Long-Horizon-Terminal-Bench: Testing the Limits of Agents on Long-Horizon Terminal Tasks with Dense Reward-Based Grading

Zongxia Li[†1,2], **Zhongzhi Li**[†1,3], **Yucheng Shi**[†1], **Ruhan Wang**[1,5], **Junyao Yang**[1,7], **Zhichao Liu**[2], **Xiyang Wu**[2], **Anhao Li**[4], **Yue Yu**[5], **Ninghao Liu**[8], **Lichao Sun**[6], **Haotao Mi**[1], **LeoweiLiang**[1]

[1]Tencent HY LLM Frontier [2]University of Maryland, College Park [3]University of Georgia [4]University of Minnesota, Twin Cities [5]Indiana University [6]Lehigh University [7]National University of Singapore [8]The Hong Kong Polytechnic University

[†] Core contribution.

zli12321@umd.edu, zongxiali@global.tencent.com,zhongzhili@global.tencent.com

AI agents have become increasingly capable of autonomously completing short, well-specified tasks. However, existing terminal benchmarks largely focus on relatively simple problems that finish within a few minutes and are typically evaluated only by their final outcome. This setup overlooks intermediate progress and partial solutions, leading to sparse reward signals and an incomplete picture of agent capability.

We introduce Long-Horizon-Terminal-Bench, a challenging terminal benchmark of 46 long-horizon tasks spanning nine categories, including experiment reproduction, software engineering, multimodal analysis, interactive games, and scientific computing. Each task follows a Terminal-Bench-style setup with a reference solution or simulation engine, but is further decomposed into fine-grained graded subtasks. This design enables dense intermediate rewards and partial credit, allowing evaluation to capture not only whether an agent reaches the final goal, but also how far it progresses on difficult, open-ended workflows.

Tasks in Long-Horizon-Terminal-Bench typically require hundreds of episodes and tens of minutes to hours of execution, stressing long-horizon planning, long-context management, and iterative debugging rather than one-shot problem solving. We evaluate 15 frontier models and find that agents consume on average 9.9M tokens per task, with roughly 231 episodes and 85.3 minutes of execution time per run, making Long-Horizon-Terminal-Bench substantially more demanding than prior terminal-based benchmarks. Even the strongest tested model achieves 15.2% pass@1 at a partial-reward threshold of 0.95 and 10.9% at a perfect-reward threshold of 1.0, while the mean pass rate across models is just 4.3% and 1.7% under the two thresholds, respectively. These results reveal substantial headroom for improvement. We further analyze common failure modes and error patterns, and release Long-Horizon-Terminal-Bench to support future progress on robust long-horizon terminal agents.

Project page: https://zli12321.github.io/LHTB/

Date: July 13, 2026

1 Introduction

AI agents built on large language models (LLMs) are rapidly improving at autonomous decision making and tool use [2, 6]. Recent work has demonstrated impressive performance on short, well-scoped tasks such as fixing a code repository issue, completing a coding ticket, or issuing a handful of shell commands [22]. However, they still cover only a narrow slice of the long-horizon, domain-specific, and practically important workflows that human experts care about in practice [22, 46, 51].

Nowadays, many real workflows are *long-horizon*: they require agents to execute hundreds of steps, maintain and update plans over tens of minutes to hours, and manage evolving long-context state [42]. Examples include reproducing results from published research papers [36], installing environments from a github repo, auditing complex multimodal datasets, debugging compiler toolchains, or shepherding a multi-stage ML training pipeline. In these settings, agents must repeatedly use terminal command lines to read and write files, run scripts, inspect partial outputs, revise their plans, and recover from mistakes. The outcome is determined not by a single action, but by sustained, coherent progress over a long sequence of decisions [21].

Existing benchmarks lack fine-grained verifiers to evaluate agents on long-horizon terminal tasks [12]. Dominant terminal and software-engineering benchmarks typically evaluate within a few minutes to at most an hour, and grade agents solely on whether the final state satisfies a test suite [3, 5]. This leaves two gaps. First, short tasks or narrow horizons understate the difficulty of real workflows that require long-horizon navigation, errored step revision, and multi-stage debugging, not just local patching [7, 22]. Second, outcome-only grading yields extremely sparse reward signals: an agent that completes most intermediate steps but fails at the final step can receive the same score as one that fails from the beginning [19]. Both issues make it difficult to understand how well current agents can sustain progress on long workflows, or where they fail.

In this paper we introduce *Long-Horizon-Terminal-Bench*, a benchmark designed explicitly for long-horizon terminal tasks. Long-Horizon-Terminal-Bench consists of 46 tasks spanning nine categories, including experiment reproduction, interactive games, software engineering, multimodal analysis, and scientific computing. Each task is hosted in a containerized terminal environment and is decomposed into graded subtasks with intermediate checks. Rather than asking agents to `solve everything or fail`, the grader assigns credit for partial progress toward a final goal, providing dense feedback along long trajectories while preserving the realism of everyday tasks.

Long-Horizon-Terminal-Bench targets domains where the completion of the final big goal depends on the optimal and completion of the sub goals, and agents must execute *many* steps over *long* time horizons,. Across 15 frontier agents evaluated, rollouts on Long-Horizon-Terminal-Bench average 231 episodes, 9.9M tokens, and 85.3 minutes of wall-clock time before exit under a 90-minute timeout. These numbers are substantially larger than prior terminal and code-editing benchmarks such as Terminal-Bench 2 tasks and SWE-Bench. Yet even in this relaxed grading, current agents struggle: our strongest reported configuration, GPT-5.5, achieves only 15.2% success rate at a 0.95 reward threshold, and the average pass rate across models is 4.3%. This suggests that long-horizon execution remains a central bottleneck for current systems.

We also conduct qualitative and quantitative analyses of agent failures on Long-Horizon-Terminal-Bench. Our analysis shows that current agents often fail not because every local step is wrong, but because they cannot reliably sustain progress, verify completion, and finish long-horizon tasks within budget. Dense rewards make these differences visible by separating timeout-driven incomplete progress from premature stopping and weak self-verification. We release Long-Horizon-Terminal-Bench and the accompanying evaluation harness to support future work on agents that can plan, verify, and execute reliably over long horizons.

2 Long-Horizon-Terminal-Bench

Long-Horizon-Terminal-Bench builds on the Terminal-Bench formulation [22], where each task is a containerized terminal environment with a natural-language instruction, all relevant files and tools, and a reference solution that can be executed entirely from the command line. Tasks span diverse domains and are explicitly designed so that progress requires *continued* work over many steps: reading and modifying code and data, running long scripts, inspecting partial outputs, and iteratively debugging.

Unlike Terminal-Bench, which evaluates tasks in a binary solved/unsolved fashion, Long-Horizon-Terminal-Bench is graded through a subtask-based scheme. Each task is decomposed into a small set of semantically meaningful subtasks with their own checks, and agents are scored by their completion rate across these subtasks. This design provides partial credit and a much denser signal about long-horizon performance, revealing how far agents get through complex workflows rather than only whether they reach the final goal.

2.1 Task formulation

A Long-Horizon-Terminal-Bench task follows the Terminal-Bench formulation [22]: it is specified as a Harbor task with (1) a natural-language instruction, (2) a Docker image that defines the terminal environment, (3) a task configuration file, and (4) an oracle implementation or simulator used for grading. The instruction describes the overall long-horizon goal (e.g., reproduce a figure, repair a robotics SLAM pipeline, audit a multimodal dataset) and is the only specification the agent sees. The Docker image contains all relevant assets, code, data, tools, and helper scripts, so that an agent can complete the task entirely from the terminal. Tasks are interactive: after the container starts, the agent issues shell commands, edits files, runs scripts, and inspects intermediate outputs over hundreds of steps until it either succeeds or times out.

While the task format matches Terminal-Bench 2, Long-Horizon-Terminal-Bench is explicitly designed so that a single subtask typically requires many reasoning steps and many minutes to hours of work and dozens to hundreds of distinct operations. Long-horizon structure comes both from the underlying domains (multi-stage ML pipelines, campaign-style games, scientific audits) and from the way we factor goals into a sequence of intermediate targets that must be discovered and executed over time.

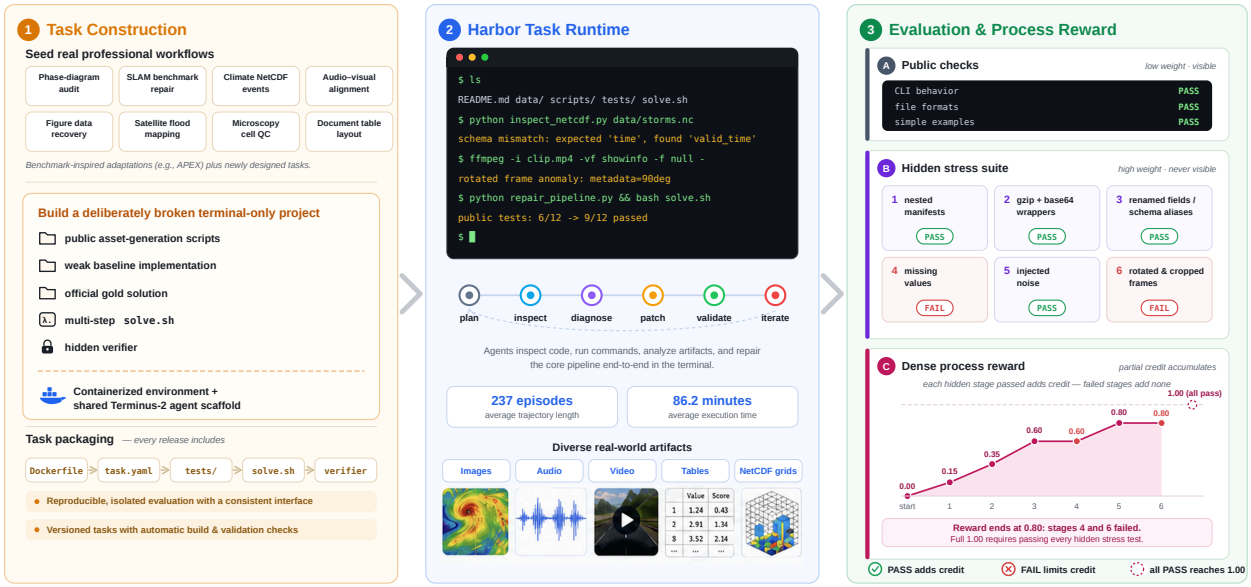

Figure 1 The overall structure of long-horizon terminal task with dense reward grading. Partial rewards show how far models are into completing the task.

2.2 Subtask-based grading

Each Long-Horizon-Terminal-Bench task is graded by a deterministic grader that runs *inside* the container at the end of a rollout. Instead of returning a strict single pass/fail, the grader computes scores for a small set of subtasks $\{s_1, \ldots, s_K\}$ that together describe the intended workflow. For every subtask s_k it outputs a normalized score $r_k \in [0, 1]$; the task reward is

$$R = \frac{\sum_{k=1}^{K} w_k \, r_k}{\sum_{k=1}^{K} w_k},$$

where w_k are non-negative weights (by default all equal, with higher weight on the final goal when needed). Subtasks are chosen to correspond to meaningful intermediate goals in the workflow, and each is evaluated using objective evidence from the final container state, such as files, outputs, test results, or simulator responses.:

- **Binary subtasks.** Checks are implemented as strict Boolean conditions over the environment state, such as whether all unit tests exit successfully, whether a service responds on the expected port, or whether required experiment scripts run without error. These subtasks return $r_k \in \{0, 1\}$ depending on whether the corresponding programmatic check passes.

- **Continuous or thresholded subtasks.** For quantitative targets (e.g., reproducing a figure, matching a metric, or achieving a given speedup) we use continuous scores. Examples include *1.0 if the reproduced metric is within tolerance of the reference, decreasing linearly to 0 as error grows*, or *fraction of held-out examples on which the model's predictions match the oracle*. This yields meaningful partial credit when an agent gets close but not perfect.

- **Episode-aggregating subtasks.** Campaign-style tasks (e.g., games or repeated audits) aggregate across episodes. A typical subtask is the fraction of episodes or levels in which the environment's internal success flag is triggered, or the mean normalized reward reported by the simulator. This measures whether an agent can perform a long-horizon behavior reliably across many episodes, rather than succeeding only once by chance.

The overall task reward R is used for all reported metrics. A task is counted as *resolved* for sucess if $R \geq \tau$[1]. We also report mean R across tasks to capture how far agents progress on problems they cannot fully solve.

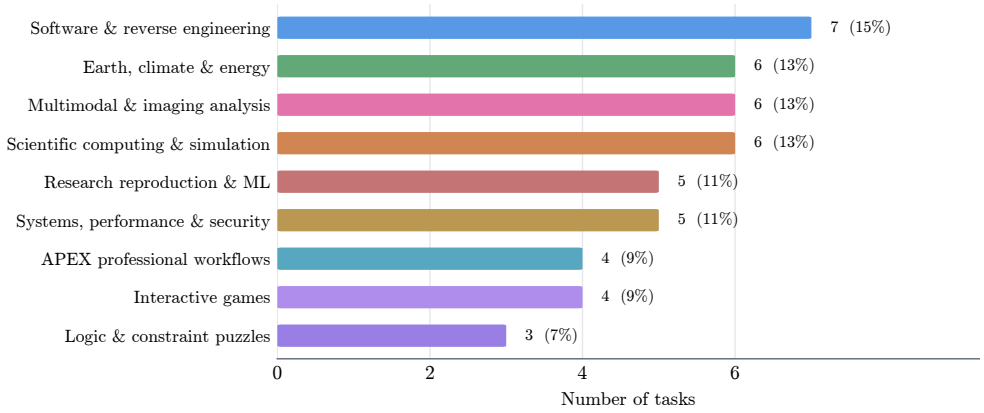

Figure 2 Task distribution across the LHTB categories. The 46 tasks span nine long-horizon domains, ranging from software and reverse engineering to scientific computing, earth and climate science, and multimodal analysis.

2.3 Dataset construction

We construct Long-Horizon-Terminal-Bench by instantiating a common recipe across a diverse set of real-world long-horizon workflows [40, 44]. Each task begins with a realistic professional data-processing or engineering problem, such as materials phase-diagram auditing, robotics SLAM benchmark repair, climate NetCDF extreme-event detection, audio–visual event alignment, scientific-figure data reconstruction, satellite flood change detection, microscopy cell-count quality control, or scanned-document table reconstruction. Some of these professional-workflow tasks are adapted from or inspired by prior long-horizon agent benchmarks such as APEX-Agents [40], while others are newly created for Long-Horizon-Terminal-Bench.

For each problem, we build a complete but deliberately broken terminal-only project. Agents must work entirely through the terminal: inspecting code, running commands, and analyzing artifacts such as images, audio, video, tables, and NetCDF files in order to diagnose and repair the core pipeline. Each task includes reproducible public asset-generation scripts, a weak baseline implementation, an official gold solution, a multi-step `solve.sh`, and a hidden verifier.

To discourage overfitting to superficial public tests, the public checks only validate command-line behavior, file formats, and a small number of simple examples, and they carry relatively low reward weight. Most of the

[1] we use a relaxed threshold such as $\tau = 0.95$

reward is assigned through hidden stress cases that dynamically generate harder inputs and schema variations, such as nested manifests, gzip-plus-base64 wrappers, renamed fields, missing values, injected noise, rotated or cropped images, anomalous frames, and alternative coordinate or time-dimension conventions. As a result, agents cannot obtain high scores by hard-coding outputs or by only patching the public cases; they must implement robust parsing, core algorithms, and end-to-end artifact generation that generalize beyond the visible examples. The official gold solution is required to achieve a verifier score of 1.0 on the full hidden evaluation suite.

Difficulty calibration and validation. We calibrate difficulty by repeatedly running Deepseek-V4-Pro [4] under 1.5-hour time budgets and adjusting task design until the tasks are challenging but still solvable in principle. We generated 120 candidate tasks with quality filtering, where the final benchmark contains 46 tasks implemented in the Harbor format [22], all with containerized environments and a shared agent harness.

2.4 Task composition

The 46 tasks in Long-Horizon-Terminal-Bench span 21 high-level categories, including interactive games, multimodal audits, software engineering, systems and performance, experiment reproduction, earth and environment, scientific computing, robotics, materials science, health and medicine, climate science, and chip design (EDA). Figure 2 shows the distribution. No single category dominates the benchmark: the largest groups are interactive games and multimodal audits, with the remaining tasks spread across a broad range of scientific and engineering workflows.

3 Experiments

Harness and agents. We evaluate models using the Harbor framework with the Terminus-2 agent harness [10], which interacts with each task environment through a single long-horizon terminal session. Additionally, for GPT-5.3, we instead use Codex [28][2] as the agent harness. These harness frameworks allow us to evaluate how existing frontier models perform on sustained terminal-based workflows.

Models. We benchmark GPT-5.5 [28], GPT-5.4, GPT-5.3 Codex, DeepSeek V4 Pro [4], Gemini 3.1 Pro [9], GLM 5.1 [8], GLM 5.2, Kimi K2.6 [24], Kimi K2.7 Code [25], MiniMax M3 [23], Qwen3.7 Max [32], Qwen3.6 Plus [31], Doubao Seed 2.1 Pro [1], Hy3 [37], and Grok 4.20 [45].

Metrics. For each model we report pass@1 (fraction of tasks whose normalized reward is at least 0.95), mean normalized reward across all tasks, number of episodes to complete the task, length of time, and estimated dollar cost and token usage per task.

3.1 Main Results

Figure 3 summarizes pass rate at $R \geq 0.9$, $R \geq 0.95$, and $R \geq 1.0$, along with mean reward across all 46 tasks. Overall, the evaluated frontier models still struggle on long-horizon tasks, even under a dense-grading evaluation that gives credit for partial progress. GPT-5.5 is the strongest reported model, but still resolves only 15.2% of tasks (7/46) at $R \geq 0.95$. MiniMax M3, Kimi K2.7 Code, and DeepSeek V4 Pro follow at 6.5% (3/46) each on the full 46-task run. Below is a mid tier at 4.3% that includes Qwen3.7 Max, Doubao Seed 2.1 Pro, Gemini 3.1 Pro, GLM 5.1, and GPT-5.3 Codex, while GLM 5.2, Qwen3.6 Plus, GPT-5.4, and Hy3 each resolve only a single task. Kimi K2.6 and Grok 4.20 solve zero tasks at $R \geq 0.95$, with Grok 4.20 also recording the lowest mean reward ($R = 0.08$) among the reported models. Mean normalized rewards follow a similar ordering, indicating that many models make partial progress on hard tasks without fully resolving them. The results suggest that long-horizon execution is not just local reasoning task, but remains a bottleneck even for top frontier models.

[2] https://github.com/openai/codex

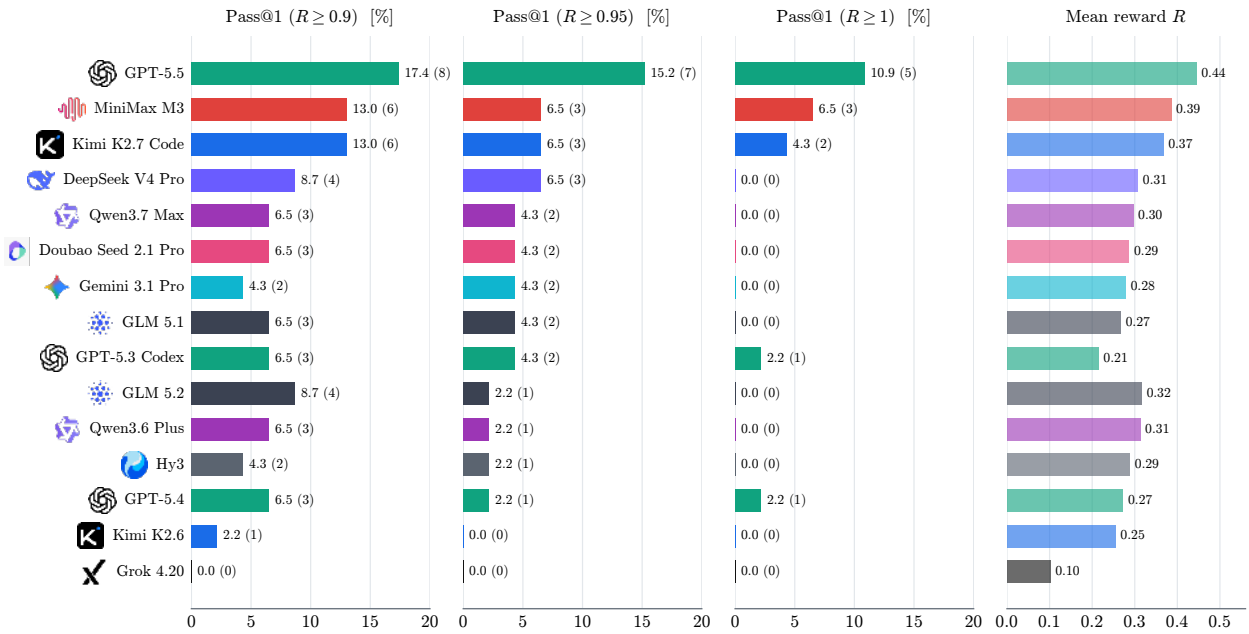

Figure 3 LHTB leaderboard. Pass@1 at reward thresholds 0.9, 0.95, and 1.0, together with mean normalized reward, for each model under a shared agent setup, shown in a single model order sorted by pass@1 at $R \geq 0.95$. Pass rate is the fraction of the 46 tasks whose overall reward R meets a threshold τ, with $R \geq 1.0$ corresponding to full completion, while mean R captures partial progress across all tasks. GPT-5.5 achieves the strongest overall performance, resolving 15.2% of tasks at $R \geq 0.95$. **Tightening the threshold to $R \geq 1.0$ collapses most models to zero pass rate, showing that partial-credit evaluation captures progress that a binary threshold misses.**

3.2 Dense Reward Is Necessary to Rank Models

A key contribution of Long-Horizon-Terminal-Bench is dense, subtask-level grading rather than binary pass/fail, which is essential at this difficulty level. Under a strict threshold ($R \geq 1.0$), 10 of the 15 models pass zero tasks (Figure 3); even at $R \geq 0.95$, only 4.3% of runs pass while 32.9% score below 0.05 (Figure 4). The remaining 64.6% of runs make real partial progress, but binary grading would count them as complete failures by giving them the same score as a run that makes no progress

Binary grading hides meaningful differences between models. Under binary pass/fail evaluation, the leaderboard collapses into large tie groups; for example, ten models each solve zero tasks at the strict $R \geq 1.0$ threshold. Mean reward resolves these ties by quantifying how far a model progresses on tasks it does not fully complete. Among the models tied at one solved task at $R \geq 0.95$, mean reward spans from 0.27 for GPT-5.4 to 0.32 for GLM 5.2. Pass rate and mean reward are positively but only moderately correlated (Spearman $\rho = 0.56$), and they can produce different model rankings because they measure different aspects of long-horizon performance: pass rate captures full task completion, whereas mean reward captures sustained partial progress.

Near-misses reveal real progress that binary pass/fail evaluation would hide. Dense grading also reveals *near-misses* ($0.75 \leq R < 0.95$) runs that are close to completion but fail some final checks. These near-misses occur more than twice as often as passes (73 vs. 30), indicating that many runs get very close to fully solving the task, but fail at the last few steps or checks. For example, Kimi K2.6 solves zero tasks at the $R \geq 0.95$ threshold, yet has five near-misses and a mean reward of 0.25. Its strongest near-miss reaches $R = 0.94$ on `grammar-fuzz-coverage-hunt`, and it also comes close on `spot-scheduler-traces` ($R = 0.90$), `poc-exploit-craft` ($R = 0.89$), and `nbody-accel-iterative` ($R = 0.89$). Under binary grading, it would appear indistinguishable from a model that makes no meaningful progress, even though it repeatedly approaches successful completion. As models improve, such near-misses are likely to provide an important

early signal of progress, often revealing more clear capability gains before they are reflected in full pass rates.

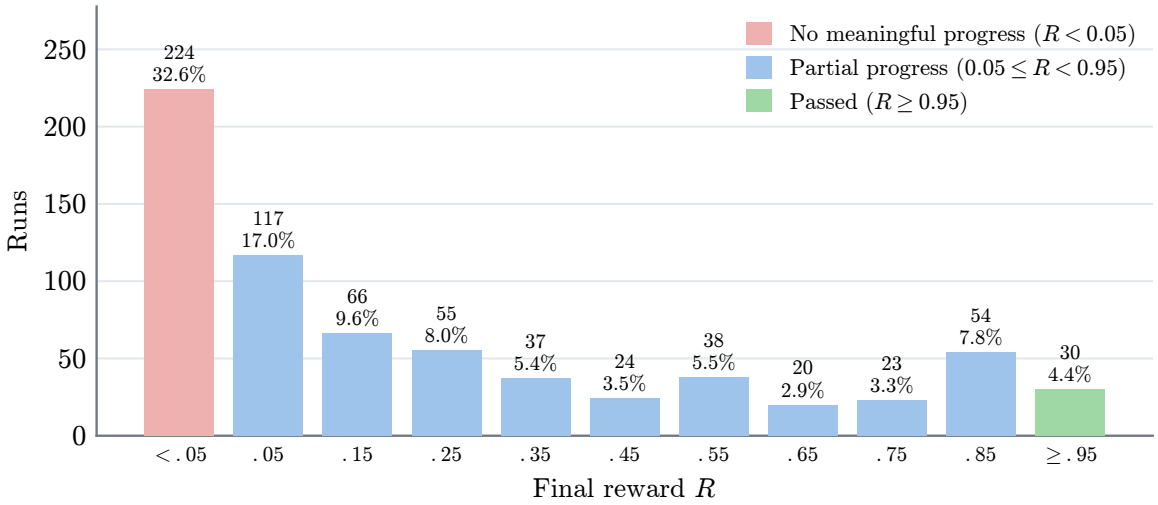

Figure 4 Distribution of final rewards over all 15×46 model–task runs. Only 30 runs (4.3%, green) pass the $R \geq 0.95$ threshold, while 227 runs (32.9%, gray) make no meaningful progress ($R < 0.05$). The remaining 433 runs (62.8%, blue) achieve partial reward but would all be counted as failures under binary pass/fail evaluation. This partial progress is often substantial rather than trivial: 180 runs (26.1%) reach $R \geq 0.5$, and the mass in $[0.85, 0.95)$ (near-complete solutions that fail only the final checks) is still nearly twice the number of full passes. **Binary grading collapses a wide range of partially successful runs into a single failure bucket, whereas dense subtask-level rewards preserve the signal needed to distinguish model progress at this difficulty level.**

3.3 Cost Analysis

Table 1 reports the cost estimate from per-task token usage and public list prices, which Figure 5 shows the pass rate via the Pareto cost–reward frontier. The average costs span from about \$2.5 to \$28 per task. Although GPT-5.5 achieves the highest pass rate but is also among the most expensive models, at roughly \$21 per task, and GPT-5.4 is the most expensive overall (about \$28 per task) with a lower pass rate. This gap arises because GPT-5.4 is the weaker model yet requires far more episodes per task (302 vs. 208 for GPT-5.5) at comparable per-M token pricing; it therefore has a higher cost while achieving a lower pass rate.

Hy3 anchors the low-cost end of the Pareto frontier at about \$2.5 per task, while Doubao Seed 2.1 Pro and MiniMax M3 also lie on the frontier, achieving pass@$R \geq 0.95$ rates of 4.3% and 6.5% at roughly \$5 and \$6 per task. DeepSeek V4 Pro sits just off the frontier, achieving the same 6.5% pass@$R \geq 0.95$ rate as MiniMax M3 at a similar cost. The remaining models, including Kimi K2.7 Code, GPT-5.3 Codex, GLM5.1, Gemini3.1 Pro, Qwen3.7 Max, GLM5.2, Kimi K2.6, and Grok 4.20, form a mid tier or lower tier with moderate to high costs but low resolution rates, indicating that higher inference spending alone is insufficient to guarantee better long-horizon performance.

Model	Tokens / task (M)	Episodes / task	Time / task (min)	Avg cost / task ($)
GPT-5.5	4.16	208	72.9	21.46
MiniMax M3	**20.20**	314	90.0	6.13
Kimi K2.7 Code	8.54	183	85.4	8.31
DeepSeek V4 Pro	14.45	**321**	83.6	6.32
Qwen3.7 Max	6.13	218	83.5	7.78
Doubao Seed 2.1 Pro	5.80	183	91.7	5.16
Gemini 3.1 Pro	3.55	148	85.0	7.61
GLM 5.1	5.84	120	**92.6**	5.13
GPT-5.3 Codex	4.57	299	80.7	8.20
GLM 5.2	8.43	195	89.3	11.93
Qwen3.6 Plus	8.67	194	88.6	4.47
Hy3	17.21	258	91.3	2.47
GPT-5.4	10.90	302	79.3	**27.57**
Kimi K2.6	10.27	188	92.5	9.94
Grok 4.20	16.23	288	69.5	20.63
Average	9.66	228	85.1	10.21

Table 1 Estimated cost per model on LHTB. On average, models spend 228 episodes and 85.1 minutes per task, at an estimated $10.2 per task. Costs are estimated from per-task token usage and public list prices (USD per million tokens) as of June 2026. Tokens/task counts both input and output tokens; all prompt tokens are billed at the full input rate, since prompt-cache discounts are not guaranteed across providers.

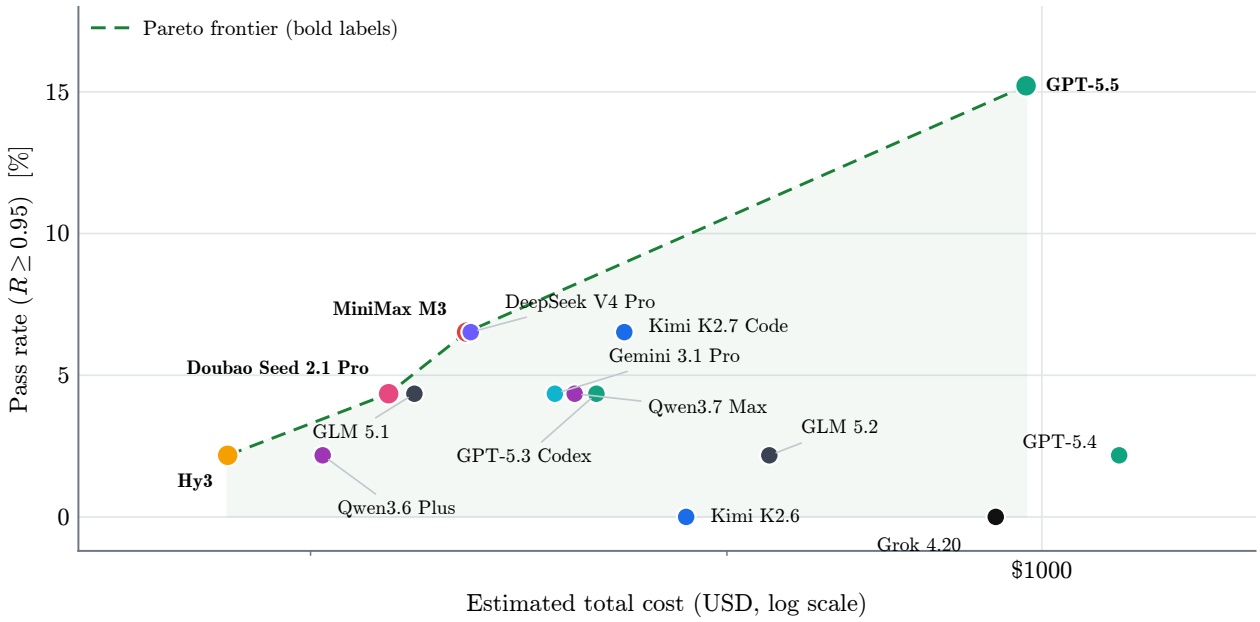

Figure 5 Cost–reward frontier on LHTB. Estimated total cost (log scale) against pass rate at $R \geq 0.95$; the dashed line traces the Pareto frontier, and models on the frontier are shown in bold. GPT-5.5 achieves the highest pass rate among the reported models, while Hy3 is at the low-cost end of the frontier.

3.4 Dense Rewards Expose Different Failure Patterns

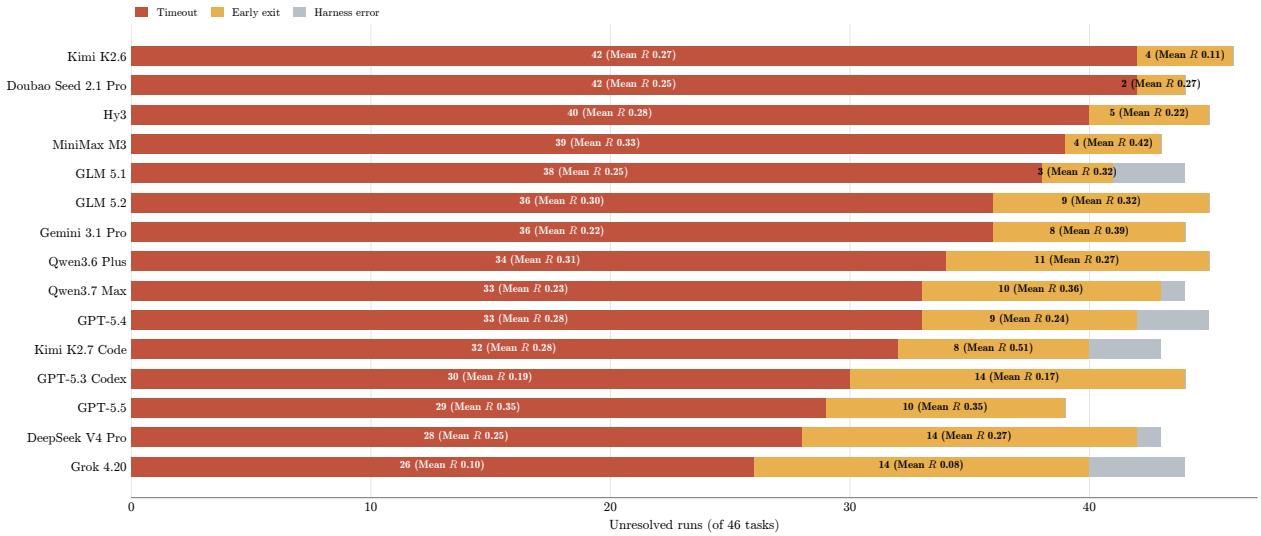

Figure 6 Composition of unresolved runs per model. Each bar decomposes a model's unresolved tasks ($R < 0.95$, out of 46) into timeouts (the agent is still working when the 90-minute budget expires), early exits (the agent terminates on its own with no harness error), and harness errors (non-timeout exceptions such as API- or verifier-side failures). The label inside each timeout segment shows the number of timed-out runs and their mean reward; the label inside each early-exit segment shows the number of early exits and their mean reward. Timeouts dominate for every model (79% of all unresolved runs), showing that many failures reflect incomplete progress rather than immediate breakdowns, while the smaller early-exit segments highlight cases where agents stop too soon despite not having fully solved the task.

Figure 6 shows unresolved runs by termination cause. Across all models, 79% of unresolved runs (518/660) end because the 90-minute budget expires while the agent is still actively working. However, these timed-out runs are not close to completion, with mean reward ranging only from 0.10 to 0.35 across models. By contrast, early exits that account for only 19% of unresolved runs often indicate overconfidence: the agent stops on its own despite not yet satisfying the hidden verifier. Just 3% are *harness errors*, failures in the agent–environment loop caused by the evaluation harness itself, rather than by the model stopping early or exhausting its time budget.

The main bottleneck is not local execution correctness, but long-horizon completion. On Terminal-Bench 2.0, most trials complete in under 20 minutes, and trajectory-level error analyses find that many failures are execution-related, including disobeying the specification, repeating steps, and missing termination conditions [22]. On Long-Horizon-Terminal-Bench, the more common failure is different: agents can often string together many locally correct actions, but they cannot reliably turn that progress into a finished artifact before the horizon expires. Short-horizon benchmarks primarily measure whether agents can *act correctly*; Long-Horizon-Terminal-Bench additionally measures whether they can *budget a long horizon*, and the latter appears to be the binding constraint.

These results suggest that rankings on Long-Horizon-Terminal-Bench reflect not only task-solving ability, but also *time efficiency*: how much reward a model can accumulate within a fixed budget. As a result, improvements that reduce redundant exploration, preserve state more effectively, or avoid repeated verification loops may yield larger gains than comparable improvements in single-step reasoning alone.

3.5 Dense Rewards Reveal False Finishes and Weak Stopping Judgment

Timeouts dominate model failures, but they are not the full story. Among the 124 runs in which an agent terminates *on its own* before the budget expires, a different limitation becomes clear: weak self-verification.

We define a *false finish* as an early exit at relatively high reward, where the agent stops despite having not yet satisfied the hidden verifier. We identify 14 such runs with $R \geq 0.75$. In these cases, the agent has completed most of the task, judges itself finished, and exits with substantial time remaining. For example, Kimi K2.7 Code stops on `duckdb-optimizer-closure` at $R = 0.92$, GLM 5.2 stops on `apex-ib244-matter` at $R = 0.90$, and seven different models stop on `apex-law433-matter` between $R = 0.80$ and 0.87, each with roughly 20 minutes still remaining. These are not hopeless attempts, but near-complete ones in which the agent stops just short of satisfying the hidden verifier.

Dense rewards are essential for exposing this pattern. A binary pass/fail metric would collapse all of these runs into the same failure bucket, whereas dense rewards show *how far* each agent gets before stopping. This reveals a clear axis of variation in stopping judgment (Figure 6). Kimi K2.7 Code's early exits average $R = 0.51$ and MiniMax M3's average $R = 0.42$, indicating that these models tend to stop later than other models ($R = 0.22$ to 0.39), after completing a substantial fraction of the task. By contrast, Kimi K2.6 exits at an average of only $R = 0.11$, abandoning tasks while most of the work remains. Early exits reveal two distinct abilities: whether a model chooses to stop prematurely, and how accurately it can judge how much work remains before the task is truly complete.

This phenomenon is the long-horizon analogue of the *verification* failures identified in Terminal-Bench 2.0, such as premature termination, weak verification, and missed final checks [22]. After the obvious public errors are fixed, they must actively search for residual defects that no visible signal reveals. The false-finish pattern suggests that current agents systematically overestimate completion and under-invest in final verification. Closing this gap for long-horizon tasks will require not only stronger local reasoning, but also better self-verification and more calibrated stopping decisions.

4 Related Work

Terminal and software-engineering agent benchmarks. Recent agent benchmarks have shifted from static question answering and isolated programming exercises toward interactive, execution-grounded tasks in realistic work environments [46, 51]. In software engineering, SWE-Bench [12] evaluates repository-level issue resolution on real GitHub projects, while LiveCodeBench [11] provides contamination-aware evaluation of code generation and related coding abilities such as execution, self-repair, and test-output prediction. Subsequent benchmarks further broaden the scope of software-agent evaluation: SWT-Bench [26] studies test generation and validation for real-world bug fixes, and Terminal-Bench [22] places agents in containerized terminals to complete realistic technical tasks such as system configuration, model training, and research-code reproduction. More recent efforts push toward substantially harder and longer workflows, including FrontierSWE [3], SWE-EVO [15], SWE-Marathon [5], and Edge-Bench, which target expert-level, multi-stage, or ultra-long-horizon tasks. These benchmarks measure progress in coding and terminal agents, but many still primarily measure end-to-end success or failure, providing limited visibility into how much useful progress an agent made before failing. Long-Horizon-Terminal-Bench uses subtask-based grading exposes partial progress even when the final objective is not fully completed.

Measuring long-horizon autonomy tasks. Recent work has increasingly characterized long-horizon autonomy in terms of the duration of tasks that agents can complete reliably, rather than aggregate performance on a fixed benchmark [14, 43]. METR's time-horizon analysis [14] makes this perspective explicit by estimating the human-time duration of tasks that frontier agents can complete at a fixed success rate. This framing treats horizon length as a first-class capability variable and is grounded in human-calibrated evaluations such as RE-Bench [43] and HCAST [34], which compare agents against human performance on realistic research-engineering, software-engineering, cybersecurity, and reasoning tasks. At the same time, it also highlights a limitation of outcome-only evaluation in long-horizon settings: a binary success signal provides little information about how much progress was made before failure or where execution began to diverge. Recent studies show that long-horizon degradation can arise from the accumulation of execution errors over extended trajectories, even when short-horizon competence appears strong [35]. Long-Horizon-Terminal-Bench is designed to complement this line of work by combining extended, many-step autonomy tasks

with subtask-level grading, thereby enabling a finer-grained characterization of partial progress and failure throughout long-horizon execution.

Beyond outcome-only grading. Process reward models provide supervision over intermediate reasoning steps rather than only final answers, yielding denser signals than outcome-only supervision [13, 18]. In parallel, rubric-based approaches decompose evaluation into individually assessable criteria or weighted dimensions of quality, making it possible to score complex outputs beyond a single holistic judgment [30, 33, 38]. Most of this literature uses fine-grained signals primarily as *training* rewards or judge-based supervision, often relying on learned verifiers or LLM evaluators [16, 39, 49, 50]. LONG-HORIZON-TERMINAL-BENCH applies the same core intuition at *evaluation* time: each task is decomposed into semantically meaningful subtasks, and each subtask is checked by deterministic, environment-grounded graders.

Agent harnesses. Benchmark outcomes depend not only on the underlying model, but also on the agent harness that mediates its interaction with the environment [17, 20, 47, 48]. Open frameworks such as SWE-agent [47], OpenHands [41], and the Terminus family introduced alongside Terminal-Bench [22] provide reusable interfaces for autonomous repository- and terminal-based work. Opensource agents such as Codex [27] and OpenClaw [29] package nontrivial scaffolding around the base model, including prompting, tool orchestration, context management, and action-selection policies.

5 Conclusion

We introduced LONG-HORIZON-TERMINAL-BENCH, a benchmark of 46 containerized terminal tasks spanning nine domains and designed to stress long-horizon execution. Unlike outcome-only benchmarks, LONG-HORIZON-TERMINAL-BENCH uses deterministic, environment-grounded subtasks to provide dense partial-credit signals, allowing evaluation to capture not only whether an agent finishes a task, but also how far it progresses.

Across 15 frontier models under a shared `terminus-2` agent, tasks in LONG-HORIZON-TERMINAL-BENCH require substantial effort, averaging 231 episodes, 9.9M tokens, $10.5 in API cost, and 85.3 minutes per run. Yet the benchmark remains far from saturated: the strongest model, GPT-5.5, achieves only a 15.2% pass rate at $R \geq 0.95$, while the mean pass rate across models is just 4.3%. Dense rewards further reveal that many runs make meaningful but incomplete progress, exposing failure patterns that binary pass/fail would miss.

Our analysis suggests that the main bottleneck is not only local reasoning, but reliable long-horizon completion. Current agents often time out after making partial progress, and even voluntary early exits frequently reflect weak self-verification rather than true completion. These results highlight the need for better planning, stronger memory and progress tracking, and more calibrated stopping decisions for long-horizon tasks. We release LONG-HORIZON-TERMINAL-BENCH and its evaluation harness to support future work on agents that can plan, verify, and execute reliably over long horizons.

References

[1] ByteDance Seed Team. Seed2.1 officially released: Advancing ai productivity. https://seed.bytedance.com/en/blog/seed2-1-officially-released-advancing-ai-productivity, June 2026. Official model release announcement (Doubao Seed 2.1 / Seed 2.1 Pro).

[2] Sadia Sultana Chowa, Riasad Alvi, Subhey Sadi Rahman, Md Abdur Rahman, Mohaimenul Azam Khan Raiaan, Md Rafiqul Islam, Mukhtar Hussain, and Sami Azam. From language to action: a review of large language models as autonomous agents and tool users. *Artificial Intelligence Review*, 2026.

[3] Evan Chu, Rajan Agarwal, Abishek Thangamuthu, Brendan Graham, Justus Mattern, Freeman Jiang, Paul Cento, Swarnim Jain, Mersad Abbasi, Mohammad Hossein Rezaei, George Wang, Alex Zhang, Simon Guo, Karina Nguyen, Danna Liu, Arash Bidgoli, Aditya Dalmia, Apoorv Dankar, Ashrut Vaddela, Calvin Chen, Keshav Kumar, Kushagra Vaish, Navid Pour, Rishyanth Kondra, Sagar Badiyani, Sidharth Giri, Snagnik Das, Soham Gaikwad, Syed Shah, Vagish Dilawari, and Vishal Agarwal. Frontierswe. *Proximal Blog*, 2026. https://frontierswe.com/blog.

[4] DeepSeek-AI. Deepseek-v4: Towards highly efficient million-token context intelligence, 2026. URL https://arxiv.org/abs/2606.19348.

[5] Rishi Desai, Jesse Hu, Joan Cabezas, Neel Harsola, Pratyush Shukla, Roey Ben Chaim, Adnan El Assadi, Omkaar Mukund Kamath, Fenil Faldu, Prannay Hebbar, et al. Swe-marathon: Can agents autonomously complete ultra-long-horizon software work? *arXiv preprint arXiv:2606.07682*, 2026.

[6] Shangheng Du, Jiabao Zhao, Jinxin Shi, Zhentao Xie, Xin Jiang, Yanhong Bai, and Liang He. A survey on the optimization of large language model-based agents. *ACM Computing Surveys*, 58(9):1–37, 2026.

[7] Yukang Feng, Jianwen Sun, Zelai Yang, Jiaxin Ai, Chuanhao Li, Zizhen Li, Fanrui Zhang, Kang He, Rui Ma, Jifan Lin, et al. Longcli-bench: A preliminary benchmark and study for long-horizon agentic programming in command-line interfaces. In *Findings of the Association for Computational Linguistics: ACL 2026*, pages 29952–29963, 2026.

[8] GLM-5-Team, :, Aohan Zeng, Xin Lv, Zhenyu Hou, Zhengxiao Du, Qinkai Zheng, Bin Chen, Da Yin, Chendi Ge, Chenghua Huang, Chengxing Xie, Chenzheng Zhu, Congfeng Yin, Cunxiang Wang, Gengzheng Pan, Hao Zeng, Haoke Zhang, Haoran Wang, Huilong Chen, Jiajie Zhang, Jian Jiao, Jiaqi Guo, Jingsen Wang, Jingzhao Du, Jinzhu Wu, Kedong Wang, Lei Li, Lin Fan, Lucen Zhong, Mingdao Liu, Mingming Zhao, Pengfan Du, Qian Dong, Rui Lu, Shuang-Li, Shulin Cao, Song Liu, Ting Jiang, Xiaodong Chen, Xiaohan Zhang, Xuancheng Huang, Xuezhen Dong, Yabo Xu, Yao Wei, Yifan An, Yilin Niu, Yitong Zhu, Yuanhao Wen, Yukuo Cen, Yushi Bai, Zhongpei Qiao, Zihan Wang, Zikang Wang, Zilin Zhu, Ziqiang Liu, Zixuan Li, Bojie Wang, Bosi Wen, Can Huang, Changpeng Cai, Chao Yu, Chen Li, Chengwei Hu, Chenhui Zhang, Dan Zhang, Daoyan Lin, Dayong Yang, Di Wang, Ding Ai, Erle Zhu, Fangzhou Yi, Feiyu Chen, Guohong Wen, Hailong Sun, Haisha Zhao, Haiyi Hu, Hanchen Zhang, Hanrui Liu, Hanyu Zhang, Hao Peng, Hao Tai, Haobo Zhang, He Liu, Hongwei Wang, Hongxi Yan, Hongyu Ge, Huan Liu, Huanpeng Chu, Jia'ni Zhao, Jiachen Wang, Jiajing Zhao, Jiamin Ren, Jiapeng Wang, Jiaxin Zhang, Jiayi Gui, Jiayue Zhao, Jijie Li, Jing An, Jing Li, Jingwei Yuan, Jinhua Du, Jinxin Liu, Junkai Zhi, Junwen Duan, Kaiyue Zhou, Kangjian Wei, Ke Wang, Keyun Luo, Laiqiang Zhang, Leigang Sha, Liang Xu, Lindong Wu, Lintao Ding, Lu Chen, Minghao Li, Nianyi Lin, Pan Ta, Qiang Zou, Rongjun Song, Ruiqi Yang, Shangqing Tu, Shangtong Yang, Shaoxiang Wu, Shengyan Zhang, Shijie Li, Shuang Li, Shuyi Fan, Wei Qin, Wei Tian, Weining Zhang, Wenbo Yu, Wenjie Liang, Xiang Kuang, Xiangmeng Cheng, Xiangyang Li, Xiaoquan Yan, Xiaowei Hu, Xiaoying Ling, Xing Fan, Xingye Xia, Xinyuan Zhang, Xinze Zhang, Xirui Pan, Xu Zou, Xunkai Zhang, Yadi Liu, Yandong Wu, Yanfu Li, Yidong Wang, Yifan Zhu, Yijun Tan, Yilin Zhou, Yiming Pan, Ying Zhang, Yinpei Su, Yipeng Geng, Yong Yan, Yonglin Tan, Yuean Bi, Yuhan Shen, Yuhao Yang, Yujiang Li, Yunan Liu, Yunqing Wang, Yuntao Li, Yurong Wu, Yutao Zhang, Yuxi Duan, Yuxuan Zhang, Zezhen Liu, Zhengtao Jiang, Zhenhe Yan, Zheyu Zhang, Zhixiang Wei, Zhuo Chen, Zhuoer Feng, Zijun Yao, Ziwei Chai, Ziyuan Wang, Zuzhou Zhang, Bin Xu, Minlie Huang, Hongning Wang, Juanzi Li, Yuxiao Dong, and Jie Tang. Glm-5: from vibe coding to agentic engineering, 2026. URL https://arxiv.org/abs/2602.15763.

[9] Google DeepMind. Gemini 3.1 pro model card. https://deepmind.google/models/model-cards/gemini-3-1-pro/, 2026.

[10] Harbor Framework Team. Harbor: A framework for evaluating and optimizing agents and models in container environments, 2026. URL https://doi.org/10.5281/zenodo.20953922.

[11] 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. 2024. URL https://arxiv.org/abs/2403.07974.

[12] Carlos E Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik Narasimhan. Swe-bench: Can language models resolve real-world github issues? In *International Conference on Learning Representations*, volume 2024, pages 54107–54157, 2024.

[13] Muhammad Khalifa, Rishabh Agarwal, Lajanugen Logeswaran, Jaekyeom Kim, Hao Peng, Moontae Lee, Honglak Lee, and Lu Wang. Process reward models that think. *arXiv preprint arXiv:2504.16828*, 2025.

[14] Thomas Kwa, Ben West, Joel Becker, Amy Deng, Katharyn Garcia, Max Hasin, Sami Jawhar, Megan Kinniment, Nate Rush, Sydney Von Arx, et al. Measuring ai ability to complete long tasks, 2025. URL https://arxiv.org/abs/2503.14499.

[15] Tue Le, Minh VT Thai, Dung Nguyen Manh, Huy Phan Nhat, and Nghi DQ Bui. Swe-evo: Benchmarking coding agents in long-horizon software evolution scenarios. *arXiv preprint arXiv:2512.18470*, 2025.

[16] Zongxia Li, Wenhao Yu, Chengsong Huang, Zhenwen Liang, Rui Liu, Fuxiao Liu, Jingxi Che, Dian Yu, Jordan Boyd-Graber, Haitao Mi, et al. Self-rewarding vision-language model via reasoning decomposition. *arXiv preprint arXiv:2508.19652*, 2025.

[17] Zongxia Li, Dawei Liu, Fuxiao Liu, Yuhang Zhou, Xiyang Wu, Jingxi Chen, Jing Xie, Xiaomin Wu, and Lichao Sun. Comfyclaw: Self-evolving skill harnesses for image generation workflows. *arXiv preprint arXiv:2607.01709*, 2026.

[18] Hunter Lightman, Vineet Kosaraju, Yura Burda, Harri Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. Let's verify step by step, 2023. URL https://arxiv.org/abs/2305.20050.

[19] Haojia Lin, Xiaoyu Tan, Yulei Qin, Zihan Xu, Yuchen Shi, Zongyi Li, Gang Li, Shaofei Cai, Siqi Cai, Chaoyou Fu, et al. Cuarewardbench: A benchmark for evaluating reward models on computer-using agent. *arXiv preprint arXiv:2510.18596*, 2025.

[20] Minhua Lin, Juncheng Wu, Zijun Wang, Zhan Shi, Yisi Sang, Bing He, Zewen Liu, Tianxin Wei, Zongyu Wu, Zhiwei Zhang, et al. Harness updating is not harness benefit: Disentangling evolution capabilities in self-evolving llm agents. *arXiv preprint arXiv:2605.30621*, 2026.

[21] Yue Liu, Yingwei Ma, Yibo Miao, Yanhao Li, Yuchong Xie, Xinlong Yang, Zhiyuan Hu, Flood Sung, Jiaheng Zhang, and Bryan Hooi. Klong: Training llm agent for extremely long-horizon tasks. *arXiv preprint arXiv:2602.17547*, 2026.

[22] Mike A Merrill, Alexander G Shaw, Nicholas Carlini, Boxuan Li, Harsh Raj, Ivan Bercovich, Lin Shi, Jeong Yeon Shin, Thomas Walshe, E Kelly Buchanan, et al. Terminal-bench: Benchmarking agents on hard, realistic tasks in command line interfaces. *arXiv preprint arXiv:2601.11868*, 2026.

[23] MiniMax. Minimax sparse attention, 2026. URL https://arxiv.org/abs/2606.13392.

[24] Moonshot AI. Kimi k2.6: From code to creation, from one to many. https://www.kimi.com/ai-models/kimi-k2-6, 2026. Official Moonshot AI model release page (Kimi K2.6).

[25] Moonshot AI. Kimi k2.7 code: Open-source 1t agentic coding model. https://kimik2ai.com/k2.7/, June 2026. Kimi K2.7 Code, released June 12, 2026; model ID `kimi-k2.7-code`.

[26] Niels Mündler, Mark N Müller, Jingxuan He, and Martin Vechev. Swt-bench: Testing and validating real-world bug-fixes with code agents. *Advances in Neural Information Processing Systems*, 37:81857–81887, 2024.

[27] OpenAI. Codex. https://github.com/openai/codex, 2025. OpenAI coding agent / CLI.

[28] OpenAI. Gpt-5 system card. https://openai.com/index/gpt-5-system-card/, 2025. System card; also arXiv:2601.03267.

[29] OpenClaw Contributors. Openclaw, 2026. URL https://github.com/openclaw/openclaw. Open-source agent platform.

[30] Tianjun Pan, Xuan Lin, Wenyan Yang, Qianyu He, Shisong Chen, Licai Qi, Wanqing Xu, Hongwei Feng, Bo Xu, and Yanghua Xiao. Rubriceval: A rubric-level meta-evaluation benchmark for llm judges in instruction following. *arXiv preprint arXiv:2603.25133*, 2026.

[31] Qwen Team. Qwen3.6. https://qwen.ai/blog?id=qwen3.6, 2026. Official Qwen model release blog (Qwen3.6 Plus).

[32] Qwen Team. Qwen3.7. https://qwen.ai/blog?id=qwen3.7, 2026. Official Qwen model release blog (Qwen3.7 Max).

[33] Delip Rao and Chris Callison-Burch. Autorubric: Unifying rubric-based llm evaluation. *arXiv preprint arXiv:2603.00077*, 2026.

[34] David Rein, Joel Becker, Amy Deng, Seraphina Nix, Chris Canal, Daniel O'Connel, Pip Arnott, Ryan Bloom, Thomas Broadley, Katharyn Garcia, et al. Hcast: Human-calibrated autonomy software tasks. *arXiv preprint arXiv:2503.17354*, 2025.

[35] Akshit Sinha, Arvindh Arun, Shashwat Goel, Steffen Staab, and Jonas Geiping. The illusion of diminishing returns: Measuring long horizon execution in llms. *arXiv preprint arXiv:2509.09677*, 2025.

[36] Dingjie Song, Hanrong Zhang, Dawei Liu, Yixin Liu, Zongxia Li, Zhengqing Yuan, Siqi Zhang, and Lichao Sun. Dr. claw: An ai research workspace from idea to paper, 2026. URL https://github.com/OpenLAIR/dr-claw.

[37] Tencent Hunyuan. Tencent hunyuan 3. https://hunyuan.tencent.com/, 2026. Official Tencent Hunyuan model site.

[38] Zihang Tian et al. Arco: Adaptive rubric with co-evolution for multi-step llm-based agents, 2026. URL https://arxiv.org/abs/2606.21262.

[39] Jonathan Uesato, Nate Kushman, Ramana Kumar, Francis Song, Noah Siegel, Lisa Wang, Antonia Creswell, Geoffrey Irving, and Irina Higgins. Solving math word problems with process-and outcome-based feedback. *arXiv preprint arXiv:2211.14275*, 2022.

[40] Bertie Vidgen, Austin Mann, Abby Fennelly, John Wright Stanly, Lucas Rothman, Marco Burstein, Julien Benchek, David Ostrofsky, Anirudh Ravichandran, Debnil Sur, et al. Apex-agents. *arXiv preprint arXiv:2601.14242*, 2026.

[41] Xingyao Wang, Boxuan Li, Yufan Song, Frank F. Xu, Xiangru Tang, Mingchen Zhuge, Jiayi Pan, Yueqi Song, Bowen Li, Jaskirat Singh, et al. Openhands: An open platform for ai software developers as generalist agents, 2024. URL https://arxiv.org/abs/2407.16741.

[42] Xinyu Jessica Wang, Haoyue Bai, Yiyou Sun, Haorui Wang, Shuibai Zhang, Wenjie Hu, Mya Schroder, Bilge Mutlu, Dawn Song, and Robert D Nowak. The long-horizon task mirage? diagnosing where and why agentic systems break. *arXiv preprint arXiv:2604.11978*, 2026.

[43] Hjalmar Wijk, Tao Lin, Joel Becker, Sami Jawhar, Neev Parikh, Thomas Broadley, Lawrence Chan, Michael Chen, Josh Clymer, Jai Dhyani, et al. Re-bench: Evaluating frontier ai r&d capabilities of language model agents against human experts. *arXiv preprint arXiv:2411.15114*, 2024.

[44] Xiyang Wu, Zongxia Li, Guangyao Shi, Alexander Duffy, Tyler Marques, Matthew Lyle Olson, Tianyi Zhou, and Dinesh Manocha. Co-evolving llm decision and skill bank agents for long-horizon tasks. *arXiv preprint arXiv:2604.20987*, 2026.

[45] xAI. Grok. https://x.ai/, 2026. Official xAI site for the Grok model family.

[46] Tianbao Xie, Danyang Zhang, Jixuan Chen, Xiaochuan Li, Siheng Zhao, Ruisheng Cao, Toh Jing Hua, Zhoujun Cheng, Dongchan Shin, Fangyu Lei, et al. Osworld: Benchmarking multimodal agents for open-ended tasks in real computer environments, 2024. URL https://arxiv.org/abs/2404.07972.

[47] John Yang, Carlos E. Jimenez, Alexander Wettig, Kilian Lieret, Shunyu Yao, Karthik Narasimhan, and Ofir Press. Swe-agent: Agent-computer interfaces enable automated software engineering, 2024. URL https://arxiv.org/abs/2405.15793.

[48] Yilun Yao, Xinyu Tan, Chao-Hsuan Liu, Yaoming Li, Zhengyang Wang, Wenhan Yu, Zhewen Tan, Yuxuan Tian, Guangxiang Zhao, Lin Sun, et al. Harness-bench: Measuring harness effects across models in realistic agent workflows. *arXiv preprint arXiv:2605.27922*, 2026.

[49] Weizhe Yuan, Richard Yuanzhe Pang, Kyunghyun Cho, Xian Li, Sainbayar Sukhbaatar, Jing Xu, and Jason Weston. Self-rewarding language models. *arXiv preprint arXiv:2401.10020*, 2024.

[50] Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric Xing, et al. Judging llm-as-a-judge with mt-bench and chatbot arena. *Advances in neural information processing systems*, 36:46595–46623, 2023.

[51] 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, 2023. URL https://arxiv.org/abs/2307.13854.

A List of Tasks in Long-Horizon-Terminal-Bench

Table 2 lists all 46 benchmark tasks using the nine-category taxonomy used throughout the paper. The difficulty label is derived from the task's mean reward across model results Figure 3: tasks with mean reward at least 0.5 are labeled *Easy*, and the rest are labeled *Hard*.

Table 2 Task list for Long-Horizon-Terminal-Bench. Each row shows a task ID, its paper-level category, a one-sentence description, and a coarse difficulty label derived from average task reward in Figure 3 (*Easy* if mean reward ≥ 0.5, otherwise *Hard*).

Task ID	Category	Description	Difficulty
2048	Interactive games	Play 2048 turn-by-turn via a terminal game server, merging tiles to reach the highest tile, scored by replaying recorded moves in a fresh seeded engine.	Hard
alp-paper-reproduction	Research reproduction & ML	Reproduce a calibrated ALP floating-point compression experiment from the paper's method and sample columns.	Hard
apex-ib244-matter	APEX professional workflows	Advise on taking EdTech firm KSchool private across 38 reactive stages, rebuilding LBO/DCF/comps models for exact figures.	Easy
apex-investment-banking-matter	APEX professional workflows	Work a private-equity take-private of Planet Fitness through 19 reactive stages, rebuilding an LBO model and answering exact figures.	Hard
apex-law433-matter	APEX professional workflows	Act as outside counsel across four legal matters and 70 reactive stages over ~600 documents, submitting deterministically-graded answers.	Easy
apex-management-consulting-matter	APEX professional workflows	Work two management-consulting client matters across 33 reactive stages, rebuilding business-case and cost-savings models for exact answers.	Hard
apex-openroad-ibex-signoff	Software & reverse engineering	Close RTL-to-GDSII signoff on the ibex CPU with OpenROAD/sky130 across reactive stages until timing, DRC, LVS, area, and power pass.	Hard
audio-visual-event-alignment	Multimodal & imaging analysis	Repair an audio-video sync audit that detects visual events, audio onsets, dropped frames, and per-clip synchronization offsets.	Hard
chess-mate	Interactive games	Play chess as White against a bundled superhuman Stockfish engine, scored on how long the agent survives before being checkmated.	Hard
climate-netcdf-extreme-event-audit	Earth, climate & energy	Repair a climate audit pipeline computing gridded heatwave, precipitation, drought, anomaly, and index metrics over NetCDF datasets.	Hard
commit0-multilib-tdd	Software & reverse engineering	Reimplement three whole Python libraries from stubs with no provided tests, writing your own suite, graded on hidden real test suites.	Hard
dicom-radiology-audit	Multimodal & imaging analysis	Audit synthetic DICOM CT series geometry, HU conversion, windowing, masks, and lesion measurements.	Hard
document-table-layout-reconstruction	Multimodal & imaging analysis	Repair a scanned-document pipeline that detects table grids, assigns noisy OCR tokens to cells, and emits CSV/JSON/-Markdown tables.	Hard
duckdb-optimizer-closure	Systems, performance & security	Iteratively patch DuckDB's query optimizer to speed up TPC-H while preserving correctness.	Hard
epa-swmm-stormwater-regression-audit	Earth, climate & energy	Audit EPA SWMM stormwater hydraulic and hydrologic simulations against official solver outputs.	Hard
epidemic-inverse-control-audit	Scientific computing & simulation	Fit and control a multi-city, age-structured SEIR-H-ICU epidemic model.	Hard

Task ID	Category	Description	Difficulty
foldseek-paper-reproduction	Research reproduction & ML	Reproduce a calibrated Foldseek-style structural protein search experiment from the Nature Biotechnology paper.	Hard
gdal-proj-raster-regression	Earth, climate & energy	Reproduce GDAL/PROJ raster reprojection, coordinate transform, point sampling, and zonal-statistics regression checks.	Hard
generals-bot-arena	Software & reverse engineering	Develop a bot for a Generals.io-style grid strategy game by iteratively playing matches and refining its strategy.	Easy
grammar-fuzz-coverage-hunt	Systems, performance & security	Iteratively build a grammar-guided fuzzer to maximize code coverage of target parsers.	Easy
great-expectations-audit	Software & reverse engineering	Build a Great Expectations-based multi-table data-quality and reconciliation audit pipeline.	Hard
langchain-version-migration	Software & reverse engineering	Migrate a legacy LangChain RAG/router app across a breaking API change with hard-fail compatibility gates.	Hard
materials-phase-diagram-audit	Scientific computing & simulation	Repair a materials phase-diagram audit reconstructing phase boundaries, invariant reactions, lever-rule fractions, and convex-hull stability.	Hard
matpower-opf-regression	Earth, climate & energy	Reproduce MATPOWER/PYPOWER power-flow and optimal-power-flow regression checks with grid-constraint diagnostics.	Hard
microscopy-cell-count-qc-audit	Multimodal & imaging analysis	Repair a microscopy QC audit that segments synthetic cell imagery, flags focus/artifact failures, and reports per-tile channel counts.	Hard
modflow6-groundwater-regression-audit	Earth, climate & energy	Audit USGS MODFLOW 6 groundwater simulations against official solver outputs.	Hard
nbody-accel-iterative	Scientific computing & simulation	Iteratively optimize an N-body gravitational simulation from O(N^2) to O(N log N) with correctness verification at multiple scales.	Easy
nrel-pysam-hybrid-renewables-audit	Earth, climate & energy	Audit renewable generation and storage dispatch against NREL PySAM official PVWatts outputs.	Hard
opensees-seismic-structural-regression-audit	Scientific computing & simulation	Audit seismic structural-dynamics results against official OpenSeesPy finite-element solver outputs.	Hard
poc-exploit-craft	Systems, performance & security	Generate proof-of-concept inputs that trigger specific vulnerability classes using sanitizer feedback.	Easy
riscv-core-debug	Software & reverse engineering	Localize and fix undisclosed injected bugs in an out-of-order RISC-V SystemVerilog CPU until simulation, compliance, and formal checks pass.	Easy
robotics-slam-benchmark-repair	Research reproduction & ML	Repair a SLAM benchmark audit that composes SE(2) poses, evaluates loop closures, reconstructs landmark maps, and reports trajectory consistency.	Hard
rush_hour_campaign	Logic & constraint puzzles	Solve a four-stage 6x6 Rush Hour campaign by hand, submitting legal move routes under length caps.	Hard
satellite-flood-change-detection-audit	Multimodal & imaging analysis	Repair a satellite flood change-detection audit that compares before/after imagery, masks clouds, and reports per-tile flood extent.	Hard
scientific-figure-data-reconstruction	Multimodal & imaging analysis	Repair a pipeline that reconstructs plotted series data from scientific figure images, captions, and partial CSV references.	Hard
snake_maze_campaign	Interactive games	Play a deterministic hard Snake variant, eating seeded apples and surviving growing obstacles as long as possible, scored by replaying the move log.	Hard
sokoban	Logic & constraint puzzles	Solve a ramp of Sokoban puzzles turn-by-turn through a game server, advancing as far as possible, scored by replaying moves in a clean engine.	Hard

Task ID	Category	Description	Difficulty
spice-ephemeris-regression	Scientific computing & simulation	Reproduce NASA SPICE ephemeris, angle, and ground-station visibility regression checks from pinned kernels.	Hard
spot-scheduler-traces	Systems, performance & security	Implement and refine a cloud spot-instance scheduling policy that minimizes cost while meeting hard deadlines.	Easy
su2-airfoil-regression	Scientific computing & simulation	Reproduce a hardened SU2 aerodynamic regression matrix from official test-case references and solver history artifacts.	Hard
sudoku-recovery	Logic & constraint puzzles	Recover seven progressively corrupted hard variant-Sudoku puzzles turn-by-turn, detecting subtle corrupt givens via global constraint reasoning.	Hard
super-mario	Interactive games	Play Super Mario Bros turn-by-turn through a game server, clearing as many levels as possible, scored by replaying frame inputs in a fresh emulator.	Hard
tabular-data-feature-covshift	Research reproduction & ML	Recover a sparse high-dimensional signal from low-dimensional observations under covariate shift, with a leakage-audited honesty gate.	Hard
unison-paper-reproduction	Research reproduction & ML	Reproduce a calibrated UNISON fat-tree network simulation experiment from the paper's method and settings.	Hard
unknown-config-semantics	Software & reverse engineering	Reverse-engineer an undocumented config format from a noisy spec and repair its normalization engine to pass hidden probes across five stages.	Hard
vector-db-iterative-build	Systems, performance & security	Build an approximate nearest-neighbor vector search service from scratch, iteratively optimizing recall and throughput.	Hard