PROBE-AND-RESUME: WHEN AND HOW TO EVALUATE AGENT EXECUTION

Anonymous authors
Paper under double-blind review

ABSTRACT

Test-time search can improve agent performance by exploring alternative actions and using a critic to select promising branches. We study this approach in terminal-based computer-use environments, which expose only a partial view of a large underlying state, and find that the Monte Carlo tree search (MCTS) configuration we evaluate does not significantly improve task success. We identify two sources of poor critic performance: the agent's observations omit environment state needed for evaluation, and intermediate states offer limited evidence of whether a trajectory will succeed. At intermediate states, critics rank continuations only modestly better than chance, and environment access improves this little. We study when to invoke a critic and what access to give it to evaluate a node. Our findings motivate a policy that invokes the critic only at nodes the agent believes are final, rather than during partial progress. Further, the policy gives the critic read, write, and execute access to a fork of the computer's current and initial state, allowing it to probe and perform potentially destructive actions to surface any hidden state the trajectory omits. We call the resulting design Probe-and-Resume: when the agent claims to be done, the critic probes a fork of its state, and a rejected agent resumes where it stopped. On Terminal-Bench 2.1, this raises success by 8.4 percentage points over MCTS with conventional critics, and approaches the success rate of the Codex CLI agent with the same underlying model at less than half its cost.

1 INTRODUCTION

Terminal agents accomplish tasks by changing a computer's state: editing files, installing packages, running programs, and configuring services. These tasks often require long sequences of actions, and a mistaken command can destroy useful state or send subsequent work down an unsuccessful path. Recently developed branchable execution environments (Xu et al., 2025; Dong et al., 2026; Wu et al., 2026) allow an agent to snapshot a running environment, restore earlier states, and explore alternatives in isolated forks. This makes test-time search practical for agents.

Unfortunately, branching alone does not tell an agent which execution trajectories to pursue. Search also needs an evaluation signal: which states are promising, which require further work, and which satisfy the task? In most automation settings, the task's ground-truth verifier is unavailable during execution, and search must rely on a critic to evaluate candidate states to guide which executions to continue and when the task is done.

This setting places importance on critic design, namely: when should the critic evaluate an execution, and what access does it need to produce a useful judgment? An intermediate state asks the critic to assess its promise and whether continuing is likely to succeed, whereas a state the actor declares is done requires the critic to assess whether the task's requirements have been met. Both are challenging in terminal environments because the actor typically only records its reasoning, the tool calls, and the outputs it observed. In contrast, the search state also includes that of the computer, which encapsulates files, running processes, memory, installed packages, and databases, which may be changing as side-effects of agent actions or concurrent processes, and are not visible to the actor or the critic.

We investigate these questions on Terminal-Bench 2.1 (Marten et al., 2026). We find that merely making branching available does not improve task success because the actor rarely uses snapshot and restore capabilities, and a Monte Carlo tree search (MCTS) search policy guided by a trajectory-only

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

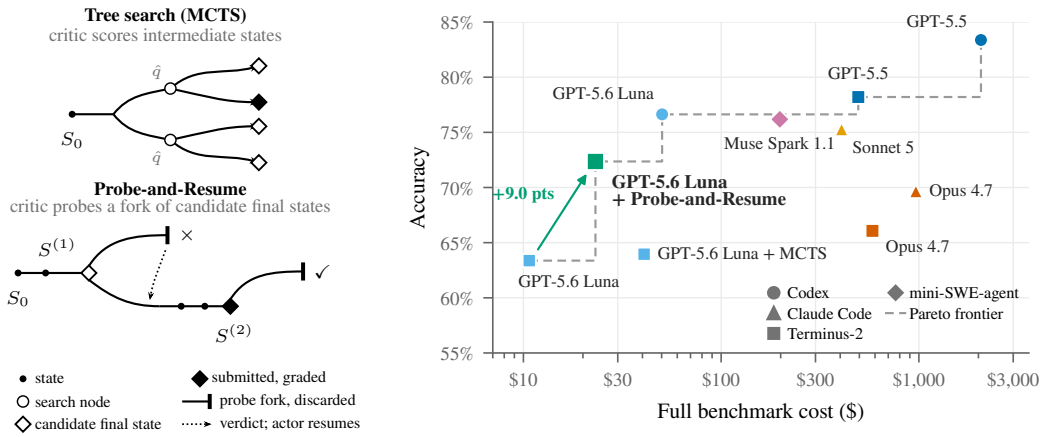


Figure 1: Left: tree search scores partial states with a critic and submits a leaf; Probe-and-Resume calls the critic only when the actor claims to be done, lets it probe a fork of that state, and resumes the actor on a rejection. Right: accuracy against the cost of running all of Terminal-Bench 2.1 (89 tasks, five trials each); the shape is the agent harness. The GPT-5.6 Luna squares are our three configurations (no search, MCTS, Probe-and-Resume), priced at the AWS Bedrock rate like GPT-5.6 Luna + Codex; the other points and their costs are from Harbor Hub (Appendix C).

critic produces little improvement. One reason for the latter is that the critics we test are poor value estimators of intermediate states, and access to a fork of the environment helps little. However, when the actor signals completion, execute access to a fork of the environment substantially improves their ability to distinguish successful from unsuccessful outcomes. These findings motivate concentrating critic computation at candidate final states, where it can test the actor's claim against the environment.

Our main observation is that *evaluating a terminal task is itself a terminal task*. Thus, neither the interaction history alone nor simply inspecting the environment is always sufficient to assess that a script, program, configuration change, or service is correct. The critic must obtain evidence by running programs and constructing checks. Branchable environments make such evaluation, which may modify or destroy state, possible by using a disposable fork. A fork of the initial environment additionally provides a reference for requirements about what must be preserved or changed.

We therefore design Probe-and-Resume (Figure 1(left)): when the actor signals completion, the critic assesses the candidate with terminal access to forks of that state and of the initial environment. If it accepts, the candidate is submitted; otherwise the actor receives its findings and resumes from the unmodified state. Unlike MCTS, the critic judges completions, not intermediate states.

On all 89 tasks of Terminal-Bench 2.1, with five trials per task and GPT-5.6 Luna as the actor and the basic Terminus-2 harness, Probe-and-Resume increases success from 63.4% at the actor's first completion claim to 72.4% at submission, expanding the accuracy–cost Pareto frontier without the need for new models nor new harnesses (Figure 1(right)). The resulting accuracy is higher than Opus 4.7 using Claude Code, and over an order of magnitude cheaper. On SWE-bench Pro, the gain is similar. Together, these results highlight the unique characteristics of critics in computer environment tasks, and how critics can turn branchable execution into useful feedback for terminal agents when ground-truth verification is unavailable.

2 Background and Related Work

Test-time search and evaluation. Search uses additional inference compute to explore candidate solutions. Policy- and value-guided search has improved performance in games (Silver et al., 2016; 2018), mathematical reasoning (Hao et al., 2023; Guan et al., 2025), and software and web tasks (Zhou et al., 2024; Koh et al., 2024; Antoniades et al., 2024). Sampling-based methods evaluate completed candidates or aggregate their answers (Cobbe et al., 2021; Wang et al., 2023; Brown et al., 2024); beam search, best-first search, and MCTS also use evaluations to allocate computation among partial solutions (Xie et al., 2023; Yao et al., 2023; Kocsis & Szepesvári, 2006; Coulom, 2007; Browne et al.,

2012). Process evaluation has proved useful in mathematical reasoning (Lightman et al., 2024; Wang et al., 2024), but imperfect assessments can cause search to discard useful candidates (Stroebl et al., 2026; Yu et al., 2025). We examine intermediate predictions and completion judgments separately, because they ask different questions about an execution state.

Feedback and iterative refinement. Self-Refine uses the same model to generate an output, critique it, and revise it (Madaan et al., 2023). Reflexion stores verbal reflections from previous trials to guide subsequent attempts (Shinn et al., 2023). Its evaluator can use environmental feedback, heuristics, model judgments, or generated tests; an evaluator and a feedback loop are therefore established components. Our method also uses feedback to revise an attempt. It preserves both the actor's environment state and trajectory when it claims the task is done, runs the critic in forks, and resumes the actor from that same candidate after rejection. This differs from resetting the environment for another trial, as in Reflexion's ALFWorld experiments (Shinn et al., 2023).

External evidence for critics. Huang et al. (2023) find that intrinsic self-correction, without external feedback, can fail to improve reasoning and can reduce accuracy in the settings they study. CRITIC obtains external evidence through tool interaction before revising an output (Gou et al., 2023), including evidence from search and code execution. Other work executes tests against candidate programs or patches (Li et al., 2022; Jain et al., 2025; Ehrlich et al., 2025; Li et al., 2025), or gives judges tools to inspect and interact with an environment (Zhuge et al., 2024; Shi et al., 2026). These methods establish the value of tool-based feedback. We study the additional choices required when the object being judged is a live environment: whether the critic should inspect intermediate or completed states, which operations it needs, and whether the current state contains the evidence required by the task. The benchmark verifier and its output are not available during search.

Judging agent trajectories. Web-agent search uses model assessments of interaction histories (Koh et al., 2024); software-agent discriminators and reward models also use trajectories and patches (Antoniades et al., 2024; Shum et al., 2025). On Terminal-Bench 2, pairwise judgments select among completed trials and recover part of the gap between a single attempt and oracle selection over five samples (Kwok et al., 2026). Judge performance varies across web-agent benchmarks (Lù et al., 2025), and execution-free patch judgments can rely on stylistic features (Jain et al., 2025). These findings motivate measuring critic errors directly, alongside task success.

Critics in branchable environments. Terminal tasks can depend on files, running processes, memory, and changes made during execution, only part of which appears in the interaction history. Revisiting a state can require replaying actions (Koh et al., 2024). Waypoint (Xu et al., 2026) instead supports snapshots, restores, and isolated forks of the filesystem, processes, memory, and terminal session. We use this capability to let a critic execute checks on a disposable fork of the candidate final state, including checks that modify or destroy state. A second fork of the initial environment provides a reference for preservation and recovery requirements. Only the critic's feedback returns to the actor; its mutations are discarded, and the actor resumes from the candidate state. Our contribution combines this evaluation procedure with an empirical analysis of when critics make useful judgments and which errors arise from missing execution access or initial-state evidence (§3).

3 Critic Design for Search in Terminal Environments

We first compare branching tools and MCTS with the no-search baseline (§3.1). We then examine three critic design choices: when to evaluate a state and what operations the critic may perform (§3.2), and whether it needs the initial state as a reference (§3.3).

Agent setup. Following policy- and value-guided search, we separate action generation, state evaluation, and search (Silver et al., 2016; Zhou et al., 2024). The actor chooses and executes actions, the critic evaluates states, and the search algorithm selects which states to continue and when to stop. Each search node stores environment state S_t and the trajectory prefix $\tau_{\leq t}$ that produced it. The environment supports $\textsc{snapshot}(S) \rightarrow i$, $\textsc{restore}(i)$, and $\textsc{fork}(i)$, where i identifies a saved state and the fork is an isolated live copy.

Experimental setup. We use all 89 tasks in Terminal-Bench 2.1 (Marten et al., 2026), run through Harbor (Harbor Framework Team, 2026) with the Terminus-2 agent and GPT-5.6 Luna. The branchable environment uses Waypoint (Xu et al., 2026), which creates, restores, and forks a Docker-like container's state. By default, the critic uses the same model as the actor and receives the task instruction and $\tau_{\leq t}$. Once search submits a final state, we run the verifier. Each task runs five times and we report pass@1 with 95% bootstrap intervals over tasks (Appendix D). Each trial gets the task's time limit T (Appendix B), and T covers everything the trial does: the actor's turns, every critic call, and every snapshot, restore and fork. Only the verifier's grading is not charged. A trial that ends with no state to submit counts as a failure. We drop the 18 trials that could not be graded, all from the two search baselines (Appendix E). Costs are model tokens only, priced at the AWS Bedrock rate (Appendix C). The configuration of each search arm is in Appendix E.

3.1 Search Baselines

We compare two search baselines. The first gives the actor SNAPSHOT and RESTORE tools, without a separate critic. The second uses MCTS with a trajectory-only critic: it calls SNAPSHOT every five turns, expands at most two children, and selects nodes using UCT (Kocsis & Szepesvári, 2006).

Table 1 shows that branching tools change pass@1 from $63.4\% \rightarrow 61.0\%$, while MCTS changes it $63.4\% \rightarrow 64.0\%$ at $3.81\times$ the model cost. Branching tools are rarely used, and the evaluated MCTS configuration provides little improvement: the actor called branching tools in 3 of 445 trials: it saved 1 snapshot and restored 3 times (twice to S_0). For instance, in the DB-WAL-RECOVERY task, restoring S_0 recovers a write-ahead log removed after the actor opens the database. In PATH-TRACING, the actor saves a state, tries a change, and restores the saved state when the change worsens the result.

These results motivate measuring critic assessments directly; they do not by themselves establish why search fails to improve performance.

Table 1: Branching tools and MCTS don't improve pass@1 on Terminal-Bench 2.1 (GPT-5.6 Luna).

	Pass@1	**Δ vs. baseline [95% CI]**	**Total model cost**
Baseline	63.4%	–	\$10.70
Actor + branching tools	61.0%	$-2.4\,[-6.7, +2.0]$	\$11.40
MCTS with trajectory-only critic	64.0%	$+0.6\,[-4.5, +5.5]$	\$40.79

3.2 What the Critic Can Do and When to Invoke It

We now vary the critic's environment access and the intermediate state it evaluates. We find that environment access improves the critics' predictions at intermediate states only slightly, but improves their judgments at candidate final states considerably. These results motivate concentrating execution-based assessment at candidate final states in our proposed search algorithm. These measurements do not rule out search with a stronger value estimator or another search policy, and do not show that completion is the optimal time to intervene.

Critic variants. C^{traj} receives no environment tools. C^{read} can inspect a fork of the state using read-only commands such as `cat`, `ls`, `grep`, and `diff`. C^{act} can execute arbitrary commands in the fork, including installing packages, writing files, running tests, and taking destructive actions (prompts in Appendix A). Comparing these variants distinguishes evidence in $\tau_{\leq t}$ from evidence obtained by inspection or execution. The two tool-enabled critics issue commands until they return an assessment or reach their limits (Appendix B).

Evaluation positions and metrics. We evaluate each critic at 25%, 50%, and 75% of the actor's trajectory and at its candidate final state. At intermediate states, the critic estimates continuation success, $\hat{q}(S_t)$. We compare this estimate with the pass rate of 5 rollouts from S_t, using the same actor and remaining task budget. At a candidate final state, the critic returns a verdict $\hat{v}$, which is compared with the verifier's output. Critics do not observe the sampled continuations or verifier outcomes. We evaluate 1111 intermediate and 406 candidate final states.

Intermediate and candidate final states have different targets. We score each kind of state by the decision search makes there, and we do not compare scores between these two states (Figure 2). At

an intermediate state, search chooses which state to continue, so the critic's ranking of states matters. We report within-task pairwise accuracy: among pairs of states from the same task and depth, the fraction that the critic orders the same way as their pass rates. A critic that guesses at random scores 50%. Each pass rate is estimated from only 5 rollouts, so sampling noise also pulls the measured accuracy towards 50% and understates the critic's ranking ability (Appendix F). At a candidate final state, search accepts or rejects, so we report the false-reject and false-accept rates.

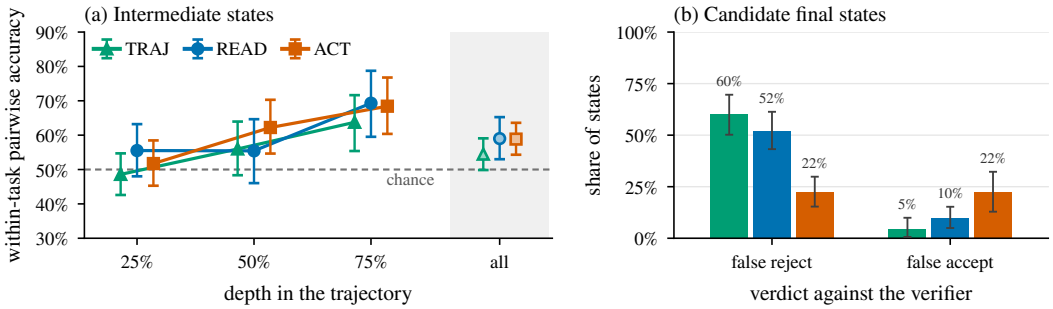


Figure 2: Critics by position in the actor's trajectory, on 89 Terminal-Bench 2.1 tasks (numbers in Appendix F); TRAJ, READ, ACT are C^{traj}, C^{read}, C^{act} (§3.2), TRAJ with the plain prompt (variants in Appendix H). **(a)** Share of same-task, same-depth pairs that $\hat{q}$ orders like the pass rates of their 5 continuations; *all* pools the depths; the dashed line marks chance (50%). **(b)** Passing states that $\hat{v}$ rejects and failing states it accepts.

Results by position. At intermediate states, the critics rank states better than chance late in the trajectory (64%–69% at 75% depth), but not early (49%–56% at 25%; Figure 2a). At 25% depth, we do not detect a statistically significant selection gain over random choice (Appendix F). Environment access changes this little: pooled over the depths, the gain of C^{act} over C^{traj} is +4.4 points $[-0.9, +9.9]$. At candidate final states, C^{act} reduces the misclassification rate from 36%–39% to 22%. Most of the gap is in false rejects: over 50% for C^{traj} and C^{read}, around 20% for C^{act} (Figure 2b).

Completion errors by verifier type. Table 2 groups tasks by what their verifier uses to evaluate: reading files in the environment, executing an artifact or probing a service, or comparing against a reference value unavailable from the environment (Appendix B). A false reject is a correct state incorrectly rejected, and a false accept is the opposite.

Table 2: Verdicts at candidate final states against the verifier, over 406 states of 89 tasks, grouped by what the verifier does to decide a task: it *reads* files and compares them with a value or format it stores (22 tasks), *runs* the artifact or probes a service (48), or *holds the answer*, a value that cannot be recovered from the environment (19). FR: share of passing states rejected. FA: share of failing states accepted. Model cost per state in Appendix G.

	reads		runs		holds the answer		all	
	FR↓	FA↓	FR↓	FA↓	FR↓	FA↓	FR↓	FA↓
C^{traj}	46%	5%	64%	3%	68%	7%	60%	5%
C^{read}	42%	16%	54%	5%	62%	12%	52%	10%
C^{act}	9%	19%	24%	19%	38%	31%	22%	22%

Even when the verifier only reads files, C^{read} rejects 42% of passing states, compared with 9% for C^{act}. In 93% of these read-only false rejects, the critic reports insufficient evidence rather than a violation of a requirement. The critic may need to compute a check even when the verifier can compare the result against a known reference.

When the verifier relies on a reference value that cannot be derived from the environment, C^{act} falsely accepts 31% of failing states, compared to 19% on tasks whose verifier executes the artifact or probes a service. The critic tends to agree with the actor's interpretation of the task completion and cannot reach the verifier's (hidden) reference.

3.3 What Information to Give the Critic

Some requirements refer to the initial state: a file must remain unchanged, a deleted input must be recovered, a program must keep its original behavior. The candidate final state alone cannot settle them, and in §3.2 such requirements account for 15 of C^{act}'s 56 false rejects. We therefore compare C^{act} with $C^{\text{act}}{+}S_0$, which receives a live fork of the initial state. We rebuild the candidate final states of 89 tasks and let each critic judge every state twice from the same snapshot (Appendix I). The two verdicts per state also measure a critic's self-consistency.

Over all 404 states (Figure 3), the fork raises accuracy from 76% to 80% (+4.0 points [+0.6, +7.5]). The gain comes from false rejects, which fall from 24% to 17% (−6.5 points [−11.3, −2.2]); false accepts do not move (+0.3 points [−4.4, +5.0]), and disagreement between a critic's two verdicts on a state drops from 18% to 9% (−9.4 points [−15.4, −3.8]).

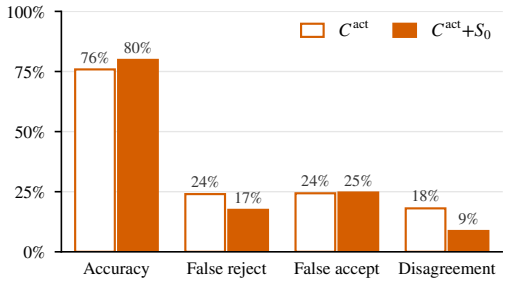


Figure 3: C^{act} against $C^{\text{act}}{+}S_0$ on all 404 candidate final states. Each critic judged every state twice, and every rate counts both verdicts. Disagreement is the share of states on which one critic's own two verdicts differ.

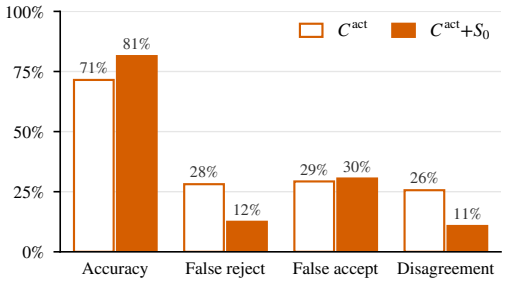


Figure 4: The same comparison on the 121 states on which $C^{\text{act}}{+}S_0$ inspected the fork of the initial state in at least one of its two evaluations.

$C^{\text{act}}{+}S_0$ uses the fork in 25% of its evaluations, on 121 states. The critic chose these states itself, so they show where the gain arises, not the effect of the fork on an average state. On them (Figure 4), accuracy rises from 71% to 81% (+9.9 points [+2.0, +18.2]) and false rejects fall from 28% to 12%; on the other 283 states, accuracy changes by +1.4 points [−1.3, +4.1]. In MODERNIZE-SCIENTIFIC-STACK, for example, C^{act} rejects because it cannot check whether a legacy file was preserved; $C^{\text{act}}{+}S_0$ compares the file's hash in both states and accepts. The initial state thus gives the critic more evidence for its checks: its verdicts become more accurate and consistent, not stricter.

4 Probe-and-Resume

The previous section identified a number of unique design decisions for critics operating in branchable terminal environments. We found that their value estimates for intermediate states are no better than chance early in a trajectory and improve little with execution access, while unrestricted execution in a fork of the current state and of the initial state helps them judge candidate final states. Based on these findings, we introduce the Probe-and-Resume test-time search policy for terminal agents, and report on its end-to-end evaluations on terminal benchmarks.

4.1 How Probe-and-Resume Works

Figure 5 shows the search policy. The actor starts in S_0 and executes as normal without knowledge of the critic. When it decides to call `task_complete` at $S^{(1)}$, we use $C^{\text{act}}{+}S_0$ (§3.3) to evaluate the proposal. The critic lists the requirements in the task and executes in the disposable forks of $S^{(1)}$ and S_0 to check each requirement. Thus, the critic can delete files, regenerate files, copy data between forks, install interpreters, start or kill processes, and take other potentially destructive actions.

If any requirements fail, the actor receives the failures, the rationale for why they failed, and how the critic checked. The actor resumes from $S^{(1)}$, and observes from the `task_complete` call that the state was rejected, the critic's feedback, and a request to either address the issues or submit a command

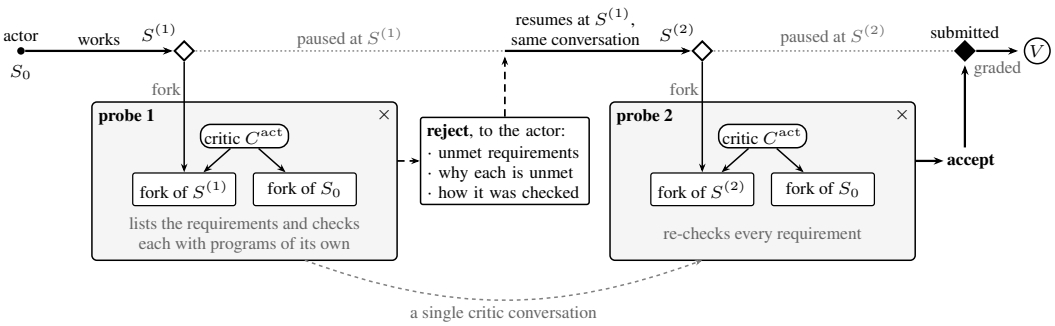


Figure 5: Probe-and-Resume. When the actor claims to be done, the critic uses forks of $S^{(r)}$ and S_0 and can freely execute actions to evaluate the state; they are then discarded ($\times$ symbol). If the critic rejects, the actor resumes at $S^{(r)}$ with the critic's feedback. Otherwise, $S^{(r)}$ goes to the verifier V.

that proves a failed requirement actually holds. The actor works until it calls `task_complete` again and then submits another proposal. We write $S^{(r)}$ for the r^{th} proposed state.

When the actor submits $S^{(2)}$, the critic continues its conversation from when it assessed $S^{(1)}$. It is told that the actor has revised its work, is shown the commands the actor ran, and is given fresh forks. It resumes from its previous requirements list and checks the entire list again. If the critic accepts the state, it is submitted to the verifier as the final result.

We execute search until the critic accepts a state, until six critic rounds have passed, or until a budget runs out. The task has a total time limit of T, shared by the actor's turns, the critic's evaluations and every snapshot, restore and fork; only the verifier's grading is not charged. After the first rejection, the actor also has at most 50 more turns. If the critic did not accept any state, the last proposed state is submitted to the verifier; if no such state exists, it is considered a failure.

4.2 TERMINAL-BENCH 2.1

We run all 89 Terminal-Bench 2.1 tasks 5 times each, using GPT-5.6 Luna as the model in all settings. We compare with a baseline that submits the first proposed final state, the Codex CLI harness, and Probe-and-Resume. Codex results are drawn from Harbor Hub's public leaderboard: Codex CLI with GPT-5.6 Luna on the same 89 tasks, five trials each, under the same per-task time limits. Two differences favor Codex. Its T covers only the agent, while ours also covers the critic, and a Codex trial that reaches T is graded on the state it left, while ours fails unless it has proposed a state. It also uses its own harness and sandboxes and calls the model through OpenAI's API rather than Bedrock. We count one Codex trial whose verifier timed out as a failure, and price its tokens at our rate.

Table 3 shows that Probe-and-Resume improves by 9 percentage points ($63\% \rightarrow 72\%$) over the baseline, and is close to Codex CLI's performance. It improves the pass rate on 30/89 tasks and regresses on 1. The improvements are comparable when using Claude Sonnet 5 (9.9 points) or GLM-5 (7.2 points) as the critic's model (Appendix J). While Probe-and-Resume increases the cost to $2.2\times$ the baseline's ($\$10.70 \rightarrow \$23.14$), it remains half the cost of Codex and only 4.2 points lower in accuracy. This is because the critic is only called when the actor believes it is done, rather than during many intermediate turns. To separate resuming from the extra round of feedback, we also restart the actor from S_0 after each rejection, with the same critic, feedback and time limit (Appendix K). Restarting costs $1.6\times$ as much as resuming and takes longer, and its pass rate is not higher (Table 3).

Detailed Analysis. To understand why the critic improves the pass rate, Figure 6a depicts a flow chart between the true pass rate of the first state the actor proposes to the critic, the critic's assessment of it, and the true pass rate of the final submitted state. Of the 136 proposals that fail, the critic correctly rejects 103 (76%), and the actor eventually is able to fix and "rescue" 43 of them (42%). Of the 282 passing proposals, the critic falsely rejects 60 (21%) and the actor eventually "rescues" 57.

Of the 123 submissions that fail, 29 were the critic's false accepts, the critic timed out on 4, and 3 were falsely rejected. The other 60 were correctly rejected by the critic, but the actor was not able to

Table 3: Pass@1 on Terminal-Bench 2.1 over 5 trials per task; cost is the model cost of one trial of every task at the AWS Bedrock rate (Appendix C). Restart from S_0 restarts the actor from S_0 after each rejection (Appendix K); Codex CLI is Harbor Hub's job (§4.2), priced at the same rate.

	Pass@1	**Δ vs. baseline [95% CI]**	**Total model cost**
Baseline	63.4%	–	$10.70
Probe-and-Resume	**72.4%**	+9.0 [+5.6, +12.6]	$23.14
Restart from S_0	66.3%	–	$38.12
Codex CLI (reference)	76.6%	–	$50.29

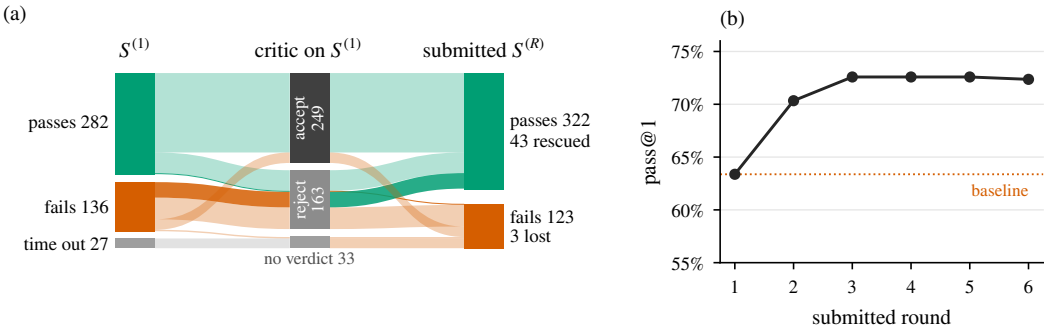


Figure 6: (a) Flow diagram that describes the true pass rate of the first state the actor proposes to the critic $S^{(1)}$, the critic's assessment of it, and the true pass rate of the final state submitted to the verifier. Trials that exceed the time limit count as failures. (b) Pass@1 if stopped after K rounds.

improve them. The remaining 27 ran out of time before the actor's first claim, so the critic never saw them. Of the trials that claim to be finished, 64% end after one round. Allowing a second claim raises the submitted pass rate from 63% to 70%, 78% of the loop's gain, and a third to 73%; further rounds add nothing measurable (Figure 6b), so a cap of three claims would lose nothing.

The critic's false accept and reject rates (21%, 21%) are comparable to those measured in §3.2 for C^{act} at candidate final states (22%, 22%). A benefit of Probe-and-Resume is that after a false rejection the actor does not need to start over from the initial state, and can resume immediately.

4.3 SWE-bench Pro

To study whether Probe-and-Resume generalizes to non-Terminal-Bench tasks, we then evaluate Probe-and-Resume (unchanged) with GPT-5.6 Luna on SWE-bench Pro (Deng et al., 2025), using the public split of its second release, V2.0.0 (642 tasks from 11 open-source repositories). We use two task sets, both defined by data released with the benchmark: *Hard*, its HARD-51 subset, and *Medium*, tasks that none of its runs of 2025 models solved. We report one trial of each of the 227 tasks we can grade offline (41 hard, 186 medium) under the 50-minute limit; Appendix M gives the exact sets and four more trials of each hard task.

V2's locked protocol runs the agent offline except for the model endpoint, as we do, but grades the agent's `git diff` applied to a fresh image. We instead run the same, unchanged verifier on a fork of the proposed state, as on Terminal-Bench. The verifier first resets the repository's test code. The hidden tests enter only that fork, and neither the actor nor the critic sees them or the grade. Since the sandbox had no network, no trial could download packages.

Figure 7 shows that while the baseline that submits the first proposed state has a pass rate of 80%, Probe-and-Resume increases the pass rate to 88%, comparable to the improvements on Terminal-Bench; the gain comes from the medium tasks, and on the hard tasks it cannot be told apart from zero (Appendix M). The benefits are similar because the critic rejects 36 of the 45 failing first proposals (80%) and the actor turns 22 of them into passing states. The critic also falsely rejects 46 of the 182 passing first proposals (25%), and the actor is able to rescue 41 of them. It costs 2.2× as much.

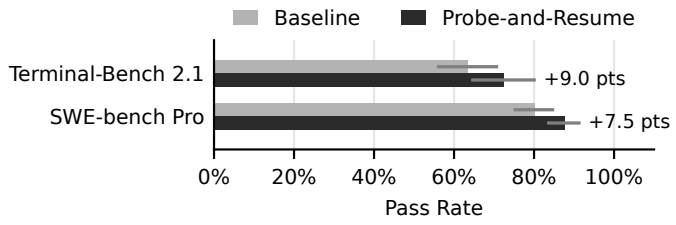


Figure 7: Probe-and-Resume on Terminal-Bench 2.1 and Medium/Hard tasks in SWE-bench Pro.

4.4 Discussion and Limitations

Branching for assessment and repair. This work shows that branching is not only useful for exploring alternative actions, but also for critics to evaluate a candidate without changing the state the actor may resume from. It also allows the critic to compare with other states as a reference (such as the initial state). We find that critics need not be fully correct, as the actor and critic pair can "rescue" seemingly failing states. The actor also avoids restarting from the beginning after a critic rejection, which lowers success by 6.1 points and increases model cost (Appendix K).

Scope and model dependence. Critics help when actors can repair mistakes they fail to identify themselves; easy tasks the actor can solve leave little room for critics to help, while tasks that are too hard may not be solvable by the actor. As weak evidence, we find no statistically significant gains using Astra and Opus as actor or critic on even hard Terminal-Bench tasks (Appendix L). Our evaluation manages environment state using Waypoint's terminal-oriented container. Other settings require branching support for their state, such as TClone for GUI workspaces (Huang et al., 2026). Forks do not isolate writes to external databases or shared services, so destructive checks require isolating those dependencies or restricting critic actions.

Sequential and parallel search. Probe-and-Resume spends test-time compute sequentially. Parallel search spends compute on breadth instead, drawing k independent trials and selecting one of them. The two are complementary as each of the k trials can itself be a Probe-and-Resume loop, and the acting critic's final verdicts can evaluate on forks of each trial's environment state to select a winner. We found that pass rates converge after three rounds (§4.2), so compute may be better spent on parallel trials. We leave studying this interaction to future work.

Towards Complex Computer Tasks. As computer tasks involve richer data and interacting systems, correctness will increasingly encompass the entire environment state *and* how it changes. Existing benchmarks already point towards this: AppWorld checks for unintended database changes (Trivedi et al., 2024), ToolSandbox evaluates dependencies and required events throughout execution (Lu et al., 2025), and STRATUS checks whether interventions worsen system health and relies on undoing them (Chen et al., 2025). Even as models increase in capability, they will likely benefit from branchable environments, as well as the critic's ability to effectively use branching capabilities to execute checks within and across intermediate and final environment states.

5 Conclusion

We studied when terminal-agent critics provide useful assessments and what access they need. The tested critics make weak predictions of success from intermediate states, even with execution access. Once the actor claims to be done, a critic that runs checks in a fork of that state judges it far more accurately. A fork of the initial state also lets it check what must be preserved or recovered. These findings motivate Probe-and-Resume: the critic checks candidate completions in disposable forks, and the actor resumes from the unchanged candidate with feedback after rejection. On Terminal-Bench 2.1, Probe-and-Resume raises pass@1 from 63.4% to 72.4%, at less than half the cost of Codex CLI with the same model. These results show that branching alone does not make search effective; it helps when a critic uses it to check completed work, because evaluating a terminal task is itself a terminal task.

AI USE STATEMENT

In this work, we used generative AI tools to provide feedback on experiment results, and implement methods. We have not used generative AI tools to help develop theoretical models or conceptual frameworks, assist in the writing of proofs, propose or refine hypotheses, clean and reformat dataset, support qualitative and thematic data analysis, interpret results, and generate synthetic data sets, formulate mathematical claims, provide critical ingredients for proving mathematical claims, assist with translation are not applicable to this work. Additionally, we used generative AI tools to create or modify scientific figures or images, sourcing/searching for information, identify relevant literature. We have reviewed all AI-assisted work. LLM-generated code was verified and tested for correctness by authors. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

ETHICS STATEMENT

We take research ethics seriously and have reviewed this work against the ICLR Code of Ethics. This work does not involve human subjects, personal data, or the release of a new dataset. All experiments run on the public verified benchmarks and the models we evaluate are used through their providers' APIs under their terms of service. We are not aware of ethical concerns specific to this work. We remain committed to conducting and reporting our research responsibly, and to releasing our code and logs so that the community can examine and build on it.

REPRODUCIBILITY STATEMENT

The framework and the semantics of its directives are given in §3, and the search policy of Probe-and-Resume in §4.1. The full text of the actor's and every critic's prompts is in the anonymous repository (Appendix A). The task corpus, the per-task timeouts, and the verifier annotation are listed in Appendix B. The statistical procedure behind every interval and paired comparison is described in Appendix D. Configurations and the search baselines are in Appendix E, and the protocol of the reference-state critic is in Appendix I. We have made every effort to make the results reproducible. Every trial was run on a fresh AWS EC2 instance, and every number in the paper is generated by scripts from the raw trial logs. An anonymous repository (`https://anonymous.4open.science/r/harbor-E9D1/`) provides the code, the prompts, the full configuration of every arm, and a CLI that launches each experiment and regenerates every table and figure from the logs.

REFERENCES

Antonis Antoniades, Albert Örwall, Kexun Zhang, Yuxi Xie, Anirudh Goyal, and William Wang. SWE-Search: Enhancing software agents with Monte Carlo tree search and iterative refinement. *arXiv preprint arXiv:2410.20285*, 2024. URL https://arxiv.org/abs/2410.20285.

Bradley Brown, Jordan Juravsky, Ryan Ehrlich, Ronald Clark, Quoc V. Le, Christopher Ré, and Azalia Mirhoseini. Large language monkeys: Scaling inference compute with repeated sampling, 2024. URL https://arxiv.org/abs/2407.21787.

Cameron B. Browne, Edward Powley, Daniel Whitehouse, Simon M. Lucas, Peter I. Cowling, Philipp Rohlfshagen, Stephen Tavener, Diego Perez, Spyridon Samothrakis, and Simon Colton. A survey of Monte Carlo tree search methods. *IEEE Transactions on Computational Intelligence and AI in Games*, 4(1):1–43, 2012. doi: 10.1109/TCIAIG.2012.2186810.

Yinfang Chen, Jiaqi Pan, Jackson Clark, Yiming Su, Noah Zheutlin, Bhavya Bhavya, Rohan R. Arora, Yu Deng, Saurabh Jha, and Tianyin Xu. STRATUS: A multi-agent system for autonomous reliability engineering of modern clouds. In *Advances in Neural Information Processing Systems*, volume 38, 2025. doi: 10.52202/085713-1671. URL https://papers.nips.cc/paper_files/paper/2025/hash/47a6e9e2c3019f13ad94a0f259fe4970-Abstract-Conference.html.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John

Schulman. Training verifiers to solve math word problems, 2021. URL https://arxiv.org/abs/2110.14168.

Rémi Coulom. Efficient selectivity and backup operators in Monte-Carlo tree search. In H. Jaap van den Herik, Paolo Ciancarini, and H. H. L. M. (Jeroen) Donkers (eds.), *Computers and Games*, pp. 72–83, Berlin, Heidelberg, 2007. Springer Berlin Heidelberg. ISBN 978-3-540-75538-8.

Xiang Deng, Jeff Da, Edwin Pan, Yannis Yiming Sun, Chen Bo Calvin Wu, Sanyam Bhutani, et al. SWE-Bench Pro: Can AI agents solve long-horizon software engineering tasks?, 2025. URL https://arxiv.org/abs/2509.16941.

Yunpeng Dong, Jingkai He, Shiqi Liu, Yuze Hou, Dong Du, Zhonghu Xu, Si Yu, Baochuan Yang, Yubin Xia, and Haibo Chen. DeltaBox: Scaling stateful AI agents with millisecond-level sandbox checkpoint/rollback. *arXiv preprint arXiv:2605.22781*, 2026. URL https://arxiv.org/abs/2605.22781.

Ryan Ehrlich, Bradley Brown, Jordan Juravsky, Ronald Clark, Christopher Ré, and Azalia Mirhoseini. CodeMonkeys: Scaling test-time compute for software engineering. *arXiv preprint arXiv:2501.14723*, 2025. URL https://arxiv.org/abs/2501.14723.

Zhibin Gou, Zhihong Shao, Yeyun Gong, Yelong Shen, Yujiu Yang, Nan Duan, and Weizhu Chen. CRITIC: Large language models can self-correct with tool-interactive critiquing, 2023. URL https://arxiv.org/abs/2305.11738.

Xinyu Guan, Li Lyna Zhang, Yifei Liu, Ning Shang, Youran Sun, Yi Zhu, Fan Yang, and Mao Yang. rStar-math: Small LLMs can master math reasoning with self-evolved deep thinking. In Aarti Singh, Maryam Fazel, Daniel Hsu, Simon Lacoste-Julien, Felix Berkenkamp, Tegan Maharaj, Kiri Wagstaff, and Jerry Zhu (eds.), *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 20640–20661. PMLR, 13–19 Jul 2025. URL https://proceedings.mlr.press/v267/guan25f.html.

Shibo Hao, Yi Gu, Haodi Ma, Joshua Jiahua Hong, Zhen Wang, Daisy Zhe Wang, and Zhiting Hu. Reasoning with language model is planning with world model, 2023. URL https://arxiv.org/abs/2305.14992.

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

Jie Huang, Xinyun Chen, Swaroop Mishra, Huaixiu Steven Zheng, Adams Wei Yu, Xinying Song, and Denny Zhou. Large language models cannot self-correct reasoning yet, 2023. URL https://arxiv.org/abs/2310.01798.

Yutong Huang, Vikranth Srivatsa, Alex Asch, Hansin Tushar Patwa, and Yiying Zhang. TClone: Low-latency forking of live GUI environments for computer-use agents. *arXiv preprint arXiv:2605.17320*, 2026. URL https://arxiv.org/abs/2605.17320.

Naman Jain, Jaskirat Singh, Manish Shetty, Liang Zheng, Koushik Sen, and Ion Stoica. R2E-Gym: Procedural environments and hybrid verifiers for scaling open-weights SWE agents. In *Conference on Language Modeling (COLM)*, 2025. URL https://arxiv.org/abs/2504.07164.

Levente Kocsis and Csaba Szepesvári. Bandit based Monte-Carlo planning. In *Machine Learning: ECML 2006*, volume 4212 of *Lecture Notes in Computer Science*, pp. 282–293. Springer, 2006.

Jing Yu Koh, Stephen McAleer, Daniel Fried, and Ruslan Salakhutdinov. Tree search for language model agents. *arXiv preprint arXiv:2407.01476*, 2024. URL https://arxiv.org/abs/2407.01476.

Jacky Kwok, Shulu Li, Pranav Atreya, Yuejiang Liu, Yixing Jiang, Chelsea Finn, Marco Pavone, Ion Stoica, and Azalia Mirhoseini. LLM-as-a-Verifier: A general-purpose verification framework, 2026. URL https://arxiv.org/abs/2607.05391.

Dacheng Li, Shiyi Cao, Chengkun Cao, Xiuyu Li, Shangyin Tan, Kurt Keutzer, Jiarong Xing, Joseph E. Gonzalez, and Ion Stoica. S*: Test time scaling for code generation, 2025. URL https://arxiv.org/abs/2502.14382.

Yujia Li, David Choi, Junyoung Chung, Nate Kushman, Julian Schrittwieser, Rémi Leblond, Tom Eccles, James Keeling, Felix Gimeno, Agustin Dal Lago, Thomas Hubert, Peter Choy, Cyprien de Masson d'Autume, Igor Babuschkin, Xinyun Chen, Po-Sen Huang, Johannes Welbl, Sven Gowal, Alexey Cherepanov, James Molloy, Daniel J. Mankowitz, Esme Sutherland Robson, Pushmeet Kohli, Nando de Freitas, Koray Kavukcuoglu, and Oriol Vinyals. Competition-level code generation with AlphaCode. *Science*, 378(6624):1092–1097, 2022. doi: 10.1126/science.abq1158. URL https://doi.org/10.1126/science.abq1158.

Hunter Lightman, Vineet Kosaraju, Yuri Burda, Harrison Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. Let's verify step by step. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://proceedings.iclr.cc/paper_files/paper/2024/file/aca97732e30bcf1303bc22ac3924fd16-Paper-Conference.pdf.

Jiarui Lu, Thomas Holleis, Yizhe Zhang, Bernhard Aumayer, Feng Nan, Haoping Bai, Shuang Ma, Shen Ma, Mengyu Li, Guoli Yin, Zirui Wang, and Ruoming Pang. ToolSandbox: A stateful, conversational, interactive evaluation benchmark for LLM tool use capabilities. In Luis Chiruzzo, Alan Ritter, and Lu Wang (eds.), *Findings of the Association for Computational Linguistics: NAACL 2025*, pp. 1160–1183, Albuquerque, New Mexico, April 2025. Association for Computational Linguistics. ISBN 979-8-89176-195-7. doi: 10.18653/v1/2025.findings-naacl.65. URL https://aclanthology.org/2025.findings-naacl.65/.

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. URL https://openreview.net/forum?id=fQcUZMPIvu.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, Shashank Gupta, Bodhisattwa Prasad Majumder, Katherine Hermann, Sean Welleck, Amir Yazdanbakhsh, and Peter Clark. Self-Refine: Iterative refinement with self-feedback, 2023. URL https://arxiv.org/abs/2303.17651.

Ryan Marten, Alex Shaw, Ivan Bercovich, Benedikt Droste, Tommaso Cerruti, Steven Dillmann, Ruiyang Wang, Dariush Wahdany, Allen Hart, Karl Krauth, ScaleAI, Snorkel AI, Turing, gNucleus AI, Boolean AI, Nicholas Carlini, Shengrui Lyu, Anjiang Wei, Arpandeep Khatua, Björn Plüster, Chaitanya Dwivedi, Christine Sutcliffe, Yuming, David Tivris, Di Wang, Hui Wen Goh, Hanwen Xing, Haowei Lin, Irakli Salia, Jaejung Seol, Jiajun Bao, Jialin Ouyang, Junha Park, Liam Walsh, Luyang Kong, Maksim Ivanov, Malte Ubl, Mikhail Liamets, Orfeas Menis, Piotr Migdal, Qingquan Bao, Raj Movva, Roey Ben Chaim, Namburi Srinath, Sergey Bogdanik, Shubham Yadav, Stephen Benjamin, Tony Kung, Walker Hughes, Xin Lan, Himanshu Gupta, Swaroop Mishra, Chenguang Wang, Hao He, Jianhong Tu, Kyle Montgomery, Zengji Tu, Atharva Naik, David Mortensen, Ivan Zhang, Yash Mathur, Emmy Liu, Karanpartap Singh, Michael Yu, Steven Feng, Varun Gangal, Zhuofu Tao, Sherry Ruan, Jonas Mueller, Joan Cabezas, Justin Bauer, Kevin Xiang Li, Robert Zhang, Aaron Feller, Alec Madayan, Leon Chen, Ben Feuer, Xiangyi Li, Boxuan Li, Harsh Raj, Samuel Galler, Lin Shi, Ivgeni Segal, Kelly Buchanan, Shreyas P., Rishi Desai, Aaron Schneider, Chris Settles, Xiangning Lin, Marianna Nezhurina, Andrew Wang, Michał Kowalczyk, Jin-Xiang Zhao, Sanyam Satia, Jessie Hu, Sherif Atef, Kobe Chen, Sam Vance, Gian Segato, Jenia Jitsev, Alex Dimakis, Mike Merrill, Andy Konwinski, and Ludwig Schmidt. Terminal-Bench, August 2026. URL https://github.com/harbor-framework/terminal-bench.

Wentao Shi, Yu Wang, Yuyang Zhao, Yuxin Chen, Fuli Feng, Xueyuan Hao, Xi Su, Qi GU, Hui Su, Xunliang Cai, and Xiangnan He. AJ-bench: Benchmarking agent-as-a-judge for environment-aware evaluation. In Maria Liakata, Viviane P. Moreira, Jiajun Zhang, and David Jurgens (eds.), *Findings of the Association for Computational Linguistics: ACL 2026*, pp. 25371–25413, San Diego, California, United States, July 2026. Association for Computational Linguistics. ISBN 979-8-89176-395-1. doi: 10.18653/v1/2026.findings-acl.1269. URL https://aclanthology.org/2026.findings-acl.1269/.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. In *Advances in Neural Information Processing Systems*, volume 36. Curran Associates, Inc., 2023.

KaShun Shum, Binyuan Hui, Jiawei Chen, Lei Zhang, X. W., Jiaxi Yang, Yuzhen Huang, Junyang Lin, and Junxian He. SWE-RM: Execution-free feedback for software engineering agents. *arXiv preprint arXiv:2512.21919*, 2025. URL https://arxiv.org/abs/2512.21919.

David Silver, Aja Huang, Chris J. Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis. Mastering the game of Go with deep neural networks and tree search. *Nature*, 529(7587):484–489, 2016. doi: 10.1038/nature16961.

David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, Timothy Lillicrap, Karen Simonyan, and Demis Hassabis. A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play. *Science*, 362(6419):1140–1144, 2018. doi: 10.1126/science.aar6404.

Benedikt Stroebl, Sayash Kapoor, and Arvind Narayanan. The limits of inference scaling through resampling. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=j8H84v6AZ1.

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 *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. ACL, 2024. URL https://aclanthology.org/2024.acl-long.850/.

Peiyi Wang, Lei Li, Zhihong Shao, Runxin Xu, Damai Dai, Yifei Li, Deli Chen, Yu Wu, and Zhifang Sui. Math-Shepherd: Verify and reinforce LLMs step-by-step without human annotations. In Lun-Wei Ku, Andre Martins, and Vivek Srikumar (eds.), *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 9426–9439, Bangkok, Thailand, August 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.acl-long.510. URL https://aclanthology.org/2024.acl-long.510/.

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc V Le, Ed H. Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. Self-consistency improves chain of thought reasoning in language models. In *The Eleventh International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=1PL1NIMMrw.

Tianyuan Wu, Chaokun Chang, Lunxi Cao, Wei Gao, and Wei Wang. Crab: A semantics-aware checkpoint/restore runtime for agent sandboxes. *arXiv preprint arXiv:2604.28138*, 2026. URL https://arxiv.org/abs/2604.28138.

Yuxi Xie, Kenji Kawaguchi, Yiran Zhao, James Xu Zhao, Min-Yen Kan, Junxian He, and Michael Xie. Self-evaluation guided beam search for reasoning. In A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine (eds.), *Advances in Neural Information Processing Systems*, volume 36, pp. 41618–41650. Curran Associates, Inc., 2023. doi: 10.52202/075280-1802. URL https://proceedings.neurips.cc/paper_files/paper/2023/file/81fde95c4dc79188a69ce5b24d63010b-Paper-Conference.pdf.

Jiakai Xu, Tianle Zhou, Eugene Wu, and Kostis Kaffes. Toward systems foundations for agentic exploration. *arXiv preprint arXiv:2510.05556*, 2025. URL https://arxiv.org/abs/2510.05556.

Jiakai Xu, Tiancheng Ge, Rohan Timmaraju, Georgios Liargkovas, Eugene Wu, and Kostis Kaffes. Waypoint, September 2026. URL https://github.com/Alex-XJK/waypoint.

Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Tom Griffiths, Yuan Cao, and Karthik Narasimhan. Tree of thoughts: Deliberate problem solving with large language models. In A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine (eds.), *Advances in Neural Information Processing Systems*, volume 36, pp. 11809–11822. Curran Associates, Inc., 2023. doi: 10.52202/075280-0517. URL https://proceedings.neurips.cc/paper_files/paper/2023/file/271db9922b8d1f4dd7aaef84ed5ac703-Paper-Conference.pdf.

Fei Yu, Yingru Li, and Benyou Wang. Scaling flaws of verifier-guided search in mathematical reasoning, 2025. URL https://arxiv.org/abs/2502.00271.

Andy Zhou, Kai Yan, Michal Shlapentokh-Rothman, Haohan Wang, and Yu-Xiong Wang. Language agent tree search unifies reasoning, acting, and planning in language models. In *Proceedings of the 41st International Conference on Machine Learning (ICML)*, volume 235 of *PMLR*, 2024. URL https://proceedings.mlr.press/v235/zhou24r.html.

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

A PROMPTS

All prompts used in this paper are in the anonymous repository at `https://anonymous.4open.science/r/harbor-E9D1/`. The actor's prompts are under `src/harbor/search/templates/`, whereas the critics' prompts and the feedback template are under `src/harbor/search/critics/templates/`. The code in the same directories shows how each prompt is built from its template, the task and the trajectory.

B TASK LIST, VERIFIER ANNOTATION AND TIMEOUTS

Table 4 lists the 89 tasks of Terminal-Bench 2.1, the time limit T each task gives the actor, and what each task's verifier does to grade it.

We read every verifier by hand. For each assertion it makes, we asked what a checker must do, at minimum, to settle it: read something that already exists in the environment (R), run something to produce a new observation (X), or compare against a value that only the verifier holds (O). R and X have finer codes. R1: file content, such as a file that exists or whose content or hash matches. R2: filesystem metadata, such as permissions, owners, timestamps or git objects. R3: runtime state, such as a live process, a listening port or an environment variable. X1: run a binary, a test or a computation and check its output. X2: probe a service, such as connecting to a port or sending a request. A task carries every code its assertions need.

§3.2 groups tasks by their strongest code: *holds the answer* if any assertion is O (19 tasks), otherwise *runs* if any is X (48), otherwise *reads* (22). The last column marks the 47 tasks whose instruction states a requirement relative to the initial state S_0, such as a file that must stay unchanged or an output that must match the original program's.

Table 4: The 89 tasks of Terminal-Bench 2.1. T: the task's time limit for the actor. *Verifier*: every code its assertions need (R1–R3 read, X1–X2 run, O compares with a value only the verifier holds); §3.2 groups tasks by the strongest one. S_0: the task states a requirement relative to the initial state (§3.3).

Task	T (min)	Verifier	S_0
adaptive-rejection-sampler	15	R1, X1	
bn-fit-modify	60	R1, O	
break-filter-js-from-html	20	R1, X1	
build-cython-ext	15	R1, X1, O	✓
build-pmars	15	R1, X1	
build-pov-ray	200	R1, X1, O	✓
caffe-cifar-10	60	R1, R2, X1	
cancel-async-tasks	15	R1, X1	
chess-best-move	15	R1	✓
circuit-fibsqrt	60	R1, X1	✓
cobol-modernization	15	R1, X1	✓
code-from-image	20	R1	✓
compile-compcert	40	R1, R2, X1	
configure-git-webserver	15	X1, X2	
constraints-scheduling	20	R1	✓
count-dataset-tokens	15	R1, O	
crack-7z-hash	30	R1	✓
custom-memory-heap-crash	30	R1, X1	✓
db-wal-recovery	15	R1	✓
distribution-search	60	R1	
dna-assembly	30	R1, X1	✓
dna-insert	30	R1, X1	✓
extract-elf	15	R1, X1, O	
extract-moves-from-video	30	R1	✓
feal-differential-cryptanalysis	30	X1, O	
feal-linear-cryptanalysis	30	R1	✓
filter-js-from-html	30	R1, X1, X2, O	
financial-document-processor	20	R1	✓
fix-code-vulnerability	15	R1, X1	✓
fix-git	15	R1	✓
fix-ocaml-gc	60	R1, X1, X2, O	
gcode-to-text	15	R1	✓
git-leak-recovery	15	R1, R2	✓
git-multibranch	15	X2	
gpt2-codegolf	15	R2, X1	✓
headless-terminal	15	R1, X1, X2	
hf-model-inference	15	R1, X1, X2	
install-windows-3.11	60	R1, R3, X2	✓
kv-store-grpc	15	R1, R3, X2	
large-scale-text-editing	20	R1, X1	✓
largest-eigenval	15	X1	
llm-inference-batching-scheduler	30	R1	✓
log-summary-date-ranges	15	R1	✓
mailman	30	R1, X2	
make-doom-for-mips	15	R1, X1	✓
make-mips-interpreter	30	R1, X1	✓
mcmc-sampling-stan	30	R1, X1	✓
merge-diff-arc-agi-task	15	R1, X1, O	✓
model-extraction-relu-logits	15	R1, X1	
modernize-scientific-stack	10	X1	✓
mteb-leaderboard	60	R1, O	
mteb-retrieve	30	R1	✓
multi-source-data-merger	15	R1	✓
nginx-request-logging	15	R1, X1, X2	
openssl-selfsigned-cert	15	R1, R2, X1	
overfull-hbox	12.5	R1, X1	✓
password-recovery	15	R1	✓
path-tracing	30	R1, X1	✓
path-tracing-reverse	30	R1, X1	✓
polyglot-c-py	15	X1	
polyglot-rust-c	15	R1, X1	
portfolio-optimization	60	R1, X1	✓
protein-assembly	30	R1, O	✓
prove-plus-comm	15	R1, X1	
pypi-server	15	R1, X1, X2	
pytorch-model-cli	15	R1, X1, O	
pytorch-model-recovery	15	R1, X1	✓
qemu-alpine-ssh	15	R3, X2	
qemu-startup	15	R3, X2	
query-optimize	15	R1, X1, O	✓
raman-fitting	15	R1	✓
regex-chess	60	R1, X1	
regex-log	15	R1, X1, O	
reshard-c4-data	60	R1, R2, X1, O	
rstan-to-pystan	30	R1	✓
sam-cell-seg	120	R1, X1, O	
sanitize-git-repo	15	R1, R2	✓
schemelike-metacircular-eval	40	R1, X1	
sparql-university	15	R1, X1, O	
sqlite-db-truncate	15	R1	✓
sqlite-with-gcov	15	R1, R2, X1	
torch-pipeline-parallelism	15	R1, X1	
torch-tensor-parallelism	15	R1, X1	
train-fasttext	60	R1, X1, O	
tune-mjcf	15	R1, X1	✓
video-processing	60	R1, X1, O	
vulnerable-secret	15	R1	✓
winning-avg-corewars	60	R1, X1	✓
write-compressor	15	R1, X1	✓

C MODEL PRICES

Every dollar figure in the paper is model cost: the tokens a trial consumed, priced at the rates in Table 5. Machine time is not counted. Prices are in dollars per million tokens as of September 2026.

Our own trials are priced at the AWS Bedrock rate of the model, read from its Bedrock model card (OpenAI models) or the public Bedrock price list (Claude, GLM). For GPT-5.6 Luna, Bedrock bills every uncached input token as a cache write, at 1.25 times the input rate; we do the same. The GPT-5.6 Luna + Codex CLI point of Figure 1 and Table 3 is priced at the AWS rate based on the token counts from Harbor Hub reports. We do not use the hub's own price for GPT-5.6 Luna, which is the rate before July 2026 ($1.00 / $0.10 / $6.00) and five times higher than AWS.

Every other point of Figure 1 uses the cost Harbor Hub reports for that job. The hub's rates match the vendor's list price for GPT-5.5 + Codex, Opus 4.7 and Muse Spark 1.1; for Sonnet 5 + Claude Code the hub bills at Sonnet 4.6's rate, 1.5 times Sonnet 5's list price; for GPT-5.5 + Terminus-2 no fixed rate reproduces the hub's figure, which we use as reported. Claude Code trials that hit the time limit have no hub cost; we price their tokens at the rate the hub used for the rest of that arm.

Table 5: Prices in dollars per million tokens. *Input*: uncached input; *cached*: cache read; a dash means the price list has no cached rate and cached tokens are billed as input.

Model	Used in	Input	Cached	Output	Source
GPT-5.6 Luna	all Luna arms; Luna + Codex	0.20 (0.25 as a cache write)	0.02	1.20	Bedrock model card, global
Claude Sonnet 5	critic, Appendix J	2.00	0.20	10.00	Anthropic list price
GLM-5	critic, Appendix J	1.00	–	3.20	Bedrock price list, us-east-1
GPT-6 Astra	Appendix L	13.75 (as cache write)	1.10	55.00	Bedrock model card, in-region
Claude Fable 5.1	Appendix L	13.75 (as cache write)	0.275	55.00	Bedrock price list, in-region
Claude Opus 5.5	Appendix L	5.50 (as cache write)	0.22	22.00	Bedrock price list, in-region
GPT-5.5 + Codex	Figure 1	5.00	0.50	30.00	Harbor Hub
Claude Opus 4.7	Figure 1	6.25 (as cache write)	0.50	25.00	Harbor Hub
Muse Spark 1.1	Figure 1	1.25	0.15	4.25	Harbor Hub
Claude Sonnet 5 + Claude Code	Figure 1	3.75	0.30	15.00	Harbor Hub (Sonnet 4.6 rate)
GPT-5.5 + Terminus-2	Figure 1	hub total, no fixed rate			Harbor Hub

D Statistical Methodology

Every interval in the paper is a 95% percentile bootstrap over tasks. Tasks are resampled with replacement 5000 times with a fixed seed; all trials, states and probes of a drawn task move together, and the statistic is recomputed on each resample. Differences between two arms are paired before resampling: by trial when both arms are read off the same trials (the baseline and the submitted state of Table 3), and by state when two critics judge the same state (§3.2, §3.3). Sample sizes are given with each table and figure.

E Search Baselines in Detail

This appendix describes the two search baselines of §3.1. Both use the actor of Table 1: GPT-5.6 Luna in the same harness with the same prompt, on all 89 tasks, five trials each, under the task's clock T. The baseline is that actor with no search: it runs until it first claims the task is done, and the verifier grades that state. Intervals are 95% bootstrap intervals over tasks (Appendix D).

E.1 Branching Tools Without a Search Policy (seq_fork)

This arm gives the actor the tool to branch. Its reply has two optional fields: `snapshot` saves the current environment under a label, and `restore` returns to a saved one. A restore replaces files, processes and shell state and keeps the conversation. The prompt says the tools may be used whenever the actor judges they help; it does not say when. There is no critic and no score. The trial ends when the actor claims completion or the clock runs out, and the verifier grades the live environment.

Pass@1 is 61.0% against 63.4% for the baseline ($\Delta = -2.4\,[-6.7, +2.0]$; Table 6). The actor touched the tools in only 3 of 445 trials: it saved 1 snapshot and restored 3 times. Two restores, in DB-WAL-RECOVERY, went back to S_0 to recover a write-ahead log the actor had destroyed by opening the database. The third, in PATH-TRACING, returned to a saved renderer after a change made it worse. Given the tools without a policy, the actor does not explore alternatives or protect its progress before a risky step; it restores only to undo damage already done. (4 trials in which the actor's commands destroyed the sandbox before grading are dropped.)

Table 6: The actor with and without snapshot and restore tools, 89 tasks of Terminal-Bench 2.1, five trials per task.

	Baseline	+ snapshot and restore tools
Pass@1 (passed / graded trials)	63.4% (282/445)	61.0% (269/441)
Trials using a tool	–	3
Snapshots saved by the actor	–	1
Restores (to S_0)	–	3 (2)
Model cost / sweep	$10.70	$11.40

E.2 Tree Search With a Trajectory-Only Critic (MCTS)

This arm adds a search policy and a critic to the actor. It builds one search tree per trial, and each node of the tree is a snapshot of the environment.

To expand a node, the search restores the node's snapshot, lets the actor run five turns, and saves the result as a new child node. A node has at most two children.

The critic is the trajectory-only critic C^{traj} (Appendix A). It sees the task and the actor's turns on the path to a node. It scores each new node once, from 0 to 100. A node's value is the mean score of all nodes in its subtree.

To choose the next node to expand, the search starts at the root. If the current node has fewer than two children, the search expands it. Otherwise it moves to the child with the highest UCT score (Kocsis & Szepesvári, 2006), with $c_p = 0.12$, and repeats. A node at which the actor declared the task finished is a candidate for submission, and the search never expands it.

The search stops when no node can be expanded or when the clock T runs out. Snapshots, restores and critic calls all count against T. The search then submits the candidate with the highest value, or the node with the highest value if there is no candidate. The verifier grades the submitted state.

Pass@1 is 64.0% for MCTS and 63.4% for the baseline ($\Delta = +0.6\,[-4.5, +5.5]$; Table 7). The interval contains zero. MCTS costs 3.81× as much as the baseline, and the critic accounts for 31.3% of that cost.

Table 7: Pass@1 of the baseline and MCTS, five trials per task. MCTS has no graded trial on KV-STORE-GRPC, so its pass@1 is over 88 tasks and the baseline's over 89. Δ is MCTS minus the baseline; its interval is a bootstrap over tasks that draws the same tasks for both arms (Appendix D).

	Pass@1	Model cost / sweep
Baseline	63.4%	$10.70
MCTS	64.0%	$40.79
Δ [95% CI]	$+0.6\,[-4.5, +5.5]$	3.81×

The search did choose between paths. It opened a second branch in 99.3% of trials. In 70.9% of trials, the submitted state came from a branch opened after the first path. In 68.3% of trials, the search ended before the clock ran out, because every leaf was a state the actor had declared finished.

We drop 9 trials lost to a failed snapshot, and the 5 trials of KV-STORE-GRPC, whose verifier could not run.

F Critic Metrics by Position

Table 8 gives the numbers behind Figure 2. It also gives the false-accept and false-reject rates at the intermediate depths, two checks of the ranking, and the mean absolute error (MAE).

All critics see the same task, the same trajectory and the same remaining budget. READ and ACT can also run commands in a fork of the state: READ only read-only commands, ACT any command (§3.2).

At an intermediate state, the outcome is the pass rate of the 5 continuations from that state. A continuation that the verifier could not grade is left out of this rate. We exclude 7 intermediate and 4 final states that have no graded outcome. As in Table 2, we also exclude the 1 final state on which the verdict of C^{read} could not be parsed.

Label noise. A pass rate from 5 continuations is a noisy label. Its standard error is at most 22 points, and 10 points in root mean square over the intermediate states. The critics never see the continuations, so this noise can flip the order of a pair but favors no critic: it pulls pairwise accuracy towards 50%. For 79% of the intermediate states, the continuations all pass or all fail. On the 117 decided pairs, where one state's continuations all pass and the other's all fail, the critics order 68%–74% of the pairs correctly. The gain of C^{act} over C^{traj} on these pairs is +3.4 points $[-5.4, +12.2]$.

Choosing a state. Search uses the critic to choose among states. Among the states of a task at one depth, the state the critic rates highest has a continuation pass rate 4–6 points above the mean of those states, which is what a random choice gets. This gain appears only late in the trajectory: at 25% depth it is -1.6 points $[-6.7, +3.7]$ for TRAJ, $+2.8$ $[-1.4, +6.9]$ for READ and $+0.2$ $[-5.3, +5.4]$ for ACT, every interval including zero; at 75% depth it is $+11.0$ to $+13.6$ points, every interval above zero (Table 8). The critics therefore carry some signal at intermediate states, but access adds little to it: the gain of C^{act} over C^{traj} is $+1.8$ points $[-2.5, +6.3]$.

Access at the two positions. Each gain from access compares two critics on the same states and outcomes. In pairwise accuracy, the gain of C^{act} over C^{traj} is $+11.5$ points $[+2.7, +20.2]$ at the final state and $+4.4$ points $[-0.9, +9.9]$ at the intermediate depths pooled. The difference between the two gains is not significant ($+7.1$ points $[-1.7, +15.5]$).

The intervals in Figure 2 are 95% bootstrap intervals over tasks (Appendix D).

Table 8: The numbers behind Figure 2. At the intermediate depths, the critic's output is its estimate $\hat{q}$ and the outcome is the pass rate of 5 continuations. At the final state, the output is the verdict $\hat{v} \in \{0, 1\}$ and the outcome is the verifier's judgment. A pair is two states of the same task at the same depth (at the final state: two final states of the same task) whose outcomes differ; pairwise accuracy is the share of pairs that the output orders like the outcome, with tied outputs counting one half. A decided pair has outcomes 0 and 1. The selection gain is the outcome of the state with the highest output among a task's states at one depth, minus the mean outcome of those states. A false accept is a failing state with output $\geq 50\%$; a false reject is a passing state with output $< 50\%$. At intermediate depths, these two rates use only the states from which all continuations pass or all fail. MAE is the mean of $|\text{output} - \text{outcome}|$ (at the final state, the misclassification rate); *best constant* is the MAE of the best single answer. Rows $X - Y$ are differences between two critics. The column *all* pools the three depths.

		intermediate state, depth				
		25%	50%	75%	all	final state
Pairwise accuracy (%) ↑	TRAJ	49	56	64	54	65
	READ	56	55	69	59	61
	ACT	52	62	68	59	76
Pairwise, decided pairs (%) ↑	TRAJ	56	73	68	68	65
	READ	67	70	78	74	61
	ACT	53	62	80	71	76
Selection gain (points) ↑	TRAJ	−2	+7	+11	+4	+14
	READ	+3	+3	+14	+6	+8
	ACT	0	+8	+13	+6	+22
False reject (%) ↓	TRAJ	2	1	1	1	60
	READ	28	11	4	14	52
	ACT	8	6	2	5	22
False accept (%) ↓	TRAJ	91	81	80	84	5
	READ	54	54	61	57	10
	ACT	75	66	59	66	22
MAE (%) ↓	TRAJ	34	32	30	32	39
	READ	34	32	28	32	36
	ACT	32	31	26	30	22
	READ−TRAJ	0	0	−1	0	−3
	ACT−TRAJ	−2	−1	−4	−2	−17
	ACT−READ	−2	−1	−2	−2	−14
	best constant	36	38	37	38	38
	TRAJ, summary + ledger	34	32	30	32	37
	TRAJ, checklist	33	33	30	32	41
States		404	333	374	1111	406
Tasks		88	83	85	88	89
Pairs		317	184	171	672	144
Decided pairs		18	30	69	117	144

G CRITIC COST

Figure 8 gives the model cost of each critic per state, at the prices of Appendix C.

At intermediate states, C^{traj} costs 0.32 cents per state, C^{read} 0.51 cents and C^{act} 0.59 cents.

At candidate final states, C^{act} costs 1.11 cents per state. This is 3.1× the cost of C^{traj} and 1.3× the cost of C^{read}.

C^{act} costs 1.9× as much at a final state as at an intermediate state, because it takes more turns there: 4.6 on average, against 3.5.

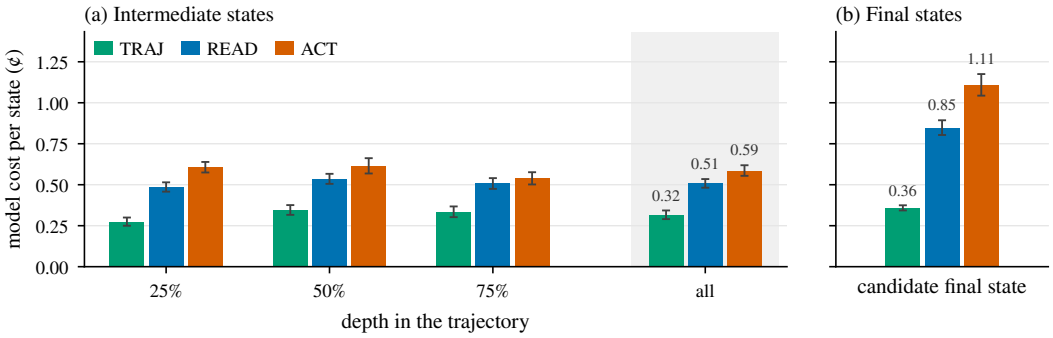


Figure 8: Model cost per state of the critics of Figure 2, on the same states. **(a)** Intermediate states at 25/50/75% depth; *all* pools the three depths. **(b)** Candidate final states. Intervals are 95% bootstrap intervals over tasks.

H TRAJECTORY-ONLY CRITIC PROMPTS

Figure 9 compares three prompts for the trajectory-only critic (Appendix A) on the states of Figure 2. None of these critics runs commands.

The *plain* prompt is the one that TRAJ uses in the main text. *Summary + ledger* adds a summary of the trajectory and a per-step ledger of facts. *Checklist* adds a list of failure modes to look for.

No variant differs significantly from TRAJ at any position. At every intermediate depth, the three are within 1 point of one another in MAE.

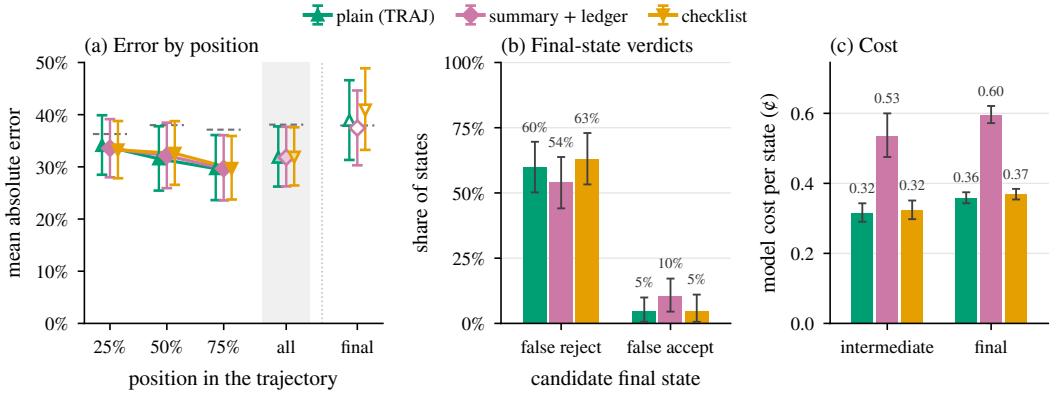


Figure 9: The three trajectory-only prompts on the states of Figure 2; *plain* is TRAJ in Figure 2. **(a)** Mean absolute error by position (Table 8); dashes mark the best constant answer. **(b)** At the candidate final state: the share of passing states rejected and the share of failing states accepted. **(c)** Model cost per state (Appendix C); for *summary + ledger* it includes the calls that write the summary and the ledger. Intervals are 95% bootstrap intervals over tasks.

I THE REFERENCE-STATE CRITIC: PROTOCOL

This appendix describes how the comparison of §3.3 was run. The snapshots of the final states of §3.2 were not kept. We rebuilt each state in a fresh environment by replaying, in order, every command the actor had issued. Of the 411 final states, we use 404. We drop the other 7 because the replay, the verifier or the fork of S_0 failed on them. To check the replay, we ran the verifier on each rebuilt state and compared its result with the one recorded in the original trial. The two results are the same on 399 states and differ on 2; the other 3 states have no recorded result to compare with. In the comparison, whether a state passes is decided by the verifier's result on the rebuilt state. Each critic judged every state twice. C^{act} uses the same prompt as the critic of §3.2. The prompt of $C^{\text{act}}{+}S_0$ adds one paragraph. It tells the critic that there is a second environment, how to run a command in it, and how to copy a file from it.

J DIFFERENT CRITIC MODELS

The critic does not have to use the actor's model. We test two other models as the critic. The actor stays GPT-5.6 Luna, and nothing else changes: the same prompts, the same clock T and the same revision budget, on the same 89 tasks.

Claude Sonnet 5. Sonnet 5 is about as strong as Luna. As an agent in Claude Code, it passes 75% [69%, 82%] of the trials on these tasks, against 77% [69%, 84%] for Luna in the Codex CLI.

The critic acts only after the first claim, so the first claims pass at about the same rate as with the Luna critic: 61% against 63%. The loop then submits a passing state in 70% of the trials. The gain is +9.9 points [+5.4, +14.6], as large as with the Luna critic (+9.0 points [+5.6, +12.6]; Figure 10a). The per-task pass rate rises on 28 tasks and falls on 2.

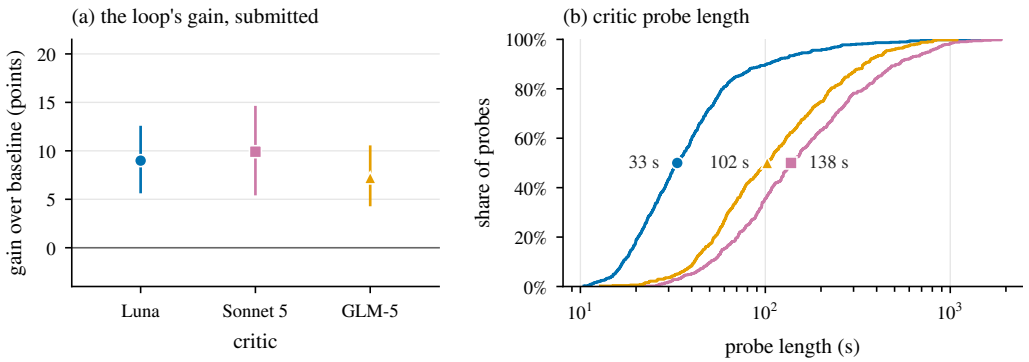


Figure 10: GPT-5.6 Luna as the actor, with each model as the critic. (a) The loop's gain over the baseline in the submitted pass rate (Table 3); bars are 95% bootstrap intervals over tasks. (b) How long the critic's probes take, over the probes that return a verdict; the markers are the medians.

The gain comes about in the same way as with Luna (§4.2). The Sonnet critic rejects 72% of the failing first claims (Luna: 76%), and the actor turns 49 of them into passing states. False rejections cost 5 passing first claims.

Sonnet's probes take longer: a median of 138 s and 12 commands, against 33 s and 4 for Luna (Figure 10b). Because the clock is shared, 59 loops run out of time during a probe, against 13 with Luna. Each of them submits its last proposed state.

GLM-5. We also use Z.AI's GLM-5 as the critic, with the same setup. The first claims pass at about the same rate, 63% against 63%. The loop submits a passing state in 70% of the trials, a gain of +7.2 points [+4.3, +10.6] (Figure 10a). The per-task pass rate rises on 22 tasks and falls on 0. The gain is smaller than with the Luna critic, but the difference, −1.8 points [−4.9, +1.6], is not significant.

GLM-5 is a more lenient critic than Luna. It rejects only 52% of the failing first claims (Luna: 76%), so fewer of them go back to the actor, which turns 33 into passing states. It also rejects fewer passing

first claims, 6% against 21%, so false rejections cost only 1 passing first claim. Its probes take a median of 102 s, between Luna's and Sonnet's (Figure 10b), and 13 commands.

K Resuming or Restarting

After a rejection, Probe-and-Resume resumes the actor at the state it claimed (§4.1). The alternative is to restart. A restart resets the environment and the actor's conversation to S_0, and the actor starts the task again, with the critic's findings added after the task instruction.

We run this restart variant on all 89 tasks of Terminal-Bench 2.1, five trials each. Nothing else changes: the same model is the actor and the critic, the critic and its feedback are the same, a trial has at most six rounds, and the actor and the critic share the time limit T. Environment failures lost 6 trials, which leaves 439.

Table 9 compares the two variants; the Probe-and-Resume row repeats Table 3. Restarting submits a passing state in 66% of trials, against 72% for resuming. The difference is -6.1 points [-10.1, -2.2].

Table 9: Resuming vs. restarting after a rejection, on the 89 tasks of Terminal-Bench 2.1, five trials per task. *Pass rate* and cost as in Table 3; time is the actor's and the critic's per trial, averaged.

	Pass rate	**Total model cost**	**Time / trial**
Probe-and-Resume	**72.4%**	$23.14	7.8 min
Restart from S_0	66.3%	$38.12	9.3 min

Restarting also costs 1.6 times as much in model calls, and a trial takes 9.3 minutes on average instead of 7.8. The critic's time is the same in both variants, so the extra time is the actor's: after a restart, the actor has to redo the work of the rejected attempt before it can act on the critic's findings. Of the 161 trials that restarted at least once, 61 ran out of time before the loop ended.

L Stronger Models as Actor and Critic

Table 3 uses GPT-5.6 Luna, which passes 63% of its first claims on Terminal-Bench 2.1. Here we repeat the loop with stronger models: GPT-6 Astra and Claude Opus 5.5 as the actor, and Claude Fable 5.1 or Opus 5.5 as the critic. On Harbor Hub, Astra in the Codex CLI passes 87% of 25 trials per task. We use three combinations of actor and critic (Table 10).

We run them on 15 tasks, chosen from the 31 tasks that Astra fails at least once in its 25 hub trials:

- ten on which its first claim passes in some of our trials and fails in others, so that the loop has something to recover;
- four that it always passes, as controls for false rejections;
- one that it never passes, as a control for false accepts.

The other settings are the same: the actor, the critic and the snapshots share the clock T, and a trial has at most six claims and 50 actor turns after the first rejection.

None of the three combinations gains significantly:

- *Astra actor, Fable 5.1 critic.* The submitted pass rate stays at the baseline: 63% against 61%, $\Delta = 1.3$ points [-6.7, 9.3].
- *Astra actor, Opus 5.5 critic.* The pass rate rises by 4.0 points [-2.7, 10.7], an interval that includes zero. 4 trials gain a passing state and 1 loses one.
- *Opus 5.5 on both sides.* The loop changes nothing. The critic accepts almost every first claim, no trial is rescued or lost, and the submitted rate equals the baseline (69%).

The mechanism of §4.2 explains this. With Luna, only 33% of first claims on these tasks pass, and the loop works on the rest. With these actors, most first claims already pass, so there is little to recover. The remaining failures are ones that a critic of the same strength does not catch either.

Table 10: Probe-and-Resume with stronger models as the actor and the critic, on the same 15 tasks of Terminal-Bench 2.1, five trials per task. Luna: GPT-5.6 Luna (the trials of Table 3 on these tasks); Astra: GPT-6 Astra; Fable, Opus: Claude Fable 5.1, Claude Opus 5.5. *Baseline* and Probe-and-Resume as in Table 3. Intervals are 95% bootstrap intervals over tasks. *Refused*: trials the actor's model refused; they count 0 in both columns. *Cost*: the total over the trials that were not refused (Appendix C).

Actor / Critic	Tasks	Trials	Baseline	Probe-and-Resume	Δ [95% CI]	Refused	Cost
Luna / Luna	15	75	33.3%	**40.0%**	+6.7 [−2.7, +16.0]	0	$4.04
Astra / Fable 5.1	15	75	61.3%	**62.7%**	+1.3 [−6.7, +9.3]	0	$126.94
Astra / Opus 5.5	15	75	69.3%	**73.3%**	+4.0 [−2.7, +10.7]	0	$65.83
Opus 5.5 / Opus 5.5	15	75	69.3%	69.3%	0.0 [0.0, 0.0]	8	$52.80

One issue comes from the model, not from the loop. Claude's content classifier answers two life-science tasks with empty replies, so 8 of the Opus 5.5 actor trials are refused; they count as 0 (column *Refused*). These trials ran to the clock on empty replies: the harness now stops an actor after six empty replies, but these trials ran before that change. Astra's Bedrock endpoint rejects some security tasks with a policy error, but no Astra trial on these tasks was refused.

M SWE-bench Pro: Tasks and Grading

This appendix gives the setup of §4.3. We use the public split of SWE-bench Pro V2.0.0, released on 22 September 2026, with the benchmark's own task images. The loop is the one of §4.1 with the same prompts and GPT-5.6 Luna as actor and critic. The actor, the critic and every snapshot, restore and fork share the task's 50-minute limit. A trial has at most six claims and 50 actor turns after the first rejection.

The two task sets. Both sets are fixed by files released with the benchmark. Neither depends on how GPT-5.6 Luna or Probe-and-Resume does on any task.

- *Hard* is the benchmark's HARD-51 list: the tasks that at least two of five frontier model families fail under V2's protocol. We leave out its one NodeBB task, which gives 50 tasks.
- *Medium* comes from the per-task results of 9 runs that the benchmark's repository released: 2025 models (Claude Sonnet 4.5, Claude Sonnet 4 twice, Claude Opus 4.1, Gemini 2.5 Pro Preview, GPT-4o, GPT-5, gpt-oss and Kimi K2) on the first release. Of the 642 V2 tasks, 397 are solved by at least one of these runs. Medium is every task that none of them solves, outside HARD-51 and NodeBB: 197 tasks. The benchmark itself has no medium tier; the name is ours. The 2025 models failed these tasks, while at most one of the five model families behind HARD-51 fails each of them.

The task lists and the released result files come with our code, and our script rebuilds both sets from the released files.

We leave out all 38 NodeBB tasks because their reference patches fail in our environment even with network access.

Tasks we do not score. A task is scored only if its reference patch passes in the same environment as our trials. This rule does not look at our trials. 20 tasks fail it (9 hard, 11 medium). In all of them the tests need the network while grading: 18 Go tasks download modules, 1 task downloads Node.js and 1 Python test opens a connection when it is loaded. V2's protocol gives the verifier network access. Our sandbox allows no traffic except to the model endpoint, for grading as well as for the agent, so these tasks cannot be graded here.

Trials. We report the first trial of each of the 227 scored tasks, so each task counts once in both sets. The hard tasks were run four more times; Table 11 reports all five. No trial, in either set, ended without a claim.

Table 11: SWE-bench Pro by task set. Baseline: the first proposed state; Probe-and-Resume: the submitted state. Gains are in points, with 95% intervals over tasks (Appendix D).

	Tasks	Trials	Baseline	Probe-and-Resume	Gain [95% CI]
Hard, first trial	41	41	70.7%	73.2%	+2.4 [−4.9, +9.8]
Hard, all five trials	41	205	65.4%	66.3%	+1.0 [−3.9, +5.9]
Medium	186	186	82.3%	90.9%	+8.6 [+3.8, +14.0]
Both, first trial (Figure 7)	227	227	80.2%	87.7%	+7.5 [+3.1, +11.9]

Grading. V2's locked protocol keeps the agent offline except for the model endpoint. It then captures the agent's `git diff`, including new files, applies it to a fresh copy of the image and runs the verifier there. Our agent phase is offline in the same way. We change one line of each task file, from no network to a public network, because Waypoint cannot create a sandbox without a network; a network namespace around the whole trial then lets traffic reach only the model endpoint. At the start of every trial, a check from inside the namespace confirms that the endpoint connects and that GitHub, PyPI, the Go module proxy, npm and direct connections by address are refused. Each trial also saved its DNS and firewall log; in all 391 trials we report, the firewall admitted only addresses of the model endpoint.

We grade differently. Each proposed state is graded by the task's unchanged V2 verifier on a fork of that state. The verifier first resets the repository's test code: it restores tracked test files and fixtures, deletes test files the agent added and removes cached bytecode. It then applies the hidden tests and runs them. The fork is discarded after grading, so the hidden tests never reach the actor's or the critic's sandbox. The loop records the grade but never reads it; only the critic's verdict decides whether the actor continues.

The difference from the protocol is what the graded state contains. A patch carries only the repository's tracked and new files. It drops files outside the repository or ignored by git, installed packages and running processes. Our fork keeps them. With no network, no trial could download a package, but a local change outside the repository would count in our grading and not in the protocol's.

The gain comes from the medium tasks. On the hard tasks it cannot be told apart from zero, with the first trial or with all five (Table 11).