PaiChecker: Uncovering and Checking PR-Issue Misalignment in SWE-Bench-Like Benchmarks

Manyi Wang
The Chinese University of Hong Kong, Shenzhen
China
manyiwang@link.cuhk.edu.cn

Junjielong Xu*
The Chinese University of Hong Kong, Shenzhen
China
junjielongxu@link.cuhk.edu.cn

Pinjia He*
The Chinese University of Hong Kong, Shenzhen
China
hepinjia@cuhk.edu.cn

Abstract

SWE-bench-like benchmarks are widely used for evaluating LLM's issue resolution capability. They typically follow a common construction pipeline: each PR (Pull Request) is paired with its linked issue by extracting issue references from the PR description; the issue description is used as the problem statement, and the PR patch serves as the test oracle. However, due to the inherent complexity of developing and maintaining large repositories, such PR-Issue pairings are often misaligned in practice. In this work, we systematically study SWE-bench Verified instances, finding that 13.6% exhibit misalignment across five patterns in eleven fine-grained scenarios. To enable reliable and scalable construction of those benchmarks in the future, we propose PaiChecker, a multi-agent system for checking PR-Issue misalignment in SWE-bench-like benchmarks. Specifically, PaiChecker adopts a three-phase design that combines specific pattern identification, cross-agent label synthesis, and code-level validation, thereby enabling more accurate, generalizable, and progressively verified detection. Experiments on SWE-Gym and SWE-bench Multilingual show that PaiChecker achieves the best performance across all four LLM backbones, reaching up to 92.12% and 91.67% binary accuracy, respectively.

ACM Reference Format:
Manyi Wang, Junjielong Xu, and Pinjia He. 2026. PaiChecker: Uncovering and Checking PR-Issue Misalignment in SWE-Bench-Like Benchmarks. In *Proceedings of the 41st IEEE/ACM International Conference on Automated Software Engineering (ASE '26), October 12–16, 2026, Munich, Germany*. ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/3832783.3837557

1 Introduction

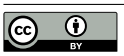
The emergence of large language models (LLMs) capable of code generation and program repair has spurred the need for rigorous, real-world benchmarks [25, 27–29, 46]. Among them, SWE-bench [29] has become one of the most influential benchmarks for evaluating LLMs on automated issue solving tasks [38, 44, 47, 49, 54]. SWE-bench constructs task instances from real-world GitHub repositories, using issue descriptions as problem statements and the corresponding pull request (PR) patches as test oracles. Following its success, several SWE-bench-like benchmarks have emerged to address various limitations, a non-exhaustive list includes SWE-bench Verified [23], SWE-Gym [32], SWE-PolyBench [35], SWE-Smith [48], and SWE-Bench-Live [52]. These benchmarks largely adopt the same construction pipeline: they filter PRs by quality criteria, then use regular expressions to extract linked issues from PR descriptions, forming PR-Issue pairs as task instances.

However, this construction pipeline rests on a critical assumption: that each PR is *well-aligned* with its linked issue, i.e., the PR exclusively and completely addresses the stated problem, and the issue fully specifies what the PR solves. In practice, this assumption is frequently violated. A PR may bundle fixes for multiple issues, serve as a follow-up to a previous incomplete fix, introduce new defects alongside the intended fix, or implement details clarified only in later discussions rather than the original description. Such misalignment renders the benchmark instance problematic: *the problem statement may not fully describe the expected solution, or the test oracle may evaluate aspects unrelated to the stated problem*, directly undermining evaluation fairness and reliability. Beyond evaluation, the same construction pipeline also underlies training-oriented datasets for code LLMs and software agents. Misaligned PR-Issue pairs therefore become noisy supervision, encouraging models to learn incomplete, unrelated, or even defective patches rather than fixes faithful to the stated issue.

In this paper, we present a systematic study of *PR-Issue misalignment* in SWE-bench-like benchmarks, where *PR-Issue misalignment* refers to discrepancies between a *pull request code implementation* and the *issue description* it is linked to in the dataset. We manually analyze all 500 instances in SWE-bench Verified [23] and find that 13.6% exhibit misalignment. We further classify these misalignments into five patterns spanning 11 fine-grained scenarios. To quantify the impact of misalignment, we correlate it with agent resolution rates and find that 41.2% of instances never resolved by any of the 131 leaderboard agents are misaligned.

These findings motivate the need for automated detection and categorization of PR–Issue misalignment at scale. However, detecting misalignment directly from the code implementation and issue description is challenging, as the intent reflected in code is inherently ambiguous. This ambiguity arises from the nature of code changes: in complex cases, the intended behavior may depend on modifications spanning hundreds or thousands of lines, while in simpler cases, the implemented behavior may still be incomplete or defective. By contrast, PR-side textual artifacts, such as PR descriptions and discussions, often summarize changes at a level of abstraction similar to issues [8–10], and their shared natural-language form makes the comparison between them far more straightforward and reliable. This asymmetry between textual artifacts and code motivates the key design principle of our approach, namely the

*Junjielong Xu and Pinjia He are the corresponding authors.



ASE '26, Munich, Germany

ACM ISBN 979-8-4007-2882-2/2026/10
https://doi.org/10.1145/3832783.3837557

text-driven, code-validation principle, which first assesses alignment based on textual evidence and then validates it against code.

According to this principle, the task is well suited to LLM reasoning and tool-augmented agents, because it is fundamentally a semantic comparison task. We therefore adapt Mini-SWE-Agent [47], augmenting it with task-specific prompts and GitHub API access, as a representative general-purpose agent baseline. However, this approach has three limitations: (1) it lacks specialization: different misalignment patterns require different inputs and reasoning workflows, but a single prompt conflates these heterogeneous requirements and often misses subtle yet critical evidence. (2) it has limited generalizability: when no predefined pattern applies, it tends to force-fit the instance into the nearest known category, even though real-world misalignments often fall outside the taxonomy. (3) it lacks verification mechanism: its predicted label may not match its own textual reasoning as well as the code implementation. We illustrate these limitations in detail in Section 2.3.

To address these limitations, we propose PaiChecker (**P**R-**I**ssue **A**lignment Checker), a three-phase multi-agent framework guided by a *text-driven, code-validation* principle. The first two phases focus on textual analysis, and the third performs code-level verification. Specifically, Phase I addresses limitation(1) through three specialized subagents, each with a focused artifact subset and tailored workflow. Phase II addresses limitation(2) via a coordinator that synthesizes cross-subagent evidence to identify misalignment beyond known patterns. For limitation(3), PaiChecker introduces a two-level *self-correction* mechanism: Phase II re-verifies each label against its textual evidence, and Phase III validates it against the actual code implementation.

We evaluate PaiChecker on SWE-Gym [32] and SWE-bench Multilingual [29], comparing it against three prompting baselines, and four agent-framework baselines. Using four state-of-the-art LLMs as backbones (GPT-5.3 Codex [31], Qwen-3.5 Plus [34], Gemini-3.1-Pro Preview [26], and Claude-Sonnet-4.6 [16]), PaiChecker achieves the best performance across all four LLM backbones, reaching up to 92.12% binary accuracy and 84.66% exact match, outperforming the strongest baseline by 5.13–12.39 accuracy points and 9.02–17.76 exact-match points on SWE-Gym, and by 2.33–5.67 accuracy points and 3.66–8.00 exact-match points on SWE-bench Multilingual. Ablation studies confirm each component's contribution.

In summary, this paper makes the following contributions:

- **Problem Formulation.** We identify PR-Issue misalignment as a systematic construction problem in SWE-bench-like benchmarks.
- **Empirical Study.** We analyze all 500 SWE-bench Verified instances, deriving a taxonomy of five patterns with 11 scenarios and finding 13.6% misaligned (Section 2).
- **Detection Framework.** We propose PaiChecker, a multi-agent framework for automated detection and categorization of PR-Issue misalignment (Section 3).
- **Evaluation.** Experiments on SWE-Gym and SWE-bench Multilingual with four state-of-the-art LLMs demonstrate the effectiveness of PaiChecker. (Sections 5).
- **Data Availability.** We release all annotated data and PaiChecker to support future benchmark curation [13].

2 Preliminary Study and Motivation

In this section, we systematically investigate the existence, prevalence, and impact of PR-Issue misalignment in SWE-bench-like benchmarks. Our findings provide the empirical foundation and motivation for our automated misalignment checker PaiChecker. This study focuses on two research questions:

- **RQ1 (Taxonomy and Prevalence):** What are the types of PR-Issue misalignments, and how prevalent are they?
- **RQ2 (Impact):** How does PR-Issue misalignment affect agent evaluation reliability?

2.1 Study Design

2.1.1 Dataset. We conduct our study on SWE-bench Verified [23], a human-validated subset of 500 SWE-Bench [29] instances. We select it for two reasons: (1) its human validation represents a high-quality baseline, so misalignments found here likely reflect broader issues in other SWE-bench-like benchmarks; (2) its official experiments repository [2] provides per-instance resolution data for 131 leaderboard agents, enabling quantitative analysis for RQ2.

2.1.2 Method. **Manual Examination Process.** Following established practices in qualitative SE research [18, 24], we apply open coding [36] to derive a misalignment taxonomy (RQ1). First, we collect per instance: issue-side artifacts (description, discussion), PR-side artifacts (description, discussion, code review, commits, changed files), and cross-reference metadata. Then, we code all 500 instances, iteratively grouping codes into higher-level categories to form a draft taxonomy; a single instance may exhibit multiple patterns. Last, the draft taxonomy was reviewed by the second author; disagreements were resolved with a third author until consensus.

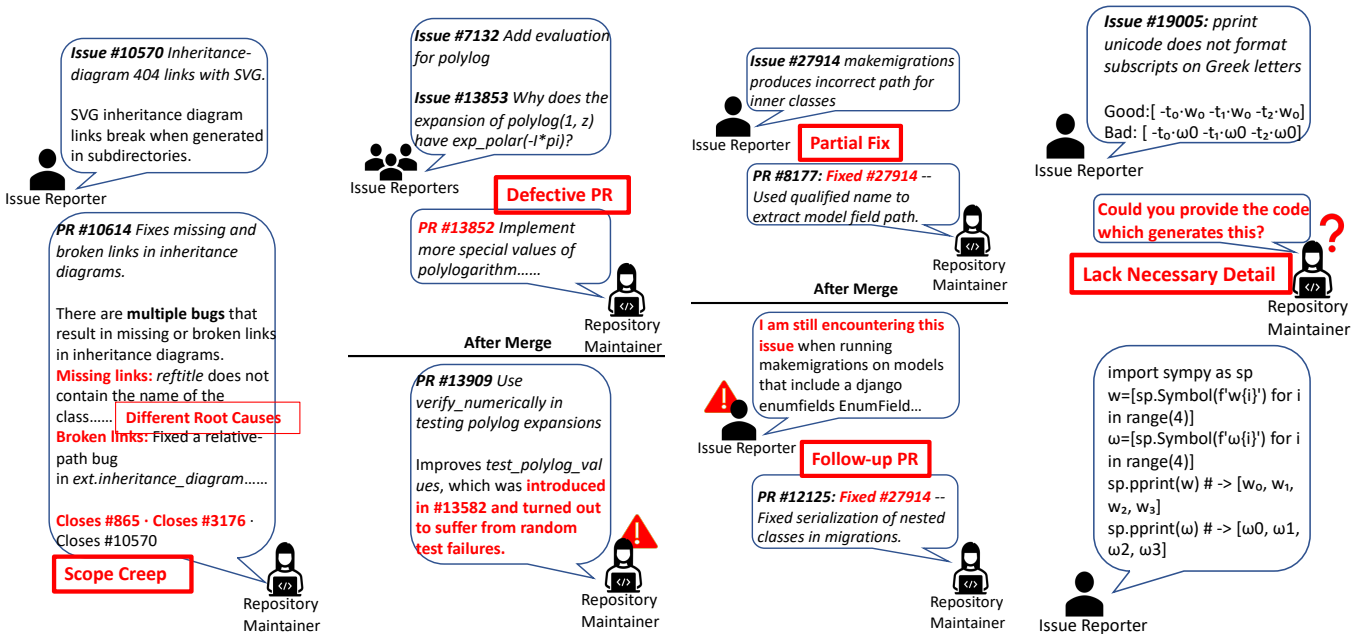
Leaderboard Data Collection. To assess the impact of misalignment on evaluation reliability, we collect per-instance resolution data from the official SWE-bench experiments repository [2], recording per-instance resolution counts for 131 leaderboard agents.

2.2 Results and Analysis

2.2.1 RQ1: Taxonomy and Prevalence of Misalignment. Table 1 summarizes the taxonomy of five patterns with 11 fine-grained scenarios, with instance counts in parentheses. The five patterns correspond to distinct root causes in the PR–Issue pairing pipeline. They are orthogonal by design: an instance may carry multiple labels when independent failure modes co-occur, while each label still denotes a separate reason for misalignment. This keeps counts interpretable as pattern-level evidence rather than catch-all defects. Our classification also follows the *text-driven, code-validation* principle (Section 1). We elaborate on each pattern below.

PR Scope Creep (SC) occurs when the PR delivers functionality beyond what the issue requests. We consider this a misalignment because it creates a mismatch between the task and its evaluation: the test patch validates requirements absent from the `problem_statement`, penalizing agents that correctly solve the stated problem. Specifically, SC-3 refers to pre-existing bugs in the repository, whereas SC-4 refers to patches addressing unrelated Issues.

Example. Figure 1a shows `sphinx-doc__sphinx-10614`: the `problem_statement` captures only issue `#10570`, but the PR closes



(a) SC-1(sphinxdoc__sphinx-10614): The PR closed three distinct issues; the problem statement covers only one.

(b) DP-1(sympy__sympy-13852): The PR introduces random test failures that require a follow-up fix.

(c) FP-2(django__django-12125): The PR addresses only the remaining bug not the full issue.

(d) IS-1(sympy__sympy-20916): The issue is too vague, confusing even the maintainer, so the model likely won't understand either.

Figure 1: Examples of four misalignment patterns.

Table 1: Taxonomy of PR-Issue misalignment patterns in SWE-bench Verified (68/500 instances). Instances may exhibit multiple patterns.

Pattern	Scenario
SC: PR Scope Creep	**SC-1 (12):** Resolves multiple issues; only one specified **SC-2 (2):** Adds features beyond issue description **SC-3 (2):** Bundles fixes for bugs not in the issue **SC-4 (6):** Extra patches for other issues alongside fix
DP: Defective PR	**DP-1 (14):** Introduces new bugs requiring follow-up fixes **DP-2 (16):** Incomplete solution requiring follow-up
IS: Incomplete Specification	**IS-1 (12):** Details supplemented by reporter in discussion **IS-2 (6):** Addresses new problem from later discussion
FP: Follow-up PR	**FP-1 (2):** Fixes bugs introduced by a previous PR **FP-2 (1):** Supplements a prior PR
UL: Unspecified Literal	**UL-1 (1):** Asserts exact literals (*e.g.* messages) not in issue

three distinct issues. An agent correctly resolving `#10570` may still fail if it does not coincidentally address the two unstated issues.

Defective PR (DP) captures cases where the PR itself is flawed—introducing new bugs or providing an incomplete fix which requires subsequent corrective work. We consider this a type of misalignment because the test patch may encode buggy behavior as the expected output or validate only a partial solution.

Example. Figure 1b shows `sympy__sympy-13852`. The PR fixes the reported issue but introduces random test failures, requiring a follow-up fix. Yet the benchmark treats the original defective PR as ground truth.

Follow-up PR (FP) is the converse of DP: the PR supplements or fixes a previous PR for the same issue. We consider this a misalignment because the codebase already contains the earlier PR's changes, so the model benefits from prior work and the evaluation does not reflect its true capability.

Example. Figure 1c shows `django__django12125`: a previous PR partially fixed the inner-class path issue for `Field` subclasses, but `Enum` serialization remained broken. The current PR addresses only that remaining gap; the codebase already contains the earlier fix.

Incomplete Specification (IS) occurs when the original issue description is supplemented or revised in later discussion. We consider this a misalignment because agents receive only the initial description, which in such cases lacks information needed for resolution. Following the standard bug report format, which includes *code to reproduce, actual behavior, and expected behavior* [4–7, 19], we distinguish two forms of incomplete specification, both grounded in maintainer consensus. (1)The issue is *incomplete* when a maintainer explicitly requests a missing component. This type of misalignment resembles the "unspecified" criterion in SWE-bench-verified [1], but is more objective: insufficiency is only flagged when the repository maintainer also fails to understand the issue. For example, an issue specified mainly through an external link is valid if developers raise no clarification request, but misaligned once such a request appears. We adopt this standard because if a maintainer, a human expert

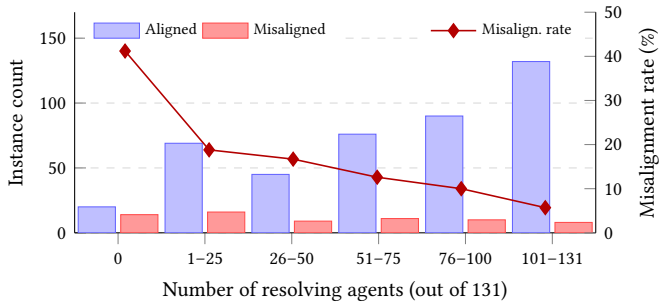


Figure 2: Misalignment rate decreases monotonically as instances are resolved by more agents. Bars show instance counts (left axis); the line shows the misalignment rate within each bucket (right axis). Among the 34 never-resolved instances, 41.2% are misaligned, compared to only 5.7% among instances solved by >100 agents.

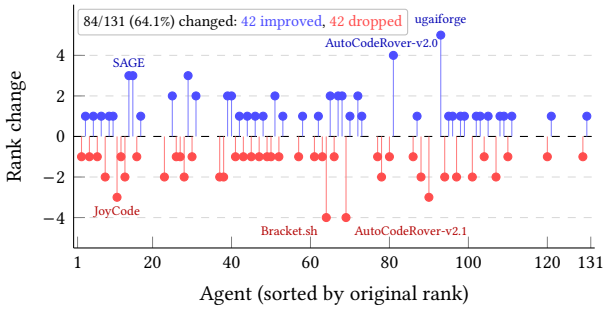


Figure 3: Per-agent rank change after excluding 68 misaligned instances, ordered by original rank. Each stem shows the magnitude and direction of rank movement (blue: improved; red: dropped).

familiar with the project, cannot infer the requirements from the description alone, a language model would likely struggle to do so as well. (2) The issue is *redefined* when a component is revised by maintainer agreement (*e.g.* the expected behavior). Unlike the incomplete case, here the original description does provide necessary components, but one or more of them are superseded through maintainer discussion. An agent following the original description would therefore target an outdated requirement.

Example. Figure 1d shows `sympy__sympy-20916`: the issue describes incorrect formatting and expected output, but lacks necessary detail. At a maintainer's request, the reporter supplies code containing key information absent from the `problem_statement`.

Unspecified Literal (UL) occurs when the test patch asserts exact literal values (*e.g.* exception messages) introduced by the PR but absent from the issue. We consider this a type of misalignment because it penalizes agents that produce semantically correct outputs differing in literal form. This reflects an overly strict test oracle that requires implementation-specific details unrecoverable from the issue description alone.

Example. `astropy__astropy-13033`: the issue requests "*an exception that tells the user which required columns are actually missing*" without specifying the exact wording. However, the test patch asserts the precise string "`TimeSeries object is invalid - expected ['time','a'] as the first columns but found ['time','b']`", rejecting correct fixes with different wording.

Finding: 13.6% (68/500) of SWE-bench Verified instances exhibit PR-Issue misalignment. Defective PR (44.1%), PR Scope Creep (32.4%), and Incomplete Specification (26.5%) are the three most prevalent pattern families.

2.2.2 RQ2: Impact on Evaluation Reliability. In this RQ, we aim to investigate the practical impact of PR-Issue misalignment. We assess impact in two ways: correlating misalignment labels with per-instance resolution counts across 131 leaderboard agents, and recomputing the leaderboard [3] after excluding misaligned instances.

Misalignment–Resolution Correlation. *(1) Methodology.* We bucket the 500 instances by the number of resolving agents and compute each bucket's misalignment rate (Figure 2). *(2) Results and Discussion.* Among the 500 instances, 34 have never been resolved by any of the 131 leaderboard agents. Of these, 14 (41.2%) exhibit PR-Issue misalignment, suggesting that the hardest instances may correlate with agents due to unstated evaluation requirements, not task complexity alone. More broadly, Figure 2 shows a monotonic decrease in misalignment rate as instances are resolved by more agents: from 41.2% among never-resolved instances to 5.7% for those resolved by over 100 agents. Well-aligned instances are solvable by many agents, whereas misaligned instances introduce extraneous difficulty that few agents overcome by coincidence. These findings are consistent with OpenAI's concurrent analysis [14], though we arrive at a similar conclusion from a distinct perspective: systematic PR-Issue misalignment in benchmark construction.

Leaderboard Re-ranking. *(1) Methodology.* Using official resolution data [2], we recompute each agent's pass rate on the 432 aligned instances after excluding the 68 misaligned ones, then re-rank agents by the new pass rate. *(2) Results and Discussion.* Figure 3 shows that 84 of 131 agents (64.1%) change rank. Most shifts are within 1–2 ranks, but outliers reach +5 (ugaiforge) and −4 (Bracket.sh, AutoCodeRover-v2.1), and 9 of the top 10 positions change. Pass rates rise by 2.77 pp on average, with gains ranging from −0.24 to +4.49 pp, indicating uneven reliance on misaligned instances.

Finding: Misalignment rate decreases from 41.2% to 5.7% as resolution count rises, and excluding 68 misaligned instances changes 64.1% of agent rankings. This correlation is not causal evidence, but shows that construction defects add extraneous difficulty and distort standings unevenly.

2.3 Motivating Examples

In this subsection, we use concrete examples to illustrate why general prompting and agentic approaches fall short for misalignment detection, motivating the design of PAICHECKER. We adapt Mini-SWE-Agent [47] as a representative tool-augmented agent by adjusting its prompt and equipping it with GitHub API Access, and include Chain-of-Thought (CoT) [43] prompting as a representative prompting technique, both on Gemini-3.1-Pro Preview. We demonstrate limitations common to these monolithic approaches.

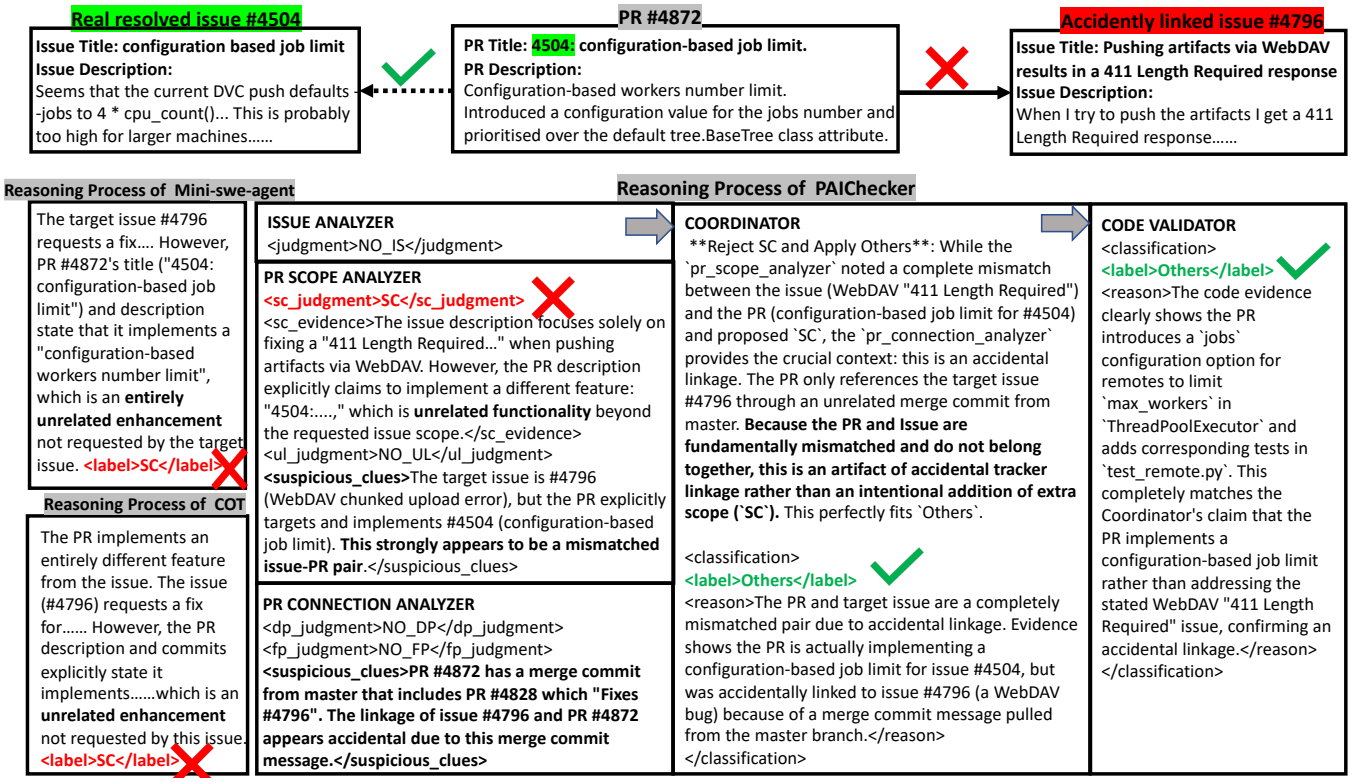


Figure 4: Motivating Example `iterative__dvc-4872`: PR #4872 is accidentally linked to issue #4796 but actually fixes issue #4504. Backbone: Gemini-3.1-Pro Preview. Mini-SWE-Agent and CoT force-classify as SC; PaiChecker's coordinator correctly drops SC and assigns *Others* by synthesizing suspicious clues from two subagents.

We observe a monolithic approach struggles when all artifacts and detection rules are packed into a single prompt, because each misalignment pattern demands a distinct reasoning workflow over a different artifact subset: SC compares issue scope against PR-side claims, DP traces cross-PR relationships, and IS tracks specification changes in discussion threads. A single prompt cannot specialize in all simultaneously: detection logic for one pattern interferes with another, diluting model's focus; and artifacts critical to one pattern, such as cross-issue references for DP, are overshadowed by more voluminous ones. *This motivates Phase I of PaiChecker: decomposing detection into three dedicated subagents, each with a focused artifact subset and workflow tailored to its target pattern.*

We further examine a beyond-taxonomy case in Figure 4, where a PR and its linked issue are fundamentally unrelated due to accidental tracker linkage. This example exposes two problems with monolithic approaches. First, both Mini-SWE-Agent and CoT fail to assign the *Others* label despite having the option, defaulting instead to SC, the nearest predefined category. *Others* requires concluding that no predefined pattern applies, a harder judgment than matching any single one. *This presents generalizability challenge and motivates Phase II of PaiChecker: a coordinator that collects suspicious clues from subagents before label commitment, allowing beyond-taxonomy signals to surface explicitly.* Second, both approaches incorrectly assign SC, creating a reasoning–label inconsistency: their reasoning describes the PR as "entirely unrelated," which directly contradicts SC. Similar inconsistencies appear in other cases (*e.g.* `MONAI-6793`), where cross-issue references are mistaken for scope creep despite no actual scope expansion. Once committed in a single pass, such errors cannot be revisited. *This presents textual-level inconsistency challenge and motivates the coordinator to also re-verify each label against its cited evidence and drop inconsistent labels.*

Following the *text-driven, code-validation* principle (Section 2.2.1), code-level verification is necessary to catch false positives that textual analysis alone cannot catch. For instance, a PR claiming to fix multiple issues may actually address duplicates sharing one root cause, or claimed extra functionality may not materialize in the code diff. *This presents code-level inconsistency challenge and motivates Phase III of PaiChecker: a code validator that cross-references textual claims against implementation evidence.* Together with the coordinator's text-level re-verification, this forms a *two-level self-correction mechanism* to ensure the accuracy of final output.

3 Methodology: PaiChecker

In this section, we present PaiChecker, a multi-agent framework for detecting and categorizing PR-Issue misalignment. As shown in Figure 5, PaiChecker has three phases: Phase I performs pattern-specific identification with three focused subagents, Phase II synthesizes their reports into preliminary labels and can assign *Others*,

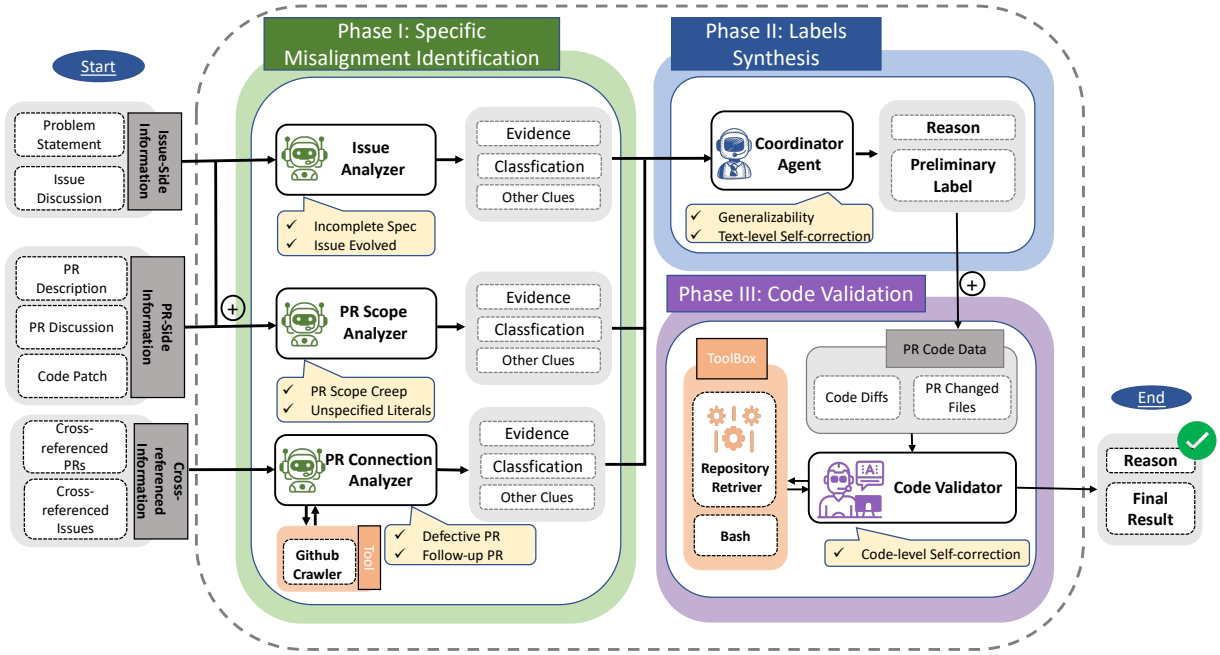


Figure 5: Workflow of PaiChecker. Phase I performs specific misalignment identification through three specialized subagents. Phase II synthesizes their outputs into a preliminary label and rationale. Phase III validates the preliminary result against code-level evidence and supplementary GitHub context to produce the final decision.

and Phase III validates the textual judgment against code-level evidence.

Across all three phases, we enforce a separation of responsibility: the authority to assign predefined labels is exclusive to Phase I; Phases II and III hold only veto power. This ensures that detection decisions are made in the focused context where relevant evidence is most concentrated, while subsequent phases serve as progressive self-correction filters.

3.1 Phase I: Specific Misalignment Identification

In this subsection, we describe Phase I, which detects the five misalignment patterns using three parallel subagents, each focused on distinct reasoning: the Issue Analyzer targets Incomplete Specification (IS), primarily drawing on issue-side artifacts; the PR Scope Analyzer targets PR Scope Creep (SC) and Unspecified Literal (UL) through issue-versus-PR scope comparison; and the PR Connection Analyzer targets Defective PR (DP) and Follow-up PR (FP) through temporal cross-PR reasoning. This grouping keeps each subagent's context compact and reasoning focused. All three produce a uniform structured output: (i) a *judgment* per target pattern, (ii) supporting *evidence* for text-level self-correction, and (iii) *suspicious clues*—anomalies outside the subagent's target patterns that support beyond-taxonomy generalization.

3.1.1 Issue Analyzer. The Issue Analyzer operationalizes IS detection by comparing the issue description against its discussion thread. As mentioned in Section 2.2.1: a well-formed bug report comprises three key components: reproduction code, actual behavior, and expected behavior. Based on this, the subagent performs two checks. First, whether the issue is incomplete: it identifies whether a maintainer explicitly requests the reporter to supplement any of these components, grounding the judgment in community consensus rather than subjective assessment. Second, whether the issue has been redefined: it checks whether any of these components differs between the original description and the discussion; a change in any component signals that the addressed problem has shifted.

3.1.2 PR Scope Analyzer. The PR Scope Analyzer detects SC and UL in a unified pass. Both patterns require establishing the issue's requested scope as a baseline and checking whether the PR exceeds it, so a single subagent establishes the baseline once and verifies both patterns together. This subagent receives the issue description as scope baseline, PR-side text for what the PR claims to deliver, and both patches for UL detection.

Following the text-driven, code-validation principle, SC is triggered only when PR-side textual artifacts explicitly claim work beyond the issue's scope. UL is triggered when all three conditions hold: (1) the test patch asserts a hardcoded literal (*e.g.* a string, number, or exception message); (2) the literal is newly added in the code patch as a fixed constant, rather than read from existing metadata; and (3) it does not appear verbatim in the issue description.

3.1.3 PR Connection Analyzer. The PR Connection Analyzer detects DP and FP, both involve temporal relationships between the current PR and other PRs for the same issue. Tracing such cross-PR dependency chains requires information beyond the current instance's artifacts, so we equip this subagent with GitHub API access to actively search for and retrieve related PRs.

For DP detection, the agent searches for PRs merged after the current one that still address the same issue. Each candidate is verified for merge status to distinguish corrective follow-ups from unrelated references. For FP detection, the agent searches for earlier merged PRs that partially addressed the same issue, checking whether the current PR is a supplementary contribution rather than an independent fix, based on merge chronology and follow-up language in PR descriptions.

3.2 Phase II: Label Synthesis

In this subsection, we describe Phase II, which deploys a coordinator agent to synthesize the three subagents' outputs into a preliminary label set. The coordinator has two responsibilities: beyond-taxonomy generalization and text-level self-correction. In both roles, it holds only veto power over predefined labels: it can remove but not add any predefined label not assigned by Phase I, because it lacks each subagent's specialized context.

For beyond-taxonomy generalization, the coordinator collects suspicious clues from all three subagents and evaluates whether they collectively point to a misalignment outside the five predefined categories; when they do, it assigns the *Others* label. This design decouples anomaly discovery from judgment: subagents surface anomalies within their focused scope, and the coordinator synthesizes all three evidence streams. For text-level self-correction, the coordinator independently assesses whether each judgment is adequately supported by its evidence; if the evidence is insufficient or self-contradictory, it drops the label, catching over-detections before they reach code validation.

3.3 Phase III: Code Validation

In this subsection, we describe Phase III, which deploys a code validator agent responsible for code-level self-correction. The preceding phases make detection decisions from textual evidence; the code validator cross-references these decisions against the actual implementation to filter false positives that textual reasoning alone cannot catch. It can only remove labels, not add them.

To perform this verification, the code validator is augmented with repository-level file retrieval capability, enabling it to download full source files and inspect changed files in their complete context. This is necessary because the `patch` contains only changed hunks, and determining whether a claimed misalignment materializes often requires inspecting surrounding code.

4 Evaluation Design

This section details the evaluation setup for PaiChecker. We aim to address the following research questions:

- **RQ3 (Effectiveness):** How effective is PaiChecker in detecting and categorizing PR-Issue misalignment?
- **RQ4 (Ablation):** How does each component of PaiChecker contribute to the overall detection performance?
- **RQ5 (Label Changes):** How and why do labels change across the three pipeline phases?

Table 2: Statistics of the evaluation dataset. An instance may carry multiple labels.

Category	SWE-Gym		SWE-bench Multilingual	
	Count	Percentage	Count	Percentage
Total instances	2,438	100%	300	100%
Aligned	1,358	55.7%	227	75.7%
Misaligned	1,080	44.3%	73	24.3%
SC	428	17.6%	33	11.0%
DP	150	6.2%	24	8.0%
IS	341	14.0%	13	4.3%
UL	387	15.9%	9	3.0%
FP	109	4.5%	2	0.7%
Others	16	0.7%	1	0.3%

4.1 Evaluation Dataset

We evaluate on two datasets disjoint from our preliminary study: SWE-Gym [32] (2,438 Python tasks) and SWE-bench Multilingual [29] (300 tasks covering C, C++, Go, Java, JavaScript/TypeScript, PHP, Ruby, and Rust). Selection followed five criteria: (i) SWE-bench-style automated PR–Issue pairing; (ii) no manual specification rewriting, so raw defects remain observable; (iii) feasible for full manual annotation yet sufficiently large; (iv) multi-repo and widely adopted; and (v) PR identifiers available for full-content access. Two annotators with development experience independently labeled all instances following our taxonomy (Section 2.2.1), guided by structured questions adapted from SWE-bench annotation instructions [1]. They achieved Cohen's Kappa [33] of $\kappa = 0.91$ on binary judgment and $\kappa = 0.86$ on fine-grained labeling; disagreements were resolved by a third author. Table 2 summarizes the dataset statistics.

Misalignment rates differ across datasets: 13.6% in SWE-bench Verified (Table 1), 44.3% in SWE-Gym, and 24.3% in SWE-bench Multilingual, largely reflecting SWE-bench Verified's human curation. Patterns tied to underspecification and test coverage drop most sharply in Verified: UL decreases from 15.9% (SWE-Gym) and 3.0% (Multilingual) to 0.2%, IS from 14.0% and 4.3% to 3.6%, and SC from 17.6% and 11.0% to 4.4%. In contrast, deeper PR–Issue relationship defects remain comparable: DP stays near 6.0–8.0%, and FP remains low across all three datasets (0.6%, 4.5%, 0.7%). These defects are harder to infer from the task alone and thus more likely to escape human curation.

4.2 Compared Techniques

We compare PaiChecker against three typical prompting techniques and four agent approaches: (1)*Zero-Shot Prompting* [21] provides the LLM with the task description, taxonomy definitions, and instance artifacts for direct classification. (2)*Few-Shot Prompting* [20] augments the zero-shot prompt with annotated examples from SWE-bench Verified. (3)*Chain-of-Thought Prompting(CoT)* [43] further adds explicit reasoning steps. (4)*Mini-SWE-Agent* [47] is a general-purpose agent baseline: we change the default *Bash* tool to *Curl* and provide the same instructions and artifact context as PaiChecker to ensure a fair comparison. (5)*OpenHands* [41] is an

agent-framework baseline evaluated with the same artifact context. (6) *Claude Code* [11] and (7) *Codex* [12] with their officially supported models.

4.3 Evaluation Metrics

We evaluate PR-Issue misalignment detection from two complementary perspectives: (1) a **binary classification** task that judges whether an instance is aligned or misaligned, and (2) a **multi-label classification** task that identifies the specific misalignment patterns, where an instance may exhibit multiple patterns simultaneously. Accordingly, we adopt the following metrics:

Binary Detection. We measure accuracy, precision, recall, and F1 for the binary detection task.

Multi-Class Categorization. Exact Match (EM) serves as the primary metric, requiring the predicted label set to exactly match the ground truth. We supplement EM with macro-averaged accuracy, precision, recall, and F1. For per-label analysis, we report only F1 due to space constraints; F1 provides a balanced measure of precision and recall, making it sufficient for diagnostic purposes.

4.4 Backbone LLMs

We evaluate PaiChecker and baselines across four state-of-the-art LLM backbones, including three closed-source models GPT-5.3-Codex [31], Claude-Sonnet-4.6 [16], Gemini-3.1-Pro-Preview [26], and one open-sourced model Qwen-3.5-Plus [34]. All models were accessed through their respective APIs.

5 Results and Analysis

5.1 RQ3: Effectiveness of PaiChecker

In this RQ, we evaluate the effectiveness of PaiChecker from two complementary perspectives: *binary misalignment detection*—whether an instance exhibits any misalignment (Section 5.1.1), and *multi-class categorization*—whether the specific misalignment patterns are correctly identified (Section 5.1.2).

5.1.1 Effectiveness of Detecting Misalignment. Table 3 reports binary detection results. PaiChecker achieves the best Accuracy and F1 across all backbones on both datasets. On SWE-Gym, it reaches 92.12% Accuracy and 91.10% F1 with Gemini; on SWE-bench Multilingual, it reaches 91.67% Accuracy and 84.28% F1 with Claude.

On SWE-Gym, PaiChecker improves Accuracy over the best baseline per backbone by 5.13–12.39 points: Mini-SWE-Agent is strongest on Claude/Gemini (82.36%/83.59%) but trails by 5.13/8.53 points, while GPT/Qwen gaps are 12.39/5.09 points. The advantage also holds on SWE-bench Multilingual, where PaiChecker reaches 85.33–91.67% Accuracy versus 83.00–88.00% for the best baseline. Because Multilingual has fewer misaligned instances (24.3% vs. 44.3% in SWE-Gym), some baselines trade recall for precision; for example, OpenHands on Qwen has higher Precision than PaiChecker (79.31% vs. 78.43%) but much lower Recall (31.51%), F1 (45.10% vs. 64.52%), and Accuracy (81.33% vs. 85.33%).

5.1.2 Effectiveness of Categorization. Table 4 shows multi-class categorization results. PaiChecker achieves the best EM and Macro F1 across all backbones on both datasets. On SWE-Gym, EM ranges from 70.59% (Qwen) to 84.66% (Gemini), and Macro F1 from 58.98% to 79.21%; on SWE-bench Multilingual, Gemini reaches 90.67% EM and 71.77% Macro F1. We omit per-label F1 for FP and Others on Multilingual, as the 2 FP and 1 Others instances are insufficient for meaningful statistical comparison.

On SWE-Gym, PaiChecker outperforms the strongest baseline by 9.02, 10.74, 15.26, and 17.76 EM points on Qwen, Claude, Gemini, and GPT, respectively. Although the "Others" category is especially challenging due to its rarity (0.7%, 16 instances), PaiChecker still achieves the best performance across all backbones. On SWE-bench Multilingual, where the lower misalignment rate (24.3%) leads to higher baseline EM, PaiChecker still improves EM by 3.66–8.00 points and achieves 42.38–71.85% Macro F1, compared with 28.64–48.03% for the strongest baseline. The metrics are less stable due to relatively few samples. For example, on Qwen, Mini-SWE-Agent attains higher Macro Precision (56.29% vs. 50.40%) through more conservative predictions, but this comes with lower Recall (22.61% vs. 37.89%), EM (73.00% vs. 83.33%), and Macro F1 (28.64% vs. 42.38%).

Finding: PaiChecker achieves the best binary Accuracy and multi-class EM across all four backbones on both datasets, leading the best baseline by 5.13–12.39 Accuracy and 9.02–17.76 EM points on SWE-Gym, and by 2.33–5.67 Accuracy and 3.66–8.00 EM points on SWE-bench Multilingual.

Table 3: Binary misalignment detection performance (%). Accuracy, Precision, Recall, and F1 are computed for the misaligned class; all differences are significant at $p < 0.001$. GPT: GPT-5.3-Codex; Claude: Claude-Sonnet-4.6; Gemini: Gemini-3.1-Pro-Preview; Qwen: Qwen-3.5-Plus. ZS: Zero-Shot; FS: Few-Shot; MSA: Mini-SWE-Agent; OH: OpenHands; CC: Claude Code; CX: Codex.

		SWE-Gym				Multilingual			
LLM	**Meth.**	**BA**	**BP**	**BR**	**BF1**	**BA**	**BP**	**BR**	**BF1**
GPT	ZS	74.36	73.14	66.57	69.70	79.67	57.32	64.38	60.65
	FS	75.72	75.31	67.22	71.04	79.33	57.33	58.90	58.11
	CoT	75.72	73.33	71.02	72.15	83.33	68.25	58.90	63.24
	MSA	72.48	83.04	47.59	60.51	88.00	84.91	61.64	71.43
	OH	70.10	79.59	48.13	59.99	83.67	83.33	41.10	55.05
	CX	62.34	61.18	44.60	51.59	82.67	81.82	36.99	50.94
	Ours	**88.11**	**83.47**	**91.20**	**87.17**	**91.00**	**85.94**	**75.34**	**80.29**
Claude	ZS	67.02	65.23	54.72	59.52	56.33	33.71	82.19	47.81
	FS	69.03	68.24	56.30	61.69	70.33	43.94	79.45	56.59
	CoT	73.54	68.20	75.46	71.65	75.33	49.54	73.97	59.34
	MSA	82.36	81.13	78.43	79.76	86.00	70.13	73.97	72.00
	OH	77.66	76.82	73.24	74.99	75.67	50.00	68.49	57.80
	CC	78.09	75.30	76.20	75.75	72.00	43.82	53.42	48.15
	Ours	**87.49**	**82.76**	**90.65**	**86.52**	**91.67**	**77.91**	**91.78**	**84.28**
Gemini	ZS	74.36	67.93	79.81	73.39	63.67	37.32	72.60	49.30
	FS	81.05	79.37	77.31	78.33	77.67	53.19	68.49	59.88
	CoT	81.09	80.49	75.65	78.00	83.33	64.94	68.49	66.67
	MSA	83.59	83.60	78.33	80.88	59.33	26.67	38.36	31.46
	OH	79.65	82.02	69.26	75.10	86.67	78.95	61.64	69.23
	Ours	**92.12**	**91.19**	**91.02**	**91.10**	**91.33**	**88.52**	**73.97**	**80.60**
Qwen	ZS	74.36	82.92	53.06	64.71	83.00	77.50	42.47	54.87
	FS	73.75	80.73	53.52	64.37	82.33	76.32	39.73	52.25
	CoT	72.76	84.90	46.85	60.38	81.67	76.47	35.62	48.60
	MSA	74.77	77.65	60.46	67.99	74.33	45.00	24.66	31.86
	OH	60.53	65.97	28.25	39.56	81.33	**79.31**	31.51	45.10
	Ours	**80.56**	**90.51**	**62.69**	**74.07**	**85.33**	78.43	**54.79**	**64.52**

5.2 RQ4: Ablation Study

In this RQ, we assess each component's contribution to PaiChecker. We run ablations on the best-performing backbone Gemini-3.1-Pro-Preview, and use phase-wise analysis across all four backbones as complementary evidence.

5.2.1 Approach. To assess component contributions, we construct ten ablated variants on SWE- Gym using Gemini-3.1-Pro-Preview. The variants cover three granularities. First, phase-level ablations remove all Phase I subagents, the Phase II coordinator, or the Phase III code validator. Second, single-subagent ablations remove the Issue Analyzer, PR Scope Analyzer, or PR Connection Analyzer while keeping the other two Phase I subagents. Third, multi-component ablations remove pairs of Phase I subagents or remove both the coordinator and code validator. We compare each variant with the full PaiChecker pipeline using Binary Accuracy and Exact Match.

5.2.2 Ablation Results. Table 5 reports the ablation results for the ten variants on Gemini-3.1-Pro-Preview.

Removing all Phase I subagents causes the largest drop (−24.29 BA, −19.88 EM), confirming the subagent ensemble as the detection backbone. Removing the coordinator (−4.71 BA, −7.22 EM) or the code validator (−3.15 BA, −5.66 EM) causes smaller but meaningful degradation. EM drops exceed BA drops by 1.5× and 1.8× respectively, indicating that Phases II and III primarily refine label-level precision.

Among individual agents, the PR Scope Analyzer has the greatest impact (−18.86 BA, −23.50 EM), consistent with the high prevalence of SC+UL (33.5% of misaligned instances). The Issue Analyzer and PR Connection Analyzer have smaller but still substantial effects (−12.02 and −9.84 EM), matching the lower prevalence of IS (14.0%) and DP+FP (10.7%). Pairwise removals show complementarity, since combined drops exceed either corresponding individual drop.

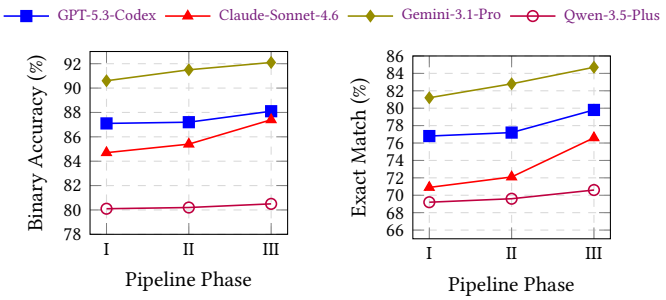
5.2.3 Phase-Wise Progression Analysis. Figure 6 visualizes how BA and EM evolve across the three pipeline phases for all four backbones. Both metrics improve monotonically across all phases and backbones. Phase I→II gains are moderate and backbone-dependent (up to +1.6 EM for Gemini), reflecting the coordinator's conservative reconciliation role. Phase II→III gains are consistently larger, with EM improvements disproportionately exceeding BA gains across all backbones.

Finding: Phase I provides the main detection signal, while Phases II and III primarily refine label-level accuracy. Individual and pairwise removals show that the three Phase I subagents contribute complementary evidence rather than substitutable signals.

5.3 RQ5: Label Changes Analysis

This RQ examines how and why labels change between pipeline phases, analyzing what types of corrections each phase contributes and whether the multi-phase design yields net positive refinement.

5.3.1 Approach. We track label changes between consecutive phases and classify each change as *IC* (Incorrect→Correct) or *CI* (Correct→Incorrect). We measure these transitions for Phase I→II and Phase II→III across all backbones, then manually inspect representative cases to explain the correction mechanisms.



(a) Binary Accuracy **(b) Exact Match**

Figure 6: Binary Accuracy and Exact Match across pipeline phases. Both metrics improve monotonically.

5.3.2 Results. As shown in Table 6, IC consistently outnumbers CI at both phases across all backbones. Phase I→II changes vary more in volume and IC rate, whereas Phase II→III changes are larger overall (55–141) and remain mostly corrective. We next analyze the mechanisms behind these corrections.

Phase I→II. Label changes mainly take two forms: adding *Others* and dropping labels whose cited evidence does not support the assigned pattern. The first often occurs in three scenarios: (1) *partial implementation with altered behavior*, where the PR substitutes a requested behavior; (2) *partial implementation with omitted behavior*, where only part of the request is implemented; and (3) *completely mismatched pairs*, where the PR and issue are unrelated. The second reflects the coordinator's text-level self-correction: aggregating subagent evidence helps identify reasoning–label inconsistencies. For example, Figure 4 shows a case where the reasoning indicates an unrelated PR, contradicting SC. Similarly, in `pandas-dev__pandas-54451`, Phase I flagged SC because the PR references both issue #54617 and #54167. The coordinator cross-referenced the PR Connection Analyzer, found the multi-issue reference to be a typo rather than genuine scope expansion, and correctly dropped SC.

Phase II→III. Label changes at this phase only drop labels whose textual claims lack implementation support. For instance, in `python__mypy-11632`, Phases I and II flagged SC because the PR claims to fix four issues. The code validator found from the diff that these issues are duplicates with one shared root cause, and correctly dropped SC, producing an exact match.

Finding: IC consistently outnumbers CI across both phases and all backbones, confirming net positive label refinement. Phase I→II mainly performs text-level evidence re-verification, while Phase II→III performs implementation-level verification.

6 Discussion

Real-world relevance. To test PaiChecker on live GitHub PR–Issue pairs, we selected nine high-star repositories spanning multiple languages (`home-assistant/core`, `microsoft/vscode`, `rust-lang/rust`, `pandas-dev/pandas`, `pydantic/pydantic`, `huggingface/transformers`, `apache/arrow`, `ray-project/ray`, `ansible/ansible`)

Table 4: Multi-class categorization results (%). EM requires the predicted label set to match the ground truth exactly; Macro metrics average all seven labels. Per-label columns report F1, and best values per backbone are in bold.

			Macro				Per-Label F1						
Backbone	**Method**	**EM**	**Acc.**	**Prec.**	**Rec.**	**F1**	**SC**	**FP**	**DP**	**IS**	**UL**	**Others**	**No Mis.**
SWE-Gym													
GPT-5.3 Codex	Zero-Shot	58.20	88.40	49.36	41.26	41.01	64.70	63.60	37.30	38.80	4.90	0.00	77.80
	Few-Shot	60.21	89.10	56.26	41.14	42.97	67.40	62.10	30.50	40.10	21.60	0.00	79.10
	Chain-of-Thought	60.34	89.20	53.59	49.37	49.81	67.10	70.90	48.70	44.80	38.80	0.00	78.40
	Mini-SWE-Agent	62.02	89.40	62.34	39.63	45.07	62.10	59.50	55.40	42.00	17.70	0.00	78.80
	OpenHands	61.59	90.00	74.80	39.27	46.61	54.16	54.01	11.85	46.96	59.22	7.40	79.37
	Codex	54.36	89.40	71.28	34.87	43.32	51.67	46.02	17.39	46.80	52.39	6.90	75.62
	PaiChecker	**79.78**	**94.80**	**78.39**	**68.07**	**69.49**	**81.40**	**79.40**	**63.40**	**78.00**	**84.20**	**11.10**	**88.90**
Claude-Sonnet 4.6	Zero-Shot	53.16	86.80	47.14	35.14	38.79	58.40	47.60	41.20	30.60	21.50	0.00	72.20
	Few-Shot	55.58	87.70	49.77	39.70	43.29	63.10	61.20	43.50	29.10	24.70	7.40	74.00
	Chain-of-Thought	57.47	88.50	57.83	50.79	53.09	65.60	63.30	50.20	58.70	37.90	20.70	75.20
	Mini-SWE-Agent	65.26	91.50	67.07	55.97	60.34	61.00	62.00	60.00	57.40	71.90	26.70	83.40
	OpenHands	65.88	91.51	67.80	54.77	58.81	65.53	68.26	45.65	63.32	71.47	15.38	82.07
	Claude Code	64.30	90.74	66.24	56.93	59.90	61.68	66.25	52.38	62.35	66.38	29.41	80.84
	PaiChecker	**76.62**	**94.20**	**71.64**	**74.39**	**72.79**	**79.60**	**77.10**	**72.30**	**76.10**	**79.70**	**36.40**	**88.30**
Gemini-3.1 Pro	Zero-Shot	56.15	88.20	58.06	56.94	54.61	67.70	71.00	50.70	61.90	34.90	20.80	75.30
	Few-Shot	66.28	91.20	64.70	59.79	59.31	66.80	63.50	53.90	61.10	65.40	21.30	83.20
	Chain-of-Thought	66.20	91.20	65.74	58.06	60.19	68.90	67.00	52.70	63.70	54.80	30.80	83.40
	Mini-SWE-Agent	69.40	92.40	69.16	57.10	61.36	61.10	68.20	73.20	61.40	70.90	9.10	85.60
	OpenHands	67.50	91.77	69.60	52.67	57.56	66.99	76.84	36.46	63.30	58.82	16.22	84.28
	PaiChecker	**84.66**	**96.20**	**79.60**	**80.46**	**79.21**	**85.10**	**80.00**	**77.60**	**82.80**	**89.60**	**46.50**	**92.90**
Qwen-3.5 Plus	Zero-Shot	62.02	89.60	67.86	43.04	49.04	61.50	58.50	44.40	53.50	13.50	25.00	79.90
	Few-Shot	61.81	89.80	74.11	41.34	49.83	58.20	52.90	40.40	50.70	38.80	21.30	79.20
	Chain-of-Thought	61.48	89.10	66.81	39.36	44.73	55.20	58.70	37.60	52.80	11.40	18.20	79.20
	Mini-SWE-Agent	61.57	89.90	70.57	42.23	50.61	54.80	44.80	51.50	47.60	51.90	25.00	78.70
	OpenHands	54.37	87.81	63.79	25.77	31.24	48.11	30.89	23.23	28.15	13.79	0.00	74.51
	PaiChecker	**70.59**	**92.05**	**75.90**	**52.24**	**58.98**	**74.20**	**68.50**	**52.20**	**54.00**	**52.40**	**27.30**	**84.40**
SWE-bench Multilingual													
GPT-5.3 Codex	Zero-Shot	73.00	92.38	41.13	34.23	30.44	67.57	-	15.38	18.18	25.64	-	86.29
	Few-Shot	75.00	92.86	41.19	35.77	31.14	72.00	-	15.38	10.00	34.29	-	86.28
	Chain-of-Thought	79.67	94.14	47.14	38.56	35.43	77.78	-	8.00	28.57	44.44	-	89.22
	Mini-SWE-Agent	83.33	95.33	49.04	35.91	38.69	79.41	-	27.59	28.57	42.86	-	92.41
	OpenHands	81.00	94.43	53.97	32.43	35.77	66.67	-	8.00	19.05	66.67	-	90.02
	Codex	80.67	94.14	57.50	30.15	34.23	58.82	-	8.00	33.33	50.00	-	89.47
	PaiChecker	**88.00**	**96.33**	**82.22**	**70.91**	**71.85**	**82.19**	-	**28.57**	**63.64**	**84.21**	-	**94.37**
Claude-Sonnet 4.6	Zero-Shot	45.33	83.76	32.56	34.49	29.48	58.14	-	55.00	25.64	4.96	-	62.64
	Few-Shot	60.33	88.81	36.56	37.91	34.33	67.61	-	60.00	24.24	10.81	-	77.66
	Chain-of-Thought	67.33	90.33	36.48	35.66	31.63	71.88	-	51.43	0.00	15.58	-	82.49
	Mini-SWE-Agent	80.67	92.29	47.60	51.28	48.03	70.97	-	79.07	40.00	53.85	-	92.31
	OpenHands	70.67	91.67	39.93	42.80	37.68	63.64	-	51.43	33.33	32.43	-	82.90
	Claude Code	68.67	90.95	37.62	35.30	33.52	61.54	-	47.06	28.57	16.67	-	80.82
	PaiChecker	**88.33**	**97.05**	**62.40**	**72.33**	**65.95**	**84.85**	-	**87.50**	**63.16**	**81.82**	-	**94.33**
Gemini-3.1 Pro	Zero-Shot	55.33	80.10	45.56	52.64	42.64	68.85	-	63.16	32.26	11.36	-	72.82
	Few-Shot	71.00	84.95	44.43	40.22	39.43	66.67	-	57.14	33.33	33.33	-	85.51
	Chain-of-Thought	78.00	89.10	46.66	41.62	42.26	77.19	-	55.56	45.71	27.27	-	90.09
	Mini-SWE-Agent	57.00	64.14	59.67	30.02	38.95	59.57	-	52.94	34.78	50.00	-	75.38
	OpenHands	82.67	95.14	53.45	41.39	42.23	78.12	-	28.57	54.55	42.86	-	91.49
	PaiChecker	**90.67**	**95.19**	**82.80**	**64.68**	**71.77**	**82.76**	-	**81.82**	**81.82**	**94.12**	-	**95.24**
Qwen-3.5 Plus	Zero-Shot	79.33	94.00	38.84	26.25	28.57	58.62	-	34.48	17.39	0.00	-	89.53
	Few-Shot	79.33	94.10	45.69	29.26	32.53	64.29	-	28.57	19.05	26.67	-	89.16
	Chain-of-Thought	78.67	93.10	43.63	24.56	26.74	52.17	-	15.38	30.77	0.00	-	88.84
	Mini-SWE-Agent	73.00	77.67	**56.29**	22.61	28.64	37.21	-	40.00	11.11	20.00	-	75.97
	OpenHands	79.67	93.48	40.26	23.88	26.69	58.82	-	28.57	10.53	0.00	-	88.93
	PaiChecker	**83.33**	**95.24**	50.40	**37.89**	**42.38**	**73.68**	-	**63.16**	**31.58**	**37.50**	-	**90.76**

and sampled 200 open PRs (post-Jan. 2026) linked to at least one issue. We ran PaiChecker on each pair and reported flagged cases as GitHub comments for maintainer confirmation or dispute. Among the 200 pairs, PaiChecker flagged 78 potential misalignments. As of writing, repository developers have responded to 17 reports, confirming 16 reports, a 94% confirmation rate among responses. This suggests that PaiChecker's detections can transfer reliably from curated benchmarks to real-world development activity. We retain original PR identifiers and outputs in our replication package [13].

Token consumption and cost. Across all backbones, PaiChecker averages 42K–59K tokens per instance, about 2× Mini-SWE-Agent (~27K) and 4× zero-shot prompting (~12K). Under current API pricing, full SWE-Gym processing costs $191–$569 in total, or $0.08–$0.23 per instance on average, which remains affordable relative to manual annotation. Costs vary with issue difficulty and PR complexity: cases resolved mostly from textual evidence cost only a few cents, whereas complex cases involving large diffs or extensive code validation can exceed $1. The code validator accounts for about 40% of per-instance tokens on average. Its added cost is typically

Table 5: Ablation study results (backbone: Gemini-3.1-Pro-Preview). BA means Binary Accuracy; EM means Exact Match. Δ denotes the change relative to the full pipeline.

Variant	BA	Δ	EM	Δ
PaiChecker (full)	**92.12**	—	**84.66**	—
w/o Phase I (all 3 subagents)	67.83	−24.29	64.78	−19.88
w/o Phase II (Coordinator)	87.41	−4.71	77.44	−7.22
w/o Phase III (Code Validator)	88.97	−3.15	79.00	−5.66
w/o Issue Analyzer	84.58	−7.54	72.64	−12.02
w/o PR Scope Analyzer	73.26	−18.86	61.16	−23.50
w/o PR Conn. Analyzer	85.77	−6.35	74.82	−9.84
w/o Issue & PR Scope Analyzers	59.32	−32.80	53.52	−31.14
w/o Issue & PR Conn. Analyzers	81.65	−10.47	71.25	−13.41
w/o PR Scope & PR Conn. Analyzers	67.91	−24.21	59.40	−25.26
w/o Coordinator & Code Validator	85.60	−6.52	77.21	−7.45

Table 6: Label change statistics per phase per backbone. IC = corrections (Incorrect→Correct), CI = regressions (Correct→Incorrect). Percentages are relative to total label changes at each phase.

	Phase I→II (Coordinator)			Phase II→III (Code Validator)		
Backbone	**IC**	**CI**	**Total**	**IC**	**CI**	**Total**
GPT-5.3-Codex	11 (91.7%)	1 (8.3%)	12	100 (72.5%)	38 (27.5%)	138
Claude-Sonnet-4.6	74 (63.2%)	43 (36.8%)	117	125 (88.7%)	16 (11.3%)	141
Gemini-3.1-Pro	58 (75.3%)	19 (24.7%)	77	67 (76.1%)	21 (23.9%)	88
Qwen-3.5-Plus	21 (63.6%)	12 (36.4%)	33	40 (72.7%)	15 (27.3%)	55

modest ($0.02–$0.1 per instance), but can approach $0.8 for outlier cases with large patches and multiple candidate labels. Each instance completes in 30–100 s, and throughput can scale linearly since instances are independent.

Backbone mixing. Our evaluation uses one backbone for all agents. Assigning different models to different agents (*e.g.* a reasoning-strong coordinator and code-proficient validator) may improve accuracy or reduce cost, but introduces a combinatorial search space.

PR description–code misalignment. Such misalignment can take two forms. First, a PR may describe changes not reflected in the code; this is already filtered by PaiChecker. Second, the code may silently introduce unrelated changes despite an aligned description. In practice, most repositories enforce code review, which our system incorporates as input. It is uncommon for entirely unrelated changes to pass review without any discussion, making such cases exceedingly rare with negligible impact on the overall results.

7 Threats to Validity

Tool-dependent Task Accessibility Task validity may depend on the tools available to an agent. For example, an issue that relies on an external link is valid under our definition if developers do not raise questions. However, it becomes invalid for agents without web search or link-following tools. Our analysis does not evaluate task validity under all possible tool configurations, which is a limitation of this study.

Internal Validity The primary threat is annotation subjectivity. We mitigate this through (1) a detailed annotation guideline with explicit boundary criteria, (2) dual-annotator independent labeling, and consensus resolution with a third annotator for disagreements.

External Validity Both datasets draw from a limited set of GitHub repositories. However, the misalignment patterns stem from general process-level factors such as how rigorously teams enforce issue–PR traceability, rather than language or domain-specific features, suggesting the taxonomy transfers to repositories with similar development workflows. The *Others* label further provides a way to capture uncategorized cases.

8 Related Work

SWE-bench-like Benchmarks. SWE-bench [29] introduced real-world GitHub issue resolution as an LLM evaluation paradigm, spawning many derivatives. SWE-bench Verified [23] adds human validation; SWE-bench-Live [52] and SWE-Rebench [17] address contamination through live updates and automated extraction; SWE-Smith [48] generates synthetic instances; SWE-Gym [32] curates runnable training instances; and SWE-Bench-Pro [25] targets harder tasks. Other variants, including SWE-PolyBench [35], SWE-Perf [27], SWE-Bench++ [40], SWE-Evo [39], SWE-Fixer [45], SWE-Flow [53], SWE-Universe [22], and SWE-Lego [37], extend the paradigm along other dimensions.

Benchmark Quality and Reliability. Benchmark reliability is a longstanding concern, with prior work highlighting test oracle insufficiency [42, 51], dataset contamination [14], and solution leakage [15]. SWE-ABS [50] strengthens test suites adversarially, while SPICE [30] automates labeling for SWE-bench-style datasets.

Comparison with OpenAI's Analysis. OpenAI's SWE-bench analysis [14] focuses on contamination and test coverage. This overlaps with parts of our taxonomy, such as PR Scope Creep, but not the construction-level root cause: a linked PR may not be intended to fix only the linked issue. Our taxonomy also covers cases beyond test coverage, such as defective PRs used as gold patches.

Comparison with SPICE. SPICE [30] follows the SWE-bench Verified annotation standard, prompting LLMs to transfer its quality criteria to SWE-Gym. In contrast, PaiChecker audits SWE-bench Verified itself and labels a more fundamental construction dimension. We compare PaiChecker with SPICE [30] through both quantitative and qualitative studies. *(1) Quantitative comparison.* We compare on the 644 instances whose instance IDs appear in both SPICE outputs and our annotations. On this overlap, SPICE flags 143 as problematic and PaiChecker flags 281 as misaligned, with 78 overlapping cases, 65 SPICE-only cases, and 203 PaiChecker-only cases.*(2) Qualitative comparison.* `dask__dask-7305` is detected only by PaiChecker: the gold PR fixes a bug but introduces a regression, which PaiChecker flags as DP; SPICE misses it because a clear issue and matching tests obscure the weakened oracle. `dask__dask-6862` is detected only by SPICE: the issue proposes `get_if_none`, and the PR implements the same semantics via `override_with`; SPICE flags poor scoping due to API differences, while PaiChecker identifies no misalignment. `conan-io__conan-16028` is flagged by both: the issue requests optional Git URL hiding, but the PR adds unrelated C-standard utilities; PaiChecker labels it SC, SPICE labels it poorly scoped tests.

9 Conclusion

This paper identifies PR-Issue misalignment as a systematic construction-level problem in SWE-bench-like benchmarks that directly undermines evaluation fairness and reliability. Manual analysis of all 500 SWE-bench Verified instances reveals 13.6% misalign- ment across five patterns and 11 fine-grained scenarios. To enable scalable de- tection, we propose PaiChecker, a three-phase multi-agent checker combining pattern-specific detection, beyond-taxonomy synthesis, and code validation. Evaluated on SWE-Gym and SWE-bench Multilingual, PaiChecker reaches up to 92.12% and 91.67% accuracy, respectively.

Acknowledgment

This paper was supported by the Guangdong Basic and Applied Basic Research Foundation (No. 2024A1515010145) and the Shenzhen Science and Technology Program (Shenzhen Key Laboratory Grant No. ZDSYS20230626091302006).

Data Availability

All data and artifacts are publicly available [13].

References

[1] 2024. SWE-bench Annotation Instructions. https://cdn.openai.com/introducing-swe-bench-verified/swe-b-annotation-instructions.pdf accessed:2026-03-06.
[2] 2024. SWE-bench Verified Official Experiments Result. https://github.com/SWE-bench/experiments accessed:2026-03-06.
[3] 2024. SWE-bench Verified Official Leaderboards. https://www.swebench.com/index.html accessed:2026-03-06.
[4] 2025. Issue Template of Astropy. https://github.com/astropy/astropy/tree/main/.github/ISSUE_TEMPLATE Accessed: 2026-03-25.
[5] 2025. Issue Template of Pytest. https://github.com/pytest-dev/pytest/tree/main/.github/ISSUE_TEMPLATE Accessed: 2026-03-25.
[6] 2025. Issue Template of Sphinx. https://github.com/sphinx-doc/sphinx/tree/master/.github/ISSUE_TEMPLATE Accessed: 2026-03-25.
[7] 2025. Issue Template of Xarray. https://github.com/pydata/xarray/tree/main/.github/ISSUE_TEMPLATE Accessed: 2026-03-25.
[8] 2025. Pull Request Template of Astropy. https://github.com/astropy/astropy/blob/main/.github/PULL_REQUEST_TEMPLATE.md Accessed: 2026-03-25.
[9] 2025. Pull Request Template of Scikit-learn. https://github.com/scikit-learn/scikit-learn/blob/main/.github/PULL_REQUEST_TEMPLATE.md Accessed: 2026-03-25.
[10] 2025. Pull Request Template of Sympy. https://github.com/sphinx-doc/sphinx/tree/master/.github/ISSUE_TEMPLATE Accessed: 2026-03-25.
[11] 2026. Claude Code. https://claude.com/product/claude-code accessed:2026-03-06.
[12] 2026. Codex. https://openai.com/codex/ accessed:2026-03-06.
[13] 2026. *PAIChecker*. https://github.com/manyifire/PAIChecker
[14] 2026. Why SWE-bench Verified no longer measures frontier coding capabilities. https://openai.com/index/why-we-no-longer-evaluate-swe-bench-verified/ accessed:2026-03-06.
[15] Reem Aleithan, Haoran Xue, Mohammad Mahdi Mohajer, Elijah Nnorom, Gias Uddin, and Song Wang. 2024. SWE-Bench+: Enhanced Coding Benchmark for LLMs. doi:10.48550/arXiv.2410.06992
[16] Anthropic. 2025. Claude Sonnet 4.6. https://www.anthropic.com/news/claude-sonnet-4-6 Accessed: 2026-03-25.
[17] Ibragim Badertdinov, Alexander Golubev, Maksim Nekrashevich, Anton Shevtsov, Simon Karasik, Andrei Andriushchenko, Maria Trofimova, Daria Litvintseva, and Boris Yangel. 2025. Swe-rebench: An automated pipeline for task collection and decontaminated evaluation of software engineering agents. *arXiv preprint arXiv:2505.20411* (2025).
[18] Shraddha Barke, Michael B James, and Nadia Polikarpova. 2023. Grounded copilot: How programmers interact with code-generating models. *Proceedings of the ACM on Programming Languages* 7, OOPSLA1 (2023), 85–111.
[19] Nicolas Bettenburg, Sascha Just, Adrian Schröter, Cathrin Weiss, Rahul Premraj, and Thomas Zimmermann. 2008. What makes a good bug report?. In *Proceedings of the 16th ACM SIGSOFT International Symposium on Foundations of software engineering (SIGSOFT '08/FSE-16)*. Association for Computing Machinery, New York, NY, USA, 308–318. doi:10.1145/1453101.1453146
[20] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. 2020. Language models are few-shot learners. In *Proceedings of the 34th International Conference on Neural Information Processing Systems* (Vancouver, BC, Canada) *(NIPS '20)*. Curran Associates Inc., Red Hook, NY, USA, Article 159, 25 pages.
[21] Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, Nick Ryder, Mikhail Pavlov, Alethea Power, Lukasz Kaiser, Mohammad Bavarian, Clemens Winter, Philippe Tillet, Felipe Petroski Such, Dave Cummings, Matthias Plappert, Fotios Chantzis, Elizabeth Barnes, Ariel Herbert-Voss, William Hebgen Guss, Alex Nichol, Alex Paino, Nikolas Tezak, Jie Tang, Igor Babuschkin, Suchir Balaji, Shantanu Jain, William Saunders, Christopher Hesse, Andrew N. Carr, Jan Leike, Josh Achiam, Vedant Misra, Evan Morikawa, Alec Radford, Matthew Knight, Miles Brundage, Mira Murati, Katie Mayer, Peter Welinder, Bob McGrew, Dario Amodei, Sam McCandlish, Ilya Sutskever, and Wojciech Zaremba. 2021. Evaluating Large Language Models Trained on Code. arXiv:2107.03374 [cs.LG] https://arxiv.org/abs/2107.03374
[22] Mouxiang Chen, Lei Zhang, Yunlong Feng, Xuwu Wang, Wenting Zhao, Ruisheng Cao, Jiaxi Yang, Jiawei Chen, Mingze Li, Zeyao Ma, Hao Ge, Zongmeng Zhang, Zeyu Cui, Dayiheng Liu, Jingren Zhou, Jianling Sun, Junyang Lin, and Binyuan Hui. 2026. SWE-Universe: Scale Real-World Verifiable Environments to Millions. doi:10.48550/arXiv.2602.02361 arXiv:2602.02361 [cs].
[23] Neil Chowdhury, James Aung, Chan Jun Shern, Oliver Jaffe, Dane Sherburn, Giulio Starace, Evan Mays, Rachel Dias, Marwan Aljubeh, Mia Glaese, Carlos E. Jimenez, John Yang, Leyton Ho, Tejal Patwardhan, Kevin Liu, and Aleksander Madry. 2024. Introducing SWE-bench Verified. https://openai.com/index/introducing-swe-bench-verified/ accessed:2026-03-06.
[24] Mariana Coutinho, Lorena Marques, Anderson Santos, Marcio Dahia, Cesar França, and Ronnie de Souza Santos. 2024. The role of generative ai in software development productivity: A pilot case study. In *Proceedings of the 1st ACM International Conference on AI-Powered Software*. 131–138.
[25] Xiang Deng, Jeff Da, Edwin Pan, Yannis Yiming He, Charles Ide, Kanak Garg, Niklas Lauffer, Andrew Park, Nitin Pasari, Chetan Rane, et al. 2025. Swe-bench pro: Can ai agents solve long-horizon software engineering tasks? *arXiv preprint arXiv:2509.16941* (2025).
[26] Google. 2025. Gemini 3.1 Pro. https://blog.google/innovation-and-ai/models-and-research/gemini-models/gemini-3-1-pro/ Accessed: 2026-03-25.
[27] Xinyi He, Qian Liu, Mingzhe Du, Lin Yan, Zhijie Fan, Yiming Huang, Zejian Yuan, and Zejun Ma. 2025. Swe-perf: Can language models optimize code performance on real-world repositories? *arXiv preprint arXiv:2507.12415* (2025).
[28] Naman Jain, King Han, Alex Gu, Wen-Ding Li, Fanjia Yan, Tianjun Zhang, Sida Wang, Armando Solar-Lezama, Koushik Sen, and Ion Stoica. 2024. LiveCodeBench: Holistic and Contamination Free Evaluation of Large Language Models for Code. arXiv:2403.07974 [cs.SE] https://arxiv.org/abs/2403.07974
[29] Carlos E Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik R Narasimhan. 2024. SWE-bench: Can Language Models Resolve Real-world Github Issues?. In *The Twelfth International Conference on Learning Representations*. https://openreview.net/forum?id=VTF8yNQM66
[30] Gustavo A. Oliva, Gopi Krishnan Rajbahadur, Aaditya Bhatia, Haoxiang Zhang, Yihao Chen, Zhilong Chen, Arthur Leung, Dayi Lin, Boyuan Chen, and Ahmed E. Hassan. 2025. SPICE: An Automated SWE-Bench Labeling Pipeline for Issue Clarity, Test Coverage, and Effort Estimation. ase 2025. doi:10.48550/arXiv.2507.09108
[31] OpenAI. 2025. Introducing GPT-5.3 Codex. https://openai.com/index/introducing-gpt-5-3-codex/ Accessed: 2026-03-25.
[32] Jiayi Pan, Xingyao Wang, Graham Neubig, Navdeep Jaitly, Heng Ji, Alane Suhr, and Yizhe Zhang. 2024. Training software engineering agents and verifiers with swe-gym. *arXiv preprint arXiv:2412.21139* (2024).
[33] Jorge Pérez, Jessica Díaz, Javier Garcia-Martin, and Bernardo Tabuenca. 2020. Systematic literature reviews in software engineering—enhancement of the study selection process using Cohen's Kappa statistic. *Journal of Systems and Software* 168 (2020), 110657.
[34] Qwen Team. 2025. Qwen 3.5. https://qwen.ai/blog?id=qwen3.5 Accessed: 2026-03-25.
[35] Muhammad Shihab Rashid, Christian Bock, Yuan Zhuang, Alexander Buchholz, Tim Esler, Simon Valentin, Luca Franceschi, Martin Wistuba, Prabhu Teja Sivaprasad, Woo Jung Kim, et al. 2025. Swe-polybench: A multi-language benchmark for repository level evaluation of coding agents. *arXiv preprint arXiv:2504.08703* (2025).
[36] Klaas-Jan Stol, Paul Ralph, and Brian Fitzgerald. 2016. Grounded theory in software engineering research: a critical review and guidelines. In *Proceedings of*

the 38th International conference on software engineering. 120–131.

[37] Chaofan Tao, Jierun Chen, Yuxin Jiang, Kaiqi Kou, Shaowei Wang, Ruoyu Wang, Xiaohui Li, Sidi Yang, Yiming Du, Jianbo Dai, Zhiming Mao, Xinyu Wang, Lifeng Shang, and Haoli Bai. 2026. SWE-Lego: Pushing the Limits of Supervised Fine-tuning for Software Issue Resolving. doi:10.48550/arXiv.2601.01426 arXiv:2601.01426 [cs].

[38] Trae Research Team, Pengfei Gao, Zhao Tian, Xiangxin Meng, Xinchen Wang, Ruida Hu, Yuanan Xiao, Yizhou Liu, Zhao Zhang, Junjie Chen, Cuiyun Gao, Yun Lin, Yingfei Xiong, Chao Peng, and Xia Liu. 2025. Trae Agent: An LLM-based Agent for Software Engineering with Test-time Scaling. doi:10.48550/arXiv.2507.23370

[39] Minh V. T. Thai, Tue Le, Dung Nguyen Manh, Huy Phan Nhat, and Nghi D. Q. Bui. 2025. SWE-EVO: Benchmarking Coding Agents in Long-Horizon Software Evolution Scenarios. doi:10.48550/arXiv.2512.18470

[40] Lilin Wang, Lucas Ramalho, Alan Celestino, Phuc Anthony Pham, Yu Liu, Umang Kumar Sinha, Andres Portillo, Onassis Osunwa, and Gabriel Maduekwe. 2025. SWE-Bench++: A Framework for the Scalable Generation of Software Engineering Benchmarks from Open-Source Repositories. doi:10.48550/arXiv.2512.17419

[41] Xingyao Wang, Boxuan Li, Yufan Song, Frank F Xu, Xiangru Tang, Mingchen Zhuge, Jiayi Pan, Yueqi Song, Bowen Li, Jaskirat Singh, et al. 2025. Openhands: An open platform for ai software developers as generalist agents. In *International Conference on Learning Representations*, Vol. 2025. 65882–65919.

[42] You Wang, Michael Pradel, and Zhongxin Liu. 2025. Are "Solved Issues" in SWE-bench Really Solved Correctly? An Empirical Study. icse 2026. doi:10.1145/3744916.3764576

[43] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed Chi, Quoc Le, and Denny Zhou. 2023. Chain-of-Thought Prompting Elicits Reasoning in Large Language Models. arXiv:2201.11903 [cs.CL] https://arxiv.org/abs/2201.11903

[44] Chunqiu Steven Xia, Yinlin Deng, Soren Dunn, and Lingming Zhang. 2024. Agentless: Demystifying LLM-based Software Engineering Agents. doi:10.48550/ARXIV.2407.01489

[45] Chengxing Xie, Bowen Li, Chang Gao, He Du, Wai Lam, Difan Zou, and Kai Chen. 2025. SWE-Fixer: Training Open-Source LLMs for Effective and Efficient GitHub Issue Resolution. In *Findings of the Association for Computational Linguistics: ACL 2025*, Wanxiang Che, Joyce Nabende, Ekaterina Shutova, and Mohammad Taher Pilehvar (Eds.). Association for Computational Linguistics, Vienna, Austria, 1123–1139. doi:10.18653/v1/2025.findings-acl.62

[46] Junjielong Xu, Ying Fu, Shin Hwei Tan, and Pinjia He. 2025. Aligning the Objective of LLM-based Program Repair. In *icse 2025*. icse 2025. doi:10.48550/arXiv.2404.08877 arXiv:2404.08877 [cs].

[47] John Yang, Carlos E Jimenez, Alexander Wettig, Kilian Lieret, Shunyu Yao, Karthik R Narasimhan, and Ofir Press. 2024. SWE-agent: Agent-Computer Interfaces Enable Automated Software Engineering. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*. https://arxiv.org/abs/2405.15793

[48] John Yang, Kilian Lieret, Carlos E Jimenez, Alexander Wettig, Kabir Khandpur, Yanzhe Zhang, Binyuan Hui, Ofir Press, Ludwig Schmidt, and Diyi Yang. 2025. Swe-smith: Scaling data for software engineering agents. *arXiv preprint arXiv:2504.21798* (2025).

[49] Zonghan Yang, Shengjie Wang, Kelin Fu, Wenyang He, Weimin Xiong, Yibo Liu, Yibo Miao, Bofei Gao, Yejie Wang, Yingwei Ma, Yanhao Li, Yue Liu, Zhenxing Hu, Kaitai Zhang, Shuyi Wang, Huarong Chen, Flood Sung, Yang Liu, Yang Gao, Zhilin Yang, and Tianyu Liu. 2025. Kimi-Dev: Agentless Training as Skill Prior for SWE-Agents. iclr 2026. doi:10.48550/arXiv.2509.23045

[50] Boxi Yu, Yang Cao, Yuzhong Zhang, Liting Lin, Junjielong Xu, Zhiqing Zhong, Qinghua Xu, Guancheng Wang, Jialun Cao, Shing-Chi Cheung, Pinjia He, and Lionel Briand. 2026. SWE-ABS: Adversarial Benchmark Strengthening Exposes Inflated Success Rates on Test-based Benchmark. arXiv:2603.00520 [cs.SE] https://arxiv.org/abs/2603.00520

[51] Boxi Yu, Yuxuan Zhu, Pinjia He, and Daniel Kang. 2025. UTBoost: Rigorous Evaluation of Coding Agents on SWE-Bench. arXiv:2506.09289 [cs.SE] https://arxiv.org/abs/2506.09289

[52] Linghao Zhang, Shilin He, Chaoyun Zhang, Yu Kang, Bowen Li, Chengxing Xie, Junhao Wang, Maoquan Wang, Yufan Huang, Shengyu Fu, et al. 2025. Swe-bench goes live! *arXiv preprint arXiv:2505.23419* (2025).

[53] Lei Zhang, Jiaxi Yang, Min Yang, Jian Yang, Mouxiang Chen, Jiajun Zhang, Zeyu Cui, Binyuan Hui, and Junyang Lin. [n. d.]. SWE-Flow: Synthesizing Software Engineering Data in a Test-Driven Manner. ([n. d.]).

[54] Yuntong Zhang, Haifeng Ruan, Zhiyu Fan, and Abhik Roychoudhury. 2024. AutoCodeRover: Autonomous Program Improvement. In *Proceedings of the 33rd ACM SIGSOFT International Symposium on Software Testing and Analysis*. ACM, Vienna Austria, 1592–1604. doi:10.1145/3650212.3680384