# SCoPR: Scaling State-Evolution Horizons for Long-Horizon Agents

**Anonymous authors**
Paper under double-blind review

## Abstract

Long-horizon coding agents must preserve and revise repository state across successive development stages. This raises two data-design questions: can agent-authored software histories provide useful training supervision, and does extending the number of dependent stages improve downstream performance? We introduce SCoPR, a pipeline that synthesizes training trajectories from chains of pull requests (PRs). Building on daVinci-Agency, SCoPR preserves transition diffs across workspace handoffs, supplies world-model feedback when execution is unavailable or unstable, and filters candidate trajectories against reference patches at intermediate stages. Across six benchmarks, the full data mixture achieves the highest reported average among the compared data configurations for both Qwen3-30B-A3B and Step-3.5-Flash. On Qwen3-30B-A3B, it improves SWE-bench Verified from 24.2% to 27.2% and Toolathlon from 3.7% to 5.6%. OpenClaw-derived data is beneficial in the mixture, but does not improve average performance when used alone. In a separate study holding the number of training samples fixed at 722, increasing the horizon from one to five or more PRs raises the reported average from 0.237 to 0.274. Together, these findings support dependent repository-state transitions as a useful unit of training-data design for long-horizon agents.

## 1 Introduction

Software engineering requires more than producing a correct patch in isolation. An agent may need to implement a feature, diagnose a regression introduced by that implementation, and adapt the resulting code to a later requirement. Each stage changes the starting point for the next. Benchmarks such as SWE-bench Pro, SWE-EVO, and FeatureBench make these extended development demands explicit, while Toolathlon examines sustained task execution across diverse tools (Deng et al., 2025; Thai et al., 2025; Zhou et al., 2026; Li et al., 2025a). These settings motivate training data that preserves not only an objective and its final solution, but also the dependent state transitions through which the solution evolves.

Public software histories offer a potential source of such supervision. They increasingly include contributions produced with coding agents: AIDev, for example, documents 932,791 agent-authored PRs across 116,211 repositories (Li et al., 2026). Yet the availability of these artifacts does not establish their value as training data. A PR records a submitted change, not necessarily the interaction sequence that produced it, and a collection of individually plausible changes need not form a coherent development trajectory. The learning problem is therefore to reconstruct useful interactions from related artifacts and retain trajectories that preserve their cross-stage dependencies.

Existing work provides a foundation for this reconstruction. SWE-bench defines repository-level issue-resolution tasks, and SWE-smith scales executable tasks and agent trajectories (Jimenez et al., 2023; Yang et al., 2025b). Such trajectories can contain many intermediate edits and tool calls within one task; their limitation here is not the absence of intermediate states, but the lack of an explicit sequence of dependent development stages. daVinci-Agency addresses that distinction by organizing related PRs into staged trajectories (Jiang et al., 2026). SCoPR builds on this chain-of-PR formulation. We focus on the practical difficulty of generating longer usable chains and on the training value of their source composition and stage count.

SCoPR combines two rollout mechanisms with reference-guided rejection sampling. *Diff-preserving handoff* carries transition artifacts across stages so that a later agent can inspect both the current



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.

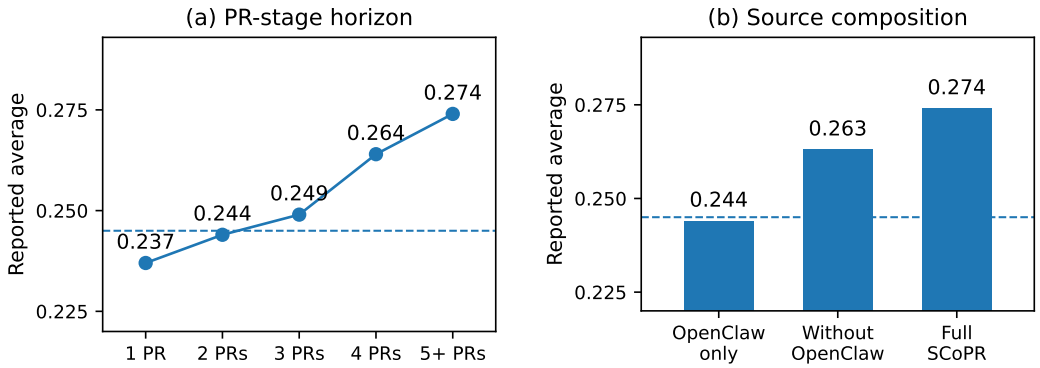


Figure 1. Two data-design comparisons on Qwen3-30B-A3B. **Left:** reported averages at a fixed sample count of 722. **Right:** reported averages for different source compositions. Dashed lines mark the baseline performance.

workspace and the edits that produced it. *World-model execution feedback* supplies provisional observations when real execution is unavailable or unstable, allowing the agent to continue diagnosing and revising its patch. A separate evaluator compares each candidate patch with the corresponding reference PR for functional alignment. Simulated observations guide generation; they are not treated as evidence of executable correctness. Together, these mechanisms support the construction of state-continuous training trajectories.

We use this pipeline to study two questions. First, does data derived from an agent-authored repository improve training when used alone or alongside other repository-derived data? Second, does a larger number of dependent PR stages improve performance when the number of training samples is held fixed? OpenClaw-only training does not improve the reported cross-benchmark average over either starting checkpoint, but the full mixture outperforms the configuration without OpenClaw on that average for both models. Separately, a 722-sample horizon study improves the average from 0.237 with one PR to 0.274 with five or more PRs. This is evidence for horizon-aware data construction at a fixed sample count, not yet an isolation of horizon from token exposure or data composition.

Our contributions:

1. We propose SCoPR, a scaling chain-of-PRs pipeline that extends a daVinci-Agency-style construction process with world-model execution feedback and state-transition diff preservation, thereby improving the information completeness of state evolution and enabling the construction of higher-signal long-horizon software engineering trajectories.
2. We study the problem of scaling agent-authored software data for long-horizon coding-agent training, showing that state-evolution data constructed from agent-authored repositories is slightly lower in quality than that constructed from human-authored repositories, but yields stronger gains when integrated with human-authored repositories.
3. We empirically investigate scaling horizon as a distinct scaling dimension for long-horizon coding agents, demonstrating that, under controlled data quality, increasing the effective horizon of rejection-sampled agent-authored chains yields additional gains on long-horizon tasks beyond simply increasing sample count.

## 2 Related Work

**Software-engineering agents and training data.** SWE-agent and OpenHands provide interfaces and environments for agents that inspect repositories, modify code, and interact with tools (Yang et al., 2024; Wang et al., 2024). Agentless, AutoCodeRover, and MASAI explore alternative ways to organize localization, repair, and validation (Xia et al., 2024; Zhang et al., 2024; Arora et al., 2024). Complementary work scales training resources: SWE-smith constructs executable tasks and interaction trajectories, R2E-Gym combines procedural environments with hybrid verification, and daVinci-Env develops the OpenSWE environment-synthesis framework (Yang et al., 2025b; Jain

et al., 2025; Fu et al., 2026). These approaches establish the value of repository-grounded supervision. Our focus is the structure of dependencies *across* development stages rather than the number of tasks or environments alone.

**State evolution and simulated feedback.** SWE-EVO and FeatureBench evaluate software changes extending beyond isolated repairs, and Toolathlon tests long-horizon execution across heterogeneous tools (Thai et al., 2025; Zhou et al., 2026; Li et al., 2025a). daVinci-Agency uses PR chains to encode decomposition, functional consistency, and iterative refinement (Jiang et al., 2026). Separately, SWE-World investigates software-agent training with learned surrogates for execution feedback (Sun et al., 2026). SCoPR draws on both directions: it combines state-preserving transitions with execution-like feedback to support chain generation, then studies repository-source composition and PR-stage horizon. This synthesis enables an empirical study of how the source and stage structure of reconstructed trajectories affect downstream training.

**Agent-authored software artifacts.** AIDev documents agent-authored PRs at scale, while Agent-Pack collects code changes co-authored by agents and humans (Li et al., 2026; Zi et al., 2025). Empirical studies also examine how agentic PRs are reviewed and integrated into software projects (Watanabe et al., 2025). These resources motivate studying agent-associated histories as a source of supervision, but authorship alone is not a quality criterion. SCoPR reconstructs trajectories from related PRs and filters their intermediate patches against reference changes. Our empirical authorship analysis uses OpenClaw as one repository-level case study; it does not establish that arbitrary agent-authored repositories will yield equally useful data.

# 3 PRELIMINARIES

## 3.1 AGENT INTERACTIONS

We describe software engineering as sequential decision making in a repository environment (Puterman, 1990). Given an initial repository state $S_0$ and a task description $L$, an agent produces an interaction trajectory

$$\tau = \big((o_j, m_j, a_j)\big)_{j=1}^{N}, \tag{1}$$

where $o_j$ is an observation, $m_j$ is the agent's reasoning or planning output, and $a_j$ is a tool action. The agent acts on the observations available through its interface, which need not expose the complete environment state. Its objective is to reach a terminal state satisfying a task-specific validation condition. Here, $N$ counts interaction steps; it is distinct from both the token length of $\tau$ and the number of development stages defined below.

## 3.2 STATE EVOLUTION AND PR-STAGE HORIZON

A staged development task consists of successive repository transformations,

$$S_0 \xrightarrow{\Delta_1} S_1 \xrightarrow{\Delta_2} \cdots \xrightarrow{\Delta_k} S_k, \tag{2}$$

where $S_t$ is the state *after* stage $t$ and $\Delta_t$ is the patch produced for its query $q_t$. Writing $\mathcal{H}_t$ for the available interaction history and transition artifacts, the induced patch-generation process is

$$\Delta_t \sim \pi_\theta(\cdot \mid q_t, S_{t-1}, \mathcal{H}_t). \tag{3}$$

The relevant dependence is operational: a later patch may extend or repair code introduced by an earlier stage, so its correctness depends on the accumulated state.

Following daVinci-Agency (Jiang et al., 2026), represent a source PR as $\mathrm{pr}_t = (x_t, \widehat{\Delta}_t)$, with natural-language context $x_t$ and reference patch $\widehat{\Delta}_t$. A chain $\mathcal{C} = (\mathrm{pr}_1, \ldots, \mathrm{pr}_k)$ links related PRs whose later stages depend on earlier changes, rather than grouping commits by chronology alone. The reference patch $\widehat{\Delta}_t$ is distinct from the rollout agent's candidate patch $\Delta_t$. We operationalize the *state-evolution horizon* as $H_{\mathrm{PR}} = k$, the number of linked PR stages. This is a proxy for cross-stage dependence, not a direct measurement of reasoning depth, token usage, or task difficulty.

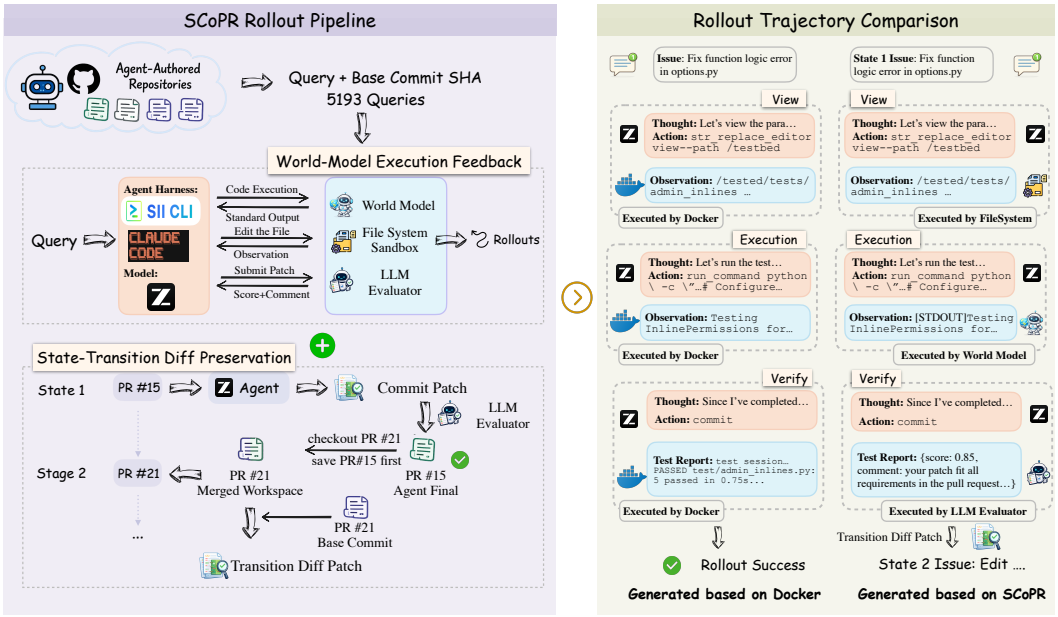


Figure 2. SCoPR rollout pipeline. File operations act on the materialized workspace, while a world model supplies provisional execution feedback when needed. A reference-guided evaluator reviews candidate patches, and transition diffs preserve information across PR stages. The illustrated score and test output are examples, not aggregate results.

# 4 SCoPR

## 4.1 Data Sources and Query Construction

Following the chain-construction approach of daVinci-Agency (Jiang et al., 2026), we process 148,655 PRs from 17 repositories and construct 5,193 task queries. Table 6 lists the source repositories and their query counts. We select related PRs to form staged development tasks in which later changes extend or refine earlier work, using the PR context to specify each local objective and the historical patch as its reference.

OpenClaw[1] is our case study of an agent-authored-data source. Its collection contains 20,469 processed PRs and 1,494 queries; we additionally collect more than 600 chains from this repository. Throughout the experiments, *OpenClaw-derived* refers to this source collection. Source authorship and trajectory generation are separate: the source PRs supply development artifacts, while a rollout agent synthesizes the training interactions for every data source.

## 4.2 State-Continuous Rollout Generation

**Workspace materialization.** For a candidate chain $\mathcal{C}$, SCoPR maintains a workspace $w_t$ associated with repository state $S_t$. It contains the file tree, patch history, repository map, dependency manifests, and a transition-diff cache. At the start of a stage, the agent receives its query, a summary of preceding stages, and relevant earlier diffs. These artifacts allow the next stage to extend or repair previous work rather than relying on a textual summary alone.

**Diff-preserving handoff.** A workspace snapshot records the current code but may obscure the edits that produced it. A checkout or workspace replacement can also discard changes needed by the next stage. SCoPR makes each transition explicit by retaining

$$\mathcal{D}_t = (\Delta_t, h_t, r_t, b_t), \tag{4}$$

[1] https://github.com/openclaw/openclaw

where $\Delta_t$ is the candidate diff, $h_t$ its header, $r_t$ the repair trace, and $b_t$ the base commit or workspace hash. During handoff, the next stage receives the materialized repository state together with relevant transition artifacts. The retained trajectory therefore records both the evolving code and the changes that connect consecutive stages. This reduces ambiguity about prior edits without assuming that a compact summary can reconstruct them.

**World-model execution feedback.** When real execution is available, the pipeline runs repository-specific commands derived from tests, continuous-integration configurations, manifests, and prior logs. When execution is unavailable or unstable, GLM-4.7 acts as an execution surrogate, following the motivation of SWE-World (Sun et al., 2026). It conditions on the candidate patch, relevant files, dependency metadata, and observed logs to predict likely failures and suggest repair directions. File inspection and editing operate on the workspace; simulated feedback supplies provisional observations for commands that cannot be executed reliably. These observations guide revision but do not certify that tests pass. Candidate patches remain subject to separate acceptance checks.

### 4.3 Reference-Guided Selection and Training Data

**Stage-level evaluation.** An evaluator compares the query, candidate patch $\Delta_t$, and reference patch $\widehat{\Delta}_t$, returning a score and actionable comments. The prompt in Section E emphasizes functional alignment, missing repair sites, and requirement violations rather than exact textual agreement. It can accept alternative implementations, although its maximum score also rewards structural alignment with the reference. Applying this evaluation at intermediate stages provides a process-aware selection signal rather than relying only on the terminal patch. The reference is privileged training-time information: acceptance is a model-based filtering decision, not proof of executable correctness.

**Rollout collection.** We use GLM-4.7 with SII-CLI and Claude Code to generate interaction trajectories (Lin et al., 2025; Anthropic, 2026). Each PR stage permits up to three attempts; if the stage remains unsuccessful, its rollout is retried up to six times. Failed attempts and retries are excluded from fine-tuning. The final collection contains 959 rollout samples. SCoPR is a training-data construction pipeline; the downstream comparisons evaluate models trained on its retained trajectories.

## 5 Experiments

### 5.1 Experimental Setup

**Models and data baselines.** We evaluate Qwen3-30B-A3B-Thinking-2507 and Step-3.5-Flash (Yang et al., 2025a; Huang et al., 2026). GLM-4.7 is the rollout and world-model backbone, distinct from these downstream models. The data comparison includes SWE-smith, CC-Bench trajectories, AFM CodeAgentDataset, and daVinci-Agency (Yang et al., 2025b; Z.ai, 2025; Li et al., 2025b; Jiang et al., 2026). These cover repository-level repair, Claude Code interactions, code question answering, and staged PR-chain supervision. The training implementations use slime and SteptronOss (Zhu et al., 2025; Huang et al., 2026). The baseline rows denote the starting checkpoints without the additional training data under study.

**Benchmarks and metrics.** We report seven metrics across six benchmarks. SWE-bench Verified measures repository-level issue resolution (Jimenez et al., 2023); FeatureBench evaluates feature-oriented development, with separate task-resolved and test-pass rates (Zhou et al., 2026). Toolathlon and the retail domain of $\tau^2$-bench measure tool-mediated task completion (Li et al., 2025a; Barres et al., 2025). DS-1000 evaluates data-science code generation (Lai et al., 2022). APTBench probes agent-relevant skills through questions derived from tasks and trajectories, rather than through end-to-end interactive execution (Qin et al., 2025). Its categories include environment setup, deep research, issue fixing, tool use, and agentic mathematics. We therefore treat its category scores as complementary diagnostics, not direct measurements of sustained agent behavior.

SWE-bench Verified uses SWE-agent as its execution scaffold (Yang et al., 2024). Toolathlon and DS-1000 follow their original evaluation settings, and $\tau^2$-bench uses GLM-4.6 as the simulated user across evaluated models. The tables retain the reported aggregate (AVG) alongside the individual

Table 1. Data comparison on Qwen3-30B-A3B. All displayed metrics are higher-is-better. SCoPR has the highest reported average and leads six metrics, with a tie on FeatureBench task resolution. AVG is the reported cross-metric summary; individual columns remain the primary evidence.

| Data | APTBench | SWE-bench Verified | FeatureBench Resolved | FeatureBench Pass Rate | $\tau^2$-bench | Toolathlon | DS-1000 | AVG |
|---|---|---|---|---|---|---|---|---|
| - | 0.456 | 0.242 | 0.00 | 0.033 | 0.579 | 0.037 | 0.370 | 0.245 |
| **SWE-smith** | 0.538 | 0.158 | 0.01 | 0.032 | 0.233 | 0.019 | 0.195 | 0.169 |
| **CC-Bench** | 0.533 | 0.200 | 0.00 | 0.036 | 0.567 | 0.000 | 0.090 | 0.204 |
| **CodeAgent** | 0.533 | 0.130 | 0.00 | 0.027 | 0.202 | 0.009 | 0.075 | 0.139 |
| **daVinci-Agency** | 0.532 | 0.262 | 0.01 | 0.047 | 0.608 | 0.046 | 0.343 | 0.264 |
| **SCoPR** | **0.545** | **0.272** | 0.01 | **0.049** | **0.610** | **0.056** | **0.377** | **0.274** |

metrics. Because the metrics measure different outcomes and FeatureBench contributes two columns, the aggregate is a descriptive summary rather than a standalone measure of long-horizon capability.

### 5.2 Main Results

SCoPR achieves the highest reported average in the Qwen3-30B-A3B data comparison, increasing it from 0.245 to 0.274 and exceeding daVinci-Agency's 0.264 shown in Table 1. It has the highest displayed score on six of the seven metrics and ties the best FeatureBench task-resolved rate. Relative to the starting checkpoint, SWE-bench Verified improves from 24.2% to 27.2%, Toolathlon from 3.7% to 5.6%, and FeatureBench test-pass rate from 3.3% to 4.9%. These are gains of 3.0, 1.9, and 1.6 percentage points, respectively. FeatureBench task resolution remains only 1%, so the pass-rate gain should not be interpreted as broad mastery of feature development.

The other data baselines improve APTBench but do not consistently improve the interactive or coding benchmarks. daVinci-Agency is the strongest comparison on the reported average, and SCoPR improves all its displayed metrics except the tied FeatureBench task-resolved rate. These results favor the complete SCoPR data configuration in the evaluated setting; they do not, by themselves, attribute the gains to either rollout mechanism in isolation.

Figure 3 breaks down APTBench performance. SCoPR performs best on `tool` and `env_setup/plan`, is competitive on `env_setup`, and improves `deepresearch` relative to the starting checkpoint. The methods differ less on `issue_fix`. This profile is consistent with transfer to some planning and tool-related skills, but the offline nature of these probes limits conclusions about actual long-horizon execution.

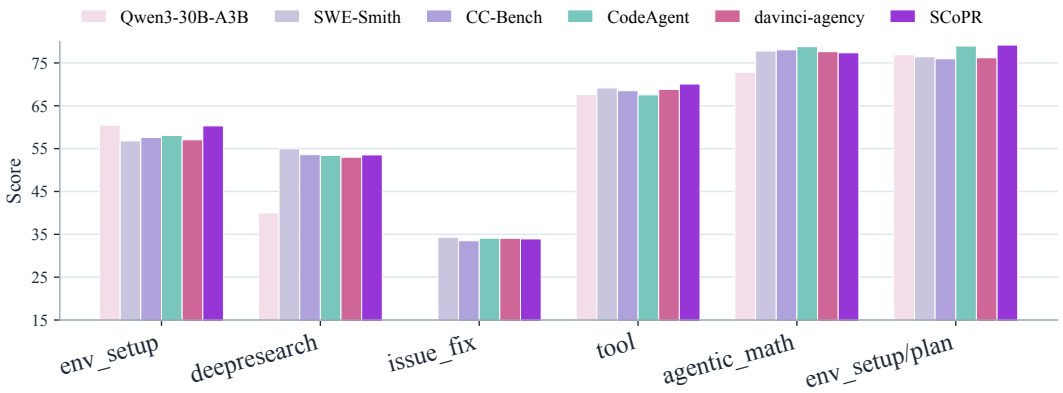


Figure 3. APTBench category scores for the Qwen3-30B-A3B data comparison. SCoPR leads on `tool` and `env_setup/plan`. These offline skill probes are distinct from end-to-end agent evaluations.

Table 2. Reported rollout acceptance rates on 30 queries spanning 210 PRs. SCoPR exceeds daVinci-Agency at each tested attempt/retry setting.

| Attempts | Retries | Success Rate | |
|---|---|---|---|
| | | daVinci-Agency | SCoPR |
| 2 | 4 | 0.067 | 0.086 |
| 3 | 4 | 0.076 | 0.100 |
| 4 | 4 | 0.076 | 0.100 |
| 3 | 3 | 0.067 | 0.095 |
| 3 | 5 | 0.071 | 0.114 |

Table 3. SWE-bench Verified performance with GLM-4.7 as the rollout model. Simulated feedback retains 87.8% of the real-execution score (0.657/0.748); this is a task-performance ratio, not execution-output agreement.

| Execution feedback | Rollout Model | SWE-bench Verified |
|---|---|---|
| Simulated (GLM-4.7) | GLM-4.7 | 0.657 |
| Real execution | GLM-4.7 | 0.748 |

Table 4. Source-composition comparison for two model families. The full SCoPR mixture has the highest reported average in both blocks, whereas OpenClaw-only training does not improve that average over either starting checkpoint. Improvements are not uniform across metrics.

| Data | APTBench | SWE-bench Verified | FeatureBench Resolved | FeatureBench Pass Rate | $\tau^2$-bench | Toolathlon | DS-1000 | AVG |
|---|---|---|---|---|---|---|---|---|
| *Qwen3-30B-A3B* | | | | | | | | |
| - | 0.456 | 0.242 | 0.00 | 0.033 | 0.579 | 0.037 | 0.370 | 0.245 |
| **SCoPR** | 0.545 | 0.272 | 0.01 | 0.049 | 0.610 | 0.056 | 0.377 | **0.274** |
| **SCoPR w/o OpenClaw** | 0.540 | 0.254 | 0.01 | 0.045 | 0.597 | 0.028 | 0.371 | 0.263 |
| **OpenClaw** | 0.533 | 0.180 | 0.00 | 0.036 | 0.566 | 0.028 | 0.365 | 0.244 |
| *Step-3.5-Flash* | | | | | | | | |
| - | 0.449 | 0.510 | 0.01 | 0.364 | 0.746 | 0.083 | 0.457 | 0.374 |
| **SCoPR** | 0.514 | 0.508 | 0.02 | 0.381 | 0.787 | 0.102 | 0.428 | **0.391** |
| **SCoPR w/o OpenClaw** | 0.517 | 0.488 | 0.02 | 0.375 | 0.780 | 0.093 | 0.424 | 0.385 |
| **OpenClaw** | 0.513 | 0.426 | 0.01 | 0.363 | 0.746 | 0.037 | 0.430 | 0.361 |

### 5.3 Rollout Acceptance and Execution Feedback

Table 2 compares rollout acceptance on 30 selected queries covering 210 PRs. SCoPR has a higher reported acceptance rate than daVinci-Agency at every tested attempt/retry setting. With three attempts and four retries, the rate increases from 0.076 to 0.100; increasing attempts to four produces no further improvement for either pipeline. With three attempts and five retries, SCoPR reaches 0.114 versus 0.071. These results support improved yield for the complete pipeline under the tested budgets, but do not separate the contributions of diff preservation and world-model feedback.

With GLM-4.7 as the rollout model, the world-model setting scores 0.657 on SWE-bench Verified, compared with 0.748 under real execution (Table 3). The former is 87.8% of the latter, with an absolute gap of 9.1 percentage points. This ratio measures downstream task performance under two feedback settings; it is *not* a measurement of agreement between simulated and real execution outputs. The comparison suggests that simulated feedback can support useful rollouts, while leaving a material gap to real execution. Direct feedback-fidelity measurements and matched-cost comparisons are needed to characterize the trade-off more fully.

### 5.4 Agent-Authored Source Composition

Table 4 compares the full mixture, the mixture without OpenClaw, and OpenClaw-only training. The full mixture has the highest reported average for both model families: 0.274 for Qwen3-30B-A3B and

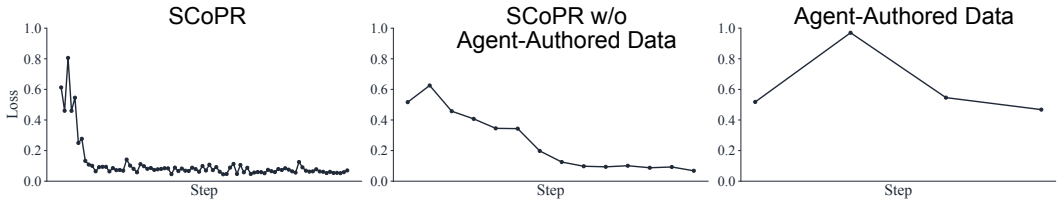


Figure 4. Reported training-loss traces for three source configurations. Loss is measured on different training distributions, so lower loss alone does not establish better data quality or generalization.

0.391 for Step-3.5-Flash, compared with 0.263 and 0.385 without OpenClaw. This pattern supports a complementary role for the OpenClaw-derived collection in the evaluated mixtures.

OpenClaw-only training, however, does *not* improve the average over the starting checkpoints: 0.244 versus 0.245 for Qwen and 0.361 versus 0.374 for Step. It improves APTBench for both models but reduces several other scores. Nor is the full mixture uniformly beneficial: Step's SWE-bench Verified score changes from 0.510 to 0.508, and its DS-1000 score decreases from 0.457 to 0.428. Thus, source complementarity in the reported aggregate is a more defensible conclusion than either standalone superiority of agent-authored data or universal cross-task improvement. The comparison also does not isolate authorship from repository content and data-mixture size.

Figure 4 shows the reported training-loss traces. The full mixture reaches a lower loss than the other configurations. Since the configurations contain different training examples, their loss values are not directly comparable measures of data quality or generalization; the downstream benchmark results provide the more relevant comparison.

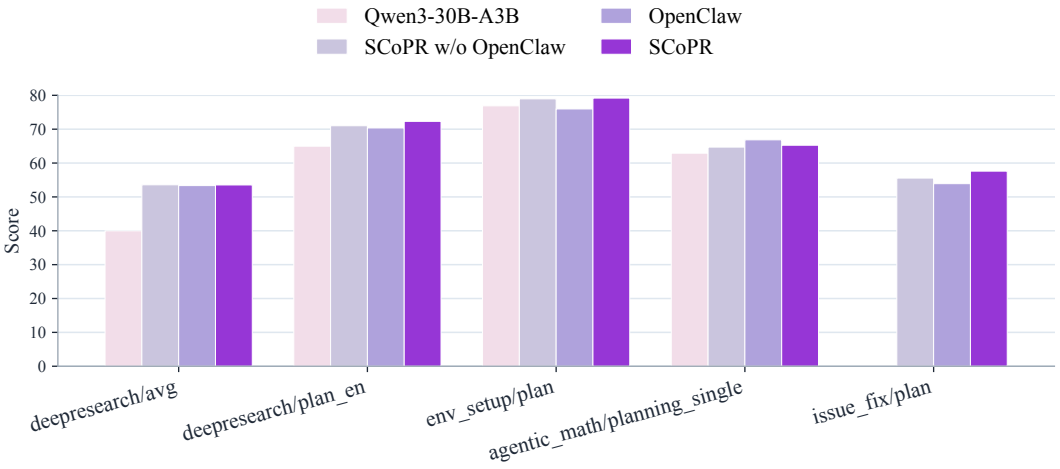


Figure 5. Source-composition effects on planning-related APTBench categories. The full mixture leads several categories; OpenClaw-only training leads `agentic_math/planning_single`.

The planning-related APTBench categories in Figure 5 likewise show a nonuniform pattern. The full mixture leads on several displayed categories, whereas OpenClaw-only training is strongest on `agentic_math/planning_single`. These diagnostic results are compatible with source-specific strengths rather than a uniform ranking of data quality.

### 5.5 Horizon Scaling at a Fixed Sample Count

We vary the PR-stage horizon while holding the number of training samples fixed at 722 (Table 5). The reported average rises monotonically from 0.237 for one PR to 0.274 for five or more PRs. Over the same range, SWE-bench Verified increases from 0.236 to 0.286, FeatureBench test-pass rate from 0.030 to 0.049, and Toolathlon from 0.009 to 0.056. APTBench changes little, and individual metrics such as DS-1000 are not monotonic across all intermediate horizons. The one- and two-PR configurations also remain below the starting checkpoint on the reported average.

Table 5. PR-stage horizon at a fixed training-sample count of 722, evaluated on Qwen3-30B-A3B. The reported average increases with horizon. Sample matching does not imply matched token exposure, compute, or task composition.

| Data horizon | Samples | APTBench | SWE-bench Verified | FeatureBench Resolved | FeatureBench Pass Rate | $\tau^2$-bench | Toolathlon | DS-1000 | AVG |
|---|---|---|---|---|---|---|---|---|---|
| - | - | 0.456 | 0.242 | 0.00 | 0.033 | 0.579 | 0.037 | 0.370 | 0.245 |
| **1 PR** | 722 | 0.540 | 0.236 | 0.00 | 0.030 | 0.479 | 0.009 | 0.366 | 0.237 |
| **2 PRs** | 722 | 0.541 | 0.252 | 0.00 | 0.036 | 0.483 | 0.019 | 0.375 | 0.244 |
| **3 PRs** | 722 | 0.540 | 0.260 | 0.00 | 0.038 | 0.513 | 0.019 | 0.372 | 0.249 |
| **4 PRs** | 722 | 0.546 | 0.266 | 0.01 | 0.044 | 0.564 | 0.037 | 0.379 | 0.264 |
| **5+ PRs** | 722 | 0.540 | 0.286 | 0.01 | 0.049 | 0.590 | 0.056 | 0.389 | 0.274 |

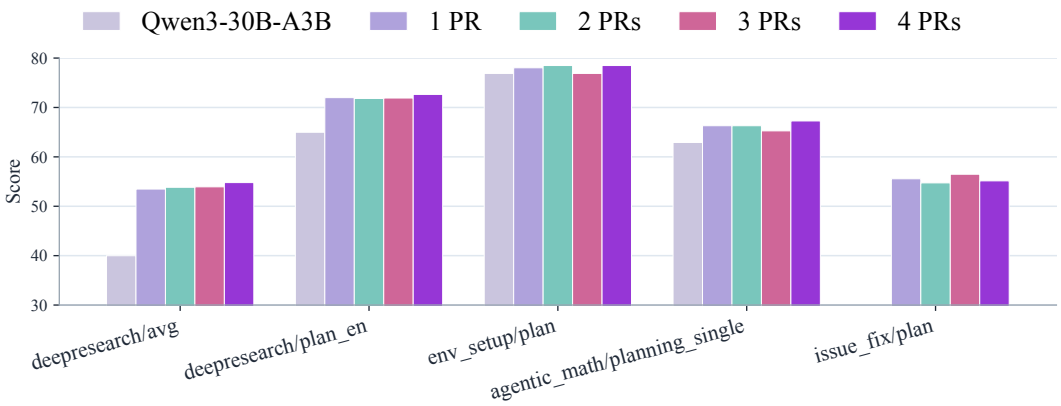


Figure 6. Planning-related APTBench scores for one through four PRs. Gains over the starting checkpoint are more consistent than differences between adjacent horizons. The five-or-more-PR setting is reported in Table 5, not shown here.

Figure 6 shows category-level APTBench results for the one- through four-PR settings. The largest contrast is often between the starting checkpoint and PR-chain training, with smaller and nonmonotonic differences among horizons. These probes therefore offer supporting context, not an independent demonstration that planning ability increases monotonically with chain length.

The horizon study controls sample count, not token count, compute, or repository/task composition. A longer chain may expose the model to more supervised tokens as well as more state transitions. Accordingly, the results establish a positive association between PR-stage horizon and several downstream scores under the reported construction protocol. They do not yet establish a token-independent scaling law or show that dependency structure alone causes the gains.

## 6 Conclusion

We present SCoPR, a data construction pipeline for scaling chain-of-PR supervision from agent-authored software artifacts. The core idea is that long-horizon training data should supervise not only final code outcomes, but also the intermediate state evolution that leads to them. To this end, SCoPR preserves state-transition diffs, provides execution-like feedback when real execution is incomplete, and applies process-aware rejection sampling to select high-quality chains. Experiments show that rejection-sampled agent-authored trajectories, while still inherently noisy, can improve over the baseline but remain insufficient to surpass other high-quality agentic datasets when used alone. In contrast, trajectory chains constructed by integrating agent-authored and human-authored repositories consistently improve post-training performance across software-engineering, tool-use, and agentic-skill benchmarks, and outperform training with purely human-authored repositories. The results further indicate that effective horizon is an important scaling dimension: under matched token budgets, longer effective trajectory chains yield stronger capability gains, suggesting SCoPR as an effective and scalable training paradigm for coding agents.

## REFERENCES


Anthropic. Claude code overview. https://code.claude.com/docs, 2026. Official Claude Code documentation. Accessed April 12, 2026.

Daman Arora, Atharv Sonwane, Nalin Wadhwa, Abhav Mehrotra, Saiteja Utpala, Ramakrishna Bairi, Aditya Kanade, and Nagarajan Natarajan. Masai: Modular architecture for software-engineering ai agents. *arXiv preprint arXiv:2406.11638*, 2024.

Victor Barres, Honghua Dong, Soham Ray, Xujie Si, and Karthik Narasimhan. $\tau^2$-bench: Evaluating conversational agents in a dual-control environment, 2025. URL https://arxiv.org/abs/2506.07982.

Xiang Deng, Jeff Da, Edwin Pan, Yannis Yiming He, Charles Ide, Kanak Garg, Niklas Lauffer, Andrew Park, Nitin Pasari, Chetan Rane, et al. Swe-bench pro: Can ai agents solve long-horizon software engineering tasks? *arXiv preprint arXiv:2509.16941*, 2025.

Dayuan Fu, Shenyu Wu, Yunze Wu, Zerui Peng, Yaxing Huang, Jie Sun, Ji Zeng, Mohan Jiang, Lin Zhang, Yukun Li, et al. davinci-env: Open swe environment synthesis at scale. *arXiv preprint arXiv:2603.13023*, 2026.

Ailin Huang, Ang Li, Aobo Kong, Bin Wang, Binxing Jiao, Bo Dong, Bojun Wang, Boyu Chen, Brian Li, Buyun Ma, et al. Step 3.5 flash: Open frontier-level intelligence with 11b active parameters. *arXiv preprint arXiv:2602.10604*, 2026.

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.

Mohan Jiang, Dayuan Fu, Junhao Shi, Ji Zeng, Weiye Si, Keyu Li, Xuefeng Li, Yang Xiao, Wenjie Li, Dequan Wang, et al. davinci-agency: Unlocking long-horizon agency data-efficiently. *arXiv preprint arXiv:2602.02619*, 2026.

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? *arXiv preprint arXiv:2310.06770*, 2023.

Yuhang Lai, Chengxi Li, Yiming Wang, Tianyi Zhang, Ruiqi Zhong, Luke Zettlemoyer, Scott Wen tau Yih, Daniel Fried, Sida Wang, and Tao Yu. Ds-1000: A natural and reliable benchmark for data science code generation, 2022. URL https://arxiv.org/abs/2211.11501.

Hao Li, Haoxiang Zhang, and Ahmed E Hassan. Aidev: studying ai coding agents on github. *arXiv preprint arXiv:2602.09185*, 2026.

Junlong Li, Wenshuo Zhao, Jian Zhao, Weihao Zeng, Haoze Wu, Xiaochen Wang, Rui Ge, Yuxuan Cao, Yuzhen Huang, Wei Liu, et al. The tool decathlon: Benchmarking language agents for diverse, realistic, and long-horizon task execution. *arXiv preprint arXiv:2510.25726*, 2025a.

Weizhen Li, Jianbo Lin, Zhuosong Jiang, Jingyi Cao, Xinpeng Liu, Jiayu Zhang, Zhenqiang Huang, Qianben Chen, Weichen Sun, Qiexiang Wang, Hongxuan Lu, Tianrui Qin, Chenghao Zhu, Yi Yao, Shuying Fan, Xiaowan Li, Tiannan Wang, Pai Liu, King Zhu, He Zhu, Dingfeng Shi, Piaohong Wang, Yeyi Guan, Xiangru Tang, Minghao Liu, Yuchen Eleanor Jiang, Jian Yang, Jiaheng Liu, Ge Zhang, and Wangchunshu Zhou. Chain-of-agents: End-to-end agent foundation models via multi-agent distillation and agentic rl, 2025b. URL https://arxiv.org/abs/2508.13167.

Jifan Lin, Xiaojie Cai, Yang Xiao, Yumin Zhuang, Qishuo Hua, Junfei Wang, and Pengfei Liu. Sii cli: Building next-generation cognitive agentic intelligence ecosystem. https://github.com/GAIR-NLP/SII-CLI, 2025. Technical Report on Cognitive Agentic Intelligence Framework.

Martin L Puterman. Markov decision processes. *Handbooks in operations research and management science*, 2:331–434, 1990.

Jiarui Qin, Yunjia Xi, Junjie Huang, Renting Rui, Di Yin, Weiwen Liu, Yong Yu, Weinan Zhang, and Xing Sun. Aptbench: Benchmarking agentic potential of base llms during pre-training. *arXiv preprint arXiv:2510.24397*, 2025.

Shuang Sun, Huatong Song, Lisheng Huang, Jinhao Jiang, Ran Le, Zhihao Lv, Zongchao Chen, Yiwen Hu, Wenyang Luo, Wayne Xin Zhao, et al. Swe-world: Building software engineering agents in docker-free environments. *arXiv preprint arXiv:2602.03419*, 2026.

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

Xingyao Wang, Boxuan Li, Yufan Song, Frank F Xu, Xiangru Tang, Mingchen Zhuge, Jiayi Pan, Yueqi Song, Bowen Li, Jaskirat Singh, et al. Openhands: An open platform for ai software developers as generalist agents. *arXiv preprint arXiv:2407.16741*, 2024.

Miku Watanabe, Hao Li, Yutaro Kashiwa, Brittany Reid, Hajimu Iida, and Ahmed E Hassan. On the use of agentic coding: An empirical study of pull requests on github. *ACM Transactions on Software Engineering and Methodology*, 2025.

Chunqiu Steven Xia, Yinlin Deng, Soren Dunn, and Lingming Zhang. Agentless: Demystifying llm-based software engineering agents. *arXiv preprint arXiv:2407.01489*, 2024.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, et al. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025a.

John Yang, Carlos E Jimenez, Alexander Wettig, Kilian Lieret, Shunyu Yao, Karthik Narasimhan, and Ofir Press. Swe-agent: Agent-computer interfaces enable automated software engineering. *Advances in Neural Information Processing Systems*, 37:50528–50652, 2024.

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*, 2025b.

Z.ai. CC-Bench trajectories. Hugging Face dataset, 2025. URL `https://huggingface.co/datasets/zai-org/CC-Bench-trajectories`. Dataset card.

Yuntong Zhang, Haifeng Ruan, Zhiyu Fan, and Abhik Roychoudhury. Autocoderover: Autonomous program improvement. In *Proceedings of the 33rd ACM SIGSOFT International Symposium on Software Testing and Analysis*, pp. 1592–1604, 2024.

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

Zilin Zhu, Chengxing Xie, Xin Lv, and slime Contributors. slime: An llm post-training framework for rl scaling. `https://github.com/THUDM/slime`, 2025. GitHub repository. Corresponding author: Xin Lv.

Yangtian Zi, Zixuan Wu, Aleksander Boruch-Gruszecki, Jonathan Bell, and Arjun Guha. Agentpack: A dataset of code changes, co-authored by agents and humans. *arXiv preprint arXiv:2509.21891*, 2025.

## A LIMITATIONS

**Scope and attribution.** The agent-authored-source analysis is limited to OpenClaw. A repository-level source label does not establish exclusive agent authorship for every PR, and differences between source collections can reflect domain, task difficulty, or contributor practices as well as authorship. Our results support complementarity of this collection in the tested mixtures, not a general ranking of human- and agent-authored software data.

**Horizon and experimental controls.** PR count is an operational proxy for state-evolution horizon. The fixed-sample study does not match token exposure, compute, or task composition. It therefore cannot establish an independent scaling law for dependency depth. The reported point estimates also do not quantify variation across training seeds or evaluation runs. Such uncertainty is especially important for benchmarks with low task-resolution rates.

**Selection and verification.** Reference-guided filtering may favor changes whose correctness is easy for a judge to assess, while underrepresenting refactoring, performance optimization, maintainability, and security-related objectives. The supplied scoring prompt also rewards structural alignment at its maximum score. World-model observations can be incorrect, and downstream performance retention does not measure their fidelity. Independent execution checks, judge calibration, and separate ablations of filtering, diff preservation, and simulated feedback are needed to establish where each mechanism is effective.

## B BROADER IMPACT

SCoPR aims to improve training data for software-engineering agents. More capable agents could reduce repetitive development work, but generated patches and simulated execution feedback should not substitute for appropriate testing and human review. Reusing repository histories also requires attention to provenance, license terms, and sensitive information in code or logs. We do not infer deployment safety or reliability from the benchmark improvements reported here.

## C SOURCE REPOSITORIES

Table 6 lists all 17 source repositories. Their histories contain 148,655 processed PRs, from which we construct 5,193 task queries. The collection spans several software-engineering domains and includes OpenClaw for the source-composition study. These are source-artifact and query counts, not counts of accepted training rollouts.

## D COMPUTE RESOURCES

Rollout generation uses 144 NVIDIA H200 GPUs for approximately 10 days. Training uses 64 H200 GPUs, and deployment and experimental evaluation use 32 H200 GPUs. These hardware allocations describe the experimental setup; they do not establish equal training or generation cost across data configurations.

Table 6. Source repositories, processed PRs, and constructed queries. Query counts are not accepted-rollout counts.

| Repo ID | Repo Name | PR Num | Query Num |
|---|---|---|---|
| 543276238 | tursodatabase_libsql | 1261 | 30 |
| 680120071 | microsoft_autogen | 1467 | 18 |
| 313853293 | cloudquery/cloudquery | 9705 | 490 |
| 62117812 | apache/pulsar | 9803 | 228 |
| 1460385 | scipy/scipy | 9511 | 238 |
| 7370241 | RIOT-OS/RIOT | 9596 | 569 |
| 309512982 | redpanda-data/redpanda | 9915 | 143 |
| 208728772 | milvus-io/milvus | 9766 | 33 |
| 8162715 | saleor/saleor | 7810 | 526 |
| 139199684 | PrefectHQ/prefect | 8433 | 215 |
| 908607 | numpy/numpy | 9355 | 354 |
| 70746484 | apache/tvm | 9378 | 176 |
| 523043277 | astral-sh/ruff | 8115 | 180 |
| 640534 | sympy/sympy | 8760 | 261 |
| 66823715 | intel/ipex-llm | 7590 | 17 |
| 34864402 | apache/mxnet | 7721 | 221 |
| 1103012935 | openclaw/openclaw | 20469 | 1494 |
| SUM | | 148655 | 5193 |

## E Evaluator Prompt

The following prompt specifies the stage-level patch-review rubric used during rejection sampling. It compares a candidate patch with the reference patch and task description, returning a score and repair comments. It prioritizes functional alignment rather than exact textual identity, while reserving the maximum score for candidates that also align structurally with the reference. We reproduce the experimental prompt unchanged.

**Evaluator Prompt**

# Code Model Judge Prompt
Return JSON only with exactly these keys and no extra prose:

```
{
  "score": 0.0,
  "comment": "Overall review summary\n...\nTop findings\n1.
  P1 `[path:line]` ...\n2. P2 `[unknown]` ...\nFix path\n1.
  ...\n2. ..."
}
```

Do not add analysis, headings outside `comment`, markdown fences, or any text before or after the JSON object.
You are an independent benchmark judge comparing a candidate patch against the official gold patch for the same SWE-bench issue.
Measure likely issue resolution, not line-by-line similarity. Different implementation shapes can still be correct when they land the same user-visible behavior safely. Your role is still benchmark measurement first, but the `comment` field must preserve the highest-signal reviewer guidance that a retrying agent can act on immediately.
## Inputs
**Gold patch**

```
{{ground_truth_patch}}
```

**Candidate patch**

```
{{diff_patch}}
```

**Issue statement**

`{{test_query}}`

## Review Principles

- Use the gold patch as a strong reference for visible repair intent, decisive fix scope, and any clearly necessary companion update. - Focus the `comment` on the highest-signal mismatches that still prevent the candidate patch from matching the gold patch's functional effect. - Use the candidate patch and gold patch as the primary source of truth for the next-try plan. If those patches expose the target file, hunk, function, method, branch, import block, or other logic block, carry those anchors into the comment instead of collapsing to a generic one-sentence diagnosis. - When the intended repair can be inferred reliably, explicitly name the file that the next attempt should modify and which function, method, or logic block in that file should change. - If the exact function or method name cannot be inferred reliably but the file can, name the surrounding logic block or code region instead of falling back to `[unknown]`. - Prefer file/line locations from the candidate patch when they can be inferred reliably. Use `path:line` with a 1-based line number whenever the edited line can be inferred from the diff. - Report at most 2 findings.

## Scoring Rules

- `score` must be a floating-point number between `0.0` and `1.0`. - Use the gold patch as a strong reference for visible repair intent, decisive fix scope, and any clearly necessary companion update. - Give a high score only when the candidate visibly lands the decisive production-code fix or a safe equivalent for the reported issue. - Keep the score below `0.85` when any of these are true: - the gold patch shows multiple decisive fix clusters, repeated repair sites, or a clearly necessary companion update, and the candidate appears to miss one of them - the candidate edits a nearby but wrong surface - the candidate changes only tests or docs while the issue clearly needs product-code changes - the candidate mixes a plausible fix with broad unrelated churn, debug artifacts, or noisy refactors that make the decisive fix uncertain - Use the `0.75` to `0.84` band only when the candidate still looks likely to resolve the issue overall and the remaining differences look non-blocking. - If you can point to a concrete missing repair site, wrong condition, wrong field/value, omitted validator/branch, hardcoded substitute, or extra behavior that may change semantics, keep the score at `0.60` or below rather than `0.80+`. - Use a medium score for meaningful but incomplete, narrow, or ambiguous fixes that likely help but do not clearly close the issue. - Use a low score for empty, malformed, reversed, unrelated, wrong-surface, or clearly non-resolving patches. - When uncertain between medium and high, choose medium. - Do not over-penalize harmless formatting, naming, helper extraction, or safe local restructuring if the same behavior still lands. - Penalize obvious logic bugs, broken API contract changes, or missing required behavior from the issue statement. - Use `1.0` only when the candidate patch is both behaviorally correct and materially aligned with the gold patch in structure and local edit shape.

## Comment Format Rules

If there are blocking or meaningful mismatches, `comment` must be a single JSON string with this exact textual structure:

```
Overall review summary
One or two sentences that name the single most important
failure mode and why the patch is still not aligned.
Top findings
1. P1 `[path:line]` concise issue statement and why it blocks
alignment.
2. P2 `[path:line]` concise issue statement and why it still
diverges.
Fix path
1. First minimal repair action.
2. Second minimal repair action.
```

Rules: - Start with the exact heading `Overall review summary`, followed by 1-2 sentences. - Then use the exact heading `Top findings`. - Then use the exact heading `Fix path`. - Each finding must contain severity, a backticked location token, and one concise ac-

tionable sentence that explains why the issue still blocks alignment. - Each `Fix path` action must repeat the exact file to edit, even if both actions stay in the same file. - Each `Fix path` action must say what concrete code change to make next in that file, not just that additional work is needed. - When the patch material makes the target inferable, each action must answer all three questions: which file to change, which function or logic block in that file to change, and what concrete edit to make there next. - Action 1 must name the highest-leverage primary anchor edit. - Action 2 must name the coordinating or close-out edit. - Prefer imperative edit verbs in `Fix path`, such as `restore`, `replace`, `keep`, `remove`, or `apply`. - Do not use generic actions such as `apply remaining corrections`, `complete the patch`, `mirror the ground truth`, `re-check requirements`, or `update other files`. - If the candidate patch is empty, malformed, reversed, edits the wrong file, or misses the target line entirely, say so explicitly in both `Overall review summary` and `Top findings`, then make `Fix path` start with the minimal corrective action. - Do not add any prose before the heading or after the last finding.
If the candidate patch is acceptable, `comment` must be exactly:

```
No blocking correctness issues found relative to the ground truth patch.
```

## Output Rules
- Return JSON only. - `comment` must preserve the exact textual structure above when there are meaningful mismatches. - Do not emit extra keys.