SCALING TRACES DIVERSITY WITH RUNTIME-DECOUPLED WORLD MODEL FOR MID-TRAINING

Anonymous authors
Paper under double-blind review

ABSTRACT

Frontier models increasingly incorporate agentic experience earlier—from post-training into mid-training—to build fundamentals in agentic behaviors and environment interaction. This stage teaches broad patterns of exploration, action, and recovery before specialized agentic SFT and RL, creating demand for both high-quality trajectories and diverse, distinct tasks. Existing execution-based pipelines provide high-quality traces but rely on curated, runnable environments and verifiable outcomes. In agentic coding, this concentrates supervision on resolved issues and merged pull requests, leaving open issues, development branches, and tasks with unusable images or non-reproducing gold solutions underrepresented. Our re-execution audit estimates that roughly one-third of SWE-rebench V2 tasks fall outside reliable execution-based supervision, while over 55M unresolved public issues offer an additional source of tasks without known solutions. Motivated by these gaps, we use Qwen-World-Model to simulate execution-dependent feedback while retaining exact repository state for deterministic actions. This enables trajectory collection without requiring every task to have a working image or known solution. We construct a corpus spanning 3M unique tasks and 100 B training tokens, increasing unique-task coverage by $4\times$ at $4\times$ the throughput of execution-based collection. In matched-budget SFT, replacing 25%, 50%, or 100% of executed traces with simulated traces yields comparable performance across four agent benchmarks, supporting their utility as training data. At matched token exposure, adding traces from new queries outperforms repeating the original collection by 4.6 pass@1 points on average, indicating that broader task coverage is more valuable than repeated rollouts at a fixed token budget. Further incorporating these diverse traces into mid-training improves pass@1 after the same SFT by +8.7, +6.2, +2.1, and +1.1 percentage points on SWE-bench Verified, Multilingual, Pro, and Terminal-Bench 2.0, respectively. These results suggest that world-model simulation can expand both the volume and breadth of agentic experience, strengthening the foundations learned during mid-training and improving downstream agent performance.

1 INTRODUCTION

Agentic capability, the ability to pursue goals through tool use and sustained environment interaction, has become a central objective of frontier model development. Frontier models such as GLM and Qwen are increasingly introducing agentic experience earlier in training, incorporating large-scale agent traces into mid-training before subsequent post-training (supervised fine-tuning and reinforcement learning) (GLM-5 Team, 2026; Cao et al., 2026; Zeng et al., 2026). This earlier exposure aims to establish transferable foundations in tool calling, observation-driven action, and recovery from failures, supporting further development of agentic capabilities through post-training.

Existing agentic mid-training recipes primarily draw on query sources used for post-training, collecting traces generated by older policies, traces on tasks that have become too easy to provide sufficient post-training learning signal as model capabilities improve, and traces excluded by post-training quality filters (Cao et al., 2026; Chen et al., 2026). However, mid-training aims to build fundamental agentic behaviors, motivating exposure to varied queries, codebases, and development states—including tasks without readily executable environments or known solution patches. Repeated

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.

Table 1: Task coverage opportunities and traces collection requirements. ✓ means the named prerequisite can be omitted; ★ denotes bypassing it through synthesis. †Detailed breakdown in Table 5 (Appendix A). ‡Source pools use > 100 stars and other source gates; refer to Table 7 for details.

Dataset	Task Source	Instance Scale	No pre-existing unit tests	No pre-existing paired fix	No task sandbox runtime
SWE-Gym	Merged issue–PR pairs	2,438	✗	✗	✗
SWE-rebench V2	Merged issue–PR pairs	32,079	✗	✗	✗
Scale-SWE	Merged PR with synthetic issue	17,202	★	✗	✗
SWE-Universe	Merged issue–PR + PR-only	807,693	✗	✗	✗
SWE-World	Merged issue–PR pairs	16,550	✗	✗	✓
World Rollout (ours)	Merged issue–PR; unverifiable sandbox (complementary)	≈10k†	✓	✓	✓
	Issue–PR development states	≈217k‡			
	Open issues; no closing-linked PR	≈2.18M‡			

rollouts diversify behavior on existing tasks but do not expand the query pool; we therefore argue for scaling task coverage alongside trajectory volume.

To identify opportunities for broader task coverage, we begin by examining how public SWE post-training tasks are collected. These resources generally pair tasks with an existing fix (pull request, PR) and unit-test evidence, then construct a runnable environment, as summarized in Table 1. SWE-Gym and SWE-rebench V2 use merged issue–PR pairs (Pan et al., 2024; Badertdinov et al., 2026); R2E-Gym and Scale-SWE synthesize task descriptions or tests from existing commits and PRs, respectively (Jain et al., 2025; Zhao et al., 2026). SWE-Universe also uses PR titles and descriptions as problem statements for PR-only tasks, while still requiring test patches and an executable sandbox (Chen et al., 2026). SWE-World collects merged issue–PR pairs and replaces native execution with simulation, but still conditions its simulators on a reference fix and supplied Fail-to-Pass/Pass-to-Pass (F2P/P2P) test contents (Sun et al., 2026). These pipelines remain centered on retrospective tasks with known reference solutions.

This reliance on reference solutions and verifiable task setups leaves three coverage gaps. First, even released tasks may lack reliable execution feedback: our SWE-rebench V2 availability audit finds that approximately 29% of tasks are not gold-resolved in the reference evaluation records, owing to gold–test mismatches or runtime/evaluation failures (Appendix A). These evaluation setups therefore cannot reliably produce execution-validated training traces for the affected tasks. Second, in-progress work on development branches falls outside merged-PR selection and is excluded before trace collection. Third, for unresolved open issues without a paired PR/fix or task-specific tests, these pipelines cannot verify the environment's feedback or certify successful rollouts, leaving these queries without execution-validated training traces. We therefore propose World Rollout to extend collection to these task states without requiring a paired fix, pre-existing tests, or a task-specific runtime.

Public GitHub offers a million-scale pool of potential tasks: 18.49M issue-linked merged PRs, 694k issue-linked open PRs, and 55.27M open issues without closing-linked PRs. To prioritize data construction, we start with repositories having > 100 stars and apply additional repository and task filters, yielding an estimated 5.58M candidate tasks across these sources (more details in Table 7). Building on SWE-World's separation of real repository state and simulated runtime feedback (Sun et al., 2026), World Rollout pairs task descriptions with pinned repository snapshots and uses the pretrained Qwen-AgentWorld model (Zuo et al., 2026) to simulate execution-dependent feedback, while navigation, reads, and edits operate through a runtime-decoupled operating-system (OS) layer. We extend the framework with strict multilingual routing: repository-dependent runtime actions are simulated or explicitly refused, never passed through to host execution. We also remove reference-fix and hidden-test dependencies from both the policy and world model (WM). After rollout generation, we apply two layers of filtering to control trace quality. **(1) World-model-independent** checks assess tool-call/observation alignment, repository-state consistency, explicit completion, and a non-empty patch. **(2) World-model-specific** checks ensure that simulated feedback follows the observation protocol and remains consistent with the recorded execution source and repository state.

To validate the training utility of the collected traces, we replace 25% and 50% of the executed SWE-trace token allocation in the SFT mixture with newly generated World Rollout traces at a fixed

training budget. We observe that both replacement settings maintain comparable average pass@1 and pass@k on SWE-bench Verified, Multilingual, and Pro, and Terminal-Bench 2.0, supporting the value of the generated traces as training supervision. We further compare training longer on the same data with expanding task coverage: doubling token exposure by repeating the original data reduces pass@1 by 3.1 points on average across the four benchmarks, suggesting that directly increasing training volume without sufficient task coverage can hurt performance, while retaining the original data and adding World Rollout traces from additional queries to broaden task coverage at the same doubled budget improves pass@1 over repetition alone on all four benchmarks, by +8.7, +6.2, +2.1, and +1.1 points, respectively. We next apply World Rollout traces to expand mid-training coverage and, after identical downstream SFT, improve average pass@1 and pass@k across the four benchmarks by 2.0 and 2.2 percentage points, respectively, over mid-training with limited coverage.

We further assess trace quality and collection efficiency through real-execution patch replay and serving and tool-observation audits (§4.1–4.2). On a SWE-rebench V2 subset, excluding unverifiable replays and counting no-patch traces as failures, World Rollout achieves a normalized real-test pass rate of 22.29%, numerically close to 22.39% for execution-based rollouts. These results demonstrate that collection without native runtime execution can produce repairs that pass real tests. Under an equal-resource planning configuration of 80 GB300 nodes, World Rollout is projected to deliver roughly 4–6$\times$ higher training-token throughput. Meanwhile, 85.1% of tool observations remain grounded in real repository state and only 12.7% are simulated, preserving grounded interaction for the majority of the collected experience.

Finally, our downstream behavioral analysis (§4.3) shows that, compared with repeating the same data at an equal token budget, broader task coverage is associated with more focused localization and editing, more active test construction and execution, and more consistent success across a larger set of tasks. We believe that exposure to diverse problems and repair strategies helps models internalize reusable problem-solving patterns. Together, these findings suggest that world-model simulation can broaden agentic experience, strengthening the behavioral foundations learned during mid-training and improving downstream problem solving.

2 Method: Runtime-Decoupled Data Collection

2.1 Framework Setup

Figure 1 organizes World Rollout collection into three stages. **(1) Task sources.** Collection requires a task $x = (q, r, c)$: a public problem description q, repository r, and pinned base commit c, without requiring *reference fixes from PRs*, *supplied unit tests*, or *task-specific sandbox environments*. We source these tasks from public datasets and GitHub issues and PRs, initializing a filesystem overlay S_0 at c, with more task construction details in § 2.3. **(2) Runtime-decoupled interaction.** A multilingual controller mediates the policy's actions. Following SWE-World (Sun et al., 2026), it separates real file operations from simulated runtime feedback: a general OS layer handles reads, searches, and structured edits without installing project dependencies, while the pretrained Qwen-AgentWorld model (Zuo et al., 2026) supplies observations for supported runtime requests. Unsupported actions are refused; only controller-owned edits change files. Similar to execution-based collection, we record actions, observations, and workspace diffs, along with additional world-model-specific diagnostic metrics. We detail routing and state updates in § 2.2. **(3) Post-filtering.** We apply two types of checks to the collected traces: basic integrity checks verify action–observation alignment and workspace-state consistency, while world-model-specific checks detect malformed simulated feedback and violations of the observation-only contract. We describe the filtering pipeline and criteria in § 2.4.

2.2 Runtime-Decoupled Interaction

Compared with execution-based collection, World Rollout reuses cached repository sources and bypasses task-specific dependency installation and native test/build execution, avoiding the need to provision a runnable project environment for each rollout. By reducing this environment-side overhead, the design enables faster, lower-cost collection and higher concurrency under a fixed resource budget.

At step t, the policy samples an action $a_t \sim \pi_\theta(\cdot \mid h_t)$ given its interaction history h_t. Let S_t denote the current filesystem overlay. A multilingual router ρ directs the action to one of three paths: deterministic repository operations through the general OS layer $\mathcal{E}_{\text{det}}$, simulation of supported

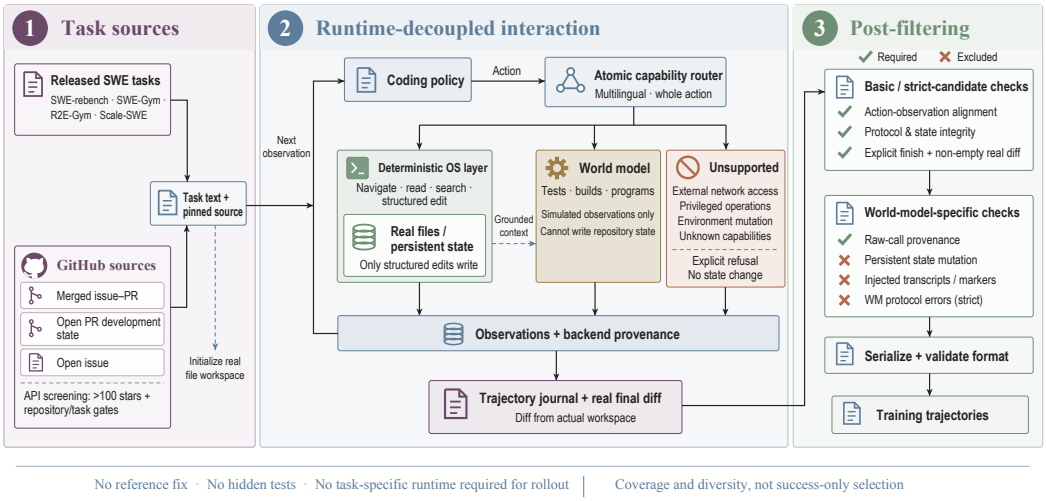

Figure 1: Overview of the three-stage World Rollout framework. We first construct tasks from public datasets and GitHub sources. We then generate traces through runtime-decoupled interaction, combining real repository operations with simulated execution feedback. Finally, basic integrity and world-model-specific checks filter the collected traces for training.

runtime feedback by the pretrained world model p_ϕ, or explicit refusal:

$$(o_{t+1}, S_{t+1}) = \begin{cases} \mathcal{E}_{\text{det}}(a_t, S_t), & \rho(a_t) = \text{DET}, \\ (\tilde{o}_{t+1}, S_t), \quad \tilde{o}_{t+1} \sim p_\phi(\cdot \mid g_t, a_t), & \rho(a_t) = \text{SIM}, \\ (\texttt{unsupported}, S_t), & \rho(a_t) = \text{REJECT}. \end{cases} \tag{1}$$

Simulation and refusal leave S_t unchanged; only controller-owned edits can modify persistent files. The world model conditions on grounded context g_t, which includes public task fields, current directory, controller-derived diff and changed paths, and completed action/observation history, but excludes reference fixes, supplied test patches, and scorer-only fields.

Building on this separation, we extend language coverage and improve rollout efficiency to support broad-coverage collection across GitHub.

Broader language coverage. We extend runtime-action detection beyond Python to Go, C/C++, Java, and CUDA-oriented compiler and build toolchains, while also covering long-tail ecosystems such as Julia, Elixir, and OCaml. The router distinguishes repository-independent OS operations from commands that require a project runtime, so runtime-dependent feedback can be simulated without installing task-specific dependencies or launching native test/build environments. This combination supports broad-coverage rollouts across heterogeneous GitHub repositories while retaining the efficiency advantage over execution-based collection.

Per-rollout workspace state. For each rollout, we maintain a private repository workspace initialized from the pinned source, together with controller-tracked working-directory and edit state. Structured edits update this workspace, so subsequent deterministic reads and searches reflect the current file contents while the original source remains unchanged. The final patch is computed as the diff between the updated workspace and the pinned source.

Execution and routing contract. To enforce this boundary, the controller executes a compound shell call deterministically only when every component is supported as deterministic. If any component requires runtime feedback, the controller routes the entire reviewed call to simulation; if any component is unsupported, it refuses the call. This all-or-nothing rule prevents partial host execution and keeps each observation consistent with the selected backend. Actions outside the supported environment contract—such as external connections (`ssh`, `curl`), privileged system operations, and filesystem-wide searches beyond the workspace—receive controller-generated error observations rather than fabricated execution results. Every observation records its backend and

Table 2: Post-filtering sample of completed rollouts from three task sources.

Source	Sampled	Non-empty diff	Passed	Other valid	Rejected	Filtered
Merged issue–PRs	5,000	4,431	4,143	842	15	82.9%
Open issues	5,000	3,784	3,449	1,536	15	69.0%
PR development states	5,000	4,026	3,854	1,144	2	77.1%
Total sample	15,000	12,241	11,446	3,522	32	76.3%

routing rule; simulated output is marked as such. The scaffold owns the tool envelope, and absent status information remains unknown rather than becoming a fabricated success code. Appendix C.2 details workspace isolation, controller-owned state updates, and the simulated-observation contract.

2.3 Constructing the expanded task pool

We construct the expanded task pool from three GitHub source states: historical merged issue–PRs, standalone open issues, and open issue-linked PRs representing development-branch states. Using the GitHub API, we retrieve public task text and repository metadata, then pair each instance with an explicitly pinned source commit for rollout. We use GitHub Search API queries to discover candidates, then repository, issue, and pull-request metadata endpoints to recover descriptions, linked relationships, and the relevant base or head commit before materialization. The raw public pools contain about 18.49M merged issue-linked PR nodes, 694k open issue-linked PR nodes, and 55.27M standalone open-issue nodes. We prioritize repositories with more than 100 stars for collection and then apply repository, task-quality, and source-materialization gates. Further pool details and the construction table are shown in Appendix .

2.4 Post-filtering

Our collection objective prioritizes task coverage and trajectory diversity over selecting only successful solutions. For tasks such as open issues, runnable verification environments are unavailable, so retention does not require execution-verified success. We also check for an explicit `finish`, a non-empty real diff, and well-formed tool calls with matching responses; training export additionally checks that traces can be converted into the required training-data format. For world-model-specific filtering, simulated observations must link to their raw WM calls and must not carry persistent state changes or claim real-execution provenance. We reject observation-only contract violations, including copied action/observation transcripts, internal prediction markers and impersonated controller refusals. Recoverable WM parsing errors are represented by safe error observations; otherwise valid traces containing them remain in the *Other valid* category.

We distinguish traces that pass all filtering criteria (*Passed*), those that remain valid but do not meet all strict criteria (*Other valid*), and those that fail integrity checks (*Rejected*), as shown in Table 2. Across the three task sources, we find that 76.3% of sampled rollouts pass all strict filtering criteria. Detailed post-filtering rules are provided in Appendix .

3 Experiments

This section aims to answer two questions: (1) can simulated traces provide training data of comparable quality to traces from execution-based collection? (2) Can a broader task pool improve mid-training performance under a fixed token budget? We introduce the experimental setup in §3.1, evaluate controlled SFT substitution in §3.2, and examine the benefits of broader task coverage through mid-training in §3.3.

3.1 Experimental setup

Model and training framework. We use the post-pretraining Nemotron 3.5 Base model (30B-A3B) to study both SFT substitution and mid-training. SFT uses an approximately 15B-token corpus for 3 epochs, while mid-training uses a 100B-token budget. We train with Megatron-LM (Narayanan et al., 2021) at a maximum sequence length of 256K tokens to accommodate long-horizon agentic trajectories. For mid-training, agentic traces account for 10% of the training tokens, with the

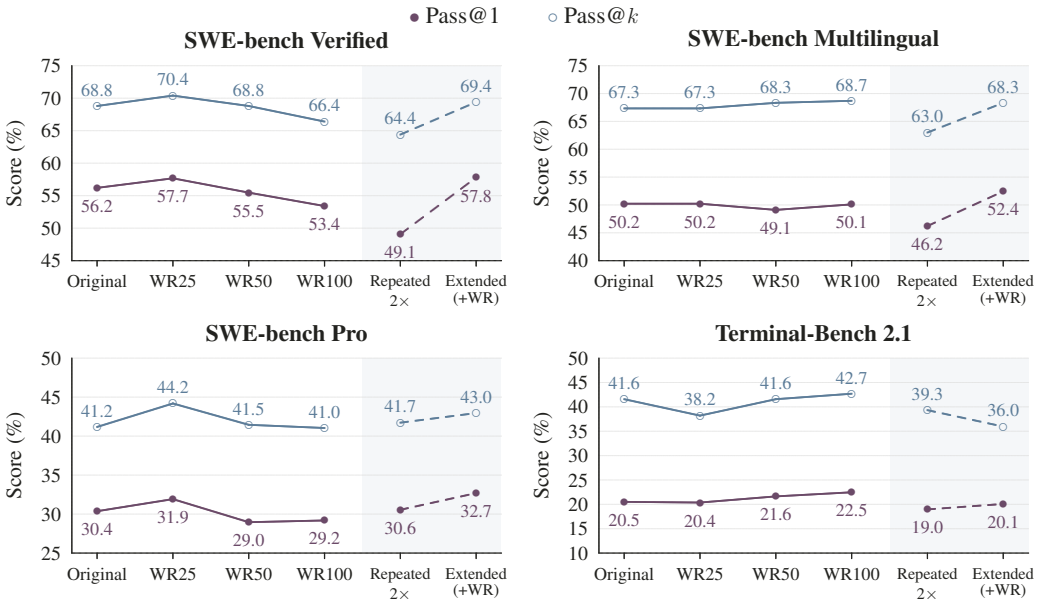

Figure 2: SFT data settings: pass@1 and pass@k under replacement and scaling. WR25/50/100 replace 25/50/100% of the execution-based traces. Shaded settings compare Repeated 2× and Extended (+WR): repeated original data versus original data plus additional-query World Rollout traces, at matched computation.

remainder drawn from a standard mixture including code diffs, repository data, and pre-training replay. Mid-training is followed by SFT with identical settings. More details are provided in Appendix C.1.1.

Evaluation. We conduct evaluation on four commonly used agentic benchmarks: SWE-bench Verified (Chowdhury et al., 2024), SWE-bench Multilingual (Yang et al., 2025), SWE-bench Pro (Deng et al., 2025), and Terminal-Bench 2.1 (Merrill et al., 2026), using real task execution. We use two metrics, pass@1 and pass@k. Pass@1 is the average success rate across all tasks and repeated runs. Pass@k is the fraction of tasks solved in at least one of the k repeats, with $k = 3$ for the SWE benchmarks and $k = 8$ for Terminal-Bench. Evaluation traces have a maximum of 200–250 agent turns and a wall-clock time limit of 1 hour per task. More details are provided in Appendix C.1.2.

3.2 Controlled SFT substitution

We replace 25%, 50%, or 100% of the execution-based traces with World Rollout traces to test whether they can match the training utility of execution-based traces. The three settings are WR25, WR50, and WR100. As shown in Figure 2, all three replacement settings retain broadly comparable overall performance, supporting the utility of simulated traces as training supervision. Averaged equally across the four benchmarks, the reported pass@1 scores are 40.0%, 38.8%, and 38.8% for WR25, WR50, and WR100, respectively, compared with 39.3% for Original. These results indicate that the generated traces can substitute for execution-based traces without a large change in aggregate performance.

We additionally test whether expanding query coverage with World Rollout traces is more effective than repeating the original data. We compare two settings at matched computation, each using twice the original token exposure. Repeated 2× revisits the original trajectory collection without expanding query coverage. Extended (+WR) retains the original data and adds World Rollout traces from additional queries, expanding the diversity and coverage of the training pool. Repeated 2× reduces average pass@1 from 39.3% to 36.2%, a drop of 3.1 percentage points, suggesting that simply training longer on a fixed data collection can hurt performance. In contrast, Extended (+WR) improves pass@1 over repetition on all three SWE benchmarks and exceeds Original on each of

Table 3: Mid-training results after identical downstream SFT: pass@1 / pass@k (%). Average weights the four benchmarks equally.

Setting	SWE-bench Verified	SWE-bench Multilingual	SWE-bench Pro	Terminal-Bench	Average
Mid Training Baseline	61.7 / 72.6	58.2 / 74.0	35.6 / **46.1**	21.9 / 41.6	44.4 / 58.6
Mid Training Expanded	**62.6 / 73.2**	**63.6 / 77.7**	**35.9 / 46.1**	**23.3 / 46.1**	**46.4 / 60.8**

Table 4: Tool-observation statistics for sampled World Rollout trajectories. (a) Sources of tool responses in collected rollouts. (b) Statistical comparison of real and simulated terminal responses. Output-length ranges span the median to the 90th percentile.

(a) Observation routing

Backend	Observations	Share
Deterministic repository operations	233,500	85.1%
Simulated runtime feedback	34,891	12.7%
Explicit refusal	6,092	2.2%

(b) Terminal-feedback distributions

Metric	Real execution	World model
Observations	8,421	16,555
Failed calls	10.4%	3.5%
Output length (chars, median–P90)	319–2,537	289–1,091

them. Across all four benchmarks, its average pass@1 reaches 40.8%, improving over Repeated 2× by 4.6 points. Together, these results support expanding query diversity and coverage as a useful direction for scaling agentic training data.

3.3 Mid-training with expanded task coverage

After validating the training utility of World Rollout traces through SFT substitution, we next apply them to mid-training. We compare mid-training with limited coverage of agentic traces to mid-training with expanded coverage, followed by identical downstream SFT. As shown in Table 3, expanded traces improve pass@1 across all four benchmarks and improve or match pass@k on every benchmark. Averaged equally across the four benchmarks, pass@1 and pass@k improve by 2.0 and 2.2 percentage points, respectively.

4 Analysis

We further analyze World Rollout from three perspectives. First, we assess the quality of generated trajectories by executing their patches on tasks with available execution environments (§4.1). Second, we examine inference efficiency and its implications for collection speed relative to execution-based rollouts (§4.2). Third, we investigate changes in downstream model behavior associated with broader training-task coverage, including improvements in problem-solving strategies and reliability (§4.3).

4.1 Trajectory quality through real-execution verification

We evaluate World Rollout and execution-based rollouts on a SWE-rebench V2 subset with available task sandboxes, applying their final patches in the real environments and running the official tests. A pass requires all fail-to-pass (F2P) and pass-to-pass (P2P) tests to pass. Excluding unverifiable replays but counting no-patch traces as failures, World Rollout achieves a normalized pass rate of 22.29%, numerically on par with execution-based rollouts at 22.39%; P2P regression shares are also close (2.86% versus 3.73%). More details on the implementation and experimental setup are provided in Appendix D.1.

4.2 Inference efficiency and tool observations

Inference efficiency. Runtime-decoupled collection avoids task-specific dependency installation and native test/build execution, allowing traces to share long-lived policy and world-model inference

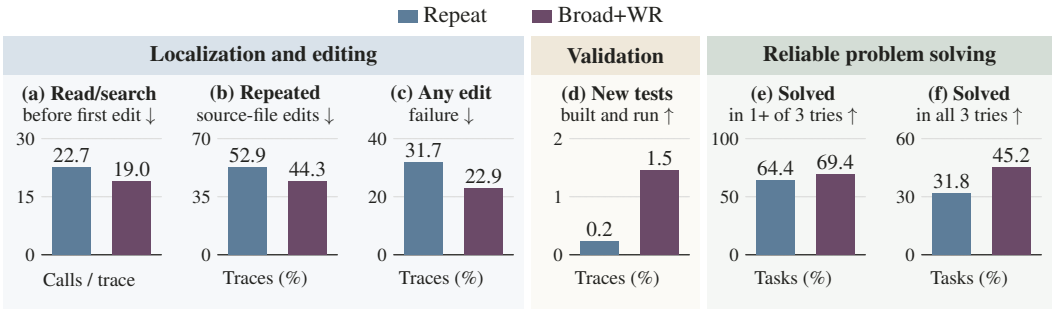

Figure 3: Behavioral improvements with broader task coverage in three categories: (a) mean read/search calls before the first successful non-test edit; (b) share of traces repeatedly editing the same non-test file; (c) share of traces with any editor failure; (d) share of traces creating and executing a new assertion-bearing Go test; (e) share of tasks solved at least once in three attempts; and (f) share of tasks solved in all three attempts.

services. This reduces environment-side overhead. Under an equal-resource planning configuration of 80 GB300 nodes, World Rollout is projected to deliver roughly **4–6× higher** training-token throughput than execution-based collection. However, WM decoding remains an important cost: WM requests account for 13.8% of model calls but 74.8% of completion tokens. Efficient model serving and observation generation are therefore important for collection speed. Appendix D.1 details the serving diagnostics, projection assumptions and sensitivity analysis.

Tool observations. Table 4 shows that 85.1% of observations remain grounded in real repository state, while 12.7% are simulated. Simulated terminal feedback has a median-to-P90 output-length range of 289–1,091 characters, compared with 319–2,537 for real execution, and fewer reported failures (3.5% versus 10.4%). These statistics suggest that failure-related feedback and long outputs make up a relatively small share of simulated observations. Appendix D.3 details the routing breakdown, observation statistics and their interpretation.

4.3 Behavioral changes with broader task coverage

We compare downstream behavior between the equal-token Repeat and Broad+WR settings. Under broader task coverage, exposure to more problems and repair strategies appears to help the model recognize issues and identify where to edit. Figure 3 summarizes the observed improvements along three dimensions: more focused localization and editing, more active test construction and execution, and more reliable problem solving over a broader set of tasks. We discuss each in turn.

More focused localization and editing. Among paired successful traces with observed non-test edits, read/search calls before the first successful edit decrease from 22.7 to 19.0, and the share of traces that repeatedly edit the same non-test file falls from 52.9% to 44.3%. Across all paired successful traces, the share with an editor failure falls from 31.7% to 22.9% (Figure 3a–c). These structured-edit records suggest more focused repairs with less search and reworking under broader task coverage.

More active test construction and execution. The model also shows more active test construction and execution in long-tail languages, as illustrated by Go. Across all SWE-bench Pro runs, the share of traces that create an assertion-bearing Go test and visibly execute the named test increases from 0.2% to 1.5% (Figure 3d). This narrow, low-frequency signal reflects more active testing, rather than merely writing a test file.

A broader and more reliable solvable task set. The share of tasks solved at least once in three attempts increases from 64.4% to 69.4%, while the share solved in all three attempts rises from 31.8% to 45.2% (Figure 3e–f). Thus, the broader-coverage setting both solves more tasks within the same sampling budget and succeeds more consistently, rather than only producing more varied attempts.

Considering these results alongside the more focused repairs, we believe that exposure to additional tasks helps internalize reusable problem-solving patterns.

5 Related work

5.1 Scaling agentic training data

Following SWE-bench's test-based formulation of repository-level issue resolution (Jimenez et al., 2024), SWE-Gym packages issue–PR pairs as executable Python training tasks (Pan et al., 2024), while SWE-rebench V2 extends executable task construction across languages and separately releases PR-derived tasks with descriptions synthesized from PR text and patches (Badertdinov et al., 2026). R2E-Gym generates descriptions and, when needed, tests from historical commits (Jain et al., 2025); Scale-SWE uses specialized agents to construct environments, tests, and issue-style descriptions from PRs (Zhao et al., 2026). SWE-Universe scales executable verifier construction from PRs, using PR titles and descriptions for a subset without linked issues (Chen et al., 2026). SWE-World replaces container execution during rollout with learned feedback, but retains known fixes and supplied F2P/P2P tests in its published pipeline (Sun et al., 2026). Across these approaches, task construction still starts from a completed commit or PR—often an issue–PR pair—and depends on a supplied or synthesized test oracle. Their executable training pipelines, other than SWE-World, also require a runnable task environment. By removing all three requirements at collection time, World Rollout supports *prospective* rollouts from unresolved issues and pinned development states while grounding file inspection and editing in real code. This broadens the eligible task pool beyond merged, test-bearing changes and enables agentic training data to scale across unresolved and in-development tasks, particularly for mid-training.

5.2 Agentic world models

In robotics, learned world models predict the effects of actions on the environment, enabling policy learning through imagined interaction (Wu et al., 2023). For LLM agents, world models instead generate tool feedback, mapping environment context, interaction history, and the current action to the next observation. SWE-World trains SWE-specific transition and reward models from Docker traces to predict execution outputs and final test results (Sun et al., 2026). Qwen-AgentWorld generalizes next-observation prediction across SWE, terminal, MCP, search, web, Android, and OS (Zuo et al., 2026). We directly use its pretrained model for execution-dependent observations. This shared action–observation interface offers flexibility to extend collection beyond SWE to search, web interaction, and general tool use by adapting the task context and tool interface.

6 Conclusion

Scaling agentic mid-training requires broader task coverage, not just more rollouts on existing queries. World Rollout expands this coverage through runtime-decoupled collection without reference fixes, supplied tests, or per-task runnable environments, reaching unresolved issues and pinned development states. Generated traces provide broadly comparable aggregate utility in SFT substitution, while adding new queries outperforms repeating existing data at matched token exposure. Expanded mid-training also improves downstream performance after identical SFT. Independent execution replay confirms that some generated patches resolve real tests, and behavioral analyses associate broader coverage with more focused problem solving and more consistent success. Together, these findings support task coverage as a key axis for scaling agentic training data and developing transferable problem-solving behaviors during mid-training.

AI Use Statement

We disclose our use of large language models (LLMs) in accordance with the ICLR 2027 policy on generative AI.

Uses requiring disclosure. In this work, LLMs were used to *generate synthetic data*: the simulated tool observations and agentic traces used as training data are produced by LLM-based world models.

By contrast, the overall data-generation framework—including the scaffolding, routing, filtering, and experimental pipeline—was designed and implemented entirely by the authors, and LLMs were not used to develop the conceptual framework or to recursively improve the method or the experiments. All experiments were run by the authors, with no LLM-driven recursive self-improvement or automated feedback loop. LLMs were not used to formulate mathematical claims, write proofs, implement the core methods, design the experimental methodology, or interpret results.

Uses with recommended disclosure. We additionally used LLMs to brainstorm and discuss research ideas and to polish the manuscript for readability. All hypotheses, methodological decisions, and analyses were made by the authors.

Review and responsibility. We have carefully reviewed all AI-assisted work. In particular, we checked LLM-generated research ideas for potential plagiarism through a manual survey of the relevant literature. The authors take full responsibility for the final content of this work, including any text, claims, or artifacts produced with the aid of generative AI.

Ethics Statement

This work involves no human subjects or sensitive personal data, and we identify no additional ethical concerns beyond applicable data-use and research-integrity requirements.

Reproducibility Statement

We plan to publicly release the complete data-generation framework, including the rollout controller, world-model interface, filtering pipeline, and evaluation scripts. The release will include the configurations and processing protocols needed to reconstruct the task pools and training traces described in Sections 2.3–2.4 and Appendix –C. Generated data will be made available to the community to support independent reproduction and follow-up research.

References

Ibragim Badertdinov, Maksim Nekrashevich, Anton Shevtsov, and Alexander Golubev. SWE-rebench V2: Language-agnostic SWE task collection at scale. *arXiv preprint arXiv:2602.23866*, 2026. URL https://arxiv.org/abs/2602.23866.

Ruisheng Cao, Mouxiang Chen, Jiawei Chen, et al. Qwen3-Coder-Next technical report, 2026. URL https://arxiv.org/abs/2603.00729.

Mouxiang Chen, Lei Zhang, Yunlong Feng, Xuwu Wang, Wenting Zhao, Ruisheng Cao, Jiaxi Yang, Jiawei Chen, Mingze Li, Zeyao Ma, Hao Ge, Zongmeng Zhang, Zeyu Cui, Dayiheng Liu, Jingren Zhou, Jianling Sun, Junyang Lin, and Binyuan Hui. SWE-Universe: Scale real-world verifiable environments to millions. *arXiv preprint arXiv:2602.02361*, 2026. URL https://arxiv.org/abs/2602.02361.

Neil Chowdhury, James Aung, Chan Jun Shern, et al. Introducing SWE-bench Verified, 2024. URL https://openai.com/index/introducing-swe-bench-verified/.

Xiang Deng, Jeff Da, Edwin Pan, et al. SWE-Bench Pro: Can AI agents solve long-horizon software engineering tasks? *arXiv preprint arXiv:2509.16941*, 2025. URL https://arxiv.org/abs/2509.16941.

GLM-5 Team. GLM-5: from vibe coding to agentic engineering, 2026. URL https://arxiv.org/abs/2602.15763.

Naman Jain, Jaskirat Singh, Manish Shetty, Liang Zheng, Koushik Sen, and Ion Stoica. R2E-Gym: Procedural environments and hybrid verifiers for scaling open-weights SWE agents. *arXiv preprint arXiv:2504.07164*, 2025. URL https://arxiv.org/abs/2504.07164.

Carlos E. Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik Narasimhan. SWE-bench: Can language models resolve real-world GitHub issues? In *International Conference on Learning Representations*, 2024. URL https://proceedings.iclr.cc/paper_files/paper/2024/hash/edac78c3e300629acfe6cbe9ca88fb84-Abstract-Conference.html.

Mike A. Merrill, Alexander G. Shaw, Nicholas Carlini, et al. Terminal-Bench: Benchmarking agents on hard, realistic tasks in command line interfaces. *arXiv preprint arXiv:2601.11868*, 2026. URL https://arxiv.org/abs/2601.11868.

Deepak Narayanan, Mohammad Shoeybi, Jared Casper, Patrick LeGresley, Mostofa Patwary, Vijay Anand Korthikanti, Dmitri Vainbrand, Prethvi Kashinkunti, Julie Bernauer, Bryan Catanzaro, Amar Phanishayee, and Matei Zaharia. Efficient large-scale language model training on GPU clusters using Megatron-LM. *arXiv preprint arXiv:2104.04473*, 2021. URL https://arxiv.org/abs/2104.04473.

Jiayi Pan, Xingyao Wang, Graham Neubig, Navdeep Jaitly, Heng Ji, Alane Suhr, and Yizhe Zhang. Training software engineering agents and verifiers with SWE-Gym. *arXiv preprint arXiv:2412.21139*, 2024. URL https://arxiv.org/abs/2412.21139.

Shuang Sun, Huatong Song, Lisheng Huang, Jinhao Jiang, Ran Le, Zhihao Lv, Zongchao Chen, Yiwen Hu, Wenyang Luo, Wayne Xin Zhao, Yang Song, Hongteng Xu, Tao Zhang, and Ji-Rong Wen. SWE-World: Building software engineering agents in docker-free environments. *arXiv preprint arXiv:2602.03419*, 2026. URL https://arxiv.org/abs/2602.03419.

Xingyao Wang, Boxuan Li, Yufan Song, et al. OpenHands: An open platform for AI software developers as generalist agents. In *International Conference on Learning Representations*, 2025. URL https://arxiv.org/abs/2407.16741.

Philipp Wu, Alejandro Escontrela, Danijar Hafner, Pieter Abbeel, and Ken Goldberg. DayDreamer: World models for physical robot learning. In *Proceedings of The 6th Conference on Robot Learning*, volume 205 of *Proceedings of Machine Learning Research*, pp. 2226–2240. PMLR, 2023. URL https://proceedings.mlr.press/v205/wu23c.html.

John Yang, Kilian Lieret, Carlos E. Jimenez, Alexander Wettig, Kabir Khandpur, Yanzhe Zhang, Binyuan Hui, Ofir Press, Ludwig Schmidt, and Diyi Yang. SWE-smith: Scaling data for software engineering agents. *arXiv preprint arXiv:2504.21798*, 2025. URL https://arxiv.org/abs/2504.21798.

Ji Zeng, Dayuan Fu, Tiantian Mi, Yumin Zhuang, Yaxing Huang, Xuefeng Li, Lyumanshan Ye, Muhang Xie, Qishuo Hua, Zhen Huang, Mohan Jiang, Hanning Wang, Jifan Lin, Yang Xiao, Jie Sun, Yunze Wu, and Pengfei Liu. daVinci-Dev: Agent-native mid-training for software engineering. *arXiv preprint arXiv:2601.18418*, 2026. URL https://arxiv.org/abs/2601.18418.

Jiale Zhao, Guoxin Chen, Fanzhe Meng, Minghao Li, Jie Chen, Hui Xu, Yongshuai Sun, Xin Zhao, Ruihua Song, Yuan Zhang, Peng Wang, Cheng Chen, Jirong Wen, and Kai Jia. Immersion in the GitHub universe: Scaling coding agents to mastery. *arXiv preprint arXiv:2602.09892*, 2026. URL https://arxiv.org/abs/2602.09892.

Yuxin Zuo, Zikai Xiao, Li Sheng, Fei Huang, Jianhong Tu, Yuxuan Liu, Tianyi Tang, Xiaomeng Hu, Yang Su, Qingfeng Lan, et al. Qwen-AgentWorld: Language world models for general agents. *arXiv preprint arXiv:2606.24597*, 2026. URL https://arxiv.org/abs/2606.24597.

A EXTENDED EXISTING TASK-AVAILABILITY AUDIT

Table 5: SWE-rebench V2 availability audit. Only the gold-resolved subset (71.36%) passes the reference check required by conventional execution-verified trace collection.

Audit category	# of Tasks	Percentage
Gold-resolved	22,891	71.36%
Gold-test mismatch	7,902	24.63%
Runtime/image access failures	1,042	3.25%
Evaluation results unavailable	244	0.76%
Total	32,079	100.00%

Table 5 summarizes reference-patch evaluation records for the pinned 32,079-task SWE-rebench V2 inventory. Gold-resolved means that the reference fix passed the strict F2P/P2P criterion. Gold–test mismatch means that even the supplied reference solution was not accepted: its observed passing set did not match the required F2P/P2P set. When the gold solution itself fails this criterion, the task's evaluator cannot be relied upon to select successful rollout traces. Such disagreement suggests a problem in the task environment or verification setup that may also compromise intermediate test and execution feedback. Reliable collection therefore requires reconciling the reference patch, tests and evaluation setup. Runtime/image access failures denote task environments that were unusable or could not be loaded in the audited setup. Evaluation results unavailable means that reference-patch evaluation results could not be retrieved or decoded.

These gaps restrict execution-verified trace collection for the affected queries: pipelines requiring a usable runtime and reliable test-based acceptance cannot consistently turn these queries into validated training traces. World Rollout can instead collect traces from their issue statements and pinned repository states, combining deterministic file operations with simulated runtime feedback and post-rollout integrity filtering. This enables structurally valid traces for otherwise excluded queries without requiring a working task runtime or a passing reference solution, complementing execution-verified collection.

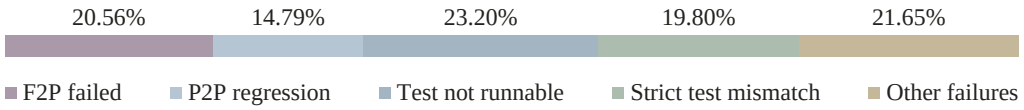

Figure 4: Detailed Breakdown of the 9,188 Originally Non-Gold-Resolved Tasks.

We additionally show a detailed breakdown of the originally non-gold-resolved instances in Figure , using their recorded rerun outcomes. *F2P failed* means that the reference patch did not make all required failing tests pass; *P2P regression* means that it failed to preserve tests that initially passed. *Test not runnable* denotes cases where the test suite could not run successfully. *Strict test mismatch* denotes disagreement with the evaluator's exact-match criterion, including extra passing tests, duplicate expected entries or test-name mismatches. *Other failures* groups image, patch-application, timeout and evaluator errors.

B ORACLE REQUIREMENTS AND COMPLEMENTARY TASK COVERAGE

Table 6 extends Table 1. The distinction is task admissibility, not a claim of larger realized repository coverage. With other source gates held fixed, removing the test-patch and known-fix requirements admits otherwise compatible historical tasks as well as no-test tasks and unresolved requests. Different source windows and repository filters do not establish a corpus-level superset.

Table 6: Oracle requirements and complementary task coverage. Supported means admissible without the named prerequisite, not verified success or measured additional coverage.

Dimension	SWE-rebench V2	SWE-World	SWE-Universe	**World Rollout**
Collection requirements and admissible task states				
Source state	Merged PR; pre-/post-fix states	Historical issue–PR; known fix	Historical PR; pre-/post-fix states	Historical/open PR; unresolved issue
Problem statement	Issue; PR-derived text in extension	Issue description	Issue or PR description	Issue or usable PR description
Gold solution patch	Required	Required by simulator	Required	Not required
Pre-existing task tests	PR test patch required	F2P/P2P contents required	PR test component required	Not required
Runnable task environment	Required	Not for simulated collection	Required	Not for rollout generation
Feedback / verification	Executed F2P/P2P tests	Learned transitions and test reward	Executed evaluation.sh	Learned observations; no correctness oracle
Unresolved issue without known fix	Not in pipeline	Not in pipeline	Not in pipeline	Supported
Open PR without accepted fix	Not in pipeline	Not in pipeline	Not established	Supported
Task without PR test changes	Excluded	Not established; tests still required	Excluded	Supported
Repository scale				
Reported corpus repos	3,617	3,763	52,960	145,306

Source-pool scale. Our raw GitHub search pool contains 55.27M public open issues without a closing-linked PR, spanning the repository frame reported in Table 6. The corresponding gated source-pool estimates are summarized in Appendix . SWE-rebench V2 and SWE-Universe are retrospective PR-anchored pipelines with test or verifier requirements, while SWE-World replaces execution with learned feedback but still starts from known fixes and supplied tests. World Rollout removes these prerequisites at collection time, allowing unresolved issues and open development states to enter the pool. Merged issue–PRs provide the largest retrospective pool, open PRs add in-progress states, and the standalone open-issue pool supplies the broadest prospective source.

B.1 UPSTREAM SOURCE-POOL ESTIMATES

Task construction and raw index. For retrospective merged issue–PRs, we pair issue text with the pinned PR-base commit and withhold the solution and test patches from rollout. For open PRs, usable linked-issue text supplies the task; otherwise we fall back to the public PR title and body. The workspace is pinned at the PR-base commit, while the unmerged head identifies the live development state rather than an accepted solution. For standalone open issues, implementation-needed issue text is paired with a pinned default-branch commit. The corresponding all-public GitHub index contains 18,490,756 merged issue-linked PRs, 693,805 open issue-linked PRs, and 55,274,088 open issues without a closing-linked PR.

B.2 POST-FILTERING RULES AND TRAINING-DATA PREPARATION

Table 7: From GitHub source pools to rollout tasks: raw index scale, eligible candidate pools, and task construction.

Source state	All-public GitHub index	Eligible pool: $>$ 100 stars + other gates	Rollout input
Retrospective merged issue–PR	18.49M PR nodes	≈3.18M PR nodes	Issue text + pinned PR-base snapshot; solution/test patches withheld.
Open-PR development state	693,805 PR nodes	≈217k PR nodes	Issue/PR text + pinned PR-base snapshot; head remains unmerged.
Standalone open issue	55.27M issue nodes	≈2.18M issue nodes	Implementation-needed issue text + pinned default-branch snapshot.

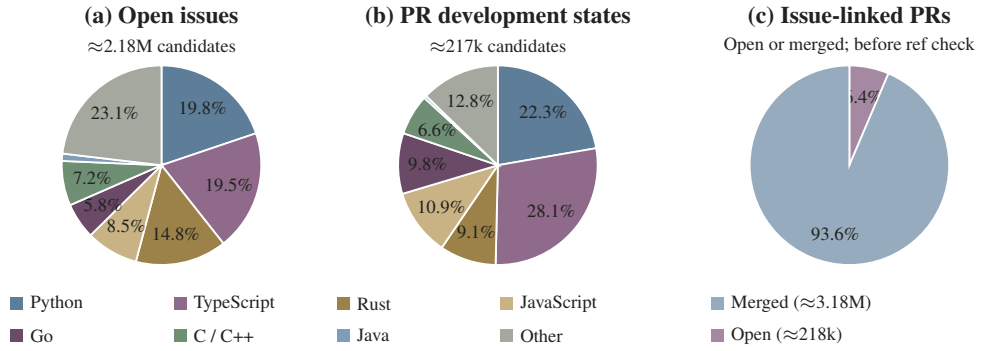

Figure 5: Source-pool composition. (a,b) Distribution of repository primary languages for open-issue and PR-development sources. (c) Distribution of open versus merged issue-linked PR nodes.

Conversation and state integrity. We first reconstruct the conversation from raw policy responses and recorded tool observations. Tool calls and responses must match in count, order, and identifier. Each simulated observation must also match its linked WM call's action, parsed response body, and available status fields. We check that backend attribution is known and consistent, and that persistent state changes correspond to controller-owned mutations. A failure of these conversation, attribution, or state-integrity checks rejects the trace.

World-model-specific checks. The quality assessor rejects unknown or inconsistent backend attribution, WM outputs carrying persistent state deltas, and simulated outputs marked as repository-grounded or real-execution verified. Its high-precision content checks detect paired `[action]` / `[observation]` transcript markers, leaked `predict_observation:` internals, and fixed strings that impersonate controller refusals. They are not a general hallucination detector. At the adapter boundary, empty/malformed predictions and reserved scaffold framing produce a fixed safe error observation. Native response bodies are accepted directly; if XML-style protocol tags occur, the entire response must match the supported protocol. An exact supported trailing terminal footer is normalized once by the scaffold. Raw responses remain auditable, but malformed contents are not copied to the policy.

Filtering categories and selection. Rules are applied in order: protocol, attribution or state-integrity failure implies *Rejected*; otherwise, traces with recorded real-execution verification are tracked separately. Among the remaining traces, explicit `finish`, a non-empty diff and no WM protocol errors imply *Passed*; all others are *Other valid*. Thus an attributed parsing-error recovery can remain *Other valid*, whereas a leaked action transcript is rejected. *Other valid* is separated from strict candidates rather than labelled structurally corrupt. A real-execution verification record indicates that verification occurred, not that tests passed. Completion is determined by an explicit `finish` action, and patch non-emptiness is determined from the final workspace diff against the pinned base, rather than from the number of edit calls or a simulated success message. An otherwise valid trace with no `finish`, an empty final diff, or a recoverable WM protocol error is therefore *Other valid*.

Training-data conversion. Before export, we check model-checkpoint lineage, prompt and dataset metadata, and tool-definition checksums. Selected traces must be convertible into the training conversation format: missing tool declarations, broken role boundaries, or failures to reconstruct the conversation with the training chat template cause conversion to fail. These conversion checks are applied in addition to the rollout-level filtering rules.

Confidence and optional model judging. The WM predicts tool observations; it does not grade the final trajectory at `finish`. For responses containing confidence, `wm_confidence` is the mean of available per-observation values; content-only responses leave it missing. The controller has a separate online guard: by default, 3 successive low-confidence WM responses below 0.3 trigger early termination. Missing confidence does not count as low confidence. This can yield an incomplete *Other valid* trace, but the aggregate confidence is not a post-filtering acceptance threshold or a resolution probability. The optional post-finish judge sees the issue, final diff, termination reason and policy transcript, and returns trajectory quality and issue–patch relevance in $[0, 1]$, plus a rationale. It receives no reference patch or hidden tests and never assigns execution reward. This module is not invoked by the default rollout quality path, and no judge-score cutoff is configured there. Unmeasured scores remain missing rather than being filled from simulated test feedback.

C EXPERIMENTAL AND IMPLEMENTATION DETAILS

C.1 EXPERIMENTAL SETUP DETAILS

C.1.1 TRAINING SETUP

Model and training budgets. We use Nemotron 3.5 Base (30B-A3B) after pre-training as the base model for the SFT substitution and mid-training studies. The training context is 256K (262,144 tokens). SFT runs for 3 epochs over an approximately 15B-token corpus. The matched-coverage mid-training comparison allocates 100B tokens to each arm.

Hyper-parameters. The recipe uses BF16 precision, global batch size 64, and micro-batch size 1. Warmup covers the first 2.5% of training samples, followed by cosine learning-rate decay from 2×10^{-5} to 5×10^{-6}. Adam uses $\beta_1 = 0.9$, $\beta_2 = 0.95$, weight decay 0.1, and gradient clipping at 1.0. Megatron-LM (Narayanan et al., 2021) is configured with tensor parallelism 2, context parallelism 4, and expert parallelism 8. Pipeline parallelism is 1. We use selective MoE activation recomputation and activation offloading to reduce memory usage, and overlap gradient reduction and parameter gathering with computation.

Training objective and masking. We perform full-parameter SFT using next-token cross-entropy on assistant-target spans, including retained reasoning and tool-call text. System and user messages, tool observations, and padding are masked from the loss. World-model-generated observations therefore provide context without becoming direct prediction targets. We also use 2-layer multi-token prediction (MTP) with an auxiliary loss scaling factor of 0.1.

Sequence packing. We pack complete materialized conversations into 256K-token sequences using the training tokenizer. The preprocessor drops conversations longer than this limit rather than splitting them. Conversation boundaries are passed to the packed attention and recurrent-state computations. The training loader adds alignment padding for context parallelism and right-truncates any overflow at the final context limit. SFT corpus token counts include masked context but exclude padding.

Mid-training protocol. Agentic traces occupy 10% of the 100B-token budget; the remaining tokens come from a standard mixture including code diffs, repository data, and pre-training replay. Across mid-training arms, we hold initialization, the background mixture, optimizer schedule, and total token exposure fixed, varying only the task allocation within the agentic component. All arms then undergo SFT with identical settings.

C.1.2 EVALUATION SETUP

The final evaluated checkpoints correspond to the completed three-epoch schedule.

Interaction budgets. We allow up to 200 agent turns on SWE-bench Verified and SWE-bench Multilingual, and 250 turns on SWE-bench Pro. Evaluation traces have a wall-clock time limit of 1 hour per task, including Terminal-Bench.

Decoding and context. Across the four benchmarks, we sample with temperature 1.0 and top-p 0.95, with reasoning enabled. We serve the models with vLLM using a nominal evaluation context of 256K (262,144 tokens).

Agent and tools. For the SWE benchmarks, we use an OpenHands agent (Wang et al., 2025) with terminal execution, file editing, reasoning, task tracking, and task-completion tools. For Terminal-Bench, we use the Terminus-2 agent through Harbor (Merrill et al., 2026). All evaluations take place in real task environments. We deploy task-specific Docker containers in AWS-hosted sandboxes, where agents interact with the task environment and execute commands.

C.2 WORKSPACE AND EXECUTION-BOUNDARY COMPARISON WITH SWE-WORLD

The configured simulator is Qwen-AgentWorld-35B-A3B.

Table 8: Implementation-level comparison with SWE-World (Sun et al., 2026). Both systems retain a real mutable repository state; the rows specify how that state is materialized, mutated, and coupled to simulated observations.

Layer	World Rollout	Released SWE-World implementation
Workspace	A read-only bare mirror materializes one immutable base tree per repository revision. Each episode receives a writable, inode-isolated copy; the episode tree contains no Git metadata.	Each instance receives a local Git clone and a second backup clone. The working clone retains Git metadata and supplies the staged agent patch.
Terminal isolation	Every deterministic shell call runs in fresh user, PID, network, IPC, UTS, and mount namespaces. The episode tree is mounted read-only, the environment is cleared, network is absent, capabilities are dropped, and missing confinement fails closed.	Commands run through a local shell in the temporary clone with the parent process environment. The released environment applies application-level lists of forbidden command strings.
Routing	A parsed three-way capability router assigns the whole action to deterministic execution, multilingual runtime simulation, or rejection. Unsupported components dominate simulated ones; otherwise any runtime component makes the full compound action simulated.	Whole-command dispatch primarily detects Python and `pytest`; commands not detected by that predicate run locally. Selected commands are string-filtered, and `pip install` receives a fixed successful observation.
Persistent mutation	The shell and world model are state-inert. Only the controller-owned structured editor can write, using atomic replacement, no-clobber creation, bounded undo state, and an explicit state delta.	The structured editing tools and locally executed non-Python shell path operate on the writable working clone. Simulated Python execution does not intentionally write working-tree files, although patch extraction stages the Git index.
Diff and provenance	Status and diff are synthesized directly from immutable base bytes and the private overlay. Each observation records backend, routing rule, component decisions, and any linked world-model call.	Patch extraction stages the working tree and reads the cached Git diff, with configured file exclusions. A raw-simulation field distinguishes simulated output when present.
Recovery	A checksummed, hash-chained turn journal advances policy messages, world-model calls, controller state, and a workspace-diff fingerprint together. Resume is refused when the overlay disagrees with the last durable boundary.	The released environment reset closes its temporary directories and creates fresh working and backup clones; the inspected path does not expose an equivalent state-checked turn-resume protocol.
Oracle interface	The simulator sees the public task, current real diff, changed paths, and completed interaction history. Gold patches, test patches, and scorer-only fields are excluded by construction.	The transition model context contains the ground-truth patch; the reward-model context additionally contains F2P/P2P tests, their contents, and the test command.

Both systems keep repository navigation and editing grounded in real files and replace repository-dependent execution with learned feedback. The distinction is therefore not the high-level sandbox/runtime split, but the state and routing contract used to realize it. We contrast our audited implementation with the public SWE-World release at revision `818b4e9`. For SWE-World, we refer to its released `SimulatedEnv` and `LocalRuntime` path; an unreported outer deployment layer could add containment beyond what is visible there.

State ownership. The central invariant in our implementation is that a persistent repository transition is equivalent to a successful controller-owned edit. A simulated observation may describe a successful test or program execution, but it cannot create files, update generated artifacts, or otherwise change the overlay. Likewise, the read-only terminal cannot introduce an unrecorded write. The next grounded observation and the final patch therefore derive from one authoritative state rather than from side effects described by the simulator.

Language coverage and action detection. The router recognizes interpreters, compilers, build tools, and test entry points across language families. Examples include `gcc`, `g++`, and `cmake` for C/C++; `java`, `javac`, `mvn`, and `gradle` for Java; and `julia`, `elixir`/`mix`, and `ocamlc`/`dune` for less common ecosystems. Deterministic execution is restricted to an explicit allowlist of repository-independent reads, searches, and controller operations, rather than serving as a default shell fallback. Supported runtime invocations and explicit project-local programs are candidates for simulation, subject to argument and capability checks; unrecognized commands are refused. These are explicit support rules, not a claim of exhaustive coverage of every language or toolchain.

Atomic multilingual policy. Routing is performed before execution over the complete shell action. The controller recognizes runtime families across the languages admitted by the task pool, while network, privilege, environment mutation, and unknown capabilities are explicit refusals. This policy prevents a mixed command from executing a deterministic prefix before a later component is simulated, and prevents an unrecognized non-Python runtime from silently reaching the host.

Worker and service deployment. CPU trace-collection workers run independently of long-lived policy and world-model services. Bounded queues, leases, and idempotent retries separate trace-collection concurrency from model serving.

Durability and auditability. The immutable base, controller-owned state deltas, and per-observation routing records provide a common identity for recovery and export. Recovery reconstructs the pinned base and replays successful controller-owned mutations from the durable turn journal. Checkpoint recovery accepts the workspace only when its diff fingerprint matches the durable turn journal; training export can in turn reconstruct every model-facing observation and identify whether it was grounded, simulated, or refused.

Task and observation contract. The simulator's task allowlist consists of instance identity, repository, base commit, problem statement and language. The controller adds grounded workspace context and completed interactions, excluding policy reasoning and scorer-only fields. Runtime simulation produces an observation body; formatting and backend provenance remain controller-owned. Simulated calls do not produce persistent state deltas. File copies have private ownership; recovery replays successful controller-owned mutations and checks the state fingerprint rather than trusting a simulated side effect.

D ANALYSIS DETAILS

D.1 TRAJECTORY QUALITY THROUGH REAL-EXECUTION VERIFICATION

Table 9: Real-execution verification outcomes on the SWE-rebench V2 subset, using the same normalization as Section 4.1. Entries are shares within each rollout cohort after excluding unverifiable replays; no-final-patch traces remain failures.

Outcome	World Rollout share	Execution-based share
Pass (F2P+P2P)	22.29%	22.39%
F2P failed	60.00%	31.34%
P2P regression	2.86%	3.73%
No final patch	14.86%	42.54%
Total	100.00%	100.00%

For the SWE-rebench V2 subset in Section 4.1, each final patch is applied to the pinned repository revision in the real task container and evaluated using the official hidden tests, test command and log parser. Pass requires all fail-to-pass (F2P) and pass-to-pass (P2P) tests to pass; additional passing tests do not invalidate success, while missing required tests count as failures. Among verifiable patches, F2P failures are classified first; P2P regression denotes a patch that passes all F2P tests but fails to preserve all P2P tests.

Table 9 normalizes each rollout cohort after excluding replays with unknown test outcomes due to infrastructure, patch-application, timeout or test-runtime failures. Traces without a final patch remain in the denominator as failures.

D.2 INFERENCE EFFICIENCY

World-model decoding cost. In a separate serving audit, WM requests account for 13.8% of policy-plus-WM calls but 74.8% of completion tokens. A WM call produces a median 10,030 completion tokens, while the parsed observation has a median 179 characters; intermediate reasoning is discarded before the observation reaches the policy. Only 0.18% of audited calls hit the output-length limit, so the short visible observations are not generally explained by output truncation. Thus, WM decoding accounts for the bulk of completion-token usage, even though most tool responses do not require simulation.

Metric and equal-resource allocation. We estimate end-to-end collection throughput in serialized training-trajectory tokens per day, not single-request inference latency. Our setting uses 80 GB300 nodes with 4 GPUs per node (320 GPUs), with the same GPU budget for both collection methods. The policy checkpoint, prompt, scaffold, sampling parameters, turn limit and task stream are assumed identical. Simulated-runtime collection allocates 64 GPU nodes to the policy and 16 to the WM. This policy–WM allocation achieved the highest rollout throughput in our empirical tests. Execution-based collection assigns all 80 to the policy. Thus, the 16 nodes saved by omitting the WM become additional policy replicas, rather than being removed from the execution baseline's budget. We assume 128 concurrent execution containers. Table 10 summarizes the configuration and projections.

Simulated-runtime projection. We estimate throughput from request concurrency and per-trajectory service demand. Each policy node supports 32 concurrent requests and each WM node supports 32, giving $S_{\mathrm{P}} = 64 \times 32 = 2,048$ policy slots and $S_{\mathrm{W}} = 16 \times 32 = 512$ WM slots. The historical serving audit records 35.2 policy calls and 5.65 WM calls per trajectory, with single-request decode times of 13.4 and 88.9 seconds, respectively. Their service demands are

$$\begin{aligned} D_{\mathrm{P}} &= 35.2 \times 13.4 \simeq 471.7 \text{ s}, \\ D_{\mathrm{W}} &= 5.65 \times 88.9 \simeq 502.3 \text{ s}. \end{aligned} \tag{2}$$

Assuming serial calls within each trajectory, the modeled service time is $T_{\mathrm{sim}} = D_{\mathrm{P}} + D_{\mathrm{W}} \simeq 974$ seconds (16.2 minutes), excluding queueing, network, prefill and other overheads. This is not a

Table 10: Equal-resource planning configuration and projected training-token throughput. Trajectory time is modeled service time for simulation and historical wall-clock mean / median for execution. Concurrency is idealized for simulation and assumes full container occupancy for execution. Both throughput columns are projections, not matched-resource measurements.

Resource / estimate	Simulated runtime	Execution-based
GPU nodes	80	80
Total GPUs	320	320
CPU nodes	4	4
Policy GPU nodes	64	80
World-model GPU nodes	16	0
Execution-container slots	–	128
Requests / node (policy / WM)	32 / 32	–
Simulated request slots (policy / WM)	2,048 / 512	–
Trajectory time (s)	974	717.9 / 519.3
Concurrent trajectories	993	128
Projected tokens/day (mean timing)	5.72B	1.00B
Projected tokens/day (median timing)	5.72B	1.38B

measured end-to-end trajectory duration. With unchanged request times at the stated concurrency and enough in-flight work to saturate the bottleneck, the ideal steady-state completion rate is

$$r_{\text{sim}} = \min\left(\frac{S_{\text{P}}}{D_{\text{P}}}, \frac{S_{\text{W}}}{D_{\text{W}}}\right) \simeq 1.019 \text{ trajectories/s.} \tag{3}$$

The WM service is the bottleneck under these historical timing inputs. The corresponding effective in-flight count is $C_{\text{eff}} = r_{\text{sim}} T_{\text{sim}} \simeq 993$, or approximately 1,000 trajectories. This derived count is neither a configured concurrency limit nor a measurement; policy and WM request slots cannot simply be added to obtain trajectory concurrency. Using 65,000 serialized training tokens per trajectory gives

$$Q_{\text{sim}} = r_{\text{sim}} \times 86{,}400 \times 65{,}000 \simeq 5.72\text{B tokens/day}, \tag{4}$$

equivalent to approximately 88,071 trajectories/day. Maintaining the historical single-request times at the stated concurrency remains an unvalidated assumption; both trajectory time and throughput are projected.

Execution-based projection. The historical sample of real-execution trajectories has a mean wall-clock duration of 717.9 seconds and a median of 519.3 seconds. Using the shared serialization assumption of 65,000 training tokens per completed trajectory and continuously occupied container slots, we estimate

$$R_{\text{exec}}(T) = 128 \times \frac{86{,}400}{T}, \qquad Q_{\text{exec}}(T) = R_{\text{exec}}(T) \times 65{,}000. \tag{5}$$

Here R_{exec} is trajectories/day and T is seconds/trajectory. Using the mean duration gives approximately 15,405 trajectories/day, or 1.00B tokens/day. Substituting the median gives the more optimistic execution estimate of 21,296 trajectories/day, or 1.38B tokens/day. This assumes that the enlarged policy service can keep all 128 containers busy and that historical timings transfer to the proposed allocation. We consider an average yield of 65,000 usable training tokens per trajectory a reasonable assumption.

Projected gain and interpretation. Dividing the simulated projection by the execution estimates gives

$$\frac{Q_{\text{sim}}}{Q_{\text{exec}}(T_{\text{median}})} \simeq 4.14\times, \qquad \frac{Q_{\text{sim}}}{Q_{\text{exec}}(T_{\text{mean}})} \simeq 5.72\times\,. \tag{6}$$

Thus, the planning estimate is 4.1–5.7×, or roughly 4–6× higher training-token throughput. The interval is a mean/median latency sensitivity range, not a confidence interval. The potential advantage is at the collection-system level: GPU batching and concurrent simulated rollouts can reduce dependence on long, sequential environment interactions. Execution-based collection faces two constraints. First, each CPU sandbox consumes dedicated resources—for example, 4 CPU cores and 8 GB of

memory—which limits how many sandboxes can run concurrently under a fixed resource budget. Second, long-running commands and blocking tool calls can prolong individual trajectories and make their completion times more variable. In contrast, World Rollout generates runtime feedback through model inference rather than waiting for these commands to execute, making feedback latency more predictable under a fixed serving configuration. Under the stated assumptions, this yields a projected average collection throughput roughly 4–6$\times$ higher than execution-based collection.

D.3 TOOL-OBSERVATION DETAILS

Observation and decoding cohorts. Table 4a covers approximately 1.2k tasks, 9.9k rollout trajectories and 274k tool observations. A task can have multiple sampled trajectories, each containing observations from multiple tool calls (about 28 observations per trajectory on average). No native task execution is routed in this corpus. Panel (b) uses a separate sample of 3k simulated trajectories. The decoding audit covers approximately 3.9k WM calls, with 0.18% reaching the output-length limit. Model-request shares use policy-plus-WM requests, not all tool observations; completion-token shares include reasoning. Token counts and character lengths have different units, and separately reported medians are not a per-call compression ratio. All three audits are historical snapshots, not one synchronized cohort or a new command-family frequency census.

Grounded interactions and routing. Table 4a separates deterministic repository operations, simulated runtime feedback and explicit refusals. Reads, searches and structured edits make up the grounded part of the interaction; tests and builds are examples of execution-dependent calls. These are backend counts, not a measured command-family breakdown. The distribution across task rollouts is heterogeneous: 27.5% of task rollouts make no WM call; the median simulated-observation share is 8.7%, rising to 25.0% at the 90th percentile. Simulation is therefore concentrated in the subset of interactions that need runtime feedback, rather than replacing every tool response.

Terminal-feedback distributions. Restricting the comparison to terminal feedback avoids treating real file edits as simulated observations (Table 4b). The median-to-P90 output-length ranges are 289–1,091 characters for WM feedback and 319–2,537 for real execution; the WM's 90th percentile is only 43.0% of the real-runtime value. Failed calls appear in 3.5% of WM observations versus 10.4% of real terminal observations. These statistics suggest that failure-rich and long-output interactions are underrepresented in the simulated traces.