# When Agents go Astray: Course-Correcting SWE Agents with PRMs

**Anonymous Author(s)**
Affiliation
Address
email

## Abstract

Large Language Model (LLM) agents are increasingly deployed for complex, multi-step software engineering (SWE) tasks. However, their trajectories often contain costly inefficiencies, such as redundant exploration, looping, and failure to terminate once a solution is reached. Prior work has largely treated these errors in a post-hoc manner, diagnosing failures only after execution. In this paper, we introduce `SWE-PRM`, an inference-time Process Reward Model (PRM) that intervenes during execution to detect and course-correct trajectory-level errors. Our PRM design leverages a taxonomy of common inefficiencies and delivers lightweight, interpretable feedback without modifying the underlying policy. On SWE-bench Verified, closed-source PRMs improve resolution from 40.0% to 50.6% (+10.6 p.p.), with the largest gains on medium and hard tasks. Among feedback strategies, taxonomy-guided PRMs outperform unguided or explicit action-prescriptive variants, increasing success rate while reducing trajectory length. These benefits come at an acceptable added inference cost of as low as $0.2, making PRMs a practical and scalable mechanism for improving SWE agents' reliability and efficiency.

## 1 Introduction

Large Language Model (LLM)-based agents are increasingly deployed for complex, multi-step software engineering (SWE) tasks, such as repository-level bug fixing and feature implementation [10, 28, 18, 13, 8, 5]. While recent advances have improved benchmark resolution rates, these gains often mask hidden inefficiencies in the agent's execution process. In particular, *trajectory-level errors*, i.e. patterns such as action looping, redundant backtracking, or drifting toward irrelevant subgoals, can accumulate over a run. On top of yielding incorrect actions, these behaviors also waste compute, inflate latency, and risk exhausting the agent's budget before task completion.

Prior work on SWE agents has largely focused on maximizing *success rate* without explicitly addressing process efficiency. For example, systems such as SWE-smith [25], SWE-gym [16], and R2E-gym [9] train an open source model to reduce inference cost, but high success rates do not guarantee low-cost, efficient execution. This gap is particularly significant because trajectory-level inefficiencies have been documented for SWE tasks [6] and noted in other sequential decision-making domains [3], suggesting that a mitigation strategy like ours could generalize beyond SWE.

Existing approaches for handling trajectory-level errors focus on *post-mortem* analysis. For example, TRAIL [6] and MAST [3] rely on dumping the entire trajectory to an LLM judge for error analysis after execution. While useful for research diagnostics, these methods are impractical in deployment: they incur substantial context-length overhead, require expensive iterative re-judging, and cannot prevent wasted computation that has already occurred. In practice, the iterative cycle often involves a human analyst reviewing error reports and manually adjusting prompts, heuristics, or control logic

Submitted to 39th Conference on Neural Information Processing Systems (NeurIPS 2025). Do not distribute.

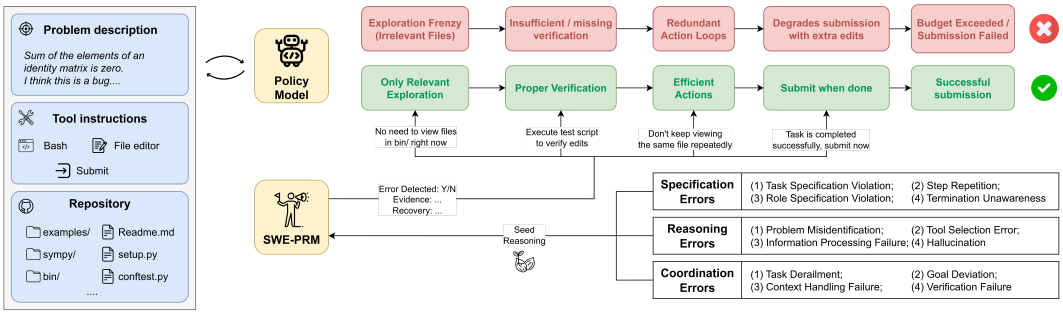



Figure 1: `SWE-PRM` helps mitigate trajectory-level suboptimalities in SWE agents.

between runs. This is fundamentally different from our setting, where the base agent remains fixed during execution, and intervention is applied only through lightweight, inference-time guidance.

Other strategies for guiding agent behaviour also have limitations. *Outcome Reward Models* (ORMs) focus solely on evaluating final solutions for correctness, ignoring process optimality and therefore missing costly but non-terminal inefficiencies [14]. Some methods use *Process Reward Models* (PRM) within Monte-Carlo Tree Search (MCTS) to score multiple future rollouts per step [1]; however, for SWE agents this is prohibitively expensive. Code-editing actions are often irreversible, making it infeasible to spin up parallel environment instances or reset to arbitrary intermediate states without high overhead.

In this work, we propose an *inference-time PRM*, `SWE-PRM`, that **prevents**, **detects**, and **course-corrects** trajectory-level errors *during* execution. The PRM is invoked periodically with a limited sliding window of past steps and is guided by a taxonomy of common error patterns. It issues actionable feedback that can be applied immediately, steering the agent back toward efficient completion without modifying its core architecture or parameters. To the best of our knowledge, this is the *first* application of PRMs for real-time trajectory-level error correction in SWE agents. Our design offers three advantages: (1) **real-time mitigation** of errors before they propagate, (2) **cost-efficiency** through sparse, targeted PRM calls, and (3) **modularity** for integration with both open-weight and proprietary LLMs, making it potentially transferable to other domains where similar inefficiencies have been observed.

We evaluate `SWE-PRM` on the SWE-bench Verified benchmark using SWE-AGENT-LM-32B, a finetuned QWEN2.5-CODER-32B-INSTRUCT model as the policy model [25]. We compare open-weight and frontier models as PRMs, with and without taxonomy guidance. Our results show that a strong PRM significantly improves resolution rate and cost-effectiveness over both a base SWE-agent and post-hoc analysis baselines, with consistent gains across all categories: easy, medium, and hard instances. Concretely, our experiments show that a taxonomy-guided PRM improves resolution from 40.0% to 50.6% on SWE-bench Verified, including +10.7 points on medium and +4.4 points on hard tasks. These gains come with shorter or comparable trajectories, translating into more efficient runs. While PRM guidance adds inference cost, the additional spend amounts to roughly $0.2 per extra resolved instance, highlighting PRMs as an attractive tradeoff between accuracy and efficiency in long-horizon SWE agents.

# 2 Related Work

## 2.1 Repository-Level Code Generation

Repository-level software engineering benchmarks have driven much of the recent progress in code agents. SWE-bench [10] provides realistic bug-fixing and feature implementation tasks from open-source repositories, with deterministic evaluation for correctness. Since SWE-bench, several new benchmarks have emerged to broaden repository-level evaluation: Multi-SWE-bench [28] extends issue-resolving tasks to multiple programming languages, SWE-PolyBench [18] introduces multi-language tasks with syntax tree analysis-based metrics, FEA-Bench [13] focuses on repository-level feature implementation, RefactorBench [8] targets multi-file refactoring, and NoCode-bench [5] evaluates natural language-driven feature addition. SWE-gym [16] offers a training and evaluation framework for coding agents and verifiers, while R2E-gym [9] introduces procedural environments with hybrid verifiers to facilitate scaling open-weight agents. However, these benchmarks and frameworks primarily aim to improve final resolution rates and do not directly address execution efficiency or trajectory-level inefficiencies, which is the focus of our approach. In an effort to replace frontier models and achieve good performance with open-source models, SWE-Smith [25] scales data generation for code agents and releases `SWE-agent-LM-32B`, the policy model we use in our experiments (a finetuned version of `Qwen2.5-Coder-32B-Instruct`).

### 2.2 Improving LLM Agents

A number of works have sought to improve the performance, robustness, and reasoning quality of LLM agents.

**Error analysis and taxonomies.** Deshpande et al. [6] introduces a comprehensive taxonomy of reasoning, execution, and planning errors in SWE agents, with human-annotated traces from SWE-bench and GAIA. Cemri et al. [3] proposes a taxonomy for multi-agent LLM systems, emphasizing coordination and reasoning failures. Both rely on post-mortem trajectory dumps to an LLM judge, often combined with human review, which limits their ability to prevent wasted computation during execution. Chen et al. [4] similarly analyses common failure modes of code agents on real-world GitHub issues, while Sung et al. [21] proposes VeriLA, a human-aligned verification framework for making agent failures more interpretable. These works highlight the need for systematic, taxonomy-guided diagnostics, but remain primarily retrospective.

**Search-based improvements.** Antoniades et al. [2] integrate Monte Carlo Tree Search (MCTS) with self-assessment to explore multiple candidate solution paths in SWE agents, yielding substantial performance gains without additional model training. Zainullina et al. [27] address search in non-serializable environments by introducing one-step lookahead and trajectory selection policies guided by learned action-value estimators, achieving improved results on SWE-bench Verified. While effective, these methods can be costly for long-horizon, irreversible settings such as repository-level code editing.

**Process optimization and recovery.** BacktrackAgent [23] introduces explicit verification, judgment, and reflection mechanisms to detect errors and revert to earlier states in GUI agents. Song et al. [19] propose exploration-based trajectory optimization that learns from failed attempts to avoid repeating mistakes. SMART [17] targets tool overuse mitigation by training agents to balance tool calls with internal reasoning, reducing unnecessary invocations while maintaining or improving performance. These approaches demonstrate the value of inference-time self-correction, though often in domains other than repository-level SWE.

**Reward models for agent improvement.** Reward modeling has been used to guide agents toward better intermediate decisions across various domains. Outcome Reward Models (ORMs) prioritize final outcome correctness in a task's result—for example, ensuring a patched program passes all tests in repository-level bug fixing [15, 16]. In contrast, Process Reward Models (PRMs) evaluate each intermediate step's quality in multi-step reasoning tasks, offering finer-grained feedback signals [11, 20, 12]. CodePRM [12] integrates execution feedback into step-level scoring for code generation, improving correctness when paired with a generate-verify-refine loop. FreePRM [20] trains PRMs without step-level labels, using pseudo-rewards inferred from final outcomes. STeCa [22] calibrates trajectories at the step level by replacing suboptimal actions with improved alternatives via LLM self-reflection. ThinkPRM [11] augments PRMs with their own reasoning chains, outperforming discriminative baselines with far less data.

While PRMs have been embedded into expensive search procedures such as MCTS, such integration is computationally prohibitive for SWE agents due to costly environment resets. To the best of our knowledge, our work is the first to apply a PRM for *real-time* trajectory-level error *prevention, detection, and course-correction* in SWE agents, using taxonomy-guided, inference-time feedback without modifying the base policy model.

# 3 Methodology

## 3.1 Task and Architecture

We study repository-level issue resolution [10]: given a natural language problem description $d$, a set of tool instructions $i$, and a snapshot of a repository $\mathcal{R}$, the agent must propose a patch $\hat{p}$ that satisfies the repository's test suite $\mathcal{S}$. The suite contains two subsets: $\mathcal{S}_{pp}$ (*pass-to-pass*) tests that must remain successful to preserve existing functionality, and $\mathcal{S}_{fp}$ (*fail-to-pass*) tests that must transition from failing to passing to confirm the requested change. A patch $\hat{p}$ is accepted iff

$$\forall\, \sigma \in \mathcal{S}_{pp},\ \ \sigma(\hat{p}(\mathcal{R})) = \texttt{pass} \quad \text{and} \quad \forall\, \sigma \in \mathcal{S}_{fp},\ \ \sigma(\hat{p}(\mathcal{R})) = \texttt{pass}.$$

The base agent follows the SWE-agent framework [24], running a ReAct-style loop [26] that records an explicit transcript of reasoning and interactions. At step $t$, the transcript is

$$\mathcal{H}_t \ = \ \big(u_1, a_1, o_1,\, u_2, a_2, o_2,\, \ldots,\, u_t, a_t, o_t\big),$$

where $u_i$ are the model's *thoughts* (free-form reasoning), $a_i$ are *actions* (tool calls), and $o_i$ are the resulting *observations* (e.g., file contents, diffs, or execution outputs). The policy $\pi_\theta$ conditions on $\mathcal{H}_t$ to generate the next thought and action, $(u_{t+1}, a_{t+1}) \sim \pi_\theta(\cdot \mid \mathcal{H}_t, d, i)$. Executing $a_{t+1}$ yields $o_{t+1}$, which is appended back to the transcript. This process is strictly sequential and continues until the agent submits a patch or reaches its step budget.

The action space is designed to simulate repository-level software engineering. The agent can (i) execute shell commands with `bash`, (ii) view or edit files through a persistent `str_replace_editor` that supports browsing paths, inserting or replacing code, creating new files, and undoing edits, and (iii) finalize its work with a `submit` action. Upon submission, the patch is evaluated in a fresh, isolated environment.

## 3.2 PRM as Course-Corrector

Process Reward Models (PRMs) are introduced as lightweight *course-correctors* within the agent's reasoning loop. Rather than replacing the base policy or dictating procedural changes, the PRM interjects periodically with natural language guidance aimed at steering the trajectory towards the next optimal action. This guidance is (1) in natural-language with demarcated sections based on taxonomy, and (2) grounded in the current context $H_t$, for the policy model to incorporate into its own reasoning.

### 3.2.1 Motivation and Taxonomy

Long-horizon software engineering agents frequently accumulate *trajectory-level inefficiencies*, patterns of reasoning and action that may not yield immediate incorrectness but gradually erode efficiency and task success. Prior work such as Trail [6] and MAST [3] introduced taxonomies of such inefficiencies, but mainly as *post-mortem analysis tools*, applied after execution to explain failure. In contrast, we operationalize inefficiency categories *during execution*, enabling a Process Reward Model (PRM) to deliver corrective natural language guidance in real time. This distinction is especially crucial in repository-level code editing on SWE-bench [10], where agents such as SWE-agent [24] often require dozens of dependent steps and small inefficiencies can compound into wasted effort or cascading failures.

The taxonomy itself is domain-general, reflecting common patterns of inefficiency that arise in long-horizon agentic reasoning. We validate it in the SWE setting since it provides a natural stress test, but the categories are broadly applicable across other domains where agents plan, reason, and act over extended horizons. The taxonomy was seeded in manual inspection of execution traces and emphasizes not only the *failure mode* but also a corresponding *recovery action*. It is organized into three families:

**Specification Errors (violations of task setup).** *Task specification violations* (ignoring explicit requirements), *role specification violations* (acting outside intended scope), *step repetition* (re-executing completed actions), and *termination unawareness* (continuing after completion criteria are met).

**Reasoning Errors (decision-making failures).** *Problem misidentification* (misunderstanding the subtask), *tool selection errors* (choosing inappropriate tools), *hallucinations* (fabricating results), and *information processing failures* (retrieving or interpreting evidence incorrectly).

**Coordination Errors (multi-step process management failures).** *Task derailment* (macro-level drift, abandoning the main task), *goal deviation* (micro-level misalignment, pursuing secondary or irrelevant subgoals), *context handling failures* (forgetting prior results), and *verification failures* (neglecting to check correctness or quality).

Each category is formally defined and paired with a corresponding recovery action, ensuring that inefficiency detection translates into actionable supervisory guidance rather than generic critique. For example, in the case of *task specification violation*, the prescribed recovery action is to redirect the agent to original task requirements. Full category definitions and recovery mappings are provided in Appendix A.1.

#### 3.2.2 Guidance Generation

At fixed intervals, the PRM is invoked to provide course-corrective feedback. Every $n$ steps, it receives as input: (i) the original problem description $d$, and (ii) the most recent $k$ steps of the agent's transcript

$$\mathcal{H}_t^{(k)} = \big(u_{t-k+1}, a_{t-k+1}, o_{t-k+1}, \ldots, u_t, a_t, o_t\big),$$

where $u_i$ are *thoughts*, $a_i$ are *actions*, and $o_i$ are the corresponding *observations*. These elements are serialized into a structured text prompt:

$$x_t = \text{serialize}(d, \mathcal{H}_t^{(k)}).$$

The PRM then produces natural language feedback

$$g_t = f_\phi(x_t, \mathcal{T}),$$

where $\mathcal{T}$ is the taxonomy of inefficiencies described in Section 3.2. The taxonomy anchors the reasoning of the PRM: guidance is framed in terms of specific inefficiency categories (e.g., looping, redundant backtracking, subgoal drift), rather than unconstrained critique. Importantly, $g_t$ is expressed in natural language that the policy model can readily integrate into its own reasoning process.

#### 3.2.3 Variants

We study different variants of `SWE-PRM` integration where the PRM provides natural language guidance to the policy model. Appendix A.1 lists the prompts corresponding to each variant. In the unified setting, where the PRM and the policy are instantiated by the same model, we vary three axes: (i) conciseness of feedback (Concise vs. Detailed), (ii) inclusion of an illustrative example (Example vs. No Example), and (iii) whether the PRM's reasoning (taxonomy-based error analysis) is provided to the policy model alongside the overall guidance (Guidance+Reasoning vs. Guidance-only). This yields the set of conditions shown in Table 1. We take $\texttt{SWE-PRM}_D$ (taxonomy-guided, detailed, with example, guidance+reasoning) as the canonical variant, since it is the richest form of feedback and aligns most directly with the intended role of a PRM. Moreover, we also study a simple PRM variant that utilizes the model's inherent understanding of trajectory-level errors, i.e. $\texttt{SWE-PRM}_S$, along with explicitly stating the next action to be taken by the policy model as part of the PRM's guidance $\texttt{SWE-PRM}_{DR}$.

In addition, we evaluate a subset of these settings with an expert PRM, where a stronger closed-source model provides guidance to a weaker open-source policy model. Specifically, we consider $\texttt{SWE-PRM}_S$, $\texttt{SWE-PRM}_D$, and $\texttt{SWE-PRM}_{DR}$, which capture the key baselines. We restrict the grid here due to the high cost of expert PRM queries, focusing on the most informative comparisons while keeping experiments tractable.

# 4 Experimental Setup

## 4.1 Dataset

We evaluate the proposed framework on SWE-BENCH VERIFIED [10], a subset of SWE-BENCH that has been verified by human annotators. As explained in Section 3.1, the task involves repository-level

Table 1: SWE-PRM variants. 'Simple' involves using the model's inherent understanding of trajectory-level errors as opposed to seeding the reasoning with the taxonomy.

| Name | Feedback Style | Example | Policy Input | Action Reco. |
|---|---|---|---|---|
| SWE-PRM$_S$ | Simple | – | Guidance+Reasoning | × |
| SWE-PRM$_C$ | Concise | ✓ | Guidance+Reasoning | × |
| SWE-PRM$_{CG}$ | Concise | ✓ | Guidance-only | × |
| SWE-PRM$_D$ | Detailed | ✓ | Guidance+Reasoning | × |
| SWE-PRM$_{DN}$ | Detailed | × | Guidance+Reasoning | × |
| SWE-PRM$_{DG}$ | Detailed | ✓ | Guidance-only | × |
| SWE-PRM$_{DNG}$ | Detailed | × | Guidance-only | × |
| SWE-PRM$_{DR}$ | Detailed | ✓ | Guidance+Reasoning | ✓ |

bug fixing with long-horizon multi-step reasoning. The benchmark contains 500 instances paired with validated ground-truth patches. Unlike synthetic tasks, these instances reflect the complexity of real-world software engineering. The dataset serves as a standardized testbed for both baseline policies and PRM-supervised variants.

### 4.2 Models and Hyperparameters

We evaluate both open-source and proprietary models. Our experiments include three representative baselines for open-weights models: SWE-AGENT-LM-32B [1], DEVSTRAL-SMALL-2505 [2], and DEVSTRAL-SMALL-2507 [3], along with CLAUDE-SONNET-4. The `temperature` was set to $0.0$ for deterministic outputs for all models and the `top_p` was set to $1.0$. For all experiments, we run the agent for a maximum of 75 steps, after which the run is auto-terminated and if a patch is generated, it is auto-submitted. For PRM-guided runs, we pass $k = 8$ most recent steps and the PRM is invoked every $n = 5$ steps. These hyperparameters balance contextual coverage with computational overhead and are fixed across all reported experiments. Two NVIDIA A100 GPUs were used to serve the models.

### 4.3 Evaluation Metrics

**Resolution Rate.** The % of instances correctly solved, both the overall rate and breakdowns by difficulty [7]: (1) Easy ($\leq$15 minutes for human developers; 194 instances, 38.8% of total), (2) Medium (15–60 minutes; 261 instances, 52.2% of total), and (3) Hard ($\geq$1 hour; 45 instances, 9.0% of total). This stratification highlights whether improvements generalize beyond the easiest cases.

**Patch Generation Rate.** The frequency with which a candidate patch is produced before the agent terminates, irrespective of correctness. This includes both, the patches submitted directly by the agent using the `submit` action, as well as auto-submissions in case of termination.

**Average Steps.** The average number of steps taken by the policy model per trajectory.

**Cost.** We report monetary cost in $ per 100 instances, including the cost of running the policy model as well as the PRM interventions. For open source models, we consider API pricing from GPU cloud platforms [4] as of July 2025. ($0.08 per million tokens). For the closed source model, CLAUDE-SONNET-4, we consider API pricing as of July 2025 ($ 3 and $ 15 per million tokens for input and output respectively).

[1] `https://huggingface.co/SWE-bench/SWE-agent-LM-32B`
[2] `https://huggingface.co/mistralai/Devstral-Small-2505`
[3] `https://huggingface.co/mistralai/Devstral-Small-2507`
[4] `https://www.together.ai/`

Table 2: Open-Source SWE-PRM variations: SWE-PRM is same as policy model. $\Delta$s in brackets compare to the corresponding base row for each policy. Resolution rate $\Delta$s: green = higher is better. Steps $\Delta$s: green = lower is better. Numbers in **bold** are best for that model.

| Setting | Policy Model | Resolution Rate (%) | Patch Generation Rate (%) | Avg Steps | Total Cost ($) per 100 instances |
|---|---|---|---|---|---|
| base | SWE-AGENT-LM-32B | **40.0** | 92.4 | 38.64 | 2.77 |
| | DEVSTRAL-SMALL-2505 | 34.0 | 92.6 | 37.97 | 2.69 |
| | DEVSTRAL-SMALL-2507 | 30.0 | 88.0 | 40.16 | 2.70 |
| SWE-PRM$_S$ | SWE-AGENT-LM-32B | 19.6 (-20.4) | 67.6 | 21.31 (-17.33) | 2.46 |
| | DEVSTRAL-SMALL-2505 | 34.4 (+0.4) | 94.9 | 41.28 (+3.31) | 4.80 |
| | DEVSTRAL-SMALL-2507 | **33.6 (+3.6)** | 93.4 | 45.54 (+5.38) | 4.84 |
| SWE-PRM$_C$ | SWE-AGENT-LM-32B | 35.6 (-4.4) | 91.4 | 34.32 (-4.32) | 3.77 |
| | DEVSTRAL-SMALL-2505 | 34.2 (+0.2) | 92.2 | 38.39 (+0.42) | 3.96 |
| | DEVSTRAL-SMALL-2507 | 30.2 (+0.2) | 90.2 | 43.46 (+3.30) | 4.46 |
| SWE-PRM$_{CG}$ | SWE-AGENT-LM-32B | 35.6 (-4.4) | 89.8 | 32.71 (-5.93) | 3.16 |
| | DEVSTRAL-SMALL-2505 | 34.2 (+0.2) | 92.8 | 37.65 (-0.32) | 3.27 |
| | DEVSTRAL-SMALL-2507 | 30.2 (+0.2) | 91.0 | 41.52 (+1.36) | 3.73 |
| SWE-PRM$_D$ | SWE-AGENT-LM-32B | 38.8 (-1.2) | 92.2 | 33.12 (-5.52) | 3.31 |
| | DEVSTRAL-SMALL-2505 | 34.2 (+0.2) | 93.4 | 37.89 (-0.08) | 3.86 |
| | DEVSTRAL-SMALL-2507 | 30.2 (+0.2) | 93.4 | 40.08 (-0.08) | 4.15 |
| SWE-PRM$_{DN}$ | SWE-AGENT-LM-32B | 30.0 (-10.0) | 79.6 | 27.54 (-11.10) | 3.18 |
| | DEVSTRAL-SMALL-2505 | 34.2 (+0.2) | 94.4 | 37.72 (-0.25) | 4.06 |
| | DEVSTRAL-SMALL-2507 | 30.2 (+0.2) | 91.6 | 39.98 (-0.18) | 4.53 |
| SWE-PRM$_{DG}$ | SWE-AGENT-LM-32B | 34.8 (-5.2) | **93.2** | 33.82 (-4.82) | 2.97 |
| | DEVSTRAL-SMALL-2505 | 34.2 (+0.2) | **95.4** | 38.58 (+0.61) | 3.47 |
| | DEVSTRAL-SMALL-2507 | 30.2 (+0.2) | 93.0 | 39.52 (-0.64) | 3.39 |
| SWE-PRM$_{DNG}$ | SWE-AGENT-LM-32B | 30.0 (-10.0) | 54.8 | **10.11 (-28.53)** | 1.23 |
| | DEVSTRAL-SMALL-2505 | 34.2 (+0.2) | 94.4 | 36.05 (-1.92) | 3.29 |
| | DEVSTRAL-SMALL-2507 | 30.4 (+0.4) | 91.8 | 39.22 (-0.94) | 3.38 |
| SWE-PRM$_{DR}$ | SWE-AGENT-LM-32B | 36.8 (-3.2) | 92.8 | 28.67 (-9.97) | 2.82 |
| | DEVSTRAL-SMALL-2505 | **36.0 (+2.0)** | 95.0 | **32.33 (-5.64)** | 3.06 |
| | DEVSTRAL-SMALL-2507 | 32.4 (+2.4) | **94.4** | **37.67 (-2.49)** | 3.87 |

Table 3: Closed-Source SWE-PRM variations: SWE-PRM is CLAUDE-SONNET-4 in all cases. Deltas in brackets compare to the base SWE-AGENT-LM-32B row.

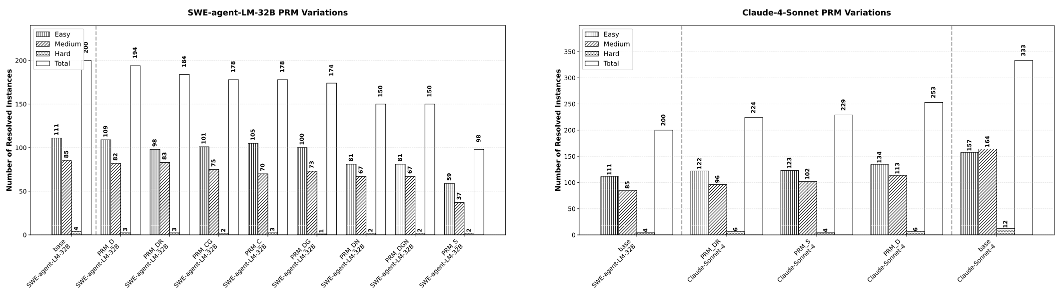
| Setting | Policy Model | Resolution Rate (%) | Patch Generation Rate (%) | Avg Steps | Total Cost ($) per 100 instances |
|---|---|---|---|---|---|
| base | SWE-AGENT-LM-32B | 40.0 | 92.4 | 38.64 | 2.77 |
| | CLAUDE-SONNET-4 | 66.6 | 100.0 | 61.72 | 121.66 |
| SWE-PRM$_S$ | SWE-AGENT-LM-32B | 45.8 (+5.8) | **98.2** | 51.54 (+12.90) | 28.42 |
| SWE-PRM$_D$ | SWE-AGENT-LM-32B | **50.6 (+10.6)** | **98.2** | 37.99 (-0.65) | 25.98 |
| SWE-PRM$_{DR}$ | SWE-AGENT-LM-32B | 44.8 (+4.8) | **98.2** | **34.38** (-4.26) | 24.53 |

# 5 Results and Analysis

We evaluate the effectiveness of SWE-PRM across four dimensions: (i) their impact on overall resolution, (ii) performance stratified by task difficulty, (iii) the relative effectiveness of different feedback strategies, and (iv) the cost–benefit tradeoffs of using SWE-PRM. Unless otherwise noted, results are reported with SWE-AGENT-LM-32B as the base policy model. Full tables are provided in Appendix A.2; here we highlight the most salient results.

## 5.1 Do off-the-shelf SWE-PRMs improve performance over base agents?

**Open-source SWE-PRM variants.** Table 2 compares the base SWE-AGENT-LM-32B with six open-source PRM-guided configurations. None improve resolution consistently: the base achieves 40.0% resolution, while open-source PRM variants range between 30.0–38.8%. In addition, these variants often introduce inefficiencies such as longer trajectories or lower patch generation rates. Similarly, the DEVSTRAL-SMALL-2505 and DEVSTRAL-SMALL-2507 show little benefit from PRM




(a) SWE-AGENT-LM-32B PRM variations

(b) CLAUDE-SONNET-4 PRM variations

Figure 2: Difficulty-wise instances resolved out of 500 SWE-bench Verified instances (194 Easy, 261 Medium, 45 Hard). $\texttt{PRM}_D$ with CLAUDE-SONNET-4 yields the strongest gains across all tiers.

guidance. These results suggest that models finetuned for SWE and agentic tasks are not inherently reliable when used as PRMs.

**Closed-source PRM variants.** In contrast, Table 3 shows that PRMs based on CLAUDE-SONNET-4 consistently raise resolution rates above the base. Improvements range from +4.8 to +10.6 percentage points, establishing a clear difference between open- and closed-source settings. The relative effectiveness of different feedback strategies is analyzed further in Section 5.3.

**Takeaway.** Open-source PRMs fail to improve performance significantly over base agents, whereas closed-source PRMs consistently provide resolution gains of 5–11 percentage points.

### 5.2 How does performance vary across difficulty levels?

We focus on SWE-AGENT-LM-32B for difficulty-stratified analysis, as it achieves the best base performance among the open-source models (40.0% resolution overall). Figure 2 shows results across Easy (194), Medium (261), and Hard (45) instances. The base agent achieves 57.2% on Easy, 32.6% on Medium, and only 8.9% on Hard, indicating a steep performance drop on more complex tasks. Open-source PRM variants (Figure 2a) do not improve this distribution. For example, $\texttt{PRM}_C$ and $\texttt{PRM}_{CG}$ reduce overall resolution, while $\texttt{PRM}_{DN}$ and $\texttt{PRM}_{DGN}$ degrade Hard-task performance further. Closed-source PRMs with CLAUDE-SONNET-4 (Figure 2b) improve across all tiers. The strongest setting, $\texttt{PRM}_D$, reaches 69.1% on Easy, 43.3% on Medium, and 13.3% on Hard. Even unguided reasoning ($\texttt{PRM}_S$) improves every tier, though it lengthens trajectories. These gains show that PRMs are particularly valuable for Medium and Hard tasks, where trajectory-level inefficiencies are most damaging.

**Takeaway.** Open-source PRMs provide no benefit across difficulty levels, while closed-source PRMs, especially $\texttt{PRM}_D$, deliver consistent improvements, with the largest relative gains on Medium and Hard tasks.

### 5.3 Which course correction strategies are most effective?

We next individually compare three feedback strategies with CLAUDE-SONNET-4: simple unguided reasoning ($\texttt{PRM}_S$), detailed taxonomy-guided reasoning with feedback ($\texttt{PRM}_D$), and detailed taxonomy-guided reasoning with explicit action recommendation ($\texttt{PRM}_{DR}$).

**Unguided reasoning (**$\texttt{PRM}_S$**)** improves resolution to 45.8% (+5.8 pp) but lengthens trajectories substantially (51.5 steps vs. 38.6 for base). Since no error detection is elicited, windows may not be explicitly flagged as suboptimal, providing no concrete signal about inefficient behavior; the empirical effect is longer, less efficient runs.

**Taxonomy-guided feedback (**$\texttt{PRM}_D$**)** is the strongest setting: resolution reaches 50.6% (+10.6 pp) while steps slightly decrease (37.99). Appendix Table 4 shows that nearly every PRM invocation marks the window as suboptimal (7.21 out of 7.24), indicating frequent detection of trajectory-level errors. This shows that structured signals help the agent truncate inefficient exploration rather than extend it.

**Taxonomy-guided with action recommendation** ($\texttt{PRM}_{DR}$) achieves the smallest resolution gain (44.8%, $+4.8$ pp). While steps reduce to 34.4, almost every invocation is still flagged suboptimal (6.37 out of 6.39), suggesting that rigid prescriptions lead to shorter but less successful runs.

Across settings, closed-source PRM variants almost always flag windows as suboptimal, reflecting strong detection of trajectory-level issues. Open-source PRMs also mark windows as suboptimal, but at lower rates, aligning with their weaker overall effectiveness. Taken together, these results demonstrate that taxonomy grounding is essential for effective guidance, and that providing explicit actions can harm resolution by constraining the agent too tightly.

**Takeaway.** $\texttt{PRM}_D$ is the most effective strategy, delivering the largest resolution rate gain with fewer steps; $\texttt{PRM}S$ lengthens runs for limited benefit, and $\texttt{PRM}DR$ shortens runs but reduces accuracy.

### 5.4 What are the cost–benefit tradeoffs of PRMs?

The final question is whether the substantial performance gains enabled by PRMs justify their additional inference cost. Table 3 reports cost per 100 instances. The base SWE-AGENT-LM-32B resolves 40.0% of instances at a cost of \$2.77. In contrast, closed-source PRMs increase resolution to as high as 50.6%, a double-digit relative improvement, while raising cost to \$24–\$28 per 100 instances.

Breaking costs down by component in Appendix A.2 shows that the increase is driven primarily by PRM queries: for example, $\texttt{PRM}_D$ spends \$3.61 per 100 on policy calls and \$22.4 on PRM calls. Crucially, this overhead translates into more instances successfully resolved. Measured as incremental cost per additional success, $\texttt{PRM}_D$ achieves the best tradeoff: \$23.2 in added cost yields 10.6 additional resolutions. $\texttt{PRM}_S$ and $\texttt{PRM}_{DR}$ are less favorable, but still surpass the base agent in absolute performance.

Viewed from this perspective, PRMs represent a deliberate performance–cost tradeoff. Without them, resolution plateaus at 40%. With taxonomy-guided feedback ($\texttt{PRM}_D$), resolution climbs above 50%. These results underscore that PRMs are a viable and practical means of unlocking further progress on complex tasks like repository-level code generation, and point to future work on making PRM calls more cost-efficient.

**Takeaway.** PRMs are not a free improvement, but they deliver clear performance gains: $\texttt{PRM}_D$ surpasses 50% resolution and offers the best cost-benefit profile, making it the most effective path to higher accuracy today.

## 6 Discussion and Conclusion

This work introduces SWE-PRM, a real-time course-corrector for software engineering agents. By anchoring feedback in a taxonomy of trajectory-level inefficiencies, SWE-PRM delivers lightweight interventions that improves agent reliability without altering the base policy model. Our results on SWE-BENCH VERIFIED demonstrate three key findings. First, while open-source PRMs offer little benefit, closed-source PRMs consistently boost resolution by 5-11 percentage points. Second, the strongest gains occur on medium and hard tasks, where trajectory-level inefficiencies are most pronounced. Third, among feedback strategies, taxonomy-guided PRMs provide the best balance: they improve the resolution rate to above 50% while maintaining or reducing the trajectory lengths.

Beyond these results, our study highlights broader implications. PRMs shift the design space from purely outcome-focused optimization toward process-aware guidance, complementing approaches like search-based planning or post-hoc trajectory analysis. Although PRMs add inference overhead, their modularity allows them to be flexibly integrated with both open-weight and proprietary models. Future work could reduce costs through adaptive invocation schedules or distillation into lighter models and extend the taxonomy to other sequential reasoning domains. In sum, PRMs represent a practical and principled path forward: they enable agents to not only solve more tasks, but to solve them more efficiently, setting the stage for more reliable deployment of LLM agents in complex software engineering environments.

## References


[1] Antonis Antoniades, Albert Örwall, Kexun Zhang, Yuxi Xie, Anirudh Goyal, and William Yang Wang. Swe-search: Enhancing software agents with monte carlo tree search and iterative refinement. *CoRR*, 2024.

[2] Antonis Antoniades, Albert Örwall, Kexun Zhang, Yuxi Xie, Anirudh Goyal, and William Wang. Swe-search: Enhancing software agents with monte carlo tree search and iterative refinement, 2025. URL `https://arxiv.org/abs/2410.20285`.

[3] Mert Cemri, Melissa Z. Pan, Shuyi Yang, Lakshya A. Agrawal, Bhavya Chopra, Rishabh Tiwari, Kurt Keutzer, Aditya Parameswaran, Dan Klein, Kannan Ramchandran, Matei Zaharia, Joseph E. Gonzalez, and Ion Stoica. Why do multi-agent llm systems fail?, 2025. URL `https://arxiv.org/abs/2503.13657`.

[4] Zhi Chen, Wei Ma, and Lingxiao Jiang. Unveiling pitfalls: Understanding why ai-driven code agents fail at github issue resolution, 2025. URL `https://arxiv.org/abs/2503.12374`.

[5] Le Deng, Zhonghao Jiang, Jialun Cao, Michael Pradel, and Zhongxin Liu. Nocode-bench: A benchmark for evaluating natural language-driven feature addition, 2025. URL `https://arxiv.org/abs/2507.18130`.

[6] Darshan Deshpande, Varun Gangal, Hersh Mehta, Jitin Krishnan, Anand Kannappan, and Rebecca Qian. Trail: Trace reasoning and agentic issue localization, 2025. URL `https://arxiv.org/abs/2505.08638`.

[7] Jatin Ganhotra. Cracking the code: How difficult are swe-bench-verified tasks really?, April 2025. URL `https://jatinganhotra.dev/blog/swe-agents/2025/04/15/swe-bench-verified-easy-medium-hard/`. Blog post.

[8] Dhruv Gautam, Spandan Garg, Jinu Jang, Neel Sundaresan, and Roshanak Zilouchian Moghaddam. Refactorbench: Evaluating stateful reasoning in language agents through code, 2025. URL `https://arxiv.org/abs/2503.07832`.

[9] 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.

[10] Carlos E Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik R Narasimhan. SWE-bench: Can language models resolve real-world github issues? In *The Twelfth International Conference on Learning Representations*, 2024. URL `https://openreview.net/forum?id=VTF8yNQM66`.

[11] Muhammad Khalifa, Rishabh Agarwal, Lajanugen Logeswaran, Jaekyeom Kim, Hao Peng, Moontae Lee, Honglak Lee, and Lu Wang. Process reward models that think, 2025. URL `https://arxiv.org/abs/2504.16828`.

[12] Qingyao Li, Xinyi Dai, Xiangyang Li, Weinan Zhang, Yasheng Wang, Ruiming Tang, and Yong Yu. CodePRM: Execution feedback-enhanced process reward model for code generation. In Wanxiang Che, Joyce Nabende, Ekaterina Shutova, and Mohammad Taher Pilehvar, editors, *Findings of the Association for Computational Linguistics: ACL 2025*, pages 8169–8182, Vienna, Austria, July 2025. Association for Computational Linguistics. ISBN 979-8-89176-256-5. doi: 10.18653/v1/2025.findings-acl.428. URL `https://aclanthology.org/2025.findings-acl.428/`.

[13] Wei Li, Xin Zhang, Zhongxin Guo, Shaoguang Mao, Wen Luo, Guangyue Peng, Yangyu Huang, Houfeng Wang, and Scarlett Li. Fea-bench: A benchmark for evaluating repository-level code generation for feature implementation, 2025. URL `https://arxiv.org/abs/2503.06680`.

[14] Hunter Lightman, Vineet Kosaraju, Yura Burda, Harri Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. Let's verify step by step. *CoRR*, 2023.

[15] Yingwei Ma, Yongbin Li, Yihong Dong, Xue Jiang, Rongyu Cao, Jue Chen, Fei Huang, and Binhua Li. Thinking longer, not larger: Enhancing software engineering agents via scaling test-time compute, 2025. URL https://arxiv.org/abs/2503.23803.

[16] Jiayi Pan, Xingyao Wang, Graham Neubig, Navdeep Jaitly, Heng Ji, Alane Suhr, and Yizhe Zhang. Training software engineering agents and verifiers with swe-gym. In *Proceedings of the 42nd International Conference on Machine Learning (ICML 2025)*, 2025. URL https://arxiv.org/abs/2412.21139. arXiv:2412.21139, accepted at ICML 2025.

[17] Cheng Qian, Emre Can Acikgoz, Hongru Wang, Xiusi Chen, Avirup Sil, Dilek Hakkani-Tür, Gokhan Tur, and Heng Ji. Smart: Self-aware agent for tool overuse mitigation, 2025. URL https://arxiv.org/abs/2502.11435.

[18] Muhammad Shihab Rashid, Christian Bock, Yuan Zhuang, Alexander Buchholz, Tim Esler, Simon Valentin, Luca Franceschi, Martin Wistuba, Prabhu Teja Sivaprasad, Woo Jung Kim, Anoop Deoras, Giovanni Zappella, and Laurent Callot. Swe-polybench: A multi-language benchmark for repository level evaluation of coding agents, 2025. URL https://arxiv.org/abs/2504.08703.

[19] Yifan Song, Da Yin, Xiang Yue, Jie Huang, Sujian Li, and Bill Yuchen Lin. Trial and error: Exploration-based trajectory optimization for llm agents, 2024. URL https://arxiv.org/abs/2403.02502.

[20] Lin Sun, Chuang Liu, Xiaofeng Ma, Tao Yang, Weijia Lu, and Ning Wu. Freeprm: Training process reward models without ground truth process labels, 2025. URL https://arxiv.org/abs/2506.03570.

[21] Yoo Yeon Sung, Hannah Kim, and Dan Zhang. Verila: A human-centered evaluation framework for interpretable verification of llm agent failures, 2025. URL https://arxiv.org/abs/2503.12651.

[22] Hanlin Wang, Jian Wang, Chak Tou Leong, and Wenjie Li. Steca: Step-level trajectory calibration for llm agent learning, 2025. URL https://arxiv.org/abs/2502.14276.

[23] Qinzhuo Wu, Pengzhi Gao, Wei Liu, and Jian Luan. Backtrackagent: Enhancing gui agent with error detection and backtracking mechanism, 2025. URL https://arxiv.org/abs/2505.20660.

[24] 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. In *Proceedings of the 38th International Conference on Neural Information Processing Systems*, NIPS '24, Red Hook, NY, USA, 2025. Curran Associates Inc. ISBN 9798331314385.

[25] John Yang, Kilian Leret, 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, 2025. URL https://arxiv.org/abs/2504.21798.

[26] Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. ReAct: Synergizing reasoning and acting in language models. In *International Conference on Learning Representations (ICLR)*, 2023.

[27] Karina Zainullina, Alexander Golubev, Maria Trofimova, Sergei Polezhaev, Ibragim Badertdinov, Daria Litvintseva, Simon Karasik, Filipp Fisin, Sergei Skvortsov, Maksim Nekrashevich, Anton Shevtsov, and Boris Yangel. Guided search strategies in non-serializable environments with applications to software engineering agents, 2025. URL https://arxiv.org/abs/2505.13652.

[28] Daoguang Zan, Zhirong Huang, Wei Liu, Hanwu Chen, Linhao Zhang, Shulin Xin, Lu Chen, Qi Liu, Xiaojian Zhong, Aoyan Li, Siyao Liu, Yongsheng Xiao, Liangqiang Chen, Yuyu Zhang, Jing Su, Tianyu Liu, Rui Long, Kai Shen, and Liang Xiang. Multi-swe-bench: A multilingual benchmark for issue resolving, 2025. URL https://arxiv.org/abs/2504.02605.

# A Appendix

## A.1 Prompts

Listing 1: Common instructions used for all runs

```
system_template: |-
      You are a helpful assistant that can interact with a computer to solve tasks.
     ↪
      <IMPORTANT>
      * If user provides a path, you should NOT assume it’s relative to the
     ↪ current working directory. Instead, you should explore the file system to
     ↪ find the file before working on it.
      </IMPORTANT>

      You have access to the following functions:

      ---- BEGIN FUNCTION #1: bash ----
      Description: Execute a bash command in the terminal.

      Parameters:
        (1) command (string, required): The bash command to execute. Can be empty
     ↪ to view additional logs when previous exit code is ‘-1‘. Can be ‘ctrl+c‘ to
     ↪  interrupt the currently running process.
      ---- END FUNCTION #1 ----

      ---- BEGIN FUNCTION #2: submit ----
      Description: Finish the interaction when the task is complete OR if the
     ↪ assistant cannot proceed further with the task.
      No parameters are required for this function.
      ---- END FUNCTION #2 ----

      ---- BEGIN FUNCTION #3: str_replace_editor ----
      Description: Custom editing tool for viewing, creating and editing files
      * State is persistent across command calls and discussions with the user
      * If ‘path‘ is a file, ‘view‘ displays the result of applying ‘cat -n‘. If ‘
     ↪ path‘ is a directory, ‘view‘ lists non-hidden files and directories up to 2
     ↪  levels deep
      * The ‘create‘ command cannot be used if the specified ‘path‘ already exists
     ↪  as a file
      * If a ‘command‘ generates a long output, it will be truncated and marked
     ↪ with ‘<response clipped>‘
      * The ‘undo_edit‘ command will revert the last edit made to the file at ‘
     ↪ path‘

      Notes for using the ‘str_replace‘ command:
      * The ‘old_str‘ parameter should match EXACTLY one or more consecutive lines
     ↪  from the original file. Be mindful of whitespaces!
      * If the ‘old_str‘ parameter is not unique in the file, the replacement will
     ↪  not be performed. Make sure to include enough context in ‘old_str‘ to make
     ↪  it unique
      * The ‘new_str‘ parameter should contain the edited lines that should
     ↪ replace the ‘old_str‘

      Parameters:
        (1) command (string, required): The commands to run. Allowed options are: ‘
     ↪ view‘, ‘create‘, ‘str_replace‘, ‘insert‘, ‘undo_edit‘.
      Allowed values: [‘view‘, ‘create‘, ‘str_replace‘, ‘insert‘, ‘undo_edit‘]
        (2) path (string, required): Absolute path to file or directory, e.g. ‘/
     ↪ repo/file.py‘ or ‘/repo‘.
        (3) file_text (string, optional): Required parameter of ‘create‘ command,
     ↪ with the content of the file to be created.
        (4) old_str (string, optional): Required parameter of ‘str_replace‘
     ↪ command containing the string in ‘path‘ to replace.
```

```
          (5) new_str (string, optional): Optional parameter of `str_replace`
      ↪ command containing the new string (if not given, no string will be added).
      ↪ Required parameter of `insert` command containing the string to insert.
          (6) insert_line (integer, optional): Required parameter of `insert`
      ↪ command. The `new_str` will be inserted AFTER the line `insert_line` of `
      ↪ path`.
          (7) view_range (array, optional): Optional parameter of `view` command
      ↪ when `path` points to a file. If none is given, the full file is shown. If
      ↪ provided, the file will be shown in the indicated line number range, e.g.
      ↪ [11, 12] will show lines 11 and 12. Indexing at 1 to start. Setting `[
      ↪ start_line, -1]` shows all lines from `start_line` to the end of the file.
        ---- END FUNCTION #3 ----


        If you choose to call a function ONLY reply in the following format with NO
      ↪ suffix:

        Provide any reasoning for the function call here.
        <function=example_function_name>
        <parameter=example_parameter_1>value_1</parameter>
        <parameter=example_parameter_2>
        This is the value for the second parameter
        that can span
        multiple lines
        </parameter>
        </function>

        <IMPORTANT>
        Reminder:
        - Function calls MUST follow the specified format, start with <function= and
      ↪  end with </function>
        - Required parameters MUST be specified
        - Only call one function at a time
        - Always provide reasoning for your function call in natural language BEFORE
      ↪  the function call (not after)
        </IMPORTANT>
instance_template: |-
        <uploaded_files>
        {{working_dir}}
        </uploaded_files>
        I've uploaded a python code repository in the directory {{working_dir}}.
      ↪ Consider the following PR description:

        <pr_description>
        {{problem_statement}}
        </pr_description>

        Can you help me implement the necessary changes to the repository so that
      ↪ the requirements specified in the <pr_description> are met?
        I've already taken care of all changes to any of the test files described in
      ↪  the <pr_description>. This means you DON'T have to modify the testing
      ↪ logic or any of the tests in any way!
        Your task is to make the minimal changes to non-tests files in the {{
      ↪ working_dir}} directory to ensure the <pr_description> is satisfied.
        Follow these steps to resolve the issue:
        1. As a first step, it might be a good idea to find and read code relevant
      ↪ to the <pr_description>
        2. Create a script to reproduce the error and execute it with `python <
      ↪ filename.py>` using the bash tool, to confirm the error
        3. Edit the source code of the repo to resolve the issue
        4. Rerun your reproduce script and confirm that the error is fixed!
        5. Think about edgecases and make sure your fix handles them as well
        Your thinking should be thorough and so it's fine if it's very long.
next_step_template: |-
        OBSERVATION:
```

```
      {{observation}}
next_step_no_output_template: |-
      Your command ran successfully and did not produce any output.
```

Listing 2: Prompt for the $\texttt{PRM}_S$ variant.

```
You are an expert at analyzing SWE-agent trajectories for solving software engineering tasks. Your role is to detect trajectory-level errors and provide corrective guidance to prevent task failure.
Analyze these recent agent steps for suboptimality.

Common suboptimal patterns include:
1. **Repetitive actions**: Viewing the same file multiple times without making progress, running the same commands repeatedly
2. **Not understanding observations**: Ignoring error messages, missing important information in outputs, not reading test failures carefully
3. **Inefficient exploration**: Looking at irrelevant files, not focusing on the error location, exploring too broadly
4. **Missing obvious next steps**: Not running tests after changes, not checking if changes work, not reading error traces
5. **Redundant operations**: Creating files that exist, re-implementing existing functionality
6. **Poor debugging strategy**: Making random changes without understanding the issue, not isolating the problem
7. **Incomplete actions**: Starting to edit but not finishing, viewing files without purpose

Recent agent steps:

{{recent_steps}}

If the recent steps are suboptimal, provide DETAILED, ACTIONABLE course-corrective feedback that:
- Explains specifically why these steps are problematic given the context
- Suggests concrete next steps the agent should take instead
- References specific files, functions, or error messages from the trajectory
- Provides a clear strategy to get back on track
- Is at least 3-5 sentences long with specific technical guidance
```

Listing 3: Prompt for the $\texttt{PRM}_C$ and $\texttt{PRM}_{CG}$ variants.

```
You are a supervisor monitoring an LLM-based coding agent. Your role is to detect trajectory-level errors and provide corrective guidance to prevent task failure.

Analyze the provided agent trace for the following error categories. For each detected error, provide specific evidence and a recovery action.

SPECIFICATION ERRORS (System Design Issues):
- Task Specification Violations: Agent fails to adhere to task constraints or requirements
- Role Specification Violations: Agent behaves outside its defined role/responsibilities
- Step Repetition: Unnecessary repetition of completed steps or actions
- Termination Condition Unawareness: Agent continues working when task completion criteria are met

REASONING ERRORS (Decision Making Issues):
- Problem Misidentification: Agent misunderstands the core problem or current subtask
- Tool Selection Errors: Agent uses inappropriate tools for the current task
- Hallucinations: Agent generates false information or fabricates tool outputs
- Information Processing Failures: Poor retrieval of relevant information or misinterpretation
```

```
COORDINATION ERRORS (Process Management Issues):
- Task Derailment: Agent deviates from intended objective or loses focus
- Goal Deviation: Agent pursues goals that don’t serve the main objective
- Context Handling Failures: Agent loses important context or forgets previous
    ↪ findings
- Verification Failures: Inadequate checking of work quality or correctness

For each error category, respond with:
DETECTED: Yes/No
EVIDENCE: Specific quote or observation from trace (if detected)
RECOVERY_ACTION: Specific instruction to correct the error (if detected)

Then provide:
TASK_STATUS: On track / Needs correction / Critical intervention required
OVERALL_GUIDANCE: 1-2 sentences of specific guidance for the agent

Recent agent steps:
{{recent_steps}}

Focus on errors that can be corrected through guidance. Be concise but precise in
    ↪ evidence citations. Only mark "DETECTED: Yes" if you have clear evidence.
```

Listing 4: Prompt for the $\text{PRM}_D$ and $\text{PRM}_{DG}$ variants.

```
You are a supervisor monitoring an LLM-based coding agent. Your role is to detect
    ↪ trajectory-level errors and provide corrective guidance to prevent task
    ↪ failure.
Analyze the provided agent trace for the following error categories. For each
    ↪ detected error, provide specific evidence and a recovery action.

SPECIFICATION ERRORS (System Design Issues)

1. Task Specification Violations
Definition: Agent fails to adhere to task constraints or requirements
Recovery: Redirect agent to original task requirements

2. Role Specification Violations
Definition: Agent behaves outside its defined role/responsibilities
Recovery: Remind agent of its specific role and boundaries

3. Step Repetition
Definition: Unnecessary repetition of completed steps or actions
Recovery: Acknowledge completed work and guide to next logical step

4. Termination Condition Unawareness
Definition: Agent continues working when task completion criteria are met
Recovery: Signal completion criteria and instruct to finalize

REASONING ERRORS (Decision Making Issues)

5. Problem Misidentification
Definition: Agent misunderstands the core problem or current subtask
Recovery: Clarify the actual problem and expected approach

6. Tool Selection Errors
Definition: Agent uses inappropriate tools for the current task
Recovery: Suggest correct tools and explain their appropriate usage

7. Hallucinations
Definition: Agent generates false information or fabricates tool outputs
Recovery: Request verification of claims against actual evidence

8. Information Processing Failures
```

```
Definition: Poor retrieval of relevant information or misinterpretation
Recovery: Guide agent to correct information sources and interpretation

COORDINATION ERRORS (Process Management Issues)

9. Task Derailment
Definition: Agent deviates from intended objective or loses focus
Recovery: Realign agent with original objectives and priorities

10. Goal Deviation
Definition: Agent pursues goals that don't serve the main objective
Recovery: Refocus on primary goals and expected outcomes

11. Context Handling Failures
Definition: Agent loses important context or forgets previous findings
Recovery: Provide context summary and key information recap

12. Verification Failures
Definition: Inadequate checking of work quality or correctness
Recovery: Instruct specific verification steps and quality checks

Response Format

For each error category, respond with:
DETECTED: Yes/No
EVIDENCE: Specific quote or observation from trace (if detected)
RECOVERY_ACTION: Specific instruction to correct the error (if detected)

Then provide:
TASK_STATUS: On track / Needs correction / Critical intervention required
OVERALL_GUIDANCE: Detailed and specific guidance for the agent

Example Response Structure

SPECIFICATION ERRORS:
1. Task Specification Violations: DETECTED: No
2. Role Specification Violations: DETECTED: No
3. Step Repetition: DETECTED: Yes
EVIDENCE: "Agent ran the same test command three times: 'pytest test_file.py'"
RECOVERY_ACTION: "The test has already been executed successfully. Proceed to
    ↪ analyze the results and move to the next development step."
4. Termination Condition Unawareness: DETECTED: No

REASONING ERRORS:
5. Problem Misidentification: DETECTED: No
6. Tool Selection Errors: DETECTED: Yes
EVIDENCE: "Agent used text editor to run Python code instead of using the Python
    ↪ interpreter"
RECOVERY_ACTION: "Use the Python interpreter tool for code execution. The text
    ↪ editor is for viewing and modifying files only."
7. Hallucinations: DETECTED: No
8. Information Processing Failures: DETECTED: No

COORDINATION ERRORS:
9. Task Derailment: DETECTED: No
10. Goal Deviation: DETECTED: No
11. Context Handling Failures: DETECTED: No
12. Verification Failures: DETECTED: No

TASK_STATUS: Needs correction
OVERALL_GUIDANCE: You are repeating actions unnecessarily and using incorrect
    ↪ tools. Specifically:
1. Stop running the same test command repeatedly - the test 'pytest test_file.py'
    ↪ has already been executed successfully three times with the same result
```

```
2. Use the Python interpreter tool for executing Python code, not the text editor
    ↪ which is only for viewing and modifying files
3. Now focus on analyzing the test results you already obtained to determine what
    ↪ the next development step should be
4. Review the test output to identify any failing tests or areas that need
    ↪ improvement
5. If all tests are passing, proceed to verify your implementation meets the
    ↪ original requirements before considering the task complete

Recent agent steps:

{{recent_steps}}

Instructions:

1. Focus on errors that can be corrected through guidance
2. Provide specific, actionable recovery instructions
3. Be concise but precise in evidence citations
4. Only mark "DETECTED: Yes" if you have clear evidence
5. Prioritize errors that most threaten task completion
```

Listing 5: Prompt for the $\mathtt{PRM}_{DN}$ and $\mathtt{PRM}_{DNG}$ variants.

```
You are a supervisor monitoring an LLM-based coding agent. Your role is to detect
    ↪ trajectory-level errors and provide corrective guidance to prevent task
    ↪ failure.
Analyze the provided agent trace for the following error categories. For each
    ↪ detected error, provide specific evidence and a recovery action.

SPECIFICATION ERRORS (System Design Issues)

1. Task Specification Violations
Definition: Agent fails to adhere to task constraints or requirements
Recovery: Redirect agent to original task requirements

2. Role Specification Violations
Definition: Agent behaves outside its defined role/responsibilities
Recovery: Remind agent of its specific role and boundaries

3. Step Repetition
Definition: Unnecessary repetition of completed steps or actions
Recovery: Acknowledge completed work and guide to next logical step

4. Termination Condition Unawareness
Definition: Agent continues working when task completion criteria are met
Recovery: Signal completion criteria and instruct to finalize

REASONING ERRORS (Decision Making Issues)

5. Problem Misidentification
Definition: Agent misunderstands the core problem or current subtask
Recovery: Clarify the actual problem and expected approach

6. Tool Selection Errors
Definition: Agent uses inappropriate tools for the current task
Recovery: Suggest correct tools and explain their appropriate usage

7. Hallucinations
Definition: Agent generates false information or fabricates tool outputs
Recovery: Request verification of claims against actual evidence

8. Information Processing Failures
Definition: Poor retrieval of relevant information or misinterpretation
Recovery: Guide agent to correct information sources and interpretation
```

```

COORDINATION ERRORS (Process Management Issues)

9. Task Derailment
Definition: Agent deviates from intended objective or loses focus
Recovery: Realign agent with original objectives and priorities

10. Goal Deviation
Definition: Agent pursues goals that don't serve the main objective
Recovery: Refocus on primary goals and expected outcomes

11. Context Handling Failures
Definition: Agent loses important context or forgets previous findings
Recovery: Provide context summary and key information recap

12. Verification Failures
Definition: Inadequate checking of work quality or correctness
Recovery: Instruct specific verification steps and quality checks

Response Format

For each error category, respond with:
DETECTED: Yes/No
EVIDENCE: Specific quote or observation from trace (if detected)
RECOVERY_ACTION: Specific instruction to correct the error (if detected)

Then provide:
TASK_STATUS: On track / Needs correction / Critical intervention required
OVERALL_GUIDANCE: Detailed and specific guidance for the agent

Recent agent steps:

{{recent_steps}}

Instructions:

1. Focus on errors that can be corrected through guidance
2. Provide specific, actionable recovery instructions
3. Be concise but precise in evidence citations
4. Only mark "DETECTED: Yes" if you have clear evidence
5. Prioritize errors that most threaten task completion
```

Listing 6: Prompt for the $\text{PRM}_{DR}$ variant.

```
You are a supervisor monitoring an LLM-based coding agent. Your role is to detect
    ↪ trajectory-level errors and provide corrective guidance to prevent task
    ↪ failure.

The agent has access to the following functions as actions -

---- BEGIN FUNCTION #1: bash ----
Description: Execute a bash command in the terminal.

Parameters:
(1) command (string, required): The bash command to execute. Can be empty to view
    ↪ additional logs when previous exit code is `-1`. Can be `ctrl+c` to
    ↪ interrupt the currently running process.
---- END FUNCTION #1 ----

---- BEGIN FUNCTION #2: submit ----
Description: Finish the interaction when the task is complete OR if the assistant
    ↪ cannot proceed further with the task.
No parameters are required for this function.
---- END FUNCTION #2 ----
```

```
---- BEGIN FUNCTION #3: str_replace_editor ----
Description: Custom editing tool for viewing, creating and editing files
* State is persistent across command calls and discussions with the user
* If ‘path‘ is a file, ‘view‘ displays the result of applying ‘cat -n‘. If ‘path‘ is a directory, ‘view‘ lists non-hidden files and directories up to 2 levels deep
* The ‘create‘ command cannot be used if the specified ‘path‘ already exists as a file
* If a ‘command‘ generates a long output, it will be truncated and marked with ‘<response clipped>‘
* The ‘undo_edit‘ command will revert the last edit made to the file at ‘path‘

Notes for using the ‘str_replace‘ command:
* The ‘old_str‘ parameter should match EXACTLY one or more consecutive lines from the original file. Be mindful of whitespaces!
* If the ‘old_str‘ parameter is not unique in the file, the replacement will not be performed. Make sure to include enough context in ‘old_str‘ to make it unique
* The ‘new_str‘ parameter should contain the edited lines that should replace the ‘old_str‘

Parameters:
(1) command (string, required): The commands to run. Allowed options are: ‘view‘, ‘create‘, ‘str_replace‘, ‘insert‘, ‘undo_edit‘.
Allowed values: [‘view‘, ‘create‘, ‘str_replace‘, ‘insert‘, ‘undo_edit‘]
(2) path (string, required): Absolute path to file or directory, e.g. ‘/repo/file.py‘ or ‘/repo‘.
(3) file_text (string, optional): Required parameter of ‘create‘ command, with the content of the file to be created.
(4) old_str (string, optional): Required parameter of ‘str_replace‘ command containing the string in ‘path‘ to replace.
(5) new_str (string, optional): Optional parameter of ‘str_replace‘ command containing the new string (if not given, no string will be added). Required parameter of ‘insert‘ command containing the string to insert.
(6) insert_line (integer, optional): Required parameter of ‘insert‘ command. The ‘new_str‘ will be inserted AFTER the line ‘insert_line‘ of ‘path‘.
(7) view_range (array, optional): Optional parameter of ‘view‘ command when ‘path‘ points to a file. If none is given, the full file is shown. If provided, the file will be shown in the indicated line number range, e.g. [11, 12] will show lines 11 and 12. Indexing at 1 to start. Setting ‘[start_line, -1]‘ shows all lines from ‘start_line‘ to the end of the file.
---- END FUNCTION #3 ----

Analyze the provided agent trace for the following error categories. For each detected error, provide specific evidence and a recovery action.

SPECIFICATION ERRORS (System Design Issues)

1. Task Specification Violations
Definition: Agent fails to adhere to task constraints or requirements
Recovery: Redirect agent to original task requirements

2. Role Specification Violations
Definition: Agent behaves outside its defined role/responsibilities
Recovery: Remind agent of its specific role and boundaries

3. Step Repetition
Definition: Unnecessary repetition of completed steps or actions
Recovery: Acknowledge completed work and guide to next logical step

4. Termination Condition Unawareness
Definition: Agent continues working when task completion criteria are met
Recovery: Signal completion criteria and instruct to finalize
```

```
REASONING ERRORS (Decision Making Issues)

5. Problem Misidentification
Definition: Agent misunderstands the core problem or current subtask
Recovery: Clarify the actual problem and expected approach

6. Tool Selection Errors
Definition: Agent uses inappropriate tools for the current task
Recovery: Suggest correct tools and explain their appropriate usage

7. Hallucinations
Definition: Agent generates false information or fabricates tool outputs
Recovery: Request verification of claims against actual evidence

8. Information Processing Failures
Definition: Poor retrieval of relevant information or misinterpretation
Recovery: Guide agent to correct information sources and interpretation

COORDINATION ERRORS (Process Management Issues)

9. Task Derailment
Definition: Agent deviates from intended objective or loses focus
Recovery: Realign agent with original objectives and priorities

10. Goal Deviation
Definition: Agent pursues goals that don't serve the main objective
Recovery: Refocus on primary goals and expected outcomes

11. Context Handling Failures
Definition: Agent loses important context or forgets previous findings
Recovery: Provide context summary and key information recap

12. Verification Failures
Definition: Inadequate checking of work quality or correctness
Recovery: Instruct specific verification steps and quality checks

Response Format

For each error category, respond with:
DETECTED: Yes/No
EVIDENCE: Specific quote or observation from trace (if detected)
RECOVERY_ACTION: Specific instruction to correct the error (if detected)

Then provide:
TASK_STATUS: On track / Needs correction / Critical intervention required
OVERALL_GUIDANCE: Detailed and specific guidance for the agent
RECOMMENDED_ACTION: Recommended next action that the agent should take

Example Response Structure

SPECIFICATION ERRORS:
1. Task Specification Violations: DETECTED: No
2. Role Specification Violations: DETECTED: No
3. Step Repetition: DETECTED: Yes
EVIDENCE: "Agent ran the same test command three times: 'pytest test_file.py'"
RECOVERY_ACTION: "The test has already been executed successfully. Proceed to
    ↪ analyze the results and move to the next development step."
4. Termination Condition Unawareness: DETECTED: No

REASONING ERRORS:
5. Problem Misidentification: DETECTED: No
6. Tool Selection Errors: DETECTED: Yes
EVIDENCE: "Agent used text editor to run Python code instead of using the Python
    ↪ interpreter"
```

```
RECOVERY_ACTION: "Use the Python interpreter tool for code execution. The text editor is for viewing and modifying files only."
7. Hallucinations: DETECTED: No
8. Information Processing Failures: DETECTED: No

COORDINATION ERRORS:
9. Task Derailment: DETECTED: No
10. Goal Deviation: DETECTED: No
11. Context Handling Failures: DETECTED: No
12. Verification Failures: DETECTED: No

TASK_STATUS: Needs correction
OVERALL_GUIDANCE: You are repeating actions unnecessarily and using incorrect tools. Specifically:
1. Stop running the same test command repeatedly - the test 'pytest test_file.py' has already been executed successfully three times with the same result
2. Use the Python interpreter tool for executing Python code, not the text editor which is only for viewing and modifying files
3. Now focus on analyzing the test results you already obtained to determine what the next development step should be
4. Review the test output to identify any failing tests or areas that need improvement
5. If all tests are passing, proceed to verify your implementation meets the original requirements before considering the task complete
RECOMMENDED_ACTION: str_replace_editor view /path/to/test_output.log

Recent agent steps:

{{recent_steps}}

Instructions:

1. Focus on errors that can be corrected through guidance
2. Provide specific, actionable recovery instructions
3. Be concise but precise in evidence citations
4. Only mark "DETECTED: Yes" if you have clear evidence
5. Prioritize errors that most threaten task completion
6. Provide a concrete recommended next action for the agent to take. This should be from the functions available to the agent.
```

## A.2 Complete Results

Table 4: All metrics for all SWE-PRM variants and policy models. Rows with "+ CLAUDE-SONNET-4" use CLAUDE-SONNET-4 for the PRM.

| Setting | Model | Resolution Rate (%) | Easy Resolution Rate (%) | Medium Resolution Rate (%) | Hard Resolution Rate (%) | Patch Generation Rate (%) | Avg Steps | Avg I/P Tokens | Avg O/P Tokens | Avg Sup. Invocations | Avg Sup. I/P Tokens | Avg Sup. O/P Tokens | Avg Optimal Windows | Avg Suboptimal Windows | Policy Model Cost ($) per 100 instances | Sup. Cost ($) per 100 instances | Total Cost ($) per 100 instances |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| base | SWE-AGENT-LM-32B | 40.0 | 57.2 | 32.6 | 8.9 | 92.4 | 38.64 | 340555 | 5744 | - | - | - | - | - | 2.77 | - | 2.77 |
| | DEVSTRAL-SMALL-2505 | 34.0 | 51.0 | 26.4 | 4.4 | 92.6 | 37.97 | 330892 | 5439 | - | - | - | - | - | 2.69 | - | 2.69 |
| | DEVSTRAL-SMALL-2507 | 30.0 | 47.4 | 21.5 | 4.4 | 88.0 | 40.16 | 332407 | 5374 | - | - | - | - | - | 2.70 | - | 2.70 |
| | CLAUDE-SONNET-4 | 66.6 | 80.9 | 62.8 | 26.7 | 100.0 | 61.72 | 37786 | 2534 | - | - | - | - | - | 121.66 | - | 121.66 |
| SWE-PRM$_S$ | SWE-AGENT-LM-32B | 19.6 | 30.4 | 14.2 | 4.4 | 67.6 | 21.31 | 254892 | 2718 | 4.12 | 29990 | 19589 | - | - | 2.06 | 0.40 | 2.46 |
| | DEVSTRAL-SMALL-2505 | 34.4 | 53.6 | 24.9 | 6.7 | 94.9 | 41.28 | 536399 | 6723 | 7.92 | 51627 | 5023 | - | - | 4.34 | 0.45 | 4.80 |
| | DEVSTRAL-SMALL-2507 | 33.6 | 50.5 | 25.3 | 8.9 | 93.4 | 45.54 | 544035 | 6492 | 8.69 | 49523 | 4651 | - | - | 4.40 | 0.43 | 4.84 |
| | SWE-AGENT-LM-32B + CLAUDE-SONNET-4 | 45.8 | 63.4 | 39.1 | 8.9 | 98.2 | 51.54 | 593077 | 7419 | 10.0 | 60192 | 3706 | - | - | 4.80 | 23.62 | 28.42 |
| SWE-PRM$_C$ | SWE-AGENT-LM-32B | 35.6 | 54.1 | 26.8 | 6.7 | 91.4 | 34.32 | 419819 | 4674 | 6.49 | 41894 | 5084 | 0.37 | 6.13 | 3.40 | 0.38 | 3.77 |
| | DEVSTRAL-SMALL-2505 | 34.2 | 54.1 | 24.9 | 2.2 | 92.2 | 38.39 | 438097 | 5326 | 7.34 | 47719 | 3801 | 0.84 | 6.50 | 3.55 | 0.41 | 3.96 |
| | DEVSTRAL-SMALL-2507 | 30.2 | 47.9 | 21.5 | 4.4 | 90.2 | 43.46 | 498381 | 6551 | 8.30 | 48540 | 3815 | 0.34 | 7.96 | 4.04 | 0.42 | 4.46 |
| SWE-PRM$_{CG}$ | SWE-AGENT-LM-32B | 35.6 | 52.1 | 28.7 | 4.4 | 89.8 | 32.71 | 344833 | 4426 | 6.19 | 40824 | 5274 | 0.64 | 5.55 | 2.79 | 0.37 | 3.16 |
| | DEVSTRAL-SMALL-2505 | 34.2 | 54.1 | 24.9 | 2.2 | 92.8 | 37.65 | 354389 | 5106 | 7.19 | 45723 | 3743 | 0.95 | 6.24 | 2.88 | 0.40 | 3.27 |
| | DEVSTRAL-SMALL-2507 | 30.2 | 47.9 | 21.5 | 4.4 | 91.0 | 41.52 | 409703 | 5887 | 7.88 | 46991 | 3633 | 0.53 | 7.36 | 3.32 | 0.40 | 3.73 |
| SWE-PRM$_D$ | SWE-AGENT-LM-32B | 38.8 | 56.2 | 31.4 | 6.7 | 92.2 | 33.12 | 360688 | 4510 | 6.18 | 44751 | 3262 | 0.42 | 5.77 | 2.92 | 0.38 | 3.31 |
| | DEVSTRAL-SMALL-2505 | 34.2 | 54.1 | 24.9 | 2.2 | 93.4 | 37.89 | 421554 | 5752 | 7.22 | 52587 | 2826 | 0.37 | 6.85 | 3.42 | 0.44 | 3.86 |
| | DEVSTRAL-SMALL-2507 | 30.2 | 47.9 | 21.5 | 4.4 | 93.4 | 40.08 | 457684 | 6338 | 7.63 | 51242 | 3391 | 0.31 | 7.32 | 3.71 | 0.44 | 4.15 |
| | SWE-AGENT-LM-32B + CLAUDE-SONNET-4 | 50.6 | 69.1 | 43.3 | 13.3 | 98.2 | 37.99 | 446185 | 5674 | 7.24 | 51443 | 4621 | 0.03 | 7.21 | 3.61 | 2.37 | 25.98 |
| SWE-PRM$_{DN}$ | SWE-AGENT-LM-32B | 30.0 | 41.8 | 25.7 | 4.4 | 79.6 | 27.54 | 350412 | 3407 | 5.21 | 36686 | 7306 | 0.47 | 4.73 | 2.83 | 0.35 | 3.18 |
| | DEVSTRAL-SMALL-2505 | 34.2 | 54.1 | 24.9 | 2.2 | 94.4 | 37.72 | 450821 | 5408 | 7.13 | 47078 | 4665 | 0.65 | 6.48 | 3.65 | 0.41 | 4.06 |
| | DEVSTRAL-SMALL-2507 | 30.2 | 47.9 | 21.5 | 4.4 | 91.6 | 39.98 | 505866 | 6266 | 7.63 | 48816 | 5092 | 0.69 | 6.93 | 4.10 | 0.43 | 4.53 |
| SWE-PRM$_{DG}$ | SWE-AGENT-LM-32B | 34.8 | 51.5 | 28 | 2.2 | 93.2 | 33.82 | 325223 | 4519 | 5.65 | 39090 | 2793 | 0.88 | 4.77 | 2.64 | 0.34 | 2.97 |
| | DEVSTRAL-SMALL-2505 | 34.2 | 54.1 | 24.9 | 2.2 | 95.4 | 38.58 | 371001 | 5405 | 7.39 | 54926 | 2854 | 1.00 | 6.39 | 3.01 | 0.46 | 3.47 |
| | DEVSTRAL-SMALL-2507 | 30.2 | 47.9 | 21.5 | 4.4 | 93 | 39.52 | 364568 | 5557 | 7.50 | 50056 | 3325 | 1.00 | 6.50 | 2.96 | 0.43 | 3.39 |
| SWE-PRM$_{DNG}$ | SWE-AGENT-LM-32B | 30.0 | 41.8 | 25.7 | 4.4 | 54.8 | 10.11 | 110855 | 1118 | 1.93 | 19379 | 21792 | 0.78 | 1.15 | 0.90 | 0.33 | 1.23 |
| | DEVSTRAL-SMALL-2505 | 34.2 | 54.1 | 24.9 | 2.2 | 94.4 | 36.05 | 354803 | 5229 | 6.80 | 46335 | 4412 | 0.99 | 5.80 | 2.88 | 0.41 | 3.29 |
| | DEVSTRAL-SMALL-2507 | 30.4 | 47.9 | 21.8 | 4.4 | 91.8 | 39.22 | 365504 | 5260 | 7.44 | 46252 | 5066 | 0.98 | 6.46 | 2.97 | 0.41 | 3.38 |
| SWE-PRM$_{DR}$ | SWE-AGENT-LM-32B | 36.8 | 50.5 | 31.8 | 6.7 | 92.8 | 28.67 | 299191 | 3900 | 5.44 | 44223 | 4767 | 0.49 | 4.95 | 2.42 | 0.39 | 2.82 |
| | DEVSTRAL-SMALL-2505 | 36.0 | 51.5 | 30.3 | 2.2 | 95.0 | 32.33 | 326033 | 4300 | 6.17 | 49445 | 2287 | 0.45 | 5.72 | 2.64 | 0.41 | 3.06 |
| | DEVSTRAL-SMALL-2507 | 32.4 | 51.0 | 23.4 | 4.4 | 94.4 | 37.67 | 418660 | 5148 | 7.14 | 56343 | 3187 | 0.37 | 6.78 | 3.39 | 0.48 | 3.87 |
| | SWE-AGENT-LM-32B + CLAUDE-SONNET-4 | 44.8 | 62.9 | 36.8 | 13.3 | 98.2 | 34.38 | 389420 | 4984 | 6.39 | 52193 | 3810 | 0.02 | 6.37 | 3.16 | 21.37 | 24.53 |

# NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: The main claims made in the abstract and introduction are backed by the results in Section 5.

   Guidelines:

   - The answer NA means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A No or NA answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: This is an inference-only based approach. The paper lists finetuning-based approaches as possible future work.

   Guidelines:

   - The answer NA means that the paper has no limitation while the answer No means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

Answer: [NA]

Justification: There are no theoretical results.

Guidelines:

- The answer NA means that the paper does not include theoretical results.
- All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
- All assumptions should be clearly stated or referenced in the statement of any theorems.
- The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
- Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
- Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

Answer: [Yes]

Justification: All prompts, model names and hyperparameters have been made available in the paper. This should be sufficient to reproduce the results of the paper even though the code has not been made explicitly available due to legal limitations. SWE-agent, the system that the experiments are based on, is an open-source system.

Guidelines:

- The answer NA means that the paper does not include experiments.
- If the paper includes experiments, a No answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
- If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
- Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
- While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
  (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
  (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
  (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
  (d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

   Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

   Answer: [No]

   Justification: The dataset SWE-bench and the architecture SWE-agent are publicly available. The code for our experiments will be open sourced in the near future. The paper provides the model names, hyperparameters and prompts that can be used to reproduce experimental results with SWE-agent, which is an open-source system.

   Guidelines:

   - The answer NA means that paper does not include experiments requiring code.
   - Please see the NeurIPS code and data submission guidelines (`https://nips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - While we encourage the release of code and data, we understand that this might not be possible, so "No" is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
   - The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://nips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
   - The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
   - At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
   - Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer, etc.) necessary to understand the results?

   Answer: [Yes]

   Justification: There is no training involved. All model hyperparameters are listed in Section 4.2

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
   - The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [No]

   Justification: The temperature is set to 0.0 which means it is a deterministic setting. Conducting several runs of these experiments would not be fruitful and would be computationally expensive.

   Guidelines:

   - The answer NA means that the paper does not include experiments.

- The authors should answer "Yes" if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
- The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
- The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
- The assumptions made should be given (e.g., Normally distributed errors).
- It should be clear whether the error bar is the standard deviation or the standard error of the mean.
- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g. negative error rates).
- If error bars are reported in tables or plots, The authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: Mentioned in Section 4.2.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes]

   Justification: The research confirms with the NeurIPS Code of Ethics.

   Guidelines:

   - The answer NA means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer No, they should explain the special circumstances that require a deviation from the Code of Ethics.
   - The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [NA]

    Justification: NA

    Guidelines:

- The answer NA means that there is no societal impact of the work performed.
- If the authors answer NA or No, they should explain why their work has no societal impact or why the paper does not address societal impact.
- Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
- The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
- The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
- If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pretrained language models, image generators, or scraped datasets)?

    Answer: [NA]

    Justification: NA

    Guidelines:

    - The answer NA means that the paper poses no such risks.
    - Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
    - Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
    - We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

    Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

    Answer: [Yes]

    Justification: Open source works such as SWE-bench, SWE-agent, etc that have been used for our work have been appropriately cited.

    Guidelines:

    - The answer NA means that the paper does not use existing assets.
    - The authors should cite the original paper that produced the code package or dataset.
    - The authors should state which version of the asset is used and, if possible, include a URL.
    - The name of the license (e.g., CC-BY 4.0) should be included for each asset.

- For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
- If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
- For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

    Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

    Answer: [NA]

    Justification: We have not yet released the code although we plan to do it in the near future.

    Guidelines:

    - The answer NA means that the paper does not release new assets.
    - Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
    - The paper should discuss whether and how consent was obtained from people whose asset is used.
    - At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

    Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

    Answer: [NA]

    Justification: NA

    Guidelines:

    - The answer NA means that the paper does not involve crowdsourcing nor research with human subjects.
    - Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
    - According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

    Answer: [NA]

    Justification: NA

    Guidelines:

    - The answer NA means that the paper does not involve crowdsourcing nor research with human subjects.

- Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
- We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
- For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

    Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does not impact the core methodology, scientific rigorousness, or originality of the research, declaration is not required.

    Answer: [Yes]

    Justification: The models and hyperparameters used as part of our experimental setup are documented in Section 4.2. The methodology is described in Section 3.

    Guidelines:

    - The answer NA means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
    - Please refer to our LLM policy (`https://neurips.cc/Conferences/2025/LLM`) for what should or should not be described.