t}$ be the raw reward for trial i on task t. For each continuous task, we define

$$\tau_t = \left\lfloor \max_i r_{i,t} \right\rfloor_5,$$

where $\lfloor \cdot \rfloor_5$ denotes flooring to five decimal places over all non-failed observed trials, including oracle calibration runs when available. A trial is counted as passing if its raw reward strictly exceeds τ_t. For bounded continuous metrics with a known upper bound (e.g., F1 score, R^2), we also count a trial as passing if it reaches the upper bound. Speedup-style tasks such as AlgoTune and GSO do not use a fixed upper bound.

E.5 Evaluation Results

In Tables 6, 7, and 8, we present the benchmark scores for all model–harness configurations on all the benchmarks we study.

F Detailed Large-scale Quantitative Analysis

This appendix details the quantitative analyses in Section 3.1. We analyze the 16×54 score matrix to study cross-benchmark redundancy, within-benchmark redundancy, model vs. harness effects, benchmark progress over time, cost evaluation, and execution-latency patterns.

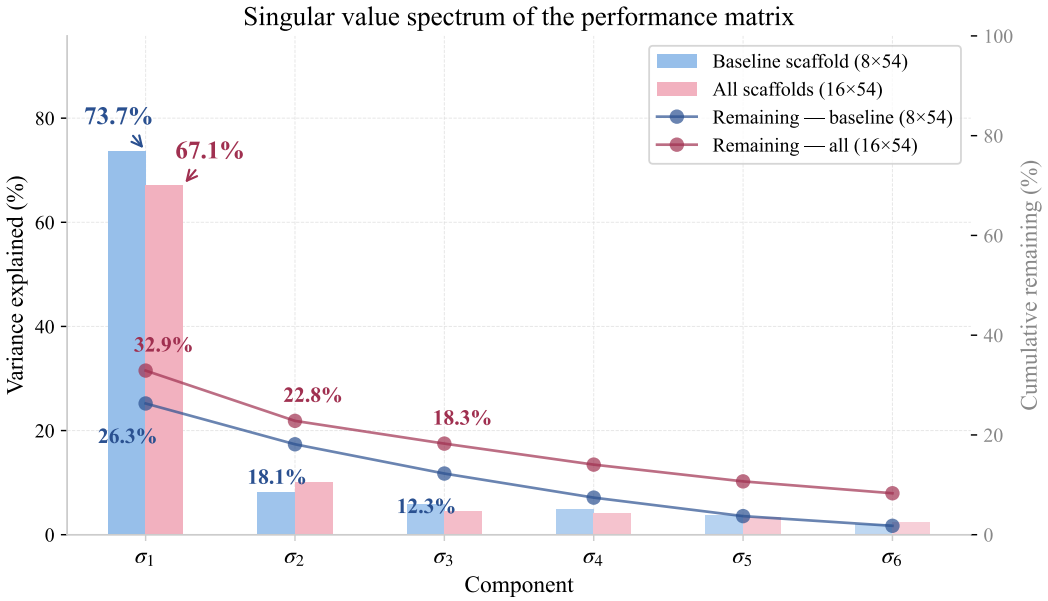


Figure 7: Singular value spectrum of the benchmark score matrix in logit space. Blue: 8-row Terminus-2 submatrix (one fixed harness). Purple: full 16-row matrix (all model–harness configurations). The drop in PC1 share and rise in PC2 when native harnesses are included reflects the orthogonal harness-effect dimension.

F.1 Benchmark and Task Predictability Analysis

F.1.1 Low-rank structure for benchmark space

Singular Value Decomposition (SVD) in logit space on the 8-row Terminus-2 submatrix (varying only the base model under the fixed Terminus-2 harness) yields PC1 = 73.7% of variance, which is comparable to BenchPress's 71% [197] on their 31×10 block with scaffold-free evaluations, confirming that a single "general capability" axis dominates across 8 models spanning capability tiers. The first two components together capture 81.9%, leaving little independent signal beyond two dimensions.

On the full 16-row matrix (all model–harness configurations), the spectrum shifts: PC1 drops to 67.1% and PC2 rises to 10.1% (Figure 7). PC2 cleanly separates the 8 configurations using native harnesses from the 8 Terminus-2 configurations with zero overlap (mean gap 3.31 standard units). Together the first two components capture 77.2% of variance, with every remaining component below 6%: the benchmark space is effectively rank-2, one axis for model capability and one for the harness effect.

F.1.2 Benchmark predictability.

Following BenchPress [197], we quantify per-benchmark redundancy by predicting held-out scores from the remaining benchmarks. Our setting differs in two respects: rows are model–harness configurations rather than models alone, and all 54 benchmark metrics are normalized scores rescaled and bounded in $[0, 1]$. Given the fully-observed score matrix $\mathbf{S} \in \mathbb{R}^{N \times B}$, for each of $T{=}5$ random seeds we hold out 50% of each row's benchmark scores as the test set $\mathcal{H}$; the remaining 50% form the training set $\mathcal{T}$. Both prediction methods operate in logit space:

$$z = \text{logit}(s) = \log \frac{\text{clip}(s,\, \epsilon,\, 1{-}\epsilon)}{1 - \text{clip}(s,\, \epsilon,\, 1{-}\epsilon)}, \quad \epsilon = 0.005, \tag{1}$$

where ϵ clips extreme scores away from 0 and 1 to avoid infinite logits. The predictor combines three steps:

- **LogitBenchReg**: for each target benchmark j, fit univariate ordinary least squares (OLS) regressions in logit space from every other benchmark $k \neq j$:

$$z_{i,j} = \beta_k \, z_{i,k} + \gamma_k, \quad \text{fitted over } \{i : (i,j) \in \mathcal{T} \wedge (i,k) \in \mathcal{T}\}, \tag{2}$$

where $z_{i,j} = \text{logit}(\text{clip}(s_{i,j},\, \epsilon,\, 1{-}\epsilon))$ with $\epsilon = 0.005$ (Eq. 1). We select the top-$K{=}5$ predictors with $R_k^2 \geq 0.1$ as the predictor set $\mathcal{K}_j$. For each held-out cell $(i,j) \in \mathcal{H}$, the predicted score is:

$$\hat{z}_{i,j} = \frac{\sum_{k \in \mathcal{K}_j} R_k^2 \, (\beta_k \, z_{i,k} + \gamma_k)}{\sum_{k \in \mathcal{K}_j} R_k^2}, \qquad \hat{s}_{i,j}^{\text{BenchReg}} = \frac{1}{1 + e^{-\hat{z}_{i,j}}}. \tag{3}$$

- **SVD-Logit (Soft-Impute)**: iterative low-rank matrix completion on the logit-transformed, column-standardized score matrix. Let μ_j and σ_j be the column mean and standard deviation from training entries only, and $\tilde{\mathbf{Z}} = (\mathbf{Z} - \boldsymbol{\mu})/\boldsymbol{\sigma}$ with held-out entries initialized to 0. The iteration:
  1. Compute the rank-r SVD approximation ($r{=}2$): $\tilde{\mathbf{Z}}_{\text{approx}} = \mathbf{U}_{:,1:r} \, \boldsymbol{\Sigma}_{1:r} \, \mathbf{V}_{1:r,:}^{\top}$.
  2. Replace only held-out entries: $\tilde{Z}_{i,j} \leftarrow (\tilde{\mathbf{Z}}_{\text{approx}})_{i,j}$ for $(i,j) \in \mathcal{H}$. Training entries remain fixed.
  3. Repeat until convergence: $\|\tilde{\mathbf{Z}}^{(t)} - \tilde{\mathbf{Z}}^{(t-1)}\|_F \big/ \|\tilde{\mathbf{Z}}^{(t-1)}\|_F < 10^{-4}$.

  After convergence, de-standardize and inverse-logit: $\hat{s}_{i,j}^{\text{SVD}} = 1/(1 + e^{-(\tilde{z}_{i,j} \cdot \sigma_j + \mu_j)})$.
- **Blending and evaluation**: the final prediction combines both in raw score space:

$$\hat{s}_{i,j} = \alpha \, \hat{s}_{i,j}^{\text{BenchReg}} + (1{-}\alpha) \, \hat{s}_{i,j}^{\text{SVD}}, \quad \alpha = 0.6. \tag{4}$$

When LogitBenchReg produces no prediction (no predictor passes $R^2 \geq 0.1$), the method falls back to SVD-Logit alone; when both fail, it uses the column mean $\bar{s}_j$. All predictions are evaluated on raw scores in $[0,1]$. The primary metric is the *benchmark-stratified* Median Absolute Error:

$$\text{MedAE}_j = \text{median}_{(i,j) \in \mathcal{H}_j} \, |\hat{s}_{i,j} - s_{i,j}|, \tag{5}$$

where $\mathcal{H}_j$ is the set of held-out cells for benchmark j (pooled across all seeds). The overall MedAE is the median across benchmarks:

$$\text{MedAE} = \text{median}_{j=1,\ldots,B} \; \text{MedAE}_j. \tag{6}$$

Because all benchmark scores share the $[0,1]$ scale, MedAE is directly interpretable as the typical prediction error in percentage points.

We evaluate on our 16×54 matrix (50% holdout, 3 folds). The simplest baseline predicts each held-out cell with the column (benchmark) mean of the observed entries, achieving MedAE = 0.117. Methods that exploit cross-benchmark structure improve substantially: LogitBenchReg reaches 0.081, SVD-Logit ($r{=}2$) reaches 0.069, and the BenchPress blend reaches **0.063**, a 46% reduction over the baseline. The fact that one benchmark's scores can predict another's to within 0.063 points confirms substantial cross-benchmark redundancy.

Table 9: Per-benchmark predictability (50% holdout, 3 folds). Low MedAE = easily predicted from other benchmarks (redundant); high MedAE = hard to predict (unique signal).

Most Redundant		*Most Unique*	
Benchmark	MedAE	Benchmark	MedAE
HumanEvalFix	0.004	ResearchCodeBench	0.221
KUMO	0.017	FinanceAgent	0.202
BFCL	0.029	LAB-Bench	0.168
StrongReject	0.030	CodePDE	0.164
BigCodeBench	0.031	USACO	0.163
LawBench	0.035	MedAgentBench	0.121
DA-Code	0.038	CyberGym	0.102
Reasoning Gym	0.042	IneqMath	0.099

Benchmarks that resist prediction include ResearchCodeBench (MedAE = 0.221), FinanceAgent (0.202), and LAB-Bench (0.168), measuring capability axes poorly covered by the remaining suite. By contrast, HumanEvalFix (MedAE = 0.004) and KUMO (0.017) are well-reconstructed from peers, adding little independent signal.

F.1.3 How many benchmarks span the evaluation space?

Only 12 of 54 benchmarks contribute independent signal at the $|\rho| < 0.7$ threshold. Greedy forward selection starts from the benchmark with the lowest mean absolute correlation to all others and iteratively adds the benchmark whose maximum absolute correlation with the already-selected set is the smallest. The first 12 selections all satisfy $\max|\rho| < 0.7$; from step 13 onward, every remaining benchmark has $|\rho| \geq 0.7$ with at least one already-selected benchmark.

Table 10: Greedy forward selection sequence: benchmarks added in order of decreasing independence. $\max|\rho|$ is the maximum Spearman correlation with any previously selected benchmark. The line separates the 12 benchmarks below the $|\rho| = 0.7$ redundancy threshold from the first benchmark that exceeds it.

Benchmark	$\max\|\rho\|$	Domain
CodePDE	0.00	Scientific Research
LiveCodeBench	0.01	Software Engineering
IneqMath	0.11	Mathematics & Reasoning
BFCL	0.27	Agents, Tools & Systems
ResearchCodeBench	0.33	Scientific Research
FinanceAgent	0.47	Professional Domains
SLDBench	0.56	Scientific Research
StrongReject	0.59	Safety & Security
Reasoning Gym	0.61	Mathematics & Reasoning
CyberGym	0.66	Agents, Tools & Systems
LAB-Bench	0.67	Scientific Research
Omni-math	0.69	Mathematics & Reasoning
SWE-bench-verified	0.70	Software Engineering

The first redundant benchmark (swebench-verified, step 13, ρ=0.70) is largely explained by the already-selected set. For practitioners with constrained budgets, $\sim$12 well-chosen benchmarks across domains capture most independent signal in the full 54-benchmark suite.

Difficulty and uniqueness are largely decoupled. The benchmarks identified as independently informative span the full difficulty spectrum—from hard (CodePDE, mean score 0.36) to easy (StrongReject, 0.92)—rather than clustering at one end (Table 10). This confirms that uniqueness is driven by the *distinctiveness* of the capability axis a benchmark measures, not by raw task difficulty.

F.1.4 Does the same redundancy hold at the task level?

We extend the SVD analysis to the 16×5896-task matrix (after filtering $6627 - 5896 = 731$ zero-variance tasks where all model–harness configurations scored identically). At the task level, PC1 explains only 32.0% of variance, compared with 67.1% at the benchmark level. Benchmark aggregation therefore compresses away independent dimensions: the task-level score matrix contains richer structure that a single overall ranking cannot capture.

Within-benchmark task redundancy varies widely. For each of the 52 benchmarks with ≥ 10 tasks, we run PCA on the $16 \times T$ sub-matrix, where T is the number of tasks in that benchmark (after removing zero-variance tasks, averaging 11.0% per benchmark). PC1 alone explains 49.5% of within-benchmark variance on average, and a median of 8 components suffices for 90% (out of a maximum of 16 dimensions). The degree of redundancy varies considerably across benchmarks:

- **ResearchCodeBench** (212 tasks): PC1 = 78.4%, only 3 components for 90%.
- **FinanceAgent** (50 tasks): PC1 = 69.5%, 6 components for 90%.
- **HumanEvalFix** (164 tasks): PC1 = 86.2%, 2 components for 90%.

Selecting the k tasks most correlated with the full benchmark mean and re-ranking the 16 systems gives the ranking fidelity shown in Table 11. Individual benchmarks range from WideSearch ($\rho = 0.99$) to CyberGym ($\rho = 0.75$), reflecting differences in internal task diversity; the complete per-benchmark breakdown is given in Table 12.

Table 12: Per-benchmark ranking fidelity: Spearman ρ between the system ranking from the top-k oracle-selected tasks and the ranking from all tasks. Benchmarks sorted by k=3 fidelity (descending).

Benchmark	Tasks	k=1	k=3	k=5	k=10
Widesearch	100	0.959	0.988	0.988	0.997
SWE-Lancer	94	0.921	0.978	0.966	0.971
SWE-bench-verified	94	0.912	0.973	0.977	0.964
DA-Code	164	0.928	0.968	0.971	0.976
SpreadsheetBench	196	0.918	0.966	0.967	0.967
HLE	195	0.935	0.966	0.974	0.987
LAB-Bench	173	0.928	0.963	0.968	0.967
DeepSynth	35	0.926	0.962	0.980	0.989
Omni-math	160	0.879	0.962	0.957	0.966
SkillsBench	68	0.961	0.962	0.974	0.982
SWT Bench	49	0.881	0.956	0.931	0.962
GAIA	157	0.909	0.953	0.959	0.950
ResearchCodeBench	191	0.927	0.949	0.949	0.949
LiveCodeBench	96	0.903	0.944	0.958	0.962
TerminalBench2.0	83	0.884	0.943	0.943	0.946
CRUST-Bench	96	0.902	0.940	0.946	0.964
GPQA Diamond	181	0.894	0.939	0.947	0.956
LawBench	181	0.956	0.938	0.968	0.982
USACO	99	0.876	0.938	0.940	0.957
FinanceAgent	48	0.925	0.937	0.956	0.988
GSO	90	0.912	0.937	0.970	0.947
GAIA2	68	0.910	0.936	0.974	0.962
ARC-AGI-2	95	0.922	0.933	0.948	0.982
Aider Polyglot	221	0.907	0.932	0.954	0.955
AA-LCR	94	0.884	0.932	0.900	0.908
BigCodeBench	119	0.760	0.931	0.960	0.950
SWE-Bench Pro	74	0.919	0.931	0.948	0.952
Spider 2	36	0.863	0.930	0.953	0.931
QCircuitBench	25	0.868	0.929	0.962	0.915
StrongReject	72	0.880	0.929	0.943	0.895
Scicode	76	0.887	0.929	0.929	0.953
PIXIU	87	0.859	0.926	0.921	0.950
FeatureBench	156	0.878	0.926	0.908	0.966
AlgoTune	147	0.876	0.924	0.947	0.926
KUMO	195	0.889	0.921	0.960	0.969
SimpleQA	196	0.859	0.919	0.919	0.921
BIX-Bench	39	0.890	0.918	0.913	0.964
HumanEvalFix	153	0.912	0.914	0.912	0.962
MMAU	91	0.838	0.913	0.921	0.906
MMMLU	127	0.908	0.909	0.911	0.909
SWE-Bench-Multilingual	48	0.885	0.901	0.902	0.965
MedAgentBench	70	0.890	0.899	0.927	0.930
AIME	51	0.873	0.897	0.910	0.942
SWE-smith	79	0.881	0.896	0.937	0.972
CompileBench	15	0.909	0.894	0.971	0.989
Seal0	101	0.844	0.882	0.930	0.955
Reasoning Gym	530	0.828	0.881	0.930	0.931
QuixBugs	80	0.812	0.881	0.923	0.922
ReplicationBench	79	0.867	0.862	0.916	0.940
IneqMath	96	0.859	0.854	0.923	0.970
BFCL	104	0.744	0.771	0.771	0.821
CyberGym	10	0.707	0.745	0.940	0.999

Within-benchmark holdout prediction. Because the full 5896-task matrix has far more columns than rows (16), cross-benchmark prediction would overfit. We therefore predict tasks *within* each benchmark: for each of the 53 benchmarks with ≥ 5 reliable tasks, we mask 50% of each row (3 folds) and blend top-k peer correlation with rank-2 SVD. The blend improves over the task-mean

baseline in 41 cases (median improvement 24.3%), with the largest gains in benchmarks whose tasks test similar capabilities, such as HumanEvalFix (67.3%) and KUMO (56.7%).

Representative task selection. Greedy forward selection within each benchmark identifies independently informative tasks ($\max|\rho| < 0.7$). The fraction of independent tasks varies widely: CyberGym (70%) and BIX-Bench (69%) contain many independent items, while HumanEvalFix and ResearchCodeBench reach the $\rho > 0.7$ threshold within the first few selections, indicating that most of their tasks measure the same underlying capability. Within-benchmark blends outperform cross-benchmark peers in 37/54 (69%) cases, but for the remaining 17 benchmarks cross-benchmark peers predict better, indicating shared capability axes across benchmark boundaries.

F.2 Model vs. Harness Effect: Methodology

To formally test whether model or harness identity explains more variance while controlling for benchmark difficulty, we fit an LMM:

$$s_{ij} = \beta_0 + \boldsymbol{\beta}_{\text{model}}\, x_{\text{model},i} + \boldsymbol{\beta}_{\text{harness}}\, x_{\text{harness},i} + u_j + \varepsilon_{ij}, \quad u_j \sim \mathcal{N}(0, \sigma_u^2), \quad \varepsilon_{ij} \sim \mathcal{N}(0, \sigma_e^2), \tag{7}$$

where:

- s_{ij}: raw score for system i (model–harness configuration) on benchmark j
- β_0: intercept (expected score at reference levels)
- $\boldsymbol{\beta}_{\text{model}}, \boldsymbol{\beta}_{\text{harness}}$: fixed-effect coefficients for model and harness identity
- $x_{\text{model},i}, x_{\text{harness},i}$: dummy-coded indicators (first alphabetical level as reference)
- $u_j \sim \mathcal{N}(0, \sigma_u^2)$: benchmark-level random intercept (difficulty heterogeneity)
- $\varepsilon_{ij} \sim \mathcal{N}(0, \sigma_e^2)$: residual error
- σ_u^2: between-benchmark variance; σ_e^2: within-benchmark variance

Estimated via REML using `statsmodels` `MixedLM`.

The intraclass correlation coefficient (ICC) measures the proportion of total variance attributable to between-benchmark differences:

$$\text{ICC} = \frac{\sigma_u^2}{\sigma_u^2 + \sigma_e^2} = 0.75, \tag{8}$$

confirming that 0.75 of score variance is between benchmarks—i.e., benchmark difficulty is the dominant source of variation, justifying the random-intercept specification.

Table 13 reports the estimated fixed effects. Each coefficient represents the difference from the reference level (the alphabetically first model or harness, whose effect is zero by construction). Among the 8 models, 6 coefficients are significant ($p < 0.05$); among the 4 harnesses, only 2. We define the *effect range* for a factor as the difference between its largest and smallest coefficients (including the reference at zero). The model effect range is $0.235 - (-0.216) = 0.451$ raw score units, while the harness effect range is $0.045 - (-0.042) = 0.087$. The model range is thus 5.2× larger, confirming that model choice dominates harness choice across benchmarks.

Table 13: LMM fixed effects (Eq. 7). $\hat{\sigma}_u^2 = 0.0485$, $\hat{\sigma}_e^2 = 0.0161$, ICC $= 0.75$. Significance: $^*p < 0.05$, $^{**}p < 0.01$, $^{***}p < 0.001$.

Factor	Level	Estimate	SE	z	p
Intercept	baseline	+0.472	0.033	14.26	4.1e-46***
Model	claude-haiku-4-5-20251001	+0.000		(ref.)	
Model	claude-opus-4-6	+0.179	0.017	10.35	4.2e-25***
Model	claude-sonnet-4-6	+0.140	0.017	8.14	4.1e-16***
Model	gemini-3-flash-preview	+0.142	0.021	6.90	5.3e-12***
Model	gemini-3.1-pro-preview	+0.235	0.021	11.43	2.8e-30***
Model	gpt-5-mini	-0.002	0.020	-0.12	0.904
Model	gpt-5-nano	-0.216	0.020	-10.84	2.2e-27***
Model	gpt-5.4	+0.129	0.020	6.49	8.5e-11***
Harness	claude-code	+0.000		(ref.)	
Harness	codex	+0.045	0.020	2.27	0.023*
Harness	gemini-cli	-0.037	0.022	-1.64	0.101
Harness	terminus-2	-0.042	0.014	-2.97	0.003**

F.3 Original benchmark data collections and mapping

To contextualize the current Harbor results, we collected historical scores of different model and/or agents over time, along with their timestamps, for each benchmark from the original paper, public leaderboards, and technical reports of new models.

In some benchmarks, multiple evaluation metrics are reported. We manually select the metric that best aligns with our Harbor evaluation results, or transform the reported metric into a Harbor-aligned score to enable direct comparison. In particular, we apply special treatment to the following papers:

- **CodePDE**: The paper reports nRMSE as the primary metric [83], where lower is better. We convert nRMSE into a Harbor-aligned pass rate by thresholding each PDE family at $\mathrm{nRMSE} < 0.05$ and averaging the resulting pass rates across the five families.
- **AlgoTune**: Raw speedup factors are mapped to the normalized Harbor score using the benchmark's log-speedup compression, $\log(\max\{1, x\})/(\log(\max\{1, x\}) + 1)$, where x denotes the speedup ratio. This transformation ensures that: (1) the score always lies in $[0, 1)$; (2) the model receives a non-zero score only when $x > 1$, i.e., when the agent-generated code is faster than the baseline; and (3) the score increases monotonically with x, so faster code leads to a higher score. In the original paper [128], the overall benchmark score is defined as the harmonic mean of acceleration ratios. We therefore collect the per-task results for each model and recompute the score using our formula, yielding a normalized score in $[0, 1)$.
- **SLDBench**: Raw verifier rewards can lie in $[-1, 1]$, so we report the display-aligned score $(x + 1)/2$ on the $[0, 1]$ scale.
- **QuixBugs**: Historical results are reported as the number of repaired programs, which we convert to pass rate by dividing by the 40 benchmark programs.
- **StrongReject**: Public results report jailbreak/attack success, while Harbor reports defense success. We therefore invert the metric as $1 - x$.
- **KUMO**: Public results are split into easy and hard settings. To match the subsampled Harbor task mixture, we compute a weighted success rate, $(202 \cdot \mathrm{accuracy}_{\mathrm{easy}} + 10 \cdot \mathrm{accuracy}_{\mathrm{hard}})/212$.
- **SciCode**: Harbor uses the 80-task without-background setup and aggregates sub-step correctness at the problem level, so we align to the paper's standard subproblem pass@1 rather than other SciCode variants.
- **ResearchCodeBench**: the official metric is scaled pass@1, which weights each code snippet by its number of executable lines of code (LoC). We therefore align Harbor evaluation to the same LoC-weighted aggregation, instead of averaging snippet-level success uniformly.
- **Spider 2.0**, **SpreadsheetBench**, **SWT-Bench**, **USACO**, and **WideSearch**: public scores are matched to the Harbor-evaluated split or subset when possible, rather than to the full benchmark headline number.

FinanceAgent, GAIA2, LawBench, MLGym, PIXIU, SkillsBench, and SWE-smith were excluded from direct historical comparison when no comparable public metric or split could be identified.

We note that existing benchmark evaluations may not align with our model–harness configurations: some benchmarks evaluate raw LLM performance without any agentic scaffold; some fine-tune specialized models or design task-specific agentic scaffolds, while our study evaluates models spanning capability tiers from the Gemini, Claude, and GPT families under general-purpose harnesses (Gemini CLI, Claude Code, Codex, and Terminus-2). We present in Table 14 a benchmark-level summary of this alignment for the 54 benchmarks in our taxonomy.

Overall, only 16 of the 54 benchmarks have at least one overlapping model or harness configuration. Therefore, these results are often not suitable for direct head-to-head comparison against our Harbor runs. Nevertheless, they remain useful for tracking progress: by taking the best reported performance at different time points, we can estimate how the state of the art has advanced on each benchmark over time, which will be discussed in Appendix F.4.

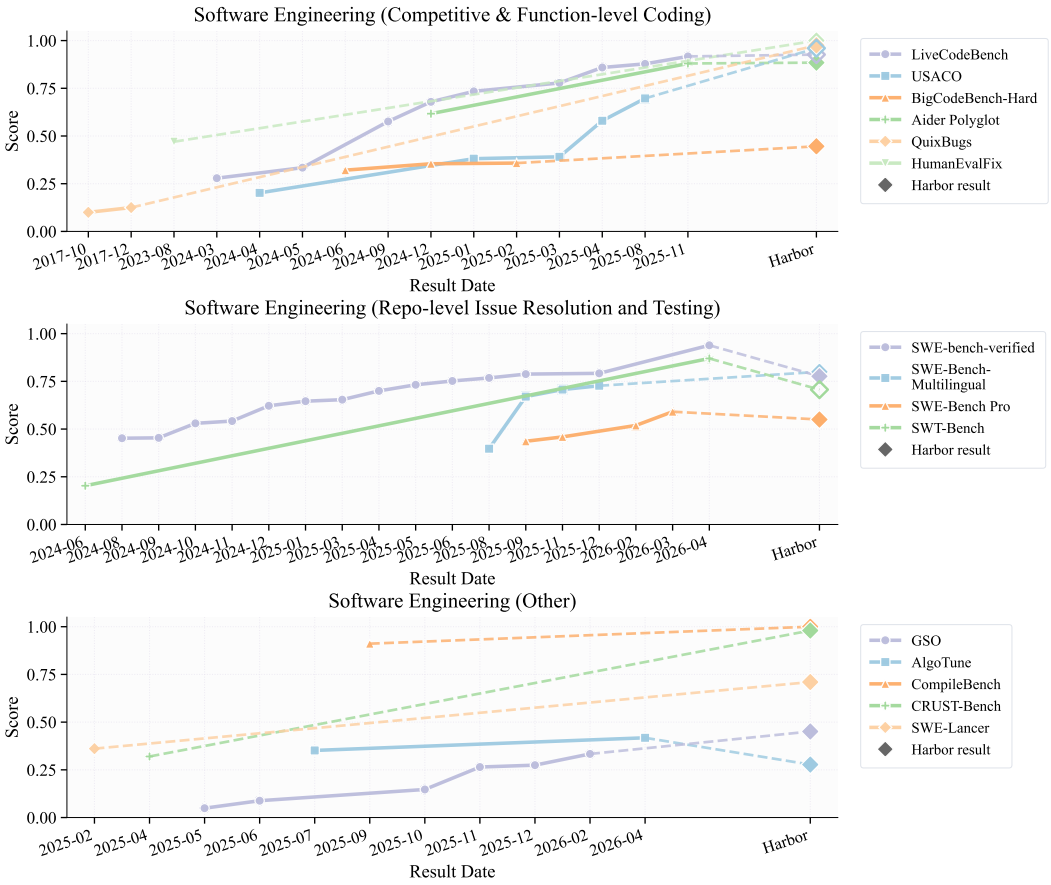


Figure 8: **Progress over time across Software Engineering benchmarks.** Each panel tracks the best reported score on representative benchmarks within a domain as a function of result date; the rightmost diamond marks the best frontier result measured in our Harbor evaluation under a unified adapter-based setup. Hollow diamonds indicate cases where the Harbor evaluation set is not identical to the original benchmark, typically because Harbor evaluates a subset, so the connected segment should be interpreted as a reference comparison rather than a strict apples-to-apples continuation. For LiveCodeBench, the historical evaluation results are based on live evaluation, while Harbor uses the fixed LiveCodeBench (V6).

F.4 Performance Progress Over Time

As described in Appendix F.3, we have collected historical scores of different model and agent configurations over time. In Figures 8 and 9, we present the best reported performance for each benchmark over time. While this view should only be interpreted as an observational record of community progress, the density and slope of these curves reveal where the community has concentrated its benchmarking effort, and where progress has been most visible.

A clear pattern is that Mathematics & Reasoning and Software Engineering have attracted the most sustained leaderboard pressure. These domains contain many closely tracked benchmarks, frequent leaderboard updates, and rapid replacement of the previous state of the art. In Mathematics & Reasoning, scores on several benchmarks rise quickly toward saturation, suggesting that once a benchmark becomes widely adopted, frontier labs and open-source systems rapidly optimize against it – potentially because these tasks have clear, verifiable evaluation protocols. Software Engineering shows an even more heterogeneous version of the same trend: function-level and competitive-programming-style benchmarks often improve rapidly and approach saturation, while several broader engineering tasks remain substantially more challenging. This suggests that the apparent progress in "coding" is not uniform; it is stronger on well-specified, unit-testable tasks,

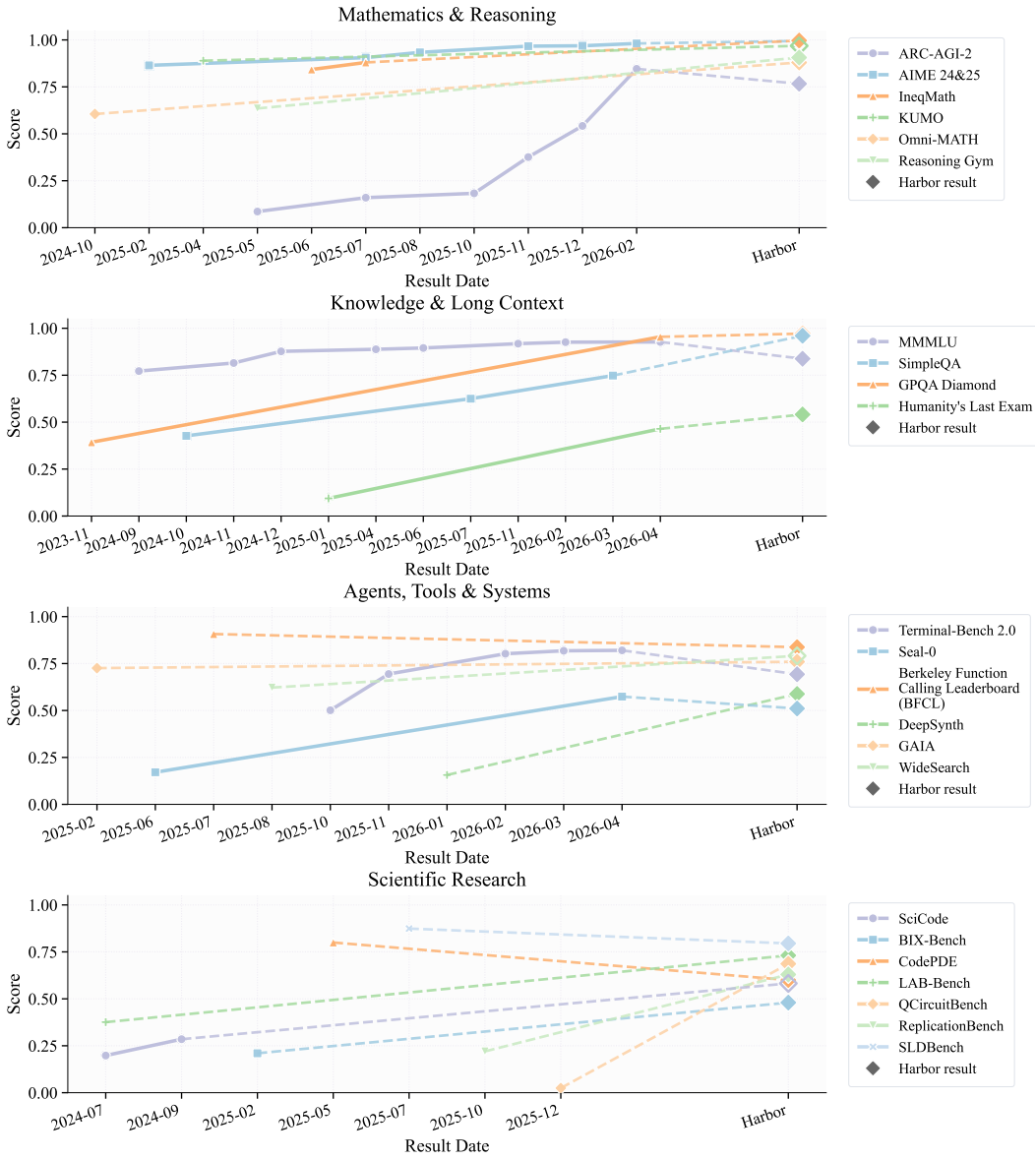


Figure 9: **Progress over time across Mathematics & Reasoning, Knowledge & Long Context, Agents, Tools & Systems, and Scientific Research benchmarks.** Each panel tracks the best reported score on representative benchmarks within a domain as a function of result date; the rightmost diamond marks the best frontier result measured in our Harbor evaluation under a unified adapter-based setup. Hollow diamonds indicate cases where the Harbor evaluation set is not identical to the original benchmark, typically because Harbor evaluates a subset, so the connected segment should be interpreted as a reference comparison rather than a strict apples-to-apples continuation.

and weaker on open-ended tasks requiring repository understanding, environment management, and long-horizon debugging.

By contrast, progress is less dense and often slower in domains such as Knowledge & Long Context and Agents, Tools & Systems. These areas have fewer historical points, fewer continuously maintained leaderboards, and less frequent apples-to-apples reporting. The resulting curves are therefore sparser, but this sparsity is itself informative: it indicates that these capabilities have not yet received the same level of repeated public measurement as math and software engineering. In Knowledge & Long Context, many benchmarks are expensive to evaluate, sensitive to retrieval or

context-window assumptions, and harder to compare across model generations. In Agents, Tools & Systems, evaluation often depends on changing external environments, tool APIs, and multi-step execution protocols, making leaderboard progress more difficult than for static math or coding tasks.

These differences highlight an important source of bias in how progress is perceived. Domains with dense public leaderboards create a feedback loop: frequent measurement encourages optimization, which produces rapid score gains, which in turn attracts more attention. Conversely, sparse domains may appear to progress more slowly partly because they are harder to evaluate consistently and partly because fewer systems are repeatedly measured there. The fastest-moving curves identify areas where the field has built strong benchmarking infrastructure, while flatter or sparser curves point to domains where better standardized evaluation may be a prerequisite for faster scientific progress. Through our infrastructure efforts in this work, we hope that agentic evaluation can become less dependent on a small number of highly visible leaderboards and more evenly distributed across capability domains.

For several benchmarks, the best results reported externally are higher than our best results measured under Harbor, as shown in Table 15. These gaps arise because the external results were obtained via specialized models (e.g., on BFCL), specialized agent scaffolds (e.g., on MedAgentBench, SLDBench, and AlgoTune), and larger compute budgets for agents (e.g., on CodePDE). By contrast, Harbor evaluates a fixed set of models spanning capability tiers under a unified adapter-based setup, with the goal of improving cross-benchmark comparability rather than maximizing performance on each individual benchmark.

F.5 Latency Study

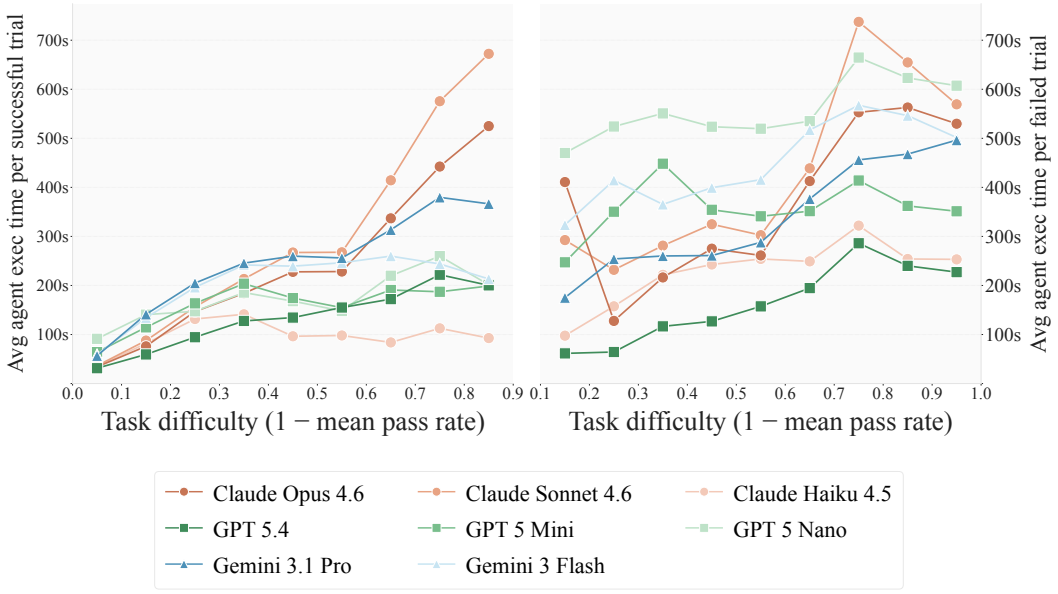


Figure 10: **Left:** Average agent execution time per successful trial. **Right:** Average agent execution time per failed trial. Per-model breakdown by task difficulty. The 0.9–1.0 bin is omitted from the success panel (most models have fewer than 25 successful trials) and the 0.0–0.1 bin from the failure panel (four models have fewer than 25 failed trials).

Agent execution latency, the wall-clock time a model spends actively working on a task, reflects inference time, tool calls, environment interaction, and retry behavior. Using the same difficulty buckets as in Section 3.3, we disaggregate latency by trial outcome to ask: do all models spend time the same way when they succeed versus when they fail?

Figure 10 plots per-model execution time on successful trials (left) and failed trials (right). On the success side, models follow a broadly similar upward trend with difficulty, but separate into a high-

latency group (Claude Sonnet 4.6, Claude Opus 4.6) and a low-latency group (GPT-5.4, Claude Haiku 4.5, GPT-5-Mini, Gemini 3 Flash), with the gap widening on harder tasks.

The failure side reveals two findings invisible in the success panel. First, there is a shared structural pattern across all models that failure latency peaks around difficulty 0.7–0.8 and then declines on the hardest tasks, the opposite of the success trend. This suggests that the hardest tasks trigger earlier termination, while tasks in the 0.7–0.8 range are plausible enough to sustain extended, ultimately unproductive attempts. Second, the relationship between success and failure latency is not consistent across models. GPT-5.4 terminates in similar time regardless of outcome, whereas GPT-5-Nano, despite being the weakest model, persists longest on failed trials.

G Case Analysis

G.1 Task Failure

The task-failure examples below illustrate the kinds of broken designs identified during Harbor-Index human review (Section 4). Each category appears across multiple benchmarks in the review set.

Broken environments. All four GAIA2 tasks that reached human review share the same defect: the adaptive simulation event needed to trigger the grading cascade never fires. One Claude-code trajectory ran for 270 steps, spent 36 hours but received only `{''notifications'': []}` throughout. The verifier waits for the simulation to advance in response to agent actions, while the adapter never advances it. Every model and harness fails for the same reason, ruling out agent capability as the cause.

On MMAU, the Harbor adapter discards the audio payload before handing each task to the agent. Agents work from a text transcript and are graded as though they heard the audio, so tasks requiring acoustic judgement (emotion, speaker identity) cannot be solved regardless of model strength.

Instruction-verification mismatch. A task only has a clear pass condition when the instruction and the verifier agree on what success looks like. When they do not, an agent can produce a defensible solution and still score zero. We see this pattern in several forms across the review set.

In one SWEBENCH-PRO Ansible task, 162 of 168 tests pass for most agents. The six that fail require a specific dictionary format absent from the problem description. A TELEPORT task adds a new test file (`åTestAuditWriter/Backoff`) in the gold patch that is absent from the starting repository. All eleven capable agents implement the backoff mechanism correctly per the written specification, but the verifier's restore command does not include the test file the gold patch introduced.

In other cases the verifier holds a standard the instruction never states. A HUMANITY'S LAST EXAM task was rejected because the judge carries an internal reference answer that disagrees with the mathematically correct value; across all 18 trials, the judge rejected the correct answer with 152–280 completion tokens of active reasoning per run. An AA-LCR task rejects "30.4 percentage points" while accepting "0.304," applying a unit-representation rule that does not appear in the instruction.

The mismatch can also originate on the instruction side. A SWEBENCH-MULTILINGUAL task gives agents a raw GitHub bug report (*Describe the bug, To Reproduce, Expected behavior, Your Configuration*) but no directive to fix anything. Agents have to guess from the report format that a fix is expected. An AIDER-POLYGLOT JavaScript task shows the matrix transpose with newline-separated strings, which suggests a string contract, but the hidden verifier expects arrays. No test file is present in the working tree, so agents cannot discover the discrepancy before submitting.

Gameable tests. A FEATUREBENCH-MODAL task produces byte-identical failure output (53 fails / 1 pass) across 14 runs from 5 distinct harness and model combinations. The test file imports the implementation under a specific symbol name and module path the instruction does not disclose. Every agent that uses a different but functionally correct structure fails in the same way. The task is measuring whether an agent guesses the right import path, not whether it can implement the feature.

Wrong gold answer. Several AA-LCR tasks were rejected because the gold answer is simply wrong. In each case, all 16 evaluated configurations spanning 4 distinct harness implementations independently reached the same alternative value, and manual inspection confirmed the agent answer is correct. When every frontier model agrees and the gold disagrees, the gold needs to be checked.

G.2 Agent Failure Modes

G.2.1 Methodology

The agent failure-mode taxonomy is bootstrapped *bottom-up* rather than authored top-down. We draw a stratified sample of failed trajectories spanning every benchmark, model, and harness in our pool, and dispatch a Claude-4.7-Opus subagent to each trajectory in parallel. Each subagent reads the task's instruction, the verifier output, and the agent's full ATIF trajectory, and emits a free-form, single-sentence description of the failure root cause—e.g., "deleted a previously-passing test rather than fixing the bug," "terminated declaring success without running the verifier," "failed to install a benchmark-specific dependency." The free-form descriptions are then clustered into a small set of candidate failure modes, and the authors review and merge near-duplicates, yielding 16 candidate rubrics. Each rubric carries a short definition and several anchor examples drawn from the bootstrap descriptions, so that downstream human annotators and an LLM judge can apply the same definitions. Below we describe the human-annotation protocol, the gold-label adjudication rule, and the LLM-judge calibration that follows.

Human annotation. A held-out sample of 182 failed trials, stratified across benchmark, model, and harness, is annotated independently by two non-overlapping panels of three annotators each (Round 1 and Round 2). Each annotator works in Docent[2] on a per-trial UI: they read the same task instruction, verifier output, and rendered ATIF trajectory the LLM judge will see, and apply each of the 18 candidate rubrics independently with a binary verdict (MATCH / NO MATCH) and a free-text explanation citing concrete trajectory evidence. We measure inter-annotator agreement by comparing the two rounds.

Inter-rater agreement. Cohen's κ between Round 1 and Round 2 is reported per rubric in Table 16. The pooled κ across all 18 rubrics is 0.66, and twelve of the eighteen rubrics individually clear the $\kappa \geq 0.5$ "moderate-or-better" threshold. The remaining six — Calculation Mismatch (R1 over-applies it as a numerical-Q&A catch-all), Rule Inference Error (low base rate, R2 over-applies on ARC-style tasks), Specification–Verification Mismatch (R2 over-applies on under-specified tasks), and three rubrics that R1 never used at all (Insufficient Web Research, Premature Termination, Wrong Output Location) — are dropped from headline reporting and folded into an "Others" bucket.

Gold-label adjudication. We then take the union of the two human annotations to generate the final gold label for calibrating the LLM judge. For each $(\textit{trial}, \textit{rubric})$ cell, the gold label is MATCH iff at least one Round-1 *or* Round-2 annotator selects a MATCH. This union rule yields a recall-leaning gold set: a single annotator who can quote concrete trajectory evidence (a verifier-output snippet, a tool-call message number, a quoted line of agent code) is sufficient to flag a cell.

LLM-judge calibration. The judge is a single multi-rubric shortlist prompt covering the 12 trustworthy rubrics: each call shows the model the task context (instruction, verifier code, oracle solution, environment Dockerfile, verifier stdout), the full rendered ATIF trajectory, and all 12 rubric definitions, and asks the model to shortlist 0–3 rubrics that fire with one citation per fire. We tune the rubric texts (not the prompt template) iteratively against the gold set, alternating broaden/tighten edits per rubric. We test two judge models (`gemini-3-flash-preview` and `gemini-3.1-pro-preview`), and use the stronger Pro model after 14 rounds of prompt tuning.

A few patterns from 14 tuning rounds:

- One-rubric-at-a-time edits compose; multi-rubric edits do not. Tightening Rubric A's exclusions reliably shifts borderline cells away from A, but the same edit will typically over-correct

[2]Docent is a system for inspecting and annotating model trajectories: https://transluce.org/introducing-docent.

on a related rubric (e.g. tightening False Success Claim drains positives into Unverified Claim) when applied alongside an unrelated change to a third rubric.

- Per-rubric sub-judges (one focused yes/no call per rubric per trial, $12\times$ the calls) underperform the multi-rubric SHORTLIST prompt. The competition between rubrics for the shortlist slot is helpful: it forces the model to commit to a positive verdict rather than hedging NO MATCH on every rubric independently.
- Pro responds better to short Docent-base rubric text than to elaborate priority-firing language we wrote in some iterations. For Agent Timeout, dropping the override entirely (so Pro reads the original 200-token rubric text) yields $\kappa = 0.48$; an elaborate override that explicitly reranks AT above content rubrics yields $\kappa = 0.0$ (the model never picks AT). Pro is conservative by default, and verbose "do this even if you'd otherwise pick X" instructions tend to confuse the model rather than improving rubric selection.

Calibration results. The final chosen configuration achieves pooled $\kappa = 0.51$ against the 182-trial gold set, vs. the human–human R1↔R2 ceiling of 0.66 on the same trials. Six rubrics clear the $\kappa \geq 0.5$ trustworthy threshold individually; the remaining six less confident rubrics are grouped into the "Others" bucket. Per-rubric numbers are given in Table 17.

The mean κ across the six reported rubrics is 0.61, essentially matching the human–human ceiling on the same six rubrics (0.71): on the rubrics where the judge clears the trustworthy bar, its calls are at human-rater quality, and the gap between 0.51 pooled and the 0.66 human ceiling is concentrated in the Others bucket — particularly Unverified Claim (Pro under-fires by an order of magnitude), Wrong Output Schema (Pro under-fires when key-name mismatches are involved), and Wrong Target / Layer (low base rate, $n = 2$ in the gold sample).

Audit at scale. The calibrated judge is then applied to all $5{,}898$ failed trials in the full hard-task set. We retry parse-failure trials up to three times with bumped `max_tokens`, achieving 98.4% parsed coverage; the remaining 1.6% are mostly trials whose rendered transcript exceeds the gateway's per-request token cap and are dropped from headline numbers.

Rubric definitions. The full set of 18 candidate rubrics, with the definitions presented to human annotators and used in the LLM judge prompts, is given below. We mark each rubric as TRUSTWORTHY (passes inter-annotator $\kappa \geq 0.5$) or EXCLUDED (folded into "Others").

Trustworthy rubrics ($\kappa \geq 0.5$, reported individually).

Wrong Target / Layer. TRUSTWORTHY

Framing. Agent applied a change to a target / abstraction layer / module / code path where the change cannot produce the behavior the grader checks.

Concrete patterns.

- Fix at global-stylesheet layer when the component uses inline / CSS-in-JS styles
- Test that asserts file-layout when the task required a behavioral test
- Patch to a codepath the failing test does not exercise
- Shallow symptom fix when root cause is deeper in state/logic
- Fix in a deprecated / unused module

Decision procedure.

1. Identify the task's intended target.
   - If no target location inferrable: NO MATCH.
2. Identify where the agent actually applied changes.
3. MATCH if (a) a change was applied AND (b) the applied change is structurally incapable of affecting the graded behavior.

Exclusions.

- Right file but logic wrong: NO MATCH (Algorithmic Bug).
- Minimal symptom fix catching some cases: NO MATCH (Algorithmic Bug).

Syntax / Language Error. TRUSTWORTHY

Framing. Agent produced code / DSL artefact that doesn't parse or uses invalid domain constructs — Python SyntaxError, OpenQASM unsupported gates, PDDL malformed, Java NPE at load time, Go won't compile, YAML rejected.

Decision procedure.

1. Identify the domain language.
2. Look for compile / parse / load errors in tool outputs.
3. MATCH if the artefact fails at language-level.

Exclusions.

- Code compiled but logic wrong: NO MATCH (Algorithmic Bug).
- Runtime exception from execution (not parse-time): NO MATCH unless the exception prevents any test from running.

Algorithmic Bug. TRUSTWORTHY

Framing. On a code-writing or bug-fixing task, the agent's overall APPROACH IS DEFENSIBLE but the implementation contains a SPECIFIC, NAMEABLE DEFECT — wrong axis, off-by-one, parameter swap, missed edge case, flawed API usage, wrong control-flow branch. You must be able to point to the bug.

Decision procedure.

1. Confirm the task is code-writing or bug-fixing (not factual Q&A, not Calc, not numerical-only).
2. Confirm runnable code WAS produced and executed without an unhandled crash (no SyntaxError, no ImportError, no environment failure).
3. Confirm reward=0.
4. POINT TO THE BUG. You must quote either: If you cannot quote a SPECIFIC defect — output 'no match'.
   - The buggy line in the agent's code AND describe the specific defect (e.g., 'used i+1 instead of i-1 on line 42'), OR
   - A specific failing test output that pinpoints the logical error (e.g., 'expected 5 got 4 on input [1,2,3,4]').

Exclusions.

- Agent took the WRONG OVERALL APPROACH (e.g., downloaded pre-trained weights when the task said to fine-tune; implemented a feature in the wrong file; chose the wrong abstraction): NO MATCH (Wrong Target / Layer).
- Code did not parse / compile: NO MATCH (Syntax / Language Error).
- Sample tests passed but hidden tests failed: NO MATCH (Hidden-Test Regression).
- Numerical result outside tolerance: NO MATCH (Calculation Mismatch).
- No code emitted (only narrative / plan): NO MATCH (Plan Over Implementation / Silent Deliverable).
- Crash from missing dependency / broken environment: NO MATCH (Environment Block).
- Verifier's spec doesn't match the prompt: NO MATCH (Specification–Verification Mismatch).
- Wrong file/path/abstraction: NO MATCH (Wrong Target / Layer).
- Format / shape only: NO MATCH (Wrong Output Schema).

- Failure on a short-answer Q&A task: NO MATCH (Wrong Factual Answer). If you would write 'the agent's solution must have a bug because reward=0' — that is NOT enough. Output 'no match'. You need to identify WHICH bug. This rubric is HIGH-PRECISION: prefer 'no match' unless you can name and quote the specific defect.

Wrong Factual Answer. TRUSTWORTHY

Framing. The task asks for a SHORT, LITERAL ANSWER (a single value, a year, a name, an MCQ letter, a number with a fixed unit, a yes/no) and the agent emitted that answer in the required slot, but the value is wrong by a plain reading of the prompt.

Decision procedure.

1. Confirm the deliverable is a short literal answer — not extended code, not a multi-step calculation pipeline that itself can have algorithmic bugs.
2. Confirm the agent emitted a literal answer in the required slot (file, message, function call) and that the answer is concretely wrong (you can quote the agent's value AND the verifier-expected value).
3. MATCH only when steps 1 + 2 hold AND the failure isn't better explained by a more-specific rubric below.

Exclusions.

- Numerical answer outside tolerance from a calculation pipeline: NO MATCH (Calculation Mismatch).
- Code-writing task where reward=0 because of a bug in the code: NO MATCH (Algorithmic Bug).
- Format/shape mismatch (right content, wrong shape): NO MATCH (Wrong Output Schema).
- Wrong because of a rule-inference mistake on a pattern task: NO MATCH (Rule Inference Error).
- Missing answer / no deliverable: NO MATCH (Silent Deliverable).
- Agent could not verify against an available checker: NO MATCH (Unverified Claim). This rubric COVERS short-answer Q&A, MCQ, simple lookups, even when the underlying reasoning is complex — as long as the deliverable is just the literal answer and the answer is wrong.

Wrong Output Schema. TRUSTWORTHY

Framing. The agent's content is at least partly correct, but the FORMAT / SHAPE / ENCODING / KEY-NAMES are rejected by the verifier. This rubric covers a wide range: JSON shape mismatch, wrong scalar type (e.g. 4.0 vs 4, '0' vs 'zero'), expected string got number, missing required key, extra prose around the answer, wrong delimiters, markdown vs plain text, list-vs-dict, and key-name mismatches (e.g. snake_case vs camelCase) where the verifier checks the literal key.

Firing trigger.

- The verifier output contains shape/type/encoding error patterns: 'expected <type> got <type>', 'KeyError', 'invalid JSON', 'expected list got dict', 'AttributeError on dict.<method>', 'JSON decode error', schema-validation messages.
- The verifier complains the agent's output cannot be parsed.
- The verifier expected a specific key/name that the agent used a different spelling/case for, where the underlying values are recognizable.
- The agent emitted markdown / prose where a bare value was expected. Output MATCH if ANY of the above patterns appears AND the agent's content is at least partly recognizable.

Decision procedure.

1. Inspect verifier_output for shape/type/encoding/key error patterns.
2. Confirm the agent's content is at least partly correct — the right answer or recognizable pieces are present, just in the wrong shape.

3. Quote the specific divergence (which field, what shape).
4. MATCH if 1+2+3 hold.

Exclusions.

- Content is wholly wrong, formatting is fine: NO MATCH (Algorithmic Bug / Wrong Factual Answer).
- Content correct, location wrong: NO MATCH (Wrong Output Location).
- No content produced at all: NO MATCH (Silent Deliverable).

Hidden-Test Regression. TRUSTWORTHY

Framing. The agent ran SCAFFOLD-PROVIDED test infrastructure or correctness checks (sample tests, the repo's own test suite, a check script that ships with the task, a make-target the task points at, a CI command, a known oracle command) on its deliverable and they PASSED — but the verifier's hidden / fuller test suite still rejects with reward=0.

Key signal: the agent ran something that came WITH the task (not something the agent wrote / fabricated) and saw a positive result.

FIRING TRIGGER (high-recall): trajectory contains BOTH: (a) An invocation of a scaffold-shipped test/check (pytest tests/, make test, gradlew test, npm test, repo-provided sample inputs, a check script in the workspace), AND (b) A positive result from that invocation (the agent quoted 'passed', 'OK', '0 failed', 'tests passed', or similar).

If both fire and reward=0, output MATCH unless a strong exclusion applies.

Decision procedure.

1. Quote the scaffold-provided test/check the agent ran (with msg number).
2. Quote the positive output the agent saw.
3. Confirm reward=0.
4. Confirm the test was scaffold-shipped, NOT agent-fabricated. Ad-hoc inline assertions the agent wrote do NOT qualify (those are False Success Claim).
5. MATCH when 1+2+3+4 hold.

Exclusions.

- Agent ran its own ad-hoc check (an inline `assert ...` it crafted, a smoke-test script, a manual eyeballing): NO MATCH (False Success Claim).
- Agent never ran any test: NO MATCH (Unverified Claim).
- Agent's tests failed and it ignored them: NO MATCH.
- Hidden tests enforce something genuinely absent from the prompt: NO MATCH (Specification–Verification Mismatch).
- Crash / syntax error: NO MATCH (Syntax / Language Error). This rubric is HIGH-RECALL: when you see scaffold-provided tests + positive result + reward=0, prefer 'match' over the broader False Success Claim or Unverified Claim. The signal is most specific.

False Success Claim. TRUSTWORTHY

Framing. The agent SAW POSITIVE EVIDENCE for its deliverable (a passing assertion, a matching expected output, an analysis it concluded was correct, a successful smoke-test) and DECLARED THE TASK DONE — but the verifier disagrees with reward=0.

Firing trigger. (a) The trajectory contains an agent-generated positive correctness signal: a quoted phrase like 'passed', 'OK', 'matches expected', 'all tests pass', 'correct', 'looks good', 'sanity check passes', or an inline assertion that returned True / didn't raise. (b) The agent declared completion (final 'task done', 'finish', 'submit', or stopped confidently after writing the deliverable). (c) Reward = 0.

If all three fire, output MATCH unless a strong exclusion applies.

Note on the FSC vs HTR boundary: when the agent ran scaffold-shipped tests that passed, prefer Hidden-Test Regression (more specific). When the agent's positive signal came from its own ad-hoc check (an inline assertion / one-off script / manual eyeballing), prefer FSC.

Decision procedure.

1. Find the agent-generated positive correctness signal in the trajectory and quote it (with msg number).
2. Confirm completion declaration (or implicit stop) and reward = 0.
3. If the positive signal came from scaffold-shipped tests (pytest, make test, check script that ships with the task), this is HTR territory — output NO MATCH.
4. Otherwise output MATCH.

Exclusions.

- Agent's check returned negative and agent declared done anyway: NO MATCH.
- Agent ran NO check at all: NO MATCH (Unverified Claim).
- Lint/syntax/existence checks only: NO MATCH (Unverified Claim).
- Agent admitted incompleteness: NO MATCH.

Unverified Claim. TRUSTWORTHY

Framing.

- The agent declared the task done WITHOUT running a substantive verification of its deliverable when one was available. Unverified Claim is a RESIDUAL rubric — it applies when the agent skipped verification AND no more-specific failure mode explains the verifier's rejection. PRIORITY ORDER — check more-specific rubrics FIRST, only fire UC if none of these apply to the failure:
- Verifier output is a syntax / compilation error → Syntax / Language Error, NOT UC.
- Verifier output is a shape / type / schema error → Wrong Output Schema, NOT UC.
- Verifier output is a missing-file / no-deliverable error → Silent Deliverable, NOT UC.
- Verifier output names a specific failing test / off-by-one / wrong axis → Algorithmic Bug, NOT UC.
- Trial timed out → Agent Timeout, NOT UC.
- Crash from missing dep / broken env → Environment Block, NOT UC.
- Agent ran a substantive check that returned positive → False Success Claim or Hidden-Test Regression, NOT UC.
- Agent emitted code in the wrong file / abstraction → Wrong Target / Layer, NOT UC. Only after ruling out the above do you consider UC.

Firing trigger.

- The agent's deliverable is a defensible attempt (not silently missing, not syntactically broken, not blatantly wrong format) AND the agent did NOT run a substantive verification before declaring complete AND a verification mechanism was available (sample tests, oracle script, the repo's pytest, or simply running the deliverable end-to-end).

Decision procedure.

1. Confirm none of the priority-rubrics above explain the failure.
2. Confirm the agent declared complete without running substantive verification.
3. Confirm a substantive verification was AVAILABLE (the task had tests, an oracle, or an end-to-end runnable check).

4. MATCH if 1+2+3 hold.

Exclusions.

- A more-specific rubric explains the failure: NO MATCH (use that rubric).
- Agent ran a substantive check successfully: NO MATCH (FSC or HTR).
- Agent admitted incompleteness: NO MATCH.
- Crashed/timed out before reaching verification: NO MATCH.
- Trial is short literal Q&A with no programmatic checker: NO MATCH.

Plan Over Implementation. TRUSTWORTHY

Framing. The agent narrates a CONCRETE NEXT ACTION ("I'll now write the file", "Let me modify X to do Y") OR an analysis/plan, and then the trajectory ends WITHOUT executing that action — no tool call, no file write, no patch applied. The required deliverable was never produced because the agent ran out of steps mid-plan.

Distinguishes from Silent Deliverable: SD = agent acted as if task was complete despite no artifact; PoI = agent stopped mid-plan, often with the next step explicitly described.

Decision procedure.

1. Identify the required deliverable.
2. Search the trajectory for tool calls that produced any version of the deliverable. If found: NO MATCH.
3. Inspect the agent's final message(s). MATCH if the agent describes a concrete next step ("I'll write...", "now let me modify...", "the next thing is...") OR finishes mid-analysis without ever executing the implementation step.

Exclusions.

- Deliverable was produced (even partially): NO MATCH.
- Agent declared 'task complete' or similar without an artifact: NO MATCH (Silent Deliverable).
- Agent admitted incompleteness, asked for help, or hit explicit timeout: NO MATCH.
- Crash before reaching implementation phase: NO MATCH.

Silent Deliverable. TRUSTWORTHY

Framing. The task asked for a CONCRETE ARTIFACT at a specific location (a file, a function definition, a saved answer, a written patch) and NO EVIDENCE that the artifact was produced exists in the trajectory. The agent did not deliver.

PRIORITY EXCLUSION (check FIRST, before the firing triggers below): if the agent's FINAL message describes a concrete next action that was not executed ("I'll now write the file", "now let me modify X", "the next step is to write...") AND the trajectory ends mid-step, OUTPUT 'no match' here — that case is Plan Over Implementation, NOT Silent Deliverable. Silent Deliverable fires when the agent treats the task as done despite no artifact; Plan Over Implementation fires when the agent stops mid-narrative without finishing.

Firing trigger.

- Verifier output explicitly reports the deliverable is missing / empty: phrases like 'No <file> found', '<file> does not exist', 'file not found', 'no answer.txt', 'missing output', 'output is empty', a Python `FileNotFoundError` / `KeyError` on the deliverable path, or any verifier message indicating the artifact wasn't there.
- Reward is null / missing AND no deliverable-producing tool call appears in the trajectory.
- The agent's trajectory lacks any tool call that would have written the deliverable (no Edit, no Write, no apply_patch, no bash redirect to the required path) AND the agent treated the task as complete.

Decision procedure.

1. Identify the required deliverable artifact and its location from the task instruction or verifier code.
2. Check the verifier output for an explicit missing-file complaint (any of the trigger phrases above). If present → output MATCH.
3. Otherwise, search every tool call (file write, edit, apply_patch, bash output, function call) for evidence the artifact was produced. If no such evidence exists AND the agent treated the task as complete → output MATCH.

Exclusions.

- Deliverable IS produced (even partially or wrongly) somewhere in the trajectory: NO MATCH (use a content rubric).
- Agent narrated a concrete next write step but trajectory was cut off mid-action: NO MATCH (Plan Over Implementation).
- Agent admitted the work was incomplete or asked for help: NO MATCH.
- Hard timeout/crash before any deliverable could be produced: NO MATCH. When in doubt — and any of the strong firing triggers fired — lean MATCH. This rubric is HIGH-RECALL on those signals; do not let broader rubrics (Algorithmic Bug, Plan Over Implementation) win when the artifact is genuinely missing.

Agent Timeout. TRUSTWORTHY

Framing. The trial ended because the agent hit its wall-clock or step-count budget. Agent Timeout MAY co-occur with content rubrics (Algorithmic Bug, Wrong Factual Answer, Wrong Output Schema) when the agent's partial output also has issues — in those cases, BOTH rubrics should be shortlisted, not just one.

Firing trigger.

- Verifier output / error fields contain `AgentTimeoutError`, `TimeoutError`, 'agent execution timed out', 'agent timeout exceeded', 'hit Ns timeout', 'timed out after N seconds'.
- Trial runtime is within ~5% of the agent's timeout budget.
- The agent's final message is mid-action ("I'll now write...", "let me run...", "next I'll...") and the trajectory was cut off without execution.
- `error_type` / `terminate_reason` indicates timeout / wall-clock / cut-off.
- `exception_info` / traceback mentions timeout-related exceptions. When ANY trigger fires, output MATCH for Agent Timeout. Note: this can co-occur with a content rubric — if the agent's partial output ALSO has a clear bug or wrong answer, shortlist BOTH (Agent Timeout AND the content rubric). The SHORTLIST allows up to 3 rubrics; co-occurring AT + content is common.

Decision procedure.

1. Inspect verifier_output / error fields / exception_info / final agent message / runtime data for ANY timeout indicator.
2. If found, output MATCH for AT. Quote the timeout signal.
3. If you'd ALSO have fired a content rubric (AB, WFA, WOS) on the partial output, you may shortlist that one as a co-rubric — but AT must be in the shortlist.

Exclusions.

- Trial finished naturally (agent declared done, no timeout signal): NO MATCH.
- Crash from missing dep / broken env: NO MATCH (Environment Block — different failure mode).
- Verifier timed out, not the agent: NO MATCH.

Environment Block. TRUSTWORTHY

Framing. Agent attempted its work in good faith but hit a real environment constraint — missing data, blocked network, broken dependency, OR a required capability that the task scaffold / adapter did not install. Includes adapter-induced under-tooling: e.g., a figure-QA task where the adapter ships raw PNGs but no OCR / vision library.

Decision procedure.

1. Scan for environment-failure signals:
   - "network is unreachable", "Connection refused", "403 Forbidden", "404 Not Found"
   - "package not found", "pip install failed", "ImportError: No module"
   - Missing file / table / input / credential
   - Agent tries vision/ASR/OCR/search tools that aren't installed (tesseract, pillow, cv2, whisper, curl)
   - dbt source tables / DB schemas not present
2. Check that the agent attempted to use a legitimate means of accomplishing the task and got blocked — not merely skipped a step it could have done.
3. MATCH if (a) a clear environment constraint documented AND (b) agent didn't work around it AND (c) no reasonable workaround was available in the scaffold.

Exclusions.

- The "environment issue" is actually agent misuse (wrong command, wrong path): NO MATCH.
- Agent worked around the constraint, failed for other reasons: NO MATCH.
- Agent timed out trying to resolve: NO MATCH (Agent Timeout).

Excluded rubrics ($\kappa < 0.5$, folded into "Others").

Calculation Mismatch. EXCLUDED

Framing. Tasks asking for a numerical result where the agent ran a pipeline to completion and produced a concrete number, but the value is outside the grader's tolerance.

Decision procedure.

1. Task asks for numerical / statistical result.
2. Agent ran a real pipeline: loaded data, applied a method, computed a number.
3. Agent wrote the resulting number to the required location.
4. reward=0, and failure is not a code crash / timeout / missing file.
5. MATCH if 1-4 hold.

Exclusions.

- Agent guessed without a pipeline: NO MATCH (Wrong Factual Answer).
- Pipeline crashed: NO MATCH (Syntax / Algorithmic).
- Task required symbolic, agent gave numerical: NO MATCH.

Rule Inference Error. EXCLUDED

Framing. The task tests the agent's ability to infer / apply a rule, scheme, mapping, or pattern from given examples. The agent demonstrably misunderstood or mis-extrapolated the rule itself. This is a content failure of REASONING, not coding, not retrieval, not formatting.

Decision procedure.

1. Confirm the task is rule-inference style: visual puzzle, IQ-puzzle, pattern-completion, instruction-following with explicit examples-then-apply structure.

- If the task is straightforward Q&A without explicit examples to generalize from: NO MATCH (Wrong Factual Answer).
- If the task is code-writing: NO MATCH.

2. Quote the specific rule the agent stated or applied, and the specific way it diverges from the correct rule (visible from the verifier output or oracle solution).
3. MATCH only when steps 1 + 2 hold and the failure is in rule inference / application — not other modes. Default: when in doubt, output 'no match'.

Specification–Verification Mismatch. EXCLUDED

Framing. The agent's deliverable is a DEFENSIBLE READING of the user-facing instruction, but the verifier rejects it because the verifier enforces a constraint NOT stated (or genuinely under-specified) in the instruction. Examples: verifier requires a specific function name not in the prompt; verifier requires a hardcoded path not in the prompt; verifier insists on the exact prose phrasing the prompt only suggested.

Decision procedure.

1. Locate the user-facing instruction (instruction.md / first user message in <task_context>).
2. Locate the verifier's actual check (verifier_code in <task_context> / verifier_output / test_sh).
3. Identify the SPECIFIC requirement the verifier enforced that caused the failure. Quote the exact failing assertion / expected value.
4. Compare against the instruction. MATCH only if BOTH: (a) The agent's deliverable reasonably satisfies the instruction as a competent reader would understand it, AND (b) The verifier's failing requirement is genuinely absent or contradictory in the instruction.

Exclusions.

- The verifier's requirement IS present in the instruction (even if obscure): NO MATCH.
- The agent's output is wrong by any plausible reading of the instruction: NO MATCH (use the appropriate content rubric).
- Agent didn't produce the deliverable at all: NO MATCH.
- Mismatch is purely formatting: NO MATCH (Wrong Output Schema / Wrong Output Location).
- Mismatch is about hidden tests probing behavior the prompt described: NO MATCH (Hidden-Test Regression).
- You cannot quote a specific verifier requirement absent from the instruction: NO MATCH. This rubric is for genuine UNDER-SPECIFICATION cases. Do not output 'match' merely because the agent failed and the verifier was strict. Default: when in doubt, output 'no match'.

Insufficient Web Research. EXCLUDED

Framing. Insufficient research applies to open-web / factual QA tasks where the agent SHOULD HAVE retrieved external information. The failure is that the retrieval was either absent when retrieval was feasible, single-sourced when corroboration was needed, or through a wrong channel.

Decision procedure.

1. Confirm the task required web research.
   - Internal tasks (code / math / local data): NO MATCH.
2. Check whether retrieval was FEASIBLE — did the scaffold provide web_search / browser / curl to external URLs?
   - Evidence of a shipped search tool: proceed.
   - No search tool available, agent had nothing to call: NO MATCH (route to Environment Block).
3. Inventory retrieval actions. MATCH if any of: (a) Search tool was available but agent made zero retrieval attempts; answered from training knowledge (b) Exactly one source consulted on an

accuracy-sensitive question without corroboration (c) Retrieval attempts uniformly blocked (403/404) and agent proceeded anyway despite alternative channels (d) Used bash-scraping when a proper web tool was available

Exclusions.

- $\geq$ 2 lookups with reconciliation, even if final answer is wrong: NO MATCH (Wrong Factual Answer).
- Tasks where training knowledge is plausibly sufficient: NO MATCH.
- Scaffold provided no retrieval tool: NO MATCH (Environment Block).

Premature Termination. EXCLUDED

Framing. Agent narrates a concrete next action ("I'll now write the file") in a final message, and the trajectory ends before the narrated action is executed. Often a scaffold artefact (e.g., codex+gpt-5-mini end-of-turn interpreted as `task_complete`).

Decision procedure.

1. Inspect the last 1-3 agent messages. Look for forward-looking commitment.
   - Final message is a conclusion ("done", "here is X") without forward-looking verb: NO MATCH.
2. Check whether the narrated action was actually executed afterward (tool call, file write).
3. MATCH if (a) concrete next-action narrated AND (b) no corresponding tool action before trajectory ended.

Exclusions.

- Narrated and executed successfully: NO MATCH.
- Timed out mid-execution: NO MATCH (Agent Timeout).
- Never narrated anything concrete: NO MATCH (Silent Deliverable).

Wrong Output Location. EXCLUDED

Framing. Agent produced correct content but wrote it to the wrong path, wrong filename, wrong cell range.

Decision procedure.

1. Identify the required output location.
2. Observe where the agent actually wrote.
3. MATCH if content was written AND location deviates from required.

Exclusions.

- Never written: NO MATCH (Silent Deliverable).
- Correct location but wrong format: NO MATCH (Wrong Output Schema).
- Correct location but wrong content: NO MATCH.

G.2.2 Agent Failure Mode Full Results

Table 18 reports per-rubric failure-mode prevalence for all six (model, harness) combinations across $N = 6{,}028$ hard-task trajectories. The six reported rubrics are those with per-rubric judge–gold $\kappa \geq 0.5$; the remaining six rubrics (Wrong Target / Layer, Agent Timeout, Unverified Claim, Plan Over Implementation, False Success Claim, Wrong Output Schema) are aggregated into Others.

Table 18: Per-rubric failure-mode prevalence (%) for all six (model, harness) combinations. WFA = Wrong Factual Answer, AB = Algorithmic Bug, HTR = Hidden-Test Regression, SD = Silent Deliverable, EB = Environment Block, Syn = Syntax / Language Error. Others aggregates six rubrics with judge–gold $\kappa < 0.5$. N is the number of hard-task trials.

Model	Harness	N	WFA	AB	HTR	SD	EB	Syn	Others
GPT-5.4	Codex	991	27.3	20.2	9.6	1.4	3.5	0.6	17.9
GPT-5.4	Terminus-2	983	26.6	18.8	4.9	4.9	9.3	1.4	18.2
Opus 4.6	Claude Code	973	26.4	22.5	6.9	4.1	2.9	0.8	17.7
Opus 4.6	Terminus-2	974	23.2	17.7	10.0	5.0	4.2	1.4	24.3
Gemini 3.1 Pro	Gemini CLI	995	25.0	17.7	8.7	6.9	3.9	1.2	16.7
Gemini 3.1 Pro	Terminus-2	973	24.5	15.3	8.8	7.3	4.1	1.4	17.9

Reading the table. A cell reports the fraction of trials in that (model, harness) row where the rubric fires; because the judge may assign multiple rubrics to a single trial, row percentages do not sum to 100%. The capability rubrics (WFA, AB, HTR) are broadly conserved across harnesses within each model family, while the operational rubrics (SD, EB) show harness-dependent variation. Environment Block is notably elevated for GPT-5.4 under Terminus-2 (9.3% vs. 3.5% under Codex), consistent with the protocol-incompatibility pattern described in §3.4. Silent Deliverable is highest on Gemini CLI (6.9%) and Opus 4.6 under Terminus-2 (5.0%), reflecting differences in submission-protocol enforcement across harnesses.

H Harbor-Index Selection Pipeline

This appendix documents the per-cell sampling protocol used to derive the difficulty filter (Section H.1), the full task-quality rubric used by both the AI auditor and human reviewers (Section H.2), the judge orchestration and prompt (Section H.3), the human-review protocol (Section H.4), and the audit-and-fix stage that yields the final 82-task release (Section H.5).

H.1 Sampling Protocol

Each candidate task is summarized by an 18-trial sample drawn from the live Harbor trial database. We define the *filtering mix* as the Cartesian product of three leading model families at the time of filtering and two harness configurations:

- **Models:** Claude Opus 4.6, GPT-5.4, Gemini 3.1 Pro.
- **Harnesses:** each model's native harness (Claude-Code for Anthropic, Codex for OpenAI, Gemini-CLI for Google), plus the cross-vendor Terminus-2 agent.

This yields 6 $(\text{model}, \text{harness})$ pairs. For each pair we keep 3 valid trials, giving 18 trials per task.

Difficulty filter. A task passes the filter iff $n_{\text{succ}} \leq 6$ over its 18-trial sample, where n_{succ} denotes the number of successful trials (i.e., the success rate over the filtering mix must be at most 33%). Success is benchmark-aware. For most benchmarks, a trial counts as a success iff reward > 0. For benchmarks whose reward semantics differ from a binary pass/fail, we follow the per-benchmark scoring cutoff established by the main experiment.

H.2 Task Quality Bar

The auditor and human reviewers evaluate each candidate against two criteria. A task is rejected if it fails either; a reviewer may also mark a task *unsure* when the evidence is mixed. The same rubric is shared between the AI judge and the human review rounds so that the two stages enforce a consistent definition of task quality.

Test–instruction alignment. Every test assertion must trace back to a requirement stated in the instruction or implied by the environment, and every requirement in the instruction must have corresponding test coverage. The criterion fails when:

- the verifier checks something the instruction never asks for or implies, such as a specific numerical value, exact error wording, file format, or schema;
- the instruction is ambiguous or under-specified about the required output while the verifier asserts a single specific answer;
- multiple correct interpretations of the instruction would each produce a different verifier outcome;
- the agent gets the right idea but the verifier rejects it on a clerical detail not foreshadowed by the instruction.

Essential difficulty. Difficulty must come from genuine reasoning, algorithmic thinking, domain expertise, long-horizon interaction, or multi-step execution, not from minor formatting details, arbitrary precision, ambiguous output schemas, or clerical detail. The criterion fails when trials repeatedly show models arriving at the right idea but being tripped up by:

- exact whitespace, decimal precision, or output column ordering not specified in the instruction;
- magic strings (function names, error messages, JSON keys) the instruction does not pin down;
- tolerance bounds the verifier applies that the instruction does not disclose.

Calibration set. The auditor receives a small set of pre-labeled rejected tasks as in-context calibration examples (e.g., a SWE-bench Multilingual task whose verifier outcome depends on wall-clock year; a SWE-bench Pro task whose verifier requires the literal wording of an `AnsibleError` not pinned down by the PR; a Spider2 task whose gold output was generated by a buggy Haversine formula and whose verifier penalizes correct fixes). Tasks exhibiting similar smells, such as verifier–instruction mismatch, fragile string equality on novel content, under-specified output format, or gold that does not pass the verifier, are rejected by analogy.

H.3 Task Audit Process

Setup. The AI auditor used at the second pipeline stage is based on Gemini-3-Flash, run inside a per-task Daytona sandbox via Claude Code in non-interactive print mode. The orchestrator packages, per task, a tarball containing the rubric, a corpus JSON file (18 trials with verifier stdouts and ATIF trajectory paths), the trajectories themselves, and the per-trial verifier stdouts; uploads it to a fresh sandbox; runs the audit; and downloads the resulting verdict. The judge is granted its native filesystem and shell tools (`Read`, `Grep`, `Bash`, `Write`) and is required to consult between two and four trajectories per task—mixing successes, failures, and exception cases—and to cite specific step IDs, tool calls, and observations in its rationale. The verdict is emitted as a single JSON object written to a file via the `Write` tool; a Python fallback parses an inline JSON object from the streamed assistant turn if the tool call is missing.

Cross-judge calibration. We ran the same harness with Opus-4.7 as the auditor model over a 74-task subset of the candidate pool to gauge inter-judge reliability against Gemini-3-Flash. The two judges agree on $66/74 = 89.2\%$ of decisions across the three-way {accept, reject, unsure} label space, with Cohen's $\kappa = 0.747$ (substantial agreement). Collapsed to the operationally relevant binary {accept, non-accept}, agreement rises to $69/74 = 93.2\%$: the two judges agree on accepts (16) and non-accepts (53) together for 69 of the 74 shared tasks, with the five disagreements concentrated in tasks Opus-4.7 marked *accept* but Gemini-3-Flash marked *reject*. Thus, the cheaper `gemini-3-flash-preview` judge is more conservative on this subset: there are no cases where it accepts a task that `opus-4.7` rejects. We use `gemini-3-flash-preview` as the production auditor on this basis.

Verdict schema. Every verdict is a JSON object with the following fields: `decision` (one of *accept*, *reject*, *unsure*); `primary_reasons` (short tags); `rubric` (per-criterion verdict and evidence); `failure_attribution` (*task_design*, *agent_weakness*, *infra_or_experiment*, or *mixed*); `trial_comparison_notes` (cross-trial observations grounded in trajectory step IDs); `trajectories_consulted`; `argument_reasoning_evidence` (detailed argumentative reasoning grounded in cited code lines, commands, or agent steps); and `summary` (a one-paragraph rationale).

Prompt. We provide the full judge prompt in Listing 1.

```
# Task quality audit

You are an expert reviewer evaluating the quality of tasks.

## What you receive

A single corpus JSON file describing one candidate task with:
- benchmark, task_path, n_succ (= number of successful trials out of 18
  frontier-model attempts; range 0-6 for the bulk pool, may be higher for
  under-rep fill-ins)
- task_files: full text of instruction.md, task.toml,
  environment/Dockerfile, solution/solve.sh, tests/test.sh
- trials: each with model, agent, reward, exception_type, test_stdout
  (verifier stdout, head/tail-clipped to ~32 KB), and trajectory_path
  (path to ATIF agent trajectory JSON)

You may also be given a task_dir path and file-system access to read
additional files in that directory if needed.

## Your job

Decide accept, reject, or unsure on the task based on quality audit. We
accept a task if it passes the rubrics that demonstrate high quality; we
reject a task if it has noticeable problems. Mark "unsure" if you are not
confident in your judgment. List detailed reasoning, evidence, and
arguments along with your final verdict.

## Quality rubric

### 1. test_instruction_alignment
Every test assertion must trace back to a requirement stated in
instruction.md or implied by the environment, and every requirement in
the instruction must have corresponding test coverage. Tests should NOT
introduce requirements beyond what the instruction describes. FAIL if:
- The verifier checks something the instruction never asks for or never
  implied (e.g., a specific numerical value, exact error wording, file
  format, or schema not stated in the instruction).
- The instruction is ambiguous or under-specified about what the agent
  should produce, while the verifier asserts a specific answer.
- Multiple correct interpretations of the instruction would each produce
  a different verifier outcome.
- The agent gets the right *idea* but the verifier rejects them on a
  clerical detail not foreshadowed by the instruction.

### 2. essential_difficulty
Difficulty must come from genuine reasoning, algorithmic thinking, domain
expertise, long-horizon interactions, multi-step execution, etc -- NOT
from formatting minutiae, arbitrary precision, ambiguous output schemas,
or clerical detail. FAIL if the trials show models repeatedly arriving
at the right idea but getting tripped up on:
- Exact whitespace / decimal precision / output column ordering not
  specified in instruction.
- Specific magic strings (function names, error messages, JSON keys) the
  instruction doesn't pin down.
- Tolerance bounds the verifier applies that the instruction doesn't
  disclose.

## How to use trial data

You have several test_stdouts plus task files. Use them to ground your
judgment:

1. Compare successful vs failed trial test_stdouts. What does the success
   pattern look like -- was it luck on a fragile verifier, or genuine
   task completion following instruction? What did the failures actually
   fail on -- insufficient domain knowledge and/or task understanding,
```

```
   wrong solution idea/execution/calculation, formatting, timeout, infra?
   If unclear, read the trajectories.
2. Compare successful trial output to solve_sh. NOTE that solve_sh is a
   rough reference solution -- it does not mean that the agent or "only
   correct" solution should do the same thing to complete the task. It
   is only material for you to compare a potentially correct approach
   against agent approaches. All your judgment should still be grounded
   in the task description and test verifiers.
3. Look for infra/experiment failures masquerading as task failures.
   Patterns to flag (mark as unsure):
   - 429 Too Many Requests, rate limiting from model API.
   - No space left on device, OOM errors mid-execution.
   - Network errors / DNS / proxy connect failures.
   - Container build failures unrelated to the task design.
   - Truncated transcripts due to step-count exhaustion.
   These are experiment issues, not task issues. If 5 of 6 trials failed
   for infra reasons and 1 succeeded cleanly, the task may still be
   high-quality -- accept if other criteria pass.

## Output format

Output a single JSON object (no preamble, no trailing markdown):

{
  "decision": "accept" | "reject" | "unsure",
  "primary_reasons": ["short tag", ...],
  "rubric": {
    "test_instruction_alignment": {"verdict": "pass"|"fail"|"na",
                                   "evidence": "..."},
    "essential_difficulty": {"verdict": "pass"|"fail"|"na",
                                   "evidence": "..."}
  },
  "failure_attribution": "task_design" | "agent_weakness" |
                         "infra_or_experiment" | "mixed",
  "trial_comparison_notes": "What you noticed comparing trajectories and
    test_stdouts (successes vs failures, cross-model). Cite specific
    step IDs / tool calls / observations from at least one trajectory
    you rendered.",
  "trajectories_consulted": ["<trial_id_1>", "<trial_id_2>", ...],
  "argument_reasoning_evidence": "Show your judgment of the task quality.
    Claim your arguments. Show detailed reasoning with concrete evidence
    (specific to words/commands/code-lines/agent-steps if helpful) to
    demonstrate. If you claim the task to be 'unsure', show what you
    are confident and unconfident about respectively, and what you want
    human reviewers to double-check.",
  "summary": "One-paragraph rationale for the decision, citing specific
    files/lines/strings."
}
```

Listing 1: Harbor-Index task-quality judge prompt (`judge_prompt.md`).

H.4 Human Review Protocol

Human review is conducted by a pool of 14 domain-experienced reviewers using the same task-quality rubric as the AI auditor. The process has three stages:

1. **Initial quality screen.** A task handled by a senior reviewer receives at least one review; a task handled by junior reviewers receives at least two reviews. Reviewers inspect the task specification, environment, verifier, and available execution evidence, and may accept, flag for repair, reject, or escalate an uncertain case. This screen leaves more than 110 candidates.
2. **Panel selection.** A panel of three senior reviewers discusses the surviving candidates and selects an intermediate pool of 100 based on task quality, difficulty, diversity, and the insight offered by model behavior.
3. **Trajectory-grounded audit.** Each of the 100 selected tasks is re-examined by at least two senior reviewers. Reviewers inspect trajectories and verifier outcomes, perform failure analysis,

and distinguish true and false positives and negatives. They repair fixable task defects and remove tasks that remain broken or become too easy after repair, producing the final 82-task release.

This process is an iterative engineering curation workflow rather than a fixed-panel annotation study: the review method was refined during construction, and reviewer counts therefore vary by task. We report the minimum review coverage and adjudication process above rather than a single human inter-rater-agreement statistic. Additional audit examples and task-level artifacts are available at `https://harbor-index.org`.

H.5 Audit-and-Fix Loop

The senior panel's initial cut yields an intermediate pool of 100 candidate tasks. We then run repeated rounds of automated audit and reviewer repair: an LLM judge grades the verifier in a fresh copy of each task's sandbox by reading agent trajectories, verifier outputs, tests, and the reference solution, and labels each rollout as a true/false positive or negative with citations to specific steps or files. We prioritize false positives that let agents pass without solving the task and false negatives that reject correct solutions. Tasks are repaired (or dropped when irreparable), the filtering-mix models are re-run, and the audit repeats until the set stabilizes. Tasks that remain broken or become too easy after repair are removed, leaving the final Harbor-Index 1.0 release of **82 tasks across 29 benchmarks**. We also tighten per-task timeouts in this stage and adopt Harbor features such as separate verifier sandboxes to close shared-container reward hacks across the suite.

Harbor-Index 1.0 Task Catalog

Table 19 summarizes the 82 release tasks by domain and benchmark. Task IDs and interactive per-task scores for all 1,476 rollouts are published at `https://harbor-index.org` and on the Harbor Hub.

I Limitations

Apart from the limitations listed in Section 6, we discuss extended limitations here.

Evaluation cost and scalability. Large-scale agent evaluation remains expensive. Our study uses substantial compute resources and token budgets, which may limit reproducibility for smaller research groups. While Harbor-Index reduces cost, it is still a proxy for the full benchmark distribution.

LLM-as-a-judge limitations. Some tasks rely on automated evaluation using language models as judges. Even though we aggregate the evaluation from 4 different models from different model providers and repeated runs for each model, these evaluations may still introduce bias, variance, or systematic errors, especially for open-ended or subjective tasks.

Ethical considerations. Our work focuses on evaluation infrastructure and does not directly address downstream societal impacts such as misuse of agents or deployment risks. Future work should consider safety, alignment, and governance aspects of agentic systems.

Addressing these limitations is an important direction for future work, including expanding benchmark diversity, improving adapter verification, reducing evaluation cost, and designing more realistic and dynamic evaluation protocols.

J Broader Impacts

This work introduces Harbor Adapters and Harbor-Index, aiming to improve the scalability, reproducibility, and coverage of agentic evaluation. We discuss potential positive impacts as well as risks and mitigation strategies.

Positive impacts. Our infrastructure lowers the barrier to evaluating language-model agents across diverse benchmarks, promoting more comprehensive and reproducible comparisons. By enabling large-scale and standardized evaluation, this work may help the community better understand

model capabilities, identify failure modes, and avoid overfitting to a small set of popular benchmarks. In the long term, improved evaluation can contribute to the development of more reliable and robust AI systems in domains such as software engineering, scientific research, and data analysis.

Risks and potential misuse. Standardized evaluation frameworks may also accelerate competitive benchmarking and optimization, potentially encouraging overfitting to benchmark suites rather than real-world performance. In addition, improved evaluation infrastructure could indirectly support the development of more capable autonomous agents, which may raise concerns around misuse, including automation of harmful tasks, large-scale exploitation, or deployment without sufficient safeguards.

Mitigation and responsible use. We partially mitigate these risks by emphasizing compactness, diversity, difficulty, and quality in Harbor-Index construction, and by releasing detailed analyses of failure modes. We encourage users to interpret results cautiously, avoid over-reliance on single benchmarks, and complement our evaluation with domain-specific and real-world testing. Future work should further incorporate safety, alignment, and governance considerations into agent evaluation frameworks.

Overall, we believe that improving evaluation infrastructure is a necessary step toward building more transparent, reliable, and accountable AI systems, but it must be accompanied by responsible use and continued scrutiny.

Table 2: 93 Benchmarks integrated via Harbor adapters, organized by domain and subdomain. Colors mark agentic character: **agentic**, **modified-agentic**, **non-agentic**.

Domain	Subdomain	Benchmarks
Software Engineering	*Repo-level Issue Resolution and Testing*	ABC-Bench [181], CooperBench [68], DevEval [80], Multi-SWE-Bench [196], SWE-Bench-Multilingual [64], SWE-Bench Pro [33], SWE-bench Verified [64], SWE-Gym [120], SWE-rebench [7], SWE-smith [183], SWT Bench [113]
	Feature Development	FeatBench [18], FeatureBench [206], ProgramBench [184], SWE-Lancer [106], WebGen-Bench [96]
	Competitive & Function-level Coding	Aider Polyglot [2], AutoCodeBench [204], BigCodeBench [210], EvoEval [176], Frontier-CS-Algorithm [102], LiveCodeBench [62], USACO [143]; CanItEdit [15], HumanEvalFix [111], QuixBugs [87], UniCode [203]
	Performance Optimization	GSO [142]; AlgoTune [128]
	Language Translation	CRUST-Bench [67]
	DevOps & Build Systems	CompileBench [132], DevOpsGym [154]
Mathematics & Reasoning	*Competition Mathematics*	AIME [191], IneqMath [94], Omni-Math [46]
	Abstract & Procedural Reasoning	SATBench [170]; ARC-AGI-2 [23], KUMO [88], Reasoning Gym [151]
Knowledge & Long Context	*Expert & Multi-subject QA*	CL-Bench [35], GPQA Diamond [135], Humanity's Last Exam [126], MMMLU [55], SimpleQA [171]
	Long-Context Reasoning	AA-LCR [155], LoCoMo [100]
Scientific Research	*End-to-end Research Workflows*	AstaBench [12], ML-Dev-Bench [119], MLR-Bench [19], ReplicationBench [190], RExBench [39], ScienceAgentBench [21], SLDBench [89]; MLGym-Bench [114]
	Scientific Computing	LLM-SRBench [145]; CodePDE [83], ResearchCodeBench [59], SciCode [157]; QCircuitBench [185]
	Biomedical Research	BIX-Bench [107]; LAB-Bench [78]
Agents, Tools & Systems	*Tool Use & Assistants*	ACE-Bench [17], GAIA [104], GAIA2 [45], LongCLIBench [43], OSWorld [179], Tau3-Bench [187]; AMA-Bench [201], BFCL [121]
	Deep Research & Web Agents	DeepResearch-Bench-II [37], DeepSynth [123], Seal-0 [125], WideSearch [173]
	Interactive Games	TextArena [50]
Data & Analytics	*Text-to-SQL & Data Science*	ADE-Bench [150], DA-Code [61], DABstep [40], KramaBench [75]; BIRD-Bench [81], DS-1000 [76], Spider 2 [79]
Professional Domains	*Finance & Trading*	FinanceAgent [162]; PIXIU [178]
	Business / Professional Work	CRMArena [60], SpreadsheetBench [98], TheAgentCompany [180]
	Healthcare & Clinical	MedAgentBench [63]
	Legal	LawBench [42]
Safety & Security	*Cybersecurity*	CyberGym [169]
	Jailbreak Robustness	StrongReject [148]
Multimodal	*Audio Understanding*	MMAU [140]
	Graphic / Visual Design	GraphDesignBench [32]
	Autonomous Driving & Sensor Reasoning	RefAV [27]

Table 3: Harbor adapter catalog (Part 1 of 2): Software Engineering, Mathematics & Reasoning, and Knowledge & Long Context. **Sc.**: integration scenario (1 = Harbor-Agent, 2 = Vanilla-LLM, 3 = Custom-Agent). **Mode**: agent interaction mode (A = Agentic, M = Modified-Agentic, N = Non-Agentic). k: number of parity runs. **Set**: $parity/full$ task count. **Orig. / Harbor**: pass rate (%) as mean ± SEM over k runs.

Benchmark	Subdomain	Sc.	Mode	Parity Agent	Parity Model	k	Set	Orig.	Harbor
SWE-Bench-Multilingual	Repo Issues	2	A	codex	gpt-5-mini	3	50/300	33.3±0.7	33.3±3.3
SWE-Bench Pro	Repo Issues	2	A	codex	gpt-5-mini	6	100/731	35.7±0.6	36.3±1.0
SWE-bench Verified	Repo Issues	1	A	mini-swe-agent	gpt-5-mini	3	499/500	56.3±0.0	54.5±0.7
SWE-smith	Repo Issues	1	A	mini-swe-agent	claude-3-haiku	–	100/100	0.0	0.0
SWT Bench	Repo Issues	1	A	claude-code	haiku-4-5	–	433/433	20.55±0.0	20.55±0.0
CooperBench	Feature Dev	1	A	openhands-sdk	gemini-3-flash	3	50/652	32.7±1.3	30.7±1.3
FeatBench	Feature Dev	1	A	trae-agent	deepseek-v3.2	3	156/156	49.8±2.3	49.4±1.1
FeatureBench	Feature Dev	1	A	codex	gpt-5-mini	2	30/200	13.3±0.0	15.0±1.7
SWE-Lancer	Feature Dev	2	A	claude-code	claude-sonnet-4	5	463/463	48.4±1.0	47.6±0.4
WebGen-Bench	Feature Dev	2	M	aider	gpt-5-mini	3	101/101	16.7±0.9	16.2±1.6
Aider Polyglot	Coding	2	N	claude-code	claude-3-haiku	1	225/225	2.7±0.0	2.7±0.0
BigCodeBench	Coding	2	M	codex	gpt-5-mini	3	145/145	32.7±0.4	34.0±1.6
FrontierCS	Coding	2	M	claude-code	opus-4-6	3	10/172	68.9±11.5	53.4±9.9
LiveCodeBench	Coding	2	N	claude-code	haiku-4-5	4	100/1055	54.5±1.5	53.3±1.0
USACO	Coding	2	N	claude-code	haiku-4-5	3	304/304	40.3±0.4	43.0±1.4
HumanEvalFix	Coding	1	N	openhands	gpt-5-mini	3	164/164	98.2±0.6	98.0±0.4
QuixBugs	Coding	2	N	codex	gpt-5-mini	3	80/80	88.4±0.4	87.9±0.4
GSO	Perf. Opt.	1	A	openhands	gpt-5.1	2	102/102	13.7±1.0	13.2±1.5
AlgoTune	Perf. Opt.	2	A	terminus-2	gpt-5-mini	3	154/154	1.23±0.02	1.23±0.02
CRUST-Bench	Lang. Trans.	2	M	codex	gpt-5-nano	3	100/100	58.3±0.7	58.3±1.2
DevOpsGym	DevOps	1	A	codex	gpt-5-mini	3	50/733	22.7±1.8	22.0±0.0
AIME	Comp. Math	2	N	codex	gpt-5-nano	–	30/30	N/A	N/A
IneqMath	Comp. Math	2	N	codex	gpt-4o-mini	3	100/100	50.0±1.5	52.0±0.7
Omni-Math	Comp. Math	2	N	codex	gpt-5.3-codex	2	100/4428	80.0±2.0	81.0±2.0
ARC-AGI-2	Abs. Reason.	2	N	codex	gpt-5.2	6	167/167	36.0±0.8	35.8±0.8
KUMO	Abs. Reason.	2	N	kumo-vanilla	gpt-5-mini	3	212/5300	88.7±0.7	89.9±0.7
Reasoning Gym	Abs. Reason.	2	N	codex	gpt-5.1-codex-mini	3	576/576	85.7±0.4	85.9±0.3
GPQA Diamond	Expert QA	2	N	codex	gpt-5.2	3	198/198	87.9±0.6	87.2±0.3
Humanity's Last Exam	Expert QA	2	N	claude-code	haiku-4-5	3	249/2500	10.7±0.9	11.0±0.4
MMMLU	Expert QA	2	N	codex-cli	gpt-5.1-codex-mini	5	150/210630	63.5±0.5	63.3±0.5
SimpleQA	Expert QA	2	N	claude-code	opus-4-6	3	50/4326	96.7±1.8	94.7±0.7
AA-LCR	Long Ctx.	2	N	codex	gpt-5-mini	1	99/99	68.0	68.7

Table 4: Harbor adapter catalog (Part 2 of 2): Scientific Research, Agents/Tools/Systems, Data & Analytics, Professional Domains, Safety & Security, and Multimodal. Columns as in Table 3.

Benchmark	Subdomain	Sc.	Mode	Parity Agent	Parity Model	k	Set	Orig.	Harbor
ReplicationBench	Research WF	1	A	claude-code	haiku-4-5	1	90/90	14.4	14.4
ScienceAgent Bench	Research WF	1	A	claude-code	haiku-4-5	5	102/102	33.7±0.9	31.8±1.2
SLDBench	Research WF	1	A	claude-code	haiku-4-5	5	8/8	38.1±9.5	39.2±10.7
MLGym-Bench	Research WF	2	M	mini-swe-agent	gpt-5	3	12/12	80.6±2.8	80.6±2.8
CodePDE	Sci. Comp.	2	M	terminus-2	haiku-4-5	8	5/5	27.5±3.7	27.5±3.7
ResearchCode Bench	Sci. Comp.	2	M	codex	gpt-4.1-mini	3	212/212	41.0±0.4	40.7±0.3
SciCode	Sci. Comp.	2	M	codex	gpt-5.1-codex-mini	3	80/80	43.3±0.6	43.8±0.6
QCircuitBench	Sci. Comp.	2	N	codex	gpt-5.2	3	28/28	60.9±2.5	61.9±2.1
BIX-Bench	Biomed.	3	A	bixbench-agent	gpt-4o-mini	3	50/205	16.0±3.1	16.7±2.4
LAB-Bench	Biomed.	2	N	codex	gpt-5-codex	3	181/181	40.0±0.2	41.1±0.5
ACE-Bench	Tool Use	1	A	claude-code	haiku-4-5	3	973/973	83.6±0.4	83.2±0.8
GAIA	Tool Use	1	A	openhands	gpt-5-mini	3	165/165	51.3±0.7	50.7±0.5
GAIA2	Tool Use	3	A	gaia2-parity-agent	gpt-5-mini	2	100/800	6.0±1.0	6.0±0.0
BFCL	Tool Use	2	N	codex	gpt-5-mini	3	123/3641	83.2±0.7	83.5±1.0
DeepSynth	Web Agents	1	A	claude-code	haiku-4-5	3	40/40	9.3±1.0	7.8±0.6
Seal-0	Web Agents	1	A	claude-code	haiku-4-5	3	111/111	33.9±3.0	33.3±3.6
WideSearch	Web Agents	1	A	claude-code	haiku-4-5	3	200/200	53.8±0.3	53.9±0.3
DA-Code	SQL & DS	3	A	da-agent	gpt-4o	3	479/479	38.7±1.1	38.9±0.2
Spider 2	SQL & DS	1	A	spider-agent	gpt-5-mini	3	64/64	18.8±0.0	18.3±0.5
FinanceAgent	Finance	3	A	finance-agent	gpt-5.2	3	50/50	78.0±1.0	80.0±0.0
PIXIU	Finance	2	M	codex	gpt-5-mini	3	435/54083	90.1±3.3	85.2±6.4
CRMArena	Business	3	A	crmarena-react	haiku-4-5	3	90/1170	69.0±1.0	68.0±2.0
SpreadsheetBench	Business	1	A	claude-code	haiku-4-5	3	400/400	68.8±0.8	68.3±1.1
MedAgentBench	Healthcare	3	A	medagent-react	gpt-4o-mini	3	300/300	58.0±0.9	57.9±0.3
LawBench	Legal	2	N	qwen-code	glm-4.7	3	120/1000	65.0±0.3	65.6±0.7
CyberGym	Cybersec.	3	A	openhands	haiku-4-5	5	10/1507	64.0±4.0	66.0±2.5
StrongReject	Jailbreak	2	N	codex	gpt-5-nano	3	150/12207	95.0±0.7	95.0±1.1
MMAU	Audio	2	N	terminus-2	gpt-4o	3	1000/1000	56.6±0.8	56.6±0.8

Table 6: GPT-series models under Terminus-2 and Codex across 54 benchmarks.

Benchmark	GPT-5.4		GPT-5-mini		GPT-5-nano	
	Terminus-2	Codex	Terminus-2	Codex	Terminus-2	Codex
SWE-bench Verified	0.690	0.713	0.307	0.460	0.160	0.110
SWE-Bench-Multilingual	0.607	0.700	0.213	0.280	0.047	0.033
SWE-smith	0.031	0.221	0.010	0.296	0.065	0.051
CRUST-Bench	0.920	0.980	0.620	0.803	0.327	0.113
GSO	0.167	0.373	0.007	0.095	0.000	0.010
SWE-Bench Pro	0.347	0.550	0.100	0.400	0.053	0.057
SWT Bench	0.540	0.707	0.147	0.307	0.073	0.147
Aider Polyglot	0.544	0.785	0.367	0.539	0.141	0.096
AlgoTune	0.166	0.275	0.130	0.144	0.050	0.009
CompileBench	0.933	1.000	0.533	0.644	0.400	0.133
FeatureBench	0.056	0.368	0.034	0.103	0.009	0.004
BigCodeBench-Hard	0.395	0.428	0.352	0.283	0.244	0.034
USACO	0.830	0.960	0.843	0.857	0.713	0.057
SWE-Lancer	0.457	0.580	0.407	0.397	0.230	0.303
LiveCodeBench	0.873	0.927	0.847	0.883	0.697	0.307
HumanEvalFix	0.998	1.000	0.988	0.998	0.941	0.470
QuixBugs	0.863	0.975	0.892	0.887	0.362	0.292
IneqMath	0.853	0.997	0.923	0.980	0.780	0.903
Omni-Math	0.543	0.852	0.767	0.833	0.710	0.622
Reasoning Gym	0.842	0.902	0.841	0.819	0.671	0.563
KUMO	0.926	0.964	0.849	0.936	0.541	0.668
AIME 24 & 25	0.711	0.989	0.889	0.967	0.756	0.739
ARC-AGI-2	0.020	0.533	0.003	0.003	0.000	0.000
AA-LCR	0.525	0.751	0.320	0.744	0.131	0.360
SimpleQA	0.748	0.945	0.645	0.955	0.158	0.762
GPQA Diamond	0.778	0.923	0.785	0.798	0.608	0.613
Humanity's Last Exam	0.203	0.519	0.131	0.312	0.058	0.111
MMMLU	0.356	0.838	0.649	0.727	0.582	0.602
GAIA	0.414	0.760	0.253	0.701	0.164	0.428
SkillsBench	0.317	0.602	0.132	0.258	0.054	0.017
BFCL	0.734	0.683	0.791	0.764	0.621	0.610
GAIA2	0.170	0.210	0.000	0.166	0.000	0.035
Terminal-Bench 2.0	0.427	0.693	0.251	0.371	0.105	0.052
DeepSynth	0.289	0.548	0.069	0.300	0.030	0.088
Seal-0	0.276	0.511	0.192	0.459	0.048	0.258
WideSearch	0.560	0.791	0.303	0.533	0.131	0.120
Spider 2	0.219	0.276	0.109	0.245	0.130	0.141
DA-Code	0.558	0.565	0.377	0.499	0.250	0.171
SpreadsheetBench	0.643	0.762	0.452	0.588	0.263	0.095
FinanceAgent	0.500	0.767	0.113	0.533	0.000	0.213
LawBench	0.586	0.638	0.486	0.425	0.321	0.110
PIXIU	0.574	0.710	0.506	0.480	0.283	0.337
MedAgentBench	0.340	0.643	0.350	0.430	0.080	0.290
LAB-Bench	0.230	0.731	0.160	0.475	0.083	0.120
SLDBench	0.673	0.795	0.542	0.707	0.225	0.075
BIX-Bench	0.300	0.407	0.193	0.313	0.073	0.073
ReplicationBench	0.256	0.607	0.081	0.233	0.041	0.041
QCircuitBench	0.496	0.620	0.358	0.177	0.142	0.000
ResearchCodeBench	0.006	0.866	0.000	0.599	0.002	0.421
SciCode	0.469	0.582	0.430	0.226	0.217	0.004
CodePDE	0.600	0.467	0.533	0.333	0.267	0.000
StrongReject	0.959	0.949	0.935	0.958	0.961	0.951
CyberGym	0.533	0.767	0.633	0.633	0.167	0.233
MMAU	0.660	0.703	0.597	0.583	0.537	0.070

Table 7: Claude-series models under Terminus-2 and Claude Code across 54 benchmarks.

Benchmark	Claude Haiku 4.5		Claude Sonnet 4.6		Claude Opus 4.6	
	Terminus-2	Claude Code	Terminus-2	Claude Code	Terminus-2	Claude Code
SWE-bench Verified	0.617	0.573	0.733	0.703	0.750	0.733
SWE-Bench-Multilingual	0.687	0.473	0.733	0.693	0.800	0.667
SWE-smith	0.187	0.231	0.415	0.289	0.582	0.412
CRUST-Bench	0.637	0.770	0.847	0.713	0.870	0.823
GSO	0.010	0.131	0.428	0.451	0.366	0.392
SWE-Bench Pro	0.080	0.450	0.473	0.490	0.487	0.537
SWT Bench	0.160	0.273	0.447	0.573	0.673	0.647
Aider Polyglot	0.299	0.301	0.572	0.560	0.719	0.686
AlgoTune	0.096	0.123	0.260	0.270	0.259	0.231
CompileBench	0.822	0.889	0.867	0.911	0.933	0.933
FeatureBench	0.034	0.126	0.189	0.277	0.405	0.323
BigCodeBench-Hard	0.393	0.345	0.446	0.437	0.444	0.416
USACO	0.273	0.380	0.727	0.750	0.800	0.780
SWE-Lancer	0.337	0.357	0.453	0.487	0.583	0.523
LiveCodeBench	0.460	0.607	0.823	0.837	0.887	0.870
HumanEvalFix	0.929	0.994	1.000	0.998	1.000	1.000
QuixBugs	0.717	0.887	0.925	0.975	0.954	0.933
IneqMath	0.733	0.953	0.970	0.997	0.860	0.860
Omni-Math	0.600	0.760	0.788	0.800	0.813	0.840
Reasoning Gym	0.803	0.807	0.869	0.818	0.875	0.851
KUMO	0.909	0.956	0.948	0.948	0.964	0.964
AIME 24 & 25	0.622	0.861	0.978	0.967	0.989	0.994
ARC-AGI-2	0.007	0.060	0.180	0.187	0.370	0.437
AA-LCR	0.606	0.549	0.663	0.741	0.616	0.747
SimpleQA	0.573	0.920	0.807	0.803	0.907	0.867
GPQA Diamond	0.535	0.709	0.823	0.854	0.843	0.884
Humanity's Last Exam	0.064	0.107	0.217	0.325	0.343	0.390
MMMLU	0.691	0.671	0.751	0.756	0.769	0.778
GAIA	0.313	0.570	0.529	0.679	0.612	0.671
SkillsBench	0.121	0.254	0.311	0.489	0.346	0.454
BFCL	0.770	0.729	0.770	0.805	0.764	0.799
GAIA2	0.056	0.213	0.319	0.384	0.357	0.358
Terminal-Bench 2.0	0.303	0.322	0.581	0.569	0.618	0.607
DeepSynth	0.062	0.112	0.398	0.454	0.492	0.461
Seal-0	0.060	0.258	0.342	0.294	0.390	0.348
WideSearch	0.326	0.574	0.511	0.673	0.617	0.758
Spider 2	0.208	0.255	0.339	0.370	0.375	0.396
DA-Code	0.504	0.510	0.578	0.580	0.616	0.580
SpreadsheetBench	0.560	0.702	0.803	0.832	0.832	0.818
FinanceAgent	0.080	0.207	0.800	0.800	0.720	0.833
LawBench	0.538	0.565	0.608	0.620	0.654	0.666
PIXIU	0.485	0.489	0.569	0.574	0.607	0.612
MedAgentBench	0.467	0.537	0.497	0.503	0.597	0.557
LAB-Bench	0.123	0.357	0.018	0.077	0.096	0.451
SLDBench	0.612	0.772	0.747	0.759	0.760	0.522
BIX-Bench	0.153	0.287	0.293	0.393	0.300	0.393
ReplicationBench	0.104	0.152	0.207	0.219	0.241	0.285
QCircuitBench	0.232	0.384	0.507	0.493	0.562	0.489
ResearchCodeBench	0.003	0.434	0.006	0.533	0.003	0.607
SciCode	0.345	0.366	0.439	0.452	0.485	0.488
CodePDE	0.400	0.333	0.600	0.333	0.600	0.400
StrongReject	0.970	0.962	0.909	0.906	0.910	0.919
CyberGym	0.633	0.733	0.700	0.500	0.567	0.633
MMAU	0.550	0.503	0.633	0.617	0.697	0.690

Table 8: Gemini-series models under Terminus-2 and Gemini CLI across 54 benchmarks.

Benchmark	Gemini 3.1 Pro Preview		Gemini 3 Flash Preview	
	Terminus-2	Gemini CLI	Terminus-2	Gemini CLI
SWE-bench Verified	0.777	0.757	0.687	0.733
SWE-Bench-Multilingual	0.767	0.747	0.747	0.767
SWE-smith	0.323	0.442	0.296	0.211
CRUST-Bench	0.940	0.923	0.687	0.853
GSO	0.373	0.431	0.245	0.408
SWE-Bench Pro	0.467	0.457	0.390	0.460
SWT Bench	0.573	0.487	0.600	0.560
Aider Polyglot	0.788	0.884	0.667	0.713
AlgoTune	0.239	0.278	0.182	0.163
CompileBench	0.933	0.956	0.956	1.000
FeatureBench	0.258	0.405	0.038	0.187
BigCodeBench-Hard	0.405	0.425	0.386	0.400
USACO	0.933	0.540	0.920	0.880
SWE-Lancer	0.630	0.710	0.477	0.500
LiveCodeBench	0.920	0.923	0.877	0.907
HumanEvalFix	1.000	1.000	0.998	0.998
QuixBugs	0.925	0.933	0.958	0.971
IneqMath	0.947	0.870	0.867	0.193
Omni-Math	0.872	0.880	0.835	0.650
Reasoning Gym	0.906	0.804	0.868	0.594
KUMO	0.969	0.967	0.953	0.964
AIME 24 & 25	0.989	0.994	0.889	0.811
ARC-AGI-2	0.603	0.767	0.350	0.377
AA-LCR	0.744	0.660	0.643	0.650
SimpleQA	0.950	0.960	0.840	0.910
GPQA Diamond	0.953	0.971	0.896	0.704
Humanity's Last Exam	0.533	0.541	0.395	0.272
MMMLU	0.804	0.769	0.824	0.740
GAIA	0.677	0.671	0.568	0.364
SkillsBench	0.374	0.582	0.260	0.564
BFCL	0.837	0.813	0.718	0.743
GAIA2	0.215	0.331	0.183	0.214
Terminal-Bench 2.0	0.689	0.633	0.513	0.524
DeepSynth	0.550	0.588	0.359	0.328
Seal-0	0.420	0.505	0.393	0.393
WideSearch	0.641	0.660	0.641	0.536
Spider 2	0.344	0.349	0.250	0.323
DA-Code	0.606	0.601	0.558	0.561
SpreadsheetBench	0.795	0.825	0.755	0.768
FinanceAgent	0.753	0.760	0.540	0.380
LawBench	0.697	0.644	0.648	0.542
PIXIU	0.748	0.748	0.711	0.704
MedAgentBench	0.637	0.550	0.507	0.307
LAB-Bench	0.313	0.709	0.311	0.678
SLDBench	0.784	0.737	0.789	0.747
BIX-Bench	0.433	0.480	0.393	0.393
ReplicationBench	0.541	0.630	0.219	0.356
QCircuitBench	0.687	0.560	0.643	0.491
ResearchCodeBench	0.000	0.833	0.002	0.711
SciCode	0.550	0.537	0.511	0.144
CodePDE	0.267	0.133	0.333	0.200
StrongReject	0.868	0.916	0.733	0.875
CyberGym	0.833	0.800	0.733	0.733
MMAU	0.627	0.610	0.627	0.507

Table 11: Ranking fidelity from a small task subset vs. the full benchmark. Values are mean Spearman ρ across 52 benchmarks with ≥ 10 tasks.

k tasks	Mean ρ	Benchmarks with $\rho > 0.95$
1	0.885	3/52
3	0.923	12/52
5	0.941	22/52
10	0.953	33/52

Table 14: Benchmark-level overlap between public benchmark evaluations and our Harbor model–harness configurations. Columns are not mutually exclusive: a benchmark can contain both overlapping and non-overlapping public configurations.

	Has overlap with ours	Has non-Harbor configuration	Total
Evaluation includes raw LLMs	10	31	38
Evaluation includes agents	7	27	33
Unique benchmarks	16	52	54

Table 15: **Comparison with original or launch-time best results on selected benchmarks.** Scores are reported on a $[0, 1]$ scale.

	BFCL	SLDBench	CodePDE	MedAgentBench	AlgoTune
Original / launch best	0.9072	0.8741	0.8000	0.6967	0.3401
Harbor best	0.8374	0.7952	0.6000	0.6433	0.2776

Table 16: Round-1 vs. Round-2 inter-annotator Cohen's κ on 182 trials with labels in both rounds. Sorted by κ. Twelve rubrics clear the $\kappa \geq 0.5$ trustworthy threshold; six are excluded.

Rubric	**R1+**	**R2+**	κ	**Tier**
Wrong Target / Layer	2	2	1.000	Trustworthy
Syntax / Language Error	3	4	0.854	Trustworthy
Agent Timeout	8	9	0.815	Trustworthy
Unverified Claim	27	21	0.809	Trustworthy
Plan Over Implementation	2	3	0.797	Trustworthy
Wrong Factual Answer	37	31	0.784	Trustworthy
Hidden-Test Regression	22	14	0.755	Trustworthy
False Success Claim	23	16	0.743	Trustworthy
Silent Deliverable	6	3	0.659	Trustworthy
Algorithmic Bug	25	26	0.659	Trustworthy
Environment Block	7	8	0.652	Trustworthy
Wrong Output Schema	12	11	0.583	Trustworthy
Calculation Mismatch	22	7	0.451	Excluded
Rule Inference Error	2	8	0.389	Excluded
Specification–Verification Mismatch	16	34	0.320	Excluded
Insufficient Web Research	0	3	0.000	Excluded
Premature Termination	0	3	0.000	Excluded
Wrong Output Location	0	1	0.000	Excluded

Table 17: Pro v12 judge agreement with the gold set on the 182-trial calibration sample. Trials where one or more parse failures dropped a verdict are excluded from the per-rubric counts. Six rubrics clear $\kappa \geq 0.5$; the rest are folded into "Others".

Rubric	**Gold+**	**LLM+**	**κ vs. gold**	**Tier**
Wrong Factual Answer	37	49	0.69	≥ 0.5 (reported)
Syntax / Language Error	4	5	0.66	≥ 0.5 (reported)
Silent Deliverable	6	6	0.66	≥ 0.5 (reported)
Hidden-Test Regression	20	11	0.62	≥ 0.5 (reported)
Environment Block	10	5	0.52	≥ 0.5 (reported)
Plan Over Implementation	3	1	0.50	≥ 0.5 (reported)
Agent Timeout	9	7	0.48	Others
False Success Claim	19	10	0.44	Others
Algorithmic Bug	28	35	0.37	Others
Wrong Output Schema	15	6	0.25	Others
Unverified Claim	21	1	0.08	Others
Wrong Target / Layer	2	2	-0.01	Others
Pooled	**183**	**138**	**0.51**	—

Table 19: Harbor-Index 1.0 task counts by domain and benchmark (82 tasks / 29 benchmarks).

Domain	Benchmark	#Tasks
Software Engineering	GSO	7
	SWE-Bench Verified	5
	AlgoTune	5
	FeatureBench	4
	SWE-Bench Pro	4
	SWE-Lancer	2
	BigCodeBench	1
	USACO	1
	SWE-smith	1
	SWT Bench	1
Scientific Research	BIX-Bench	5
	LAB-Bench	4
	SciCode	3
	SLDBench	1
	Replication Bench	1
	CodePDE	1
	QCircuit Bench	1
Agents, Tools & Systems	GAIA2	5
	GAIA	3
	Terminal Bench 2	3
	SkillsBench	2
	WideSearch	1
Knowledge	HLE	8
	GPQA Diamond	1
Mathematics & Reasoning	ARC-AGI-2	5
	OmniMath	2
Data & Analytics	Spider 2	2
	DA-Code	1
Safety & Security	Cyber Gym	2